INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

KNOWLEDGE-BASED OPTIMIZATION OF MINERAL GRINDING CIRCUITS

Akbar Farzanegan

Department of Mining and Metallurgical Engineering
McGill University
Montreal, Canada

August, 1998

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

ABSTRACT

The performance of mineral grinding circuits strongly affects downstream processes such as flotation and cyanidation, and grinding is often the single most expensive unit operation. Hence, optimization efforts must be made on a regular basis to maintain and improve its technical and economic efficiency. The focus of this thesis, off-line optimization of grinding circuits, is based on the mathematical modelling of process units such as ball mills and hydrocyclones.

To complete an optimization task, a mineral process engineer must possess skills and knowledge pertaining to the different stages involved in such effort, available software tools and interpretation of results. A prototype knowledge-based system, *G*rinding *C*ircuits *O*ptimization *S*upervisor (GCOS), has been developed in CLIPS (*C* *L*anguage *I*ntegrated *P*roduction *S*ystem) to assist a non-expert mineral process engineer to do off-line optimization studies.

Due to the importance of the correct estimation of back-calculated mill selection function in grinding optimization studies, a spline curve fitting algorithm has been used to improve their quality. The linkage of the algorithms for the selection function estimation, spline curve fitting, selection function scaling for different ball sizes and single ball mill simulation has provided a useful tool, *N*umerical *G*rinding *O*ptimization *T*ools in *C* (NGOTC) for circuit analysis and grinding media size optimization. The smoothed estimated or scaled selection functions can be used subsequently in *B*all *M*illing *C*ircuits *S*imulator (BMCS) to perform full circuit simulations.

Data from a number of mineral processing plants including Agnico Eagle (La Ronde Division), Les Mine Selbaie, Les Mines Casa Berardi, Lupin Mine, Dome Mine

and Louvicourt Mine were used to develop and test NGOTC, BMCS and GCOS. The results of data analysis and circuit simulations of some of these plants are presented, and the impact of some suggested actions is given and discussed.

RÉSUMÉ

L'efficacité d'un circuit de broyage se répercute souvent sur le procédé aval, comme la flottation ou la cyanuration; le broyage lui-même représente souvent le procédé minéralurgique unitaire le plus coûteux. Il est donc normal de chercher à maintenir et même améliorer cette efficacité sur une base régulière. Cette thèse présente un outil permettant de le faire en discontinu, à l'aide d'un modèle des circuits de broyage comprenant comme procédés unitaires broyeurs à boulets et hydrocyclones.

Un ingénieur minéralurgiste chargé d'optimiser un circuit de broyage doit être doté d'un certain savoir et savoir-faire couvrant toutes les étapes d'une telle démarche, l'acquisition de donnée, la simulation et les logiciels disponibles, et l'interprétation des résultats. Nous avons mis au point un système à base de connaissance, "*G*rinding *C*ircuits *O*ptimisation *S*upervisor" (GCOS), rédigé en CLIPS (*C L*anguage *I*ntegrated *P*roduction *S*ystem), pour aider l'ingénieur non-spécialiste dans son travail d'optimisation en discontinu.

Vu l'importance de bien estimer la valeur de la fonction de sélection de broyeurs industriels pour fins d'optimisation, nous avons développé un algorithme de lissage de telles courbes par fonction spline. L'intégration des algorithmes d'estimation de la fonction de sélection, de lissage par spline, d'ajustement de la fonction de sélection en fonction de la dimension des boulets d'ajout, et de simulation d'un broyeur à boulets unitaire a produit un outil de travail permettant l'optimisation l'analyse de circuit et l'optimisation de taille des boulets d'ajout, le logiciel NGOTC (*N*umerical *G*rinding *O*ptimization *T*ools in *C*). Les fonctions de sélection obtenues, estimées directement de données d'opération ou ajustées pour une taille différente de boulets, peuvent ensuite être utilisées par le logiciel BMCS (*B*all *M*illing *C*ircuits *S*imulator) pour simuler dans circuits

de broyage complets.

Nous avons utilisé, pour développer et vérifier les logiciels NGOTC, BMCS et GCOS, une base de données provenant d'Agnico-Eagle (Division Laronde), les Mines Selbaie, les Mines Casa Berardi, Lupin Mine, Dome Mine et Louvicourt Mine. Nous présentons l'analyse d'une partie de ces données et de certaines simulations. Puisque certaines recommandations de GCOS ont été mises en place, nous en présentons et discutons l'impact.

To my dear wife and parents

ACKNOWLEDGEMENTS

I am indebted to many people and organizations for contributing to my research program; without their help it was impossible for me to start and complete my Ph.D. study.

I specially thank my dear wife, Nikta Nobakht Sohi, for her help, understanding and patience during my study.

I thank my supervisor, Professor André R. Laplante, for his kindness, constant support and teaching me fundamental concepts of mineral process engineering. I am very grateful to my co-supervisor, Professor David A. Lowther, for his invaluable instructions especially concerning development of knowledge-based systems. I was privileged to do my Ph.D. thesis under his co-supervision. I must also thank Dr. Gregory Carayannis who introduced me to the field of Artificial Intelligence at the early stages of my work.

I thank Professor Jim A. Finch for offering helpful lectures and short courses in our department and also making many comments and suggestions concerning my Ph.D. program over past years. I also thank Professor Zhenghe Xu and Dr. Ram Rao for their friendship and very informative discussions about carrying out research and development in mineral processing field.

I am grateful to my fellow graduate students in Mining and Metallurgical Engineering Department, particularly Mr. Jinghong Ling and Mrs. Jie Xiao, for creating a friendly environment. I also thank Mr. Raymond Langlois for his help concerning laboratory tests.

Last, I would like to thank the Ministry of Culture and Higher Eduction of Iran for awarding me a scholarship to continue my education. Also, I appreciate Agnico Eagle, Les Mine Selbaie, Les Mines Casa Berardi, Lupin Mine, Louvicourt Mine and the Natural Science and Engineering Research Council of Canada for the financial and technical support of my research.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

CHAPTER 2

OPTIMIZATION OF MINERAL GRINDING

CIRCUITS

CHAPTER 3

KNOWLEDGE-BASED SYSTEMS THEORY AND APPLICATIONS

CHAPTER 4

NUMERICAL GRINDING OPTIMIZATION TOOLS IN C (NGOTC)

CHAPTER 5

BALL MILLING CIRCUITS SIMULATOR (BMCS)

CHAPTER 6

GRINDING CIRCUITS OPTIMIZATION SUPERVISOR (GCOS)

CHAPTER 7

TESTING OF GRINDING CIRCUITS OPTIMIZATION SUPERVISOR

CHAPTER 8

SUMMARY AND CONCLUSIONS

List of Figures

List of Tables

List of Appendices

CHAPTER 1
INTRODUCTION

1.1 Background

The optimization of mineral grinding circuits using steady-state simulators is an established methodology in both theory and practice. Mineral processors are increasingly using computer simulations to assess the effect of changing critical process variables such as fresh feed flow rate, % solids, grinding media size and ore grindability on the circuit performance. Although current mineral processing simulators, particularly those constructed for comminution, incorporate the necessary mathematical models of processing units, their routine use by practising engineers faces several hurdles such as the complexity of concepts on which models are based and developed and the difficult use of various software tools required for building process models and subsequent simulations. In general, *K*nowledge-*B*ased *S*ystems (KBSs) have been used to remove such obstacles in computer-aided design and optimization systems by providing a better tool to communicate with the system users.

The development of mathematical models and simulation structures of grinding circuits has been an active research area in the field of mineral processing over the past three decades [Napier-Munn and Lynch 1992]. A number of models have been reported in the literature which can represent ball mills in open circuits and predict their grinding performance adequately. These models are well capable of simulating the effect of changes in the most critical operating variables such as feed size distribution and rate. Nevertheless, one of the limitations of the current ball mill simulators [Morrell 1990] is their inability to take into the consideration the effect of ball size, which is very

important in plant optimization. Earlier simulators at McGill also lacked this capability.

Recent applications of KBSs for further optimization of mineral processing plants have proved their merits in achieving increased productivity [Herbst et al. 1989; Harris and Meech, 1990; Herbst et al. 1995]. Inevitably, more industrial plants will use this technology to implement automatic monitoring and control systems. KBSs have been used at the optimization level of the hierarchy of control systems. Generally speaking, the increased productivity achieved by KBSs, even when they are applied to the best-operated plants, stems from their capability in solving problems that cannot be addressed appropriately by the traditional computer programming techniques. Theoretically, a combination of KBSs and computer simulation technologies must provide a more powerful tool for optimization studies.

1.2 Optimization of Grinding Circuits

Ideally, the software used for grinding circuit optimization must provide tools critical to model building and simulation such as those for mass balancing, parametric estimation, process unit modelling and full circuit simulation. Simulation-based optimization of grinding circuits primarily requires building mathematical models of all the processing units involved. Then, the models are linked together in an appropriate simulation structure which must be capable of predicting the effect of circuit configuration and operating conditions on the metallurgical performance. Obviously, the software becomes more complex by introducing additional tools for increased capabilities. To get the most benefits, the end users, i.e. process engineers, have to learn how to use this software effectively.

An overview of the modelling of important process units in fine grinding circuits, for example ball mills and hydrocyclones, and the main aspects involved in a full grinding circuit simulation is given in the following sections.

1.2.1 Modelling of Ball Mills

Ball mills are normally used for fine grinding. The most important design and operating parameters of ball mills are shown in Figure 1.1.

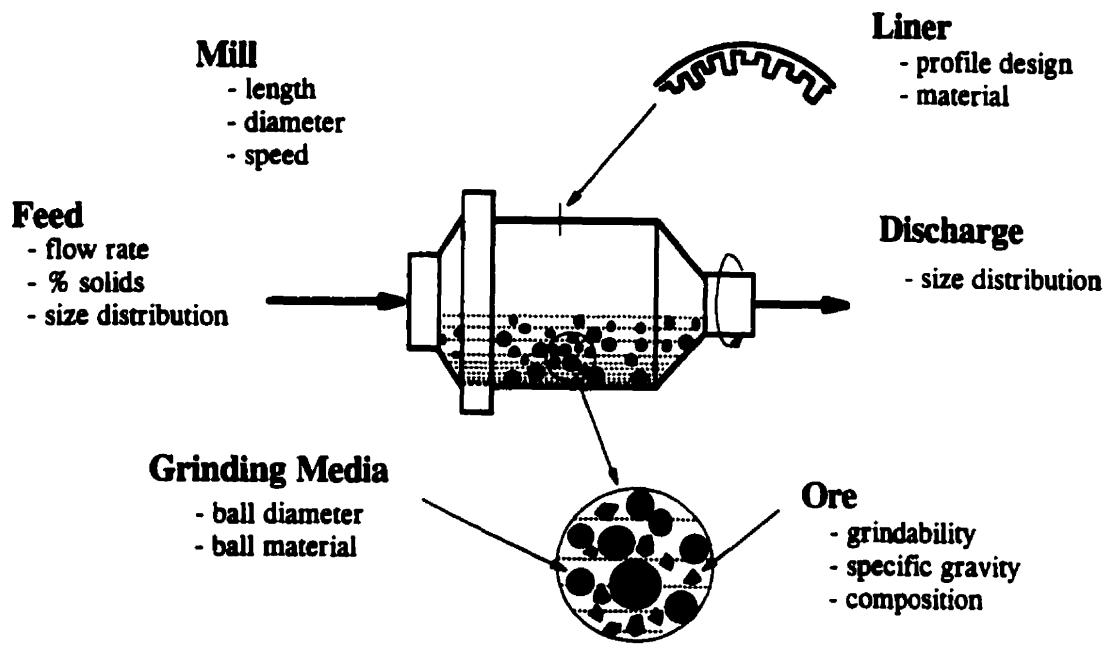


Figure 1.1 Major design and operating parameters of ball mills

Rittinger, Kick and Bond developed equations referred to as the "laws" of grinding which relate energy and size reduction [Kelly and Spottiswood 1982]. The third law, Bond's, is an empirical relationship often used for designing ball mills and analysing grinding performance. As an extension to the Bond work index analysis, McIvor et al. [1990] proposed an approach called functional performance analysis which uses circuit size and classification data to improve the performance of ball milling circuits.

Nevertheless, for the last thirty years, many researchers have worked on the

modelling of ball mills based on the principles of *P*opulation *B*alance *M*odelling (PBM) theory. These models are mostly referred to by the name of developers or the nature of models. Knowledge about these models can be found in scientific journals on mineral processing and related fields. From a software engineering point of view, a grinding optimization software should ideally include more than one model for ball mills. This is of course a great help in off-line optimization studies as there is a possibility that the investigator is unaware of the already developed models and the circumstances under which these models could be applied.

1.2.2 Modelling of Hydrocyclones

Hydrocyclones are used to classify solid particles into fine and coarse products based on a cut size. These devices are normally operated in closed circuit with ball mills and their performance significantly affects the overall grinding circuit efficiency; therefore, their modelling must be considered as an integral part of simulation studies. Figure 1.2 shows the most critical design and operating parameters of hydrocyclone classifiers. The performance of hydrocyclones depends on parameters such as ore composition, cyclone geometry and operating conditions (e.g. feed rate and % solids). Plitt [1976] and Lynch and Rao [1975] have developed empirical models of hydrocyclones which have been used successfully in industrial simulation.

1.2.3 Simulation of Grinding Circuits

Industrial closed-circuit grinding operations consist of ball mill and hydrocyclone units configured based on various flowsheet designs such as those shown in Figure 1.3. The circuit operation totally depends on performance of individual process units. To study a grinding circuit as a single system, a steady-state simulator may be used to predict circuit performance under various configurations and operating conditions. Circuit simulation needs the linking of mathematical counterparts of processing units using information about the material streams that connect the units in the circuit [Motard, Shacham and Rosen 1975]. Sequential simulation is one of the methods used for

flowsheet analysis. In these simulators, an iterative procedure is used to solve recycle streams [Richardson, Coles and White 1981].

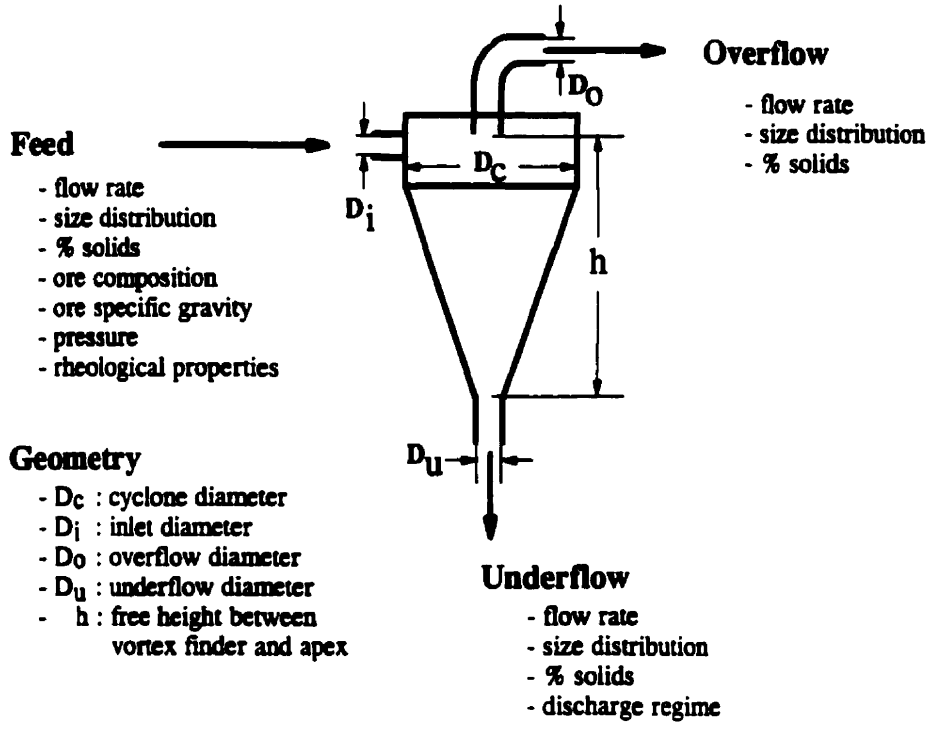


Figure 1.2 Major design and operating parameters of hydrocyclones

1.3 KBSs

Although accurate models of process units are fundamental to building successful grinding simulation and optimization tools, they would have no real use if they could not be incorporated in computer programs developed based on the standard software engineering practices. In reality, engineering problem solving strategies require the ability to process both numeric and symbolic knowledge. *A*rtificial *I*ntelligence (AI) has mainly attempted to enable computers to process symbolic knowledge in addition to their

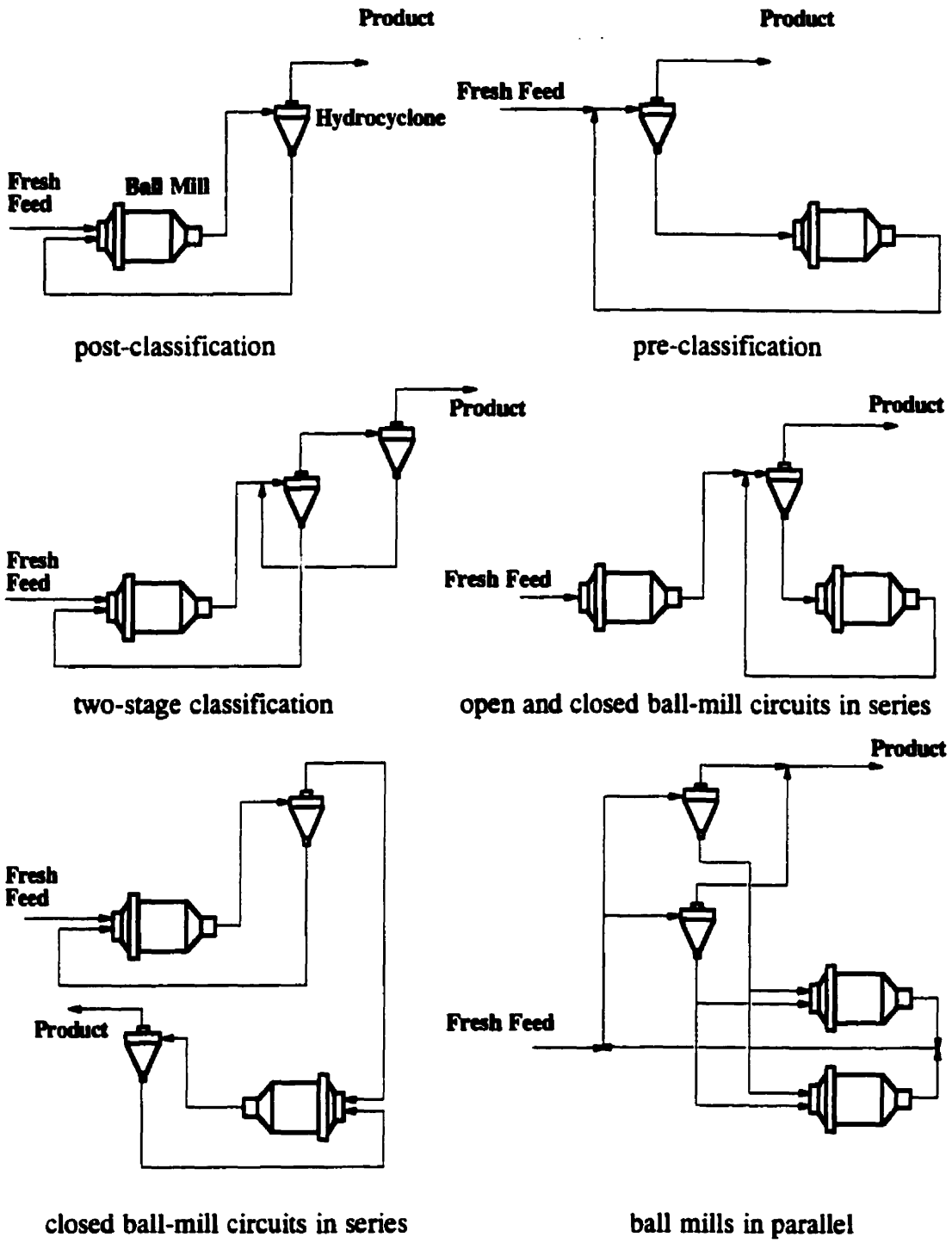


Figure 1.3 Typical configurations of ball mill/hydrocyclone circuits

traditional capability to process numerical data. In recent years, KBSs have been successfully applied to many real world problems in different fields.

1.3.1 Concepts of KBSs

KBSs are intelligent computer programs that use knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solutions [Barr and Feigenbaum 1981]. Unlike conventional computer simulation systems that model a physical process, a KBS models a human expert. In other words, a KBS models the problem solving capability of a human expert.

1.3.2 Structure of KBSs

A simple structure of KBSs has been shown in Figure 1.4. The complexity of the system architecture depends on the application. A full-fledged KBS may have other components such as natural language understanding, an explanation facility, and a knowledge acquisition tool.

The knowledge base of a KBS is a repository of domain knowledge in different forms such as facts, frames (objects) and rules. KBSs are often classified according to their dominant knowledge representation scheme such as *R*ule-*B*ased *S*ystems (RBSs) and *F*rame-*B*ased *S*ystems (FBS). Rule-based representation is considered as one of the best available means for codifying the problem-solving know-how of human experts [Hayes-Roth 1985]. In many domains, however, frames are needed to represent knowledge about objects. A frame [Fikes and Kehler 1985] provides a structured representation of an object or a class of objects. The attributes and behavioral properties of an object are represented as slots and methods in the related frame.

The inference engine models the process of human reasoning [Durkin 1994]. Its main purpose is finding matches between facts in the working memory and the domain knowledge in the knowledge base to draw new conclusions about the problem. A

glossary of KBSs is provided in Appendix A.

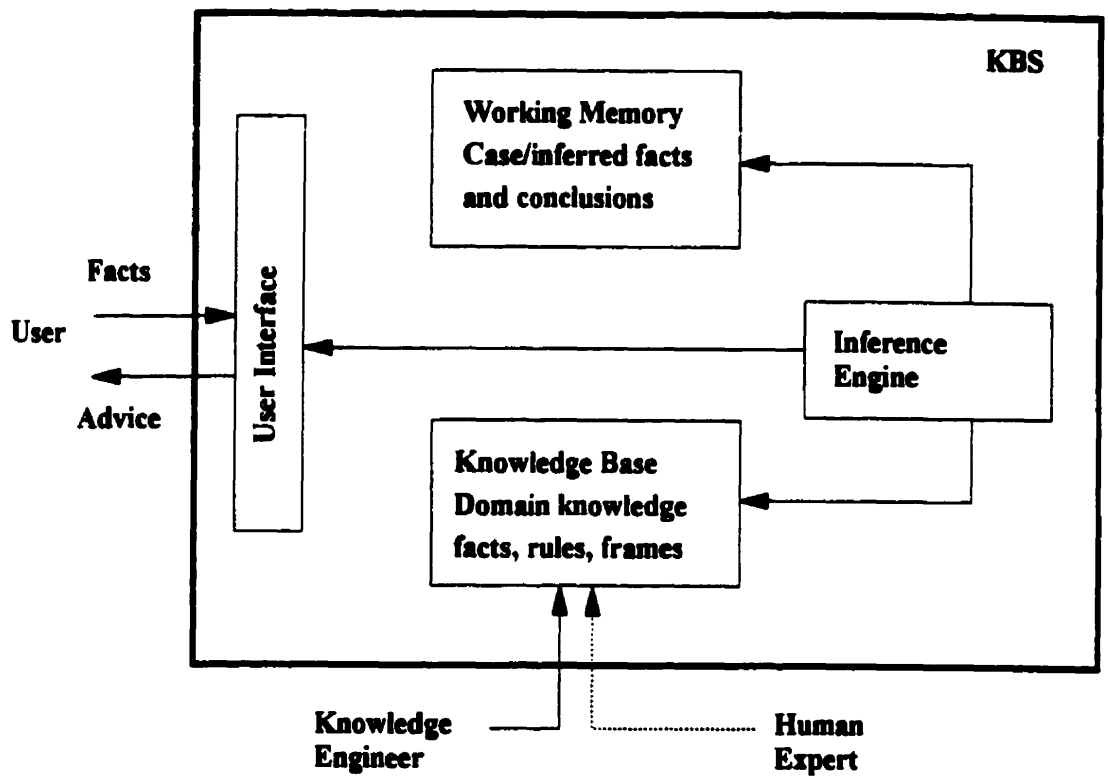


Figure 1.4 A typical structure of knowledge-based systems

1.3.3 Development Tools of KBSs

Developers can use numerical-oriented languages such as FORTRAN and PASCAL, or symbol-manipulation languages such as LISP and PROLOG to implement KBSs projects [Waterman 1986]. Also, knowledge engineering languages, e.g., shells, and general-purpose tools which have a ready-made inference engine and support specific knowledge representation methods, offer many facilities for building KBSs. While programming languages provide more flexibility for designing KBSs, shells and general-

purpose tools have many advantages which may greatly reduce development time and cost.

Knowledge engineers are faced with the critical decision of selecting the most suitable KBS development tool for project implementation, which requires much information about the knowledge structure of the problem domain. For this reason, knowledge engineers must perform a task analysis which helps to specify the necessary features of the KBS tool required. More details about KBSs development tools can be found elsewhere [Hayes-Roth, Waterman and Lenat 1983; Waterman 1986].

CLIPS (*C Language Integrated Production System*) was selected as the KBS development tool in this project. Although CLIPS lacks some of the features of industrial KBSs, it is currently used for research projects by universities. The relevant aspects of CLIPS will be discussed in Sections 3.7.1 and 6.4.1.

1.4 Knowledge Engineering of Grinding Optimization

In grinding optimization, a mineral processor performs a number of general engineering tasks such as synthesis, analysis, interpretation, diagnosis, monitoring and control. KBSs have been used to solve these kind of problems. Regardless of their internal architecture, KBSs use both qualitative (heuristic or shallow knowledge) and quantitative (equation or numerical-oriented or deep knowledge) methods to accomplish tasks involved in optimization problems [Reuter and Van Deventer 1992].

Figure 1.5 shows various stages of a grinding circuit optimization study. A study is started by defining the problem and setting optimization objectives. Then, a plant survey is designed to get samples from various streams and collect all required information such as plant design specification and operating conditions. The samples must be analyzed to determine their particle size distributions and % solids. A mass balance is normally performed to adjust size distributions and % solids data. Next, the

parameters of the selected grinding and classification models are estimated. The model predictions then will be validated by comparing them with the measured data. If models are validated, they will be used for circuit simulations.

A knowledge-based system is designed to guide a novice process engineer to do each stage.

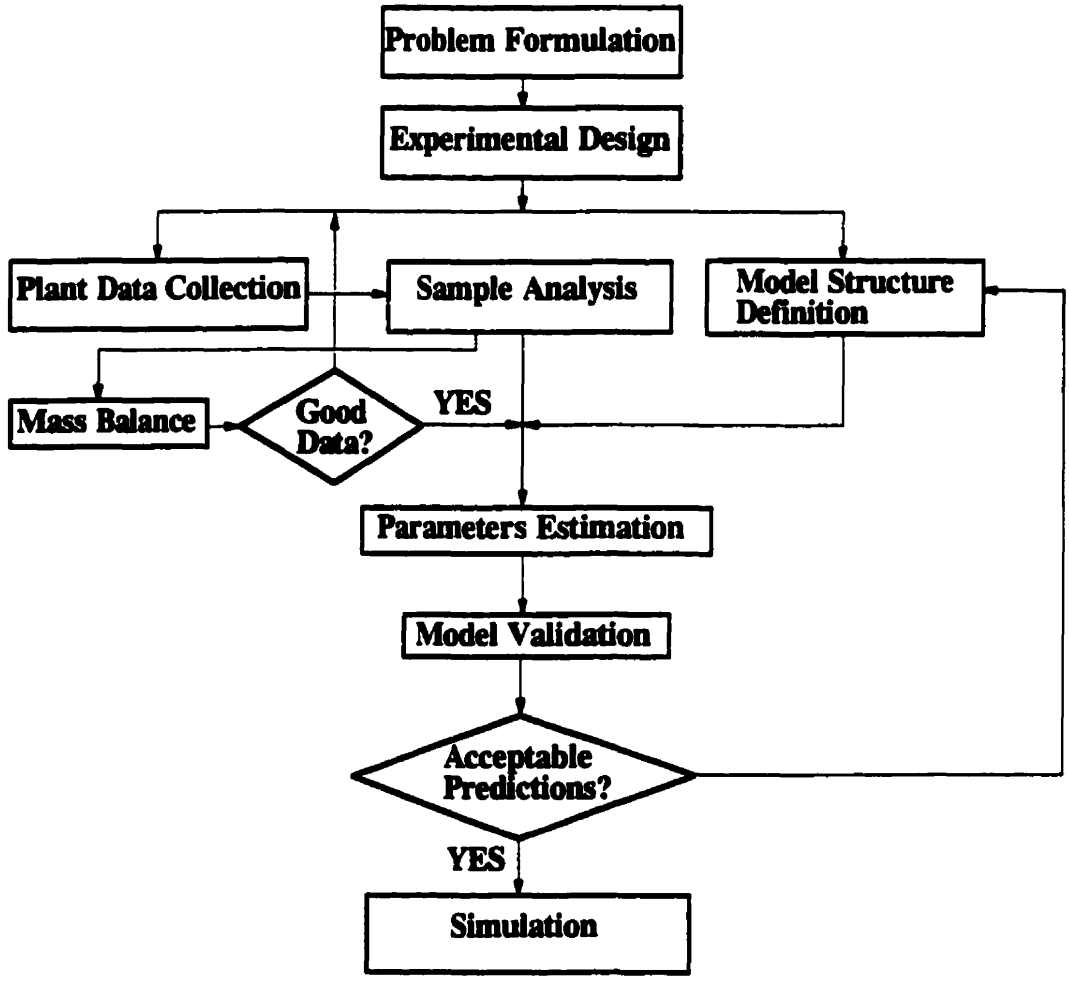


Figure 1.5 Various steps in mineral grinding circuits optimization

The present research has been primarily focused on the use of KBSs for the off-line optimization of grinding circuits rather than the on-line monitoring and control applications. Figure 1.6 shows how KBSs can be integrated with the grinding analysis and simulation tools at various levels of complexity. While currently available level-1 grinding simulation packages, such as SPOC [Laguitton 1985], JKSimMet [Napier-Munn and Lynch 1992] and USIM PAC [Durance et al. 1993], have made remarkable contributions to off-line optimization of mineral processing plants, systems which integrate these conventional software and KBSs should provide more coherent problem solving tools.

Simulation-based grinding optimization requires special software tools developed for parametric estimation and subsequent simulations (level 1). Optimization can also be guided by a KBS; in simple packages, the analysis and simulation software are not explicitly linked to the KBS (level 2). The various programs should be linked under a single interface and capable of interacting with each other (level 3). At the next level, the system would be equipped with a natural language processor capable of communicating in natural language. The present research work is mainly oriented towards applications at level 3.

1.5 Scope of the Research

The thesis is focused on grinding circuit optimization based on the concepts of PBM and KBSs. In terms of implementation of these concepts, a series of computer programs was developed, using procedural and rule-based languages, which can be used to solve off-line grinding optimization problems. The first program, *N*umerical *G*rinding *O*ptimization *T*ools in *C* (NGOTC), is a package of several software modules written in C, and is used for estimating selection function parameters and optimizing ball size. The second program, *B*all *M*illing *C*ircuits *S*imulator (BMCS), is a sequential-modular simulator used to calculate the performance of ball mill/hydrocyclone circuits. The third program, *G*rinding *C*ircuits *O*ptimization *S*upervisor (GCOS), was developed using

CLIPS to assist a novice mineral process engineer to complete a grinding optimization task by giving appropriate advice during each step of the exercise. Several industrial plant investigations were performed as a means of knowledge acquisition for developing and testing GCOS, NGOTC and BMCS. In addition, a number of industrial grinding circuits will be studied whose data were also used in testing the developed software.

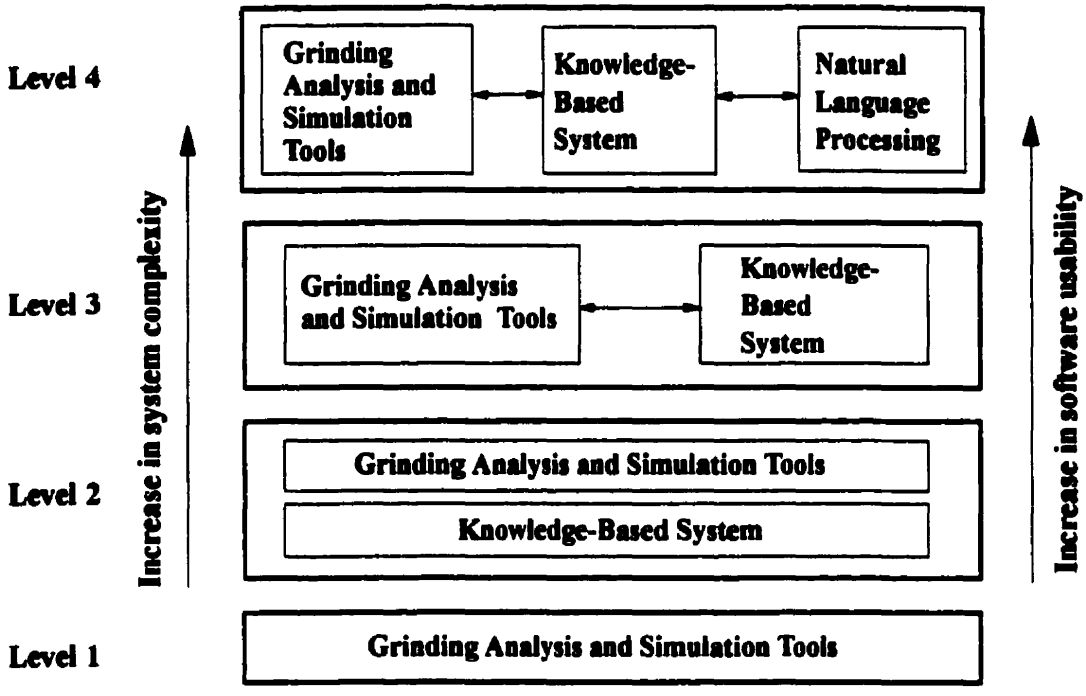


Figure 1.6 The integration of grinding analysis and simulation tools with knowledge-based systems

1.6 Contributions to Original Knowledge

This research is multidisciplinary in nature, as it includes concepts of mineral grinding optimization and knowledge-based systems. The thesis targets the concepts of computer-aided grinding optimization and how these concepts can be accommodated

effectively within software structures. The development of mathematical models of process units, which is the fundamental part of constructing grinding simulation programs, has been discussed elsewhere [Mular 1972] and is not addressed in this thesis. Neither will problems which inherently belong to the pure artificial intelligence and knowledge-based systems fields.

The most important contribution is a coupling between KBSs and conventional grinding optimization tools, which will be used to demonstrate and study how the qualitative problem solving ability of grinding human experts can be incorporated in the present simulation packages. It should be noted that expert systems developed so far have not approached the model building process itself and off-line steady-state simulation as demonstrated in this thesis. Indeed, they have been implemented for both off-line and on-line applications from a plant monitoring and control point of view rather than a help for off-line optimization studies.

The other contribution made is adding a new tool for ball size optimization within an integrated grinding optimization environment which includes selection function estimation and ball mill simulation functionalities. The ball size optimization software has been developed by the integration of several algorithms: selection function estimation, Morrell's scaling procedure, spline curve fitting and ball mill simulation.

In summary, the following contributions are claimed have been made to mineral processing field as a result of this thesis research:

1. applying KBSs approach to automate or computerize problem solving knowledge or expertise used in optimization of mineral grinding circuits
2. development of a rule-based system to supervise the grinding optimization process
3. development of a program to smooth the estimated industrial mill selection

functions using a spline curve fitting algorithm

4. development of the source code to optimize grinding media size using Morrell's selection function scaling procedure and its integration with the selection function estimation and ball mill simulation programs
5. doing case studies both to expand the database and evaluate developed programs

1.7 Claims of Originality

1. identification of types of knowledge required for off-line optimization of grinding process from a knowledge engineering point of view
2. development of a knowledge base, GCOS, using CLIPS which incorporates the formalized grinding optimization knowledge in form of rules and frames to function as a front end of the simulation packages, NGOTC and BMCS
3. development of NGOTC as an integrated software for grinding optimization studies which allows mill selection functions to be estimated for currently used ball size and then scaled if a new make-up or top ball size used.
4. integration of a spline curve fitting algorithm in NGOTC which permits the use of the standard deviations associated with size-by-size selection function data when multiple sets of estimated selection function are available. This allows more accurate estimations or predictions of selection functions to be made.
5. development of a new sequential-modular simulation package, BMCS, which is capable of predicting performance of circuits without any restrictions on circuit flowsheet complexity in terms of number of ball mills, hydrocyclones and streams
6. design of an integrated software for grinding circuits modelling and steady-state simulation supervised by a knowledge-based system approach

1.8 Organization of the Thesis

The thesis is presented in two volumes. Volume 1 is the main text which documents various aspects of the thesis. Volume 2 contains the source codes of the three computer programs developed by the candidate. The first volume is divided into eight chapters. An introduction is given in Chapter 1. The theory and practice of mineral grinding optimization will be reviewed in Chapter 2. Then, the underlying concepts of KBSs and their applications in industrial grinding operations will be explained in Chapter 3. A detailed description of NGOTC, BMCS and GCOS programs and their applications in a number of optimization studies are given in Chapters 4, 5 and 6, respectively. The results of testing GCOS using information obtained from the industrial plant data analyses will be presented in Chapter 7. The thesis will be summarized and concluded in Chapter 8. The second volume is divided into three sections which include the source codes of NGOTC, BMCS and GCOS.

CHAPTER 2
OPTIMIZATION OF MINERAL GRINDING CIRCUITS

2.1 Introduction

In this chapter some theoretical aspects of grinding optimization will be discussed. Before attempting to apply a *K*nowledge-*B*ased *S*ystems (KBS) approach to the grinding optimization domain, the underlying theories should be well understood. Most of the mathematical equations described in next sections to model ball mills and hydrocyclones have been used to develop *N*umerical *G*rinding *O*ptimization *T*ools in *C* (NGOTC) and *B*all *M*illing *C*ircuits *S*imulator (BMCS) software. In practice, application of mathematical unit models in process analysis and simulation requires a good engineering judgement which can be effectively automated by KBSs presented in Chapter 3.

A wet industrial grinding circuit consists of grinding and classification units configured according to a specific arrangement. Mathematical modelling of ball mills and hydrocyclone classifiers has allowed the study of ball milling circuits under various configurations and operating conditions, which significantly affect grinding performance. For example, using two-stage classification instead of single-stage would result in more efficient operation in terms of increased energy savings and reduced overgrinding [Dahlstrom and Kam 1988, 1992]. Operating variables such as the fresh feed rate, the circulating load and slurry density are also important factors that govern the optimal behaviour of grinding circuits and can be studied by unit process modelling and circuit simulation.

The mathematical modelling of grinding as a breakage process has evolved around two central points of view: (1) an energy-based modelling approach which considers energy input and its relationship to size reduction and (2) a *P*opulation *B*alance *M*odelling (PBM) approach which considers the principle of population conservation* to describe breakage.

Optimization techniques target improving plant performance through better utilization of energy and equipment. In plant operation, optimization leads to better yields of valuable products (or reduced yields of contaminants), reduced energy and reagent consumption, higher production rates, and fewer shutdowns. In this thesis, optimization generally refers to any attempt to improve a grinding circuit design and operation, though the result is not guaranteed to be the best possible (optimal) state of the circuit design or operation. Optimization can be effected by applying off-line (circuit analysis and simulations) or on-line (manual or automatic control strategies) techniques. Optimization can be limited to a particular segment of a plant, local optimization; or it can be plant-wide, global optimization. The main focus of this thesis is off-line optimization investigations, applied either separately to breakage and classification processes as sub-systems of the underlying grinding circuit or both as a whole system.

2.2 Energy-Based Modelling

Early attempts to model comminution process were based on the empirical relationships between energy and size reduction. Rittinger, Kick and Bond formulated equations referred to as "laws" of grinding [Kelly and Spottiswood 1982] based on experimental observations.

Rittinger's equation relates the breakage energy required to produce a specified amount of fresh surface per unit of mass:

*What actually is considered to be conserved is particles' mass rather than their number (or population).

$$E = K\left(\frac{1}{x_o} - \frac{1}{x_i}\right) \tag{2.1}$$

where E is specific energy, K a proportionality constant, x_i particle size before breakage and x_o particle size after breakage. Kick proposed the following equation which relates specific energy to change in size of particles:

$$E = K \ln\left(\frac{x_i}{x_o}\right) \tag{2.2}$$

Bond has also developed a relationship between specific energy and size reduction after performing a significant amount of test work. His equation is as follows:

$$E = K\left(\frac{1}{\sqrt{x_o}} - \frac{1}{\sqrt{x_i}}\right) \tag{2.3}$$

Bond's equation has been used widely for sizing crushers, rod and ball mills.

A generalized differential form of the grinding laws has been given by Walker et al. [1937]:

$$dE = -C\frac{dx}{x^n} \tag{2.4}$$

where C and n are constants. Rittinger, Kick and Bond's equations can be obtained by integration of Eq. 2.4 with n equal to 2, 1 and 1.5 over particle size limits.

Although the Bond equation has been used extensively for sizing comminution

units [Austin 1973; Herbst, Grandy and Fuerstenau 1973; Hodouin, Bérubé and Everell 1978], it lacks the ability for taking into account the individual mechanisms of grinding such as the breakage kinetics of individual size classes and the efficiency of classification. The Bond law has been used for optimization purposes that will be explained in Section 2.5.1.

2.3 Population Balance Modelling

This approach has received a lot of attention over the past three decades and has been applied not only to circuit design, but also circuit analysis, optimization and control. These models are phenomenological, i.e. the values of model parameters or constants must be determined experimentally [Herbst and Mular 1979].

A number of population balance models have been published for both batch and continuous grinding with various degrees of complexity. Laplante, Finch and del Villar [1987] have discussed applications of simplified models for the simulation of closed grinding circuits. What will now be presented are concepts common to nearly all PBM efforts. A time continuous, size-discrete description and notation will be used.

2.3.1 The Breakage Function

When a single brittle particle breaks into smaller pieces, a range of particle sizes will be produced. Conceptually, the breakage function, or more exactly the breakage distribution function, is a mathematical description of distribution of fragments into a number of size classes. A particle is considered completely broken if all produced fragments leave the original particle size class and appear in smaller size classes. For partial breakage the most common approach is to consider that material remaining in the original (parent) size class unbroken, and that which reports to finer size classes broken. Kelly and Spottiswood [1990] defined the breakage distribution function as the average size distribution resulting from the fracture of a single particle.

A number of methods have been proposed to estimate the breakage function. Herbst and Fuerstenau [1968] have devised a laboratory method to determine the breakage function. The method is based on the concept of zero order production of fine sizes during grinding and the following relationship between the selection and breakage parameters which is used to determine the breakage function values:

$$B_{ij} = \frac{F_i}{S_j} \qquad j=1 \text{ to } i-1 \tag{2.5}$$

where B_{ij} is the cumulative breakage function, F_i is the fines production rate of size class i and S_j is the selection function of size class j (the parent class).

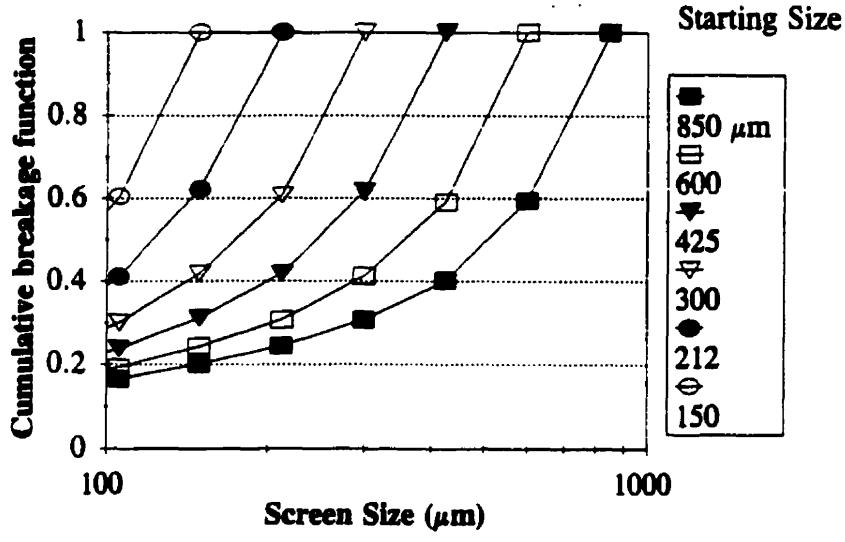
The values of breakage function for each size class obtained from laboratory tests can be fitted with the following equation [Austin, Klimpel and Luckie 1984]:

$$B_{ij} = \Phi_j\left(\frac{x_{i-1}}{x_j}\right)^{\gamma} + (1-\Phi_j)\left(\frac{x_{i-1}}{x_j}\right)^{\beta} \qquad 0 \le \Phi_j \le 1 \tag{2.6}$$

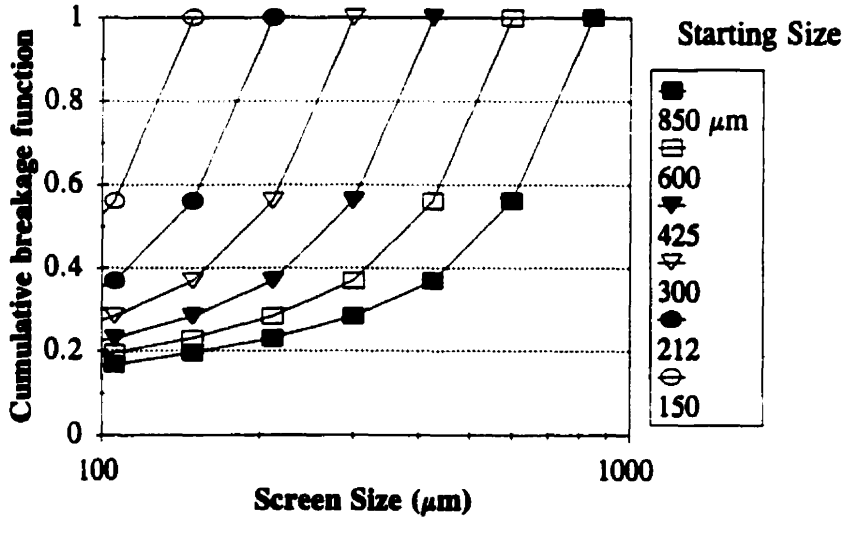
where Φ_j , γ and β are parameters of the model and are material dependent. When the breakage function is independent of the initial particle size, it can be normalized in respect to the initial particle size, and is said to be normalizable. The cumulative breakage function data are often presented by plots such as those shown in Figure 2.1. In normalized form, the breakage function vs. particle size curves for various top or parent size classes coincide.

2.3.2 The Selection Function

The selection function or specific rate of breakage is a measure of grinding process kinetics. In other words, it is an indication of how fast the material breaks. There is ample experimental evidence that batch grinding of brittle and homogeneous



Non-normalizable breakage function, $B_{ij} \neq B_{i+1,\,j+1}$



Normalizable breakage function, $B_{ij} = B_{i+1,\,j+1}$

Figure 2.1 Typical representation of breakage (distribution) function data

materials follows a first order kinetics [Klimpel and Austin 1970]:

$$\frac{dm_i(t)}{dt} = -S_i m_i(t) \tag{2.7}$$

where $m_i(t)$ is the mass of material in size class i at time t, S_i is the rate constant or selection function for size class i. The grinding kinetics data are usually presented by the selection function vs. particle size curves, Figure 2.2. The curves can assume various shapes such as straight and parabolic lines. The selection function values will be dimensionless if a normalized (dimensionless) time is used in Eq. 2.7.

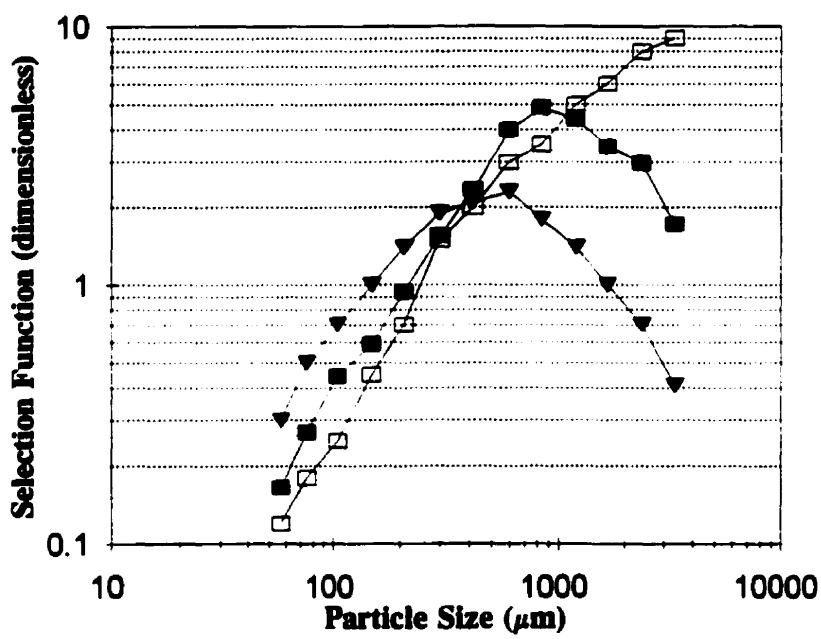


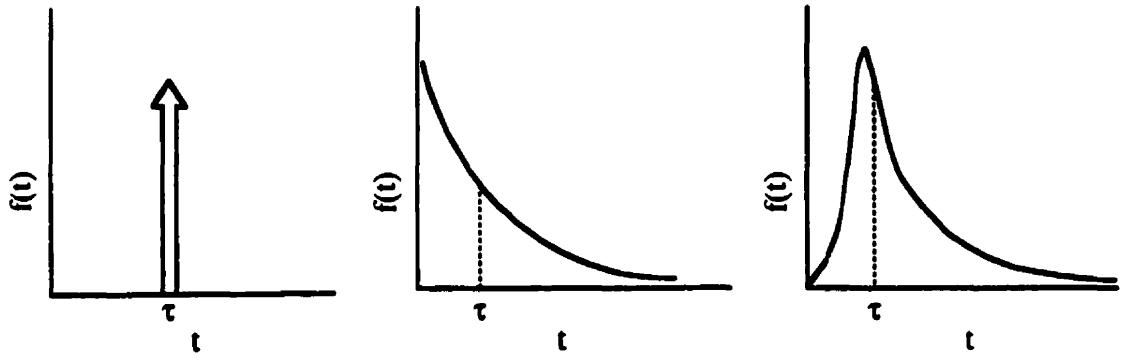
Figure 2.2 Typical representation of grinding (breakage) kinetics data

2.3.3 Residence Time Distribution

In continuous grinding, there is a material flow through the mill. To model these

types of grinding mills, this flow must be described mathematically. The *R*esidence *T*ime *D*istribution (RTD) is a statistical tool which allows the evaluation of material transport and mixing phenomena in a reactor vessel [Marchand, Hodouin and Everell 1980].

The procedure to determine the RTD of an industrial grinding mill is to first perform a tracer test and then fit an RTD model to the measured response. Figure 2.3 shows three types of RTD curves. The first and second curves represent two ideal cases, i.e plug flow and perfectly mixed systems. The third curve represents real mixing systems which is considered as a combination of the first two ideal cases. There are a number of arrangements of plug flow and perfectly mixed systems reported in the literature which can closely fit various experimental RTD data [Bazin and Hodouin 1988].



(a) ideal plug flow systems (b) ideal perfectly mixed systems (c) real systems

Figure 2.3 Typical residence time distribution (RTD) curves

Weller [1980] has proposed a model which assumes a mill as a series of tanks: one plug flow, two small perfect mixers and a large perfect mixer unit. The model is:

$$f(t) = \left(\frac{\alpha}{\tau_l}\right)\left[-\frac{t-\tau_{pf}}{\tau_s}\exp\left(-\frac{t-\tau_{pf}}{\tau_s}\right)-\alpha\exp\left(-\frac{t-\tau_{pf}}{\tau_s}\right)+\alpha\exp\left(-\frac{t-\tau_{pf}}{\tau_l}\right)\right], \quad \alpha = \frac{\tau_l}{\tau_l-\tau_s} \quad (2.8)$$

where τ_{pf} ,τ_s and τ_l are the mean retention times of the plug flow, small perfect mixers and the large perfect mixer, respectively. The best estimates of τ_{pf} ,τ_s and τ_l can be determined by fitting Eq. 2.8 to the measured curve using a numerical optimization tool.

Another commonly used model considers a mill as a series of one plug flow unit and several perfect mixers:

$$f(t) = \frac{n^n(t-\tau_{pf})^{n-1}\exp\left(-n\frac{t-\tau_{pf}}{\tau}\right)}{\tau^n(n-1)!} \quad (2.9)$$

where n is the number of perfect mixers and τ is the total mean retention time of perfect mixers.

In simulation of continuous mills, the dimensionless form of a residence time distribution is combined with a batch comminution model to calculate the particle size distribution of the mill discharge. For a more complete treatment of the subject of residence time distribution, the reader is referred to the works by Austin, Luckie and Ateya [1971]; Weller [1980]; and Marchand, Hodouin and Everell [1980].

2.3.4 Batch Grinding Model

Batch grinding can be modelled based on the concepts of breakage function and selection function defined in Sections 2.3.1 and 2.3.2. In a batch mill, all particles are subjected to the grinding process for the same grinding time. Assuming that grinding is

a rate process and follows a first order kinetics, the size-discretized differential form of the batch grinding equation can be obtained by a mass balance for the i-th size interval at time t as below [Austin 1971]:

$$\frac{dm_i(t)}{dt} = -S_i m_i(t) + \sum_{j=1}^{i-1} b_{ij} S_j m_j(t) \tag{2.10}$$

Writing Eq. 2.10 for n size class yields a system of equations which can be represented by the following single matrix equation:

$$\frac{d\mathbf{m}}{dt} = (B-I)S\mathbf{m} \tag{2.11}$$

Assuming that there are no two equal selection functions, Eq. 2.11 can be solved to obtain the product size distribution as a function of time:

$$\mathbf{m}(t) = T\exp(-St)T^{-1}\mathbf{m}(0) \tag{2.12}$$

where

$$T_{ij} = \begin{cases} 0 & i<j \\ 1 & i=j \\ \dfrac{1}{s_i - s_j}\displaystyle\sum_{k=1}^{i-1} b_{ik} s_k T_{kj} & i>j \end{cases}$$

and

$$T_{ij}^{-1} = \begin{cases} 0 & i<j \\ s_j^{-1} & i=j \\ \dfrac{-\displaystyle\sum_{k=1}^{i-1} T_{ik} T_{kj}^{-1}}{s_i} & i>j \end{cases}$$

2.3.5 Continuous Grinding Model

Continuous grinding can be modeled by taking into consideration the residence time distribution of the solid phase. The steady-state product size distribution of a continuous mill in open circuit is the weighted average of the size distributions produced by batch grinding (Eq. 2.12) for different times corresponding to the particles residence time distribution [Spring, Larsen and Mular 1985]:

$$m_{i,c}(t) = \int_{t=0}^{\infty} f(t)\, m_{i,b}(t)\, dt \tag{2.13}$$

where $m_{i,c}$ and $m_{i,b}$ are the retained mass corresponding to size class i at time t for continuous and batch grinding, respectively and f(t) is the residence time distribution. By considering Weller's model [1980] as a fit to the measured RTD, the product size distribution of a continuous grinding unit can be obtained:

$$M_d = T(I+S\tau_s)^{-2}(I+S\tau_l)^{-1}\exp(-S\tau_{pf})T^{-1}M_f \tag{2.14}$$

where:

M_d and M_f: column matrices representing product and feed size distributions
T and T^{-1}: lower triangular matrices defined in Eq. 2.12
I: identity matrix
S: a diagonal matrix representing the selection function

Eq. 2.14 is based on several implicit assumptions such as the ones used to derive Eq. 2.10 and (1) residence time distributions for different particle sizes are identical, (2) the breakage and selection functions are constant as particles move through the mill, and (3) no internal classification takes place.

2.4 Modelling of Hydrocyclones

A hydrocyclone is a classification device which separates solid particles based on their settling behaviour in the suspending liquid. The settling velocity mainly depends on the size, density, shape of the particles and the rheological properties of the feed slurry, which change in the cyclone. When particles have the same density and shape, for example in homogenous ores, their separation is based exclusively on their size.

Plitt [1976] published his original model of hydrocyclone classifiers based on a significant amount of experimental data. The model describes the operation of hydrocyclones using four empirical equations. The main indicators of hydrocyclone performance are the corrected cut size, flow split between overflow and underflow, separation sharpness and pressure drop. These parameters are calculated in terms of the operating and design variables of the hydrocyclone.

Lynch and Rao [1975] also proposed similar equations to model hydrocyclones which have been widely used in commercial grinding circuit simulators. They worked on the concept of reduced efficiency curves that can be converted into the corrected and actual efficiency curves by predicting d_{50c} and R_f. A number of tests and scale-up procedures have been recommended by Lynch and Rao [1975] to use their model for hydrocyclone design purposes.

As Plitt's model has been used in this thesis to develop the *B*all *M*illing *C*ircuits *S*imulator (BMCS), a review of this model is presented here. The reader is referred to Lynch and Rao [1975], Lynch et al. [1977] for a discussion of Lynch and Rao's hydrocyclone model.

2.4.1 Classification Performance Curves

The operational performance of a cyclone can be illustrated by plotting the size-by-size recovery of solids to the cyclone underflow versus a characteristic particle size

as shown in Figure 2.4. The data points are obtained by sampling the streams around the cyclone and performing particle size analysis tests on each of the samples. A partition curve (also called selectivity or Tromp curve) is a graphical representation of a mathematical model that adequately fits the experimental points and characterize cyclone performance in terms of cut size, the recovery of fluid to the cyclone underflow and the sharpness of separation.

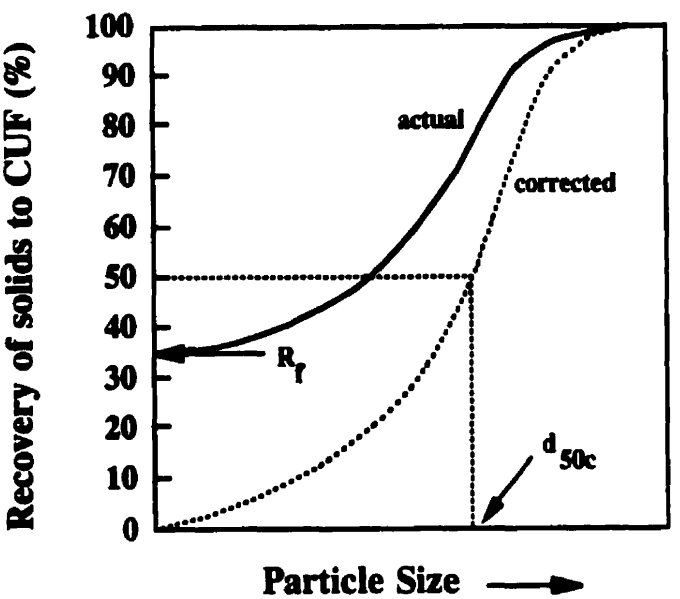


Figure 2.4 Usual classification performance curves

For process analysis and simulation, Plitt [1976] has given the following relationship:

$$R_i = R_f + (1-R_f)\left\{1-\exp\left[-0.693\left(\frac{x_i}{d_{50c}}\right)^m\right]\right\} \qquad (2.15)$$

where:

R_i: recovery of solids to cyclone underflow for size class i

R_f: recovery of fluid to cyclone underflow

x_i: characteristic particle size for size class i

d_{50c}: corrected cut size

m: separation sharpness

The three parameters of Plitt's model, R_f, d_{50c} and m, are indications of classification performance and can be estimated by non-linear least-square fit. The corrected cut size or d_{50c} is the size at which a particle has equal probability to report to either cyclone overflow or underflow streams of the hydrocyclone by true classification. The corrected cut size is related to the hydrocyclone geometrical variables as below:

$$d_{50c} = \frac{50.5 D_c^{0.46} D_i^{0.6} D_o^{1.21} \exp(0.063\phi)}{D_u^{0.71} h^{0.38} Q^{0.45} (\rho_s - \rho)^{0.5}} \qquad (2.16)$$

where:

D_c: internal diameter of cyclone at the bottom of vortex finder in centimeters

D_i: internal diameter of cyclone inlet in centimeters

D_o: internal diameter of cyclone overflow or vortex finder in centimeters

D_u: internal diameter of cyclone underflow or apex in centimeters

h: free vortex hight of cyclone in centimeters

Q: volumetric flow rate of slurry feed to the cyclone in liters per minute

ϕ: volumetric fraction of solids in the feed slurry in per cent

ρ_s and ρ: density of solid and liquid in grams per cubic centimeters

The corrected d_{50} is a more fundamental parameter than the uncorrected one, as it is a measure of the true classification, rather than its combination with the short

circuiting.

The flow Split is the ratio of volumetric underflow rate to volumetric overflow rate and can be calculated using the following regression equation:

$$S = \frac{1.9(D_u/D_o)^{3.31} h^{0.54} \left(D_u^2 + D_o^2\right)^{0.36} \exp(0.0054\phi)}{H^{0.24} D_c^{1.11}} \qquad (2.17)$$

where S is the flow split (dimensionless) and H the pressure drop across the cyclone expressed in terms of the head of feed slurry in meters.

The sharpness of separation, m, indicates how well the cyclone separates fine and coarse particles. This parameter is calculated by the following regression equation:

$$m = 1.94 \exp\left(-1.58 \frac{S}{S+1}\right)\left(\frac{D_c^2 h}{Q}\right)^{0.15} \qquad (2.18)$$

A cyclone with a value of m over 3 is considered to have a sharp classification. A value of m less than 2 represents poor classification [Plitt 1976]. Separation sharpness has been represented also by another term called "probable error" which is defined as [Plitt, Finch and Flintoff 1980]:

$$E_p = \frac{d_{75c} - d_{25c}}{2} \qquad (2.19)$$

where

E_p: probable error

d_{75c} and d_{25c}: corrected particle sizes having probabilities equal to 75 per cent and 25 per cent to report to cyclone underflow, respectively

Imperfection is the ratio of the probable error to the cut size, d_{50c}, and is also a measure of cyclone inefficiency:

$$I = \frac{E_p}{d_{50c}} \tag{2.20}$$

where I is the imperfection (dimensionless). The normal value of imperfection is in the range of 0.2 to 0.8 and it is approximately related to the sharpness of separation parameter, m, of Plitt's model by:

$$I = \frac{0.77}{m} \qquad 3.5 > m > 1.1 \tag{2.21}$$

The pressure drop is another parameter which must be known in order to design the pumping system for a given capacity or to determine the capacity for existing cyclones [Plitt 1976]. It is also needed to calculate flow split.

$$P = \frac{1.88 Q^{1.78} \exp(0.0055\phi)}{D_c^{0.37} D_i^{0.94} h^{0.28} \left(D_u^2 + D_o^2\right)^{0.87}} \tag{2.22}$$

where P is pressure drop in kPa. The pressure drop is not a critical parameter in process analysis. Nevertheless, for simulation purposes it is required to compute the value of S.

Recovery of fluid to the cyclone underflow (or bypass), R_f, is calculated by:

$$R_f = \frac{\frac{S}{(1+S)} - \frac{R_s \phi}{100}}{1 - \frac{\phi}{100}} \tag{2.23}$$

R_s is the recovery of feed solids to cyclone underflow which can be calculated by:

$$R_s = \frac{\Sigma R_f m_i}{\Sigma m_i} \tag{2.24}$$

where m_i is amount of feed solids in size class i.

Equations 2.15, 2.23 and 2.24 show that calculation of R_f is an iterative process as R_s itself is a function of R_f. Plitt, Conil and Broussaud [1990] have proposed a new procedure which simplifies calculation of R_f by eliminating the iterative solution.

Plitt's hydrocyclone model has applications in process analysis, simulation, design and on-line size analysis [Flintoff, Plitt and Turak 1987]. In process analysis, samples are taken from the cyclone feed, underflow and overflow streams. After determining the size distribution of each sample and knowing operating data such as feed rate and percent solids, Plitt's model can be fitted to the data, to evaluate cyclone performance.

In simulation, Plitt's model can be used to predict the size distribution of the underflow and overflow streams and flow rates based on the geometry of the cyclone and feed flow rate to the cyclone.

2.4.2 Operating Constraints

Implicitly, Plitt's model (Eqs. 2.16, 2.17, 2.18 and 2.22) assumes that any cyclone operates under the spray discharge (normal) regime. Consequently, the model cannot be readily used to simulate cyclone operation under rope discharge regime. Rope

discharge is caused by overloading the apex orifice. The modelling of cyclone performance under roping is of importance since there is a tendency among the plant operators to run cyclones at boundary conditions near to rope discharge.

Plitt, Flintoff and Neale [1986] and Plitt, Flintoff and Stuffco [1987] investigated the operation of hydrocyclones under conditions that cause rope discharge from the apex. They found that roping occurs at a certain fraction of solids in the underflow discharge which depends on many factors such as slurry rheology, cyclone geometry, operating conditions and feed characteristics. Experimental results obtained by Plitt, Flintoff and Stuffco [1987] indicate that transition of cyclone operation from spray to rope discharge occurs with an increase in the cut size (d_{50c}) and little change in the sharpness of operation. The following empirical relationship has been used [Flintoff, Plitt and Turak 1987] in SPOC* to determine the critical value of volumetric solids concentration in the underflow:

$$L_u = L_{u20} + 0.2(\phi - 20) \tag{2.25}$$

where

L_u: critical volumetric solids concentration at which roping occurs

L_{u20}: volumetric solids concentration at ϕ equal to 20 per cent

ϕ: volumetric solids concentrate in the feed

If rope discharge conditions are detected, when L_u predicted by the cyclone model is greater than or equal to the L_u calculated by Eq. 2.25, the performance of the cyclone will be predicted by the modification of model parameters or structure.

Though they are not discussed here, there are other operating constraints that must

*Simulated Processing of Ore and Coal, a simulation package developed by CANMET, Canada

be considered in hydrocyclone modelling and simulation such as cyclone blockage and cyclone capacity. More details can be found in Plitt, Flintoff and Neale [1986].

2.5 Methodologies of Grinding Optimization

According to the literature, grinding optimization methodologies have been constantly under evolution since the introduction of comminution laws by Rittinger, Kick and Bond. Concerning the grinding process itself, the early approaches were based on the energy-based models of size reduction. Later on, new methods were developed using PBM which is founded on one of the first principles of physics, i.e., conservation of mass.

2.5.1 Operating Work Index Optimization

The concept of plant work index or operating work index, W_{io}, and its comparison with the laboratory (standard) work index, W_i, to indicate grinding operation efficiency have been discussed by several workers [Bond 1952, Rowland 1973 and Levin 1992]. To evaluate grinding performance using the operating work index, Bond grindability tests should be run on a sample taken from the mill feed. The operating conditions at the time of sampling must be recorded for calculating the operating work index.

The published values of W_{io}/W_i for a number of ball mills in single-stage or rod/ball circuits are in the range of 0.89 to 1.16 and 0.78 to 1.29, respectively [Rowland 1973]. Therefore, depending on the value of W_{io}/W_i, the performance of an industrial mill could be worse or better than the average performance of the mills observed by Bond. Grinding efficiency improvements can be measured by using W_{io} as a performance index.

2.5.2 Functional Performance Optimization

McIvor et al. [1990] have presented an analytical approach for functional performance of grinding based on the concepts of the Bond work index and operating

work index. It has been proposed that the overall performance of a ball mill-classifier circuit can be defined by the quantification of the classification system and ball mill breakage efficiencies, Figure 2.5. To measure the efficiency of the classification system or fines removal system, a coarse solids inventory is used which represents the fraction of the mill volume effectively used for grinding of coarse particles.

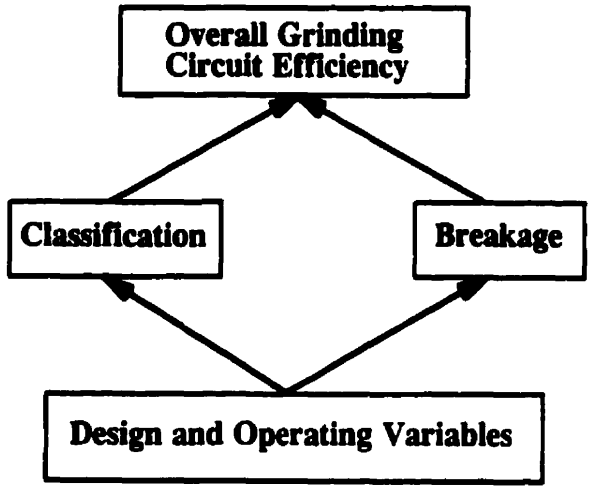


Figure 2.5 The concept of functional performance analysis of grinding process [McIvor et al. 1990]

Despite presenting new concepts, this approach still includes the limitations of Bond's method and lacks the capabilities of PBM. For example, the population balance modelling of the grinding process permits constructing accurate predictive models of ball mills which can be used to simulate the particle size distribution of a ball mill discharge or to determine the size-by-size grinding kinetics which indicates the process performance and can be used for grinding media size optimization.

2.5.3 Simulation-Based Optimization

Computer simulation is a powerful tool for designing new grinding circuits, and

the study, evaluation and optimization of existing plants [Laguitton 1982]. Hodouin, Bérubé and Everell [1978] have illustrated the use of mathematical simulation techniques for design purposes. Different approaches for simulation and scale-up of wet ball milling have been discussed by several researchers [Herbst and Fuerstenau 1980, Austin and Weller 1982]. Steady-state simulators have been used for process analysis and off-line circuit design and optimization [Laguitton et al. 1984]. Dynamic simulators have been useful in on-line optimization of grinding circuits for designing effective control strategies [Herbst, Alba, Pate and Oblad 1988]. USIM-PAC [Durance et al. 1993] and JKSimMet [Napier-Munn and Lynch 1992] are among the commercial software which are capable of simulating a wide range of unit operations including grinding circuits.

Steady-state simulators for off-line studies of grinding circuits are based on the population balance models of grinding and empirical models of classification units. Although the development of process unit models which are the building blocks of any circuit simulator, is a fundamental step, these models should be linked appropriately in a simulation structure. At McGill University, A number of models have been developed, and most are not widely used; for example, Del Villar and Laplante [1985] developed programs in BASIC to simulate open and closed circuit ball milling.

Grinding simulators can be used to study the effect of circuit configuration or operating conditions on the circuit capacity or product specifications. For instance, a circuit simulation can be used to compare the grinding efficiencies of series vs. parallel ball mill installations or the performances of a two-stage classification vs. a single stage classification. Herbst, Schena and Fu [1989] have discussed the structure of a grinding simulator and its use in flowsheeting.

2.6 Behaviour of Individual Minerals in Grinding Circuits

In complex ores (multi-component) that include individual minerals having various grindabilities and densities, the study of mineral performances in size reduction and

classification environments helps in better understanding of the process and any further optimization. The behaviour of minerals in the size reduction process can be characterized in terms of their grinding kinetics (selection function). In classification, the behaviour of each mineral can be illustrated by its classification performance curve.

2.6.1 Mineral Size Reduction Kinetics

It is well accepted that minerals in a multi-component material exhibit individual behaviours both in grinding and classification [Finch and Matwijenko 1977, Finch and Ramirez-Castro 1981]. In relation to grinding, the study of breakage kinetics on a component basis (mineral by mineral) explains process performance in more detail; however, it requires more effort due to size-by-size mineral assays.

The study of gold behaviour, for example, demonstrates the effects of individual mineral properties on the overall plant performance. Banisi, Laplante and Marois [1991] have investigated the behaviour of gold in the grinding circuit of Hemlo Mines Ltd. They found that the gold selection function is much lower than that of silica, due to the malleability of gold particles. It also reports preferentially to the cyclone underflow owing to its density. The slow grinding kinetics and propensity to report to the cyclone coarse product of gold increase its circulating load.

2.6.2 Mineral Classification Performance Curves

There are a number of reported cases [Finch and Matwijenko 1977, Laplante and Finch 1984] of measured classification partition curves that do not have the usual smooth shape, but exhibit a plateau or dip in intermediate size range as shown in Figure 2.6.

The overall curve for the ore can be calculated mathematically as the weighted sum of the component curves in respect to their fraction in each size interval [Laplante and Finch 1984]. The plateau is amplified in secondary cyclones, whose feed is largely comprised of coarse lights and fine heavies. Recognizing the conditions causing unusual

hydrocyclone partition curves prevents flaws in interpretation of plant data and also misleading predictions obtained by applying a single-component model while a two-component model must be used.

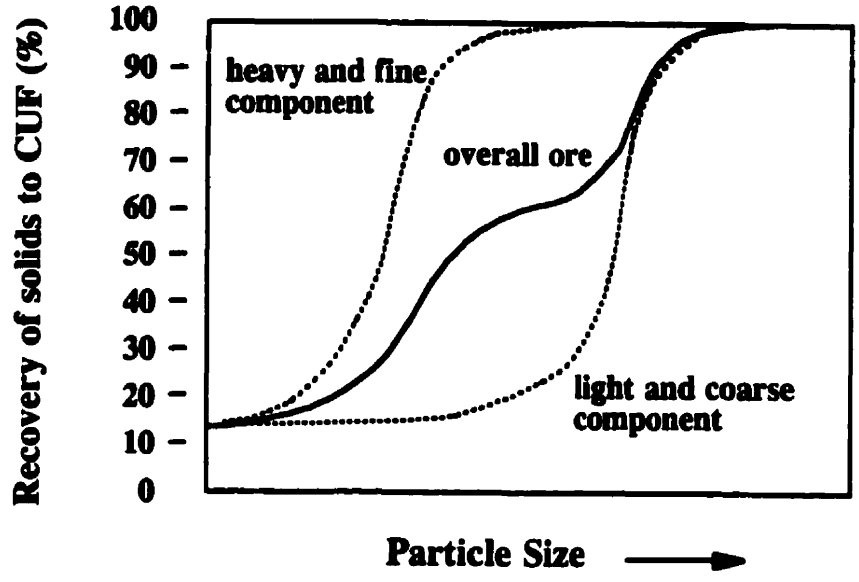


Figure 2.6 Unusual classification performance curve with a plateau at intermediate particle sizes

2.7 Grinding Media Size

The size of grinding media has a considerable effect on the efficiency and capacity of tumbling grinding mills [Bond 1958]. Bond has given empirical equations to calculate the optimal size of grinding media that should be added regularly to rod and ball mills. The equation to calculate the size of make-up balls is:

$$d_b = \left(\frac{F_{80}}{K}\right)^{0.5}\left(\frac{S_g W_i}{100 C_s D_m^{0.5}}\right)^{0.33} \tag{2.26}$$

where F_{80} is 80% passing size of the make-up feed in micrometers, K is the wet grinding constant, S_g is the specific gravity of ore, W_i is the Bond work index in kiloWatt hours per short ton, C_s is the fraction of critical speed and D_m is the mill diameter in meter.

Azzaroni [1981] has also proposed the following relationship to estimate make-up ball size:

$$d_b = 5.8 \frac{(F_{80})^{0.29} (W_i)^{0.4}}{(ND_m)^{0.25}} \tag{2.27}$$

where N is mill speed in rpm.

The following formulae have been quoted by Lowrison, 1974:

$$d_b = K(feed\ size)^{\frac{1}{2}} \tag{2.28}$$

$$d_b = K(feed\ size) \tag{2.29}$$

$$d_b = K(feed\ size)^{\frac{2}{3}} \tag{2.30}$$

It is well known that a spectrum of ball sizes is established in a mill after grinding for a period of time due to the wear of balls and continual addition of larger ball sizes [Bond 1958]. However, the choice of the mixture of ball sizes to be used in a ball mill is a difficult question [Austin, Shoji and Luckie 1976]. Grinding efficiency is directly affected by the equilibrium mix of balls. The optimum mixture of balls is best determined by an accurate mill simulation [Austin, Klimpel and Luckie 1984].

The newer approach to select grinding media size is based on grinding kinetics evaluation [Concha, Magne and Austin 1992; Cooper et al. 1993; Staples, Cooper and Grant 1997]. Grinding kinetics, represented by selection function or specific rate of breakage, is strongly linked with the equilibrium mix of balls. The effect of using a certain grinding media size on grinding kinetics can be investigated either by changing media size in real mill or by performing computer simulations.

Assuming two major types of fracture mechanisms (impact and attrition) prevailing in a ball mill, Morrell [1990] has developed procedure to predict selection function values for a new ball size, by estimating selection function values for the current ball size. When coupled with a spline curve fitting algorithm, Morrell's scaling procedure provides a useful tool for studying the effect of grinding media size on grinding kinetics. Additionally, this tool can be integrated with a ball mill simulator to investigate the effect of ball size on final product size distribution.

CHAPTER 3
KNOWLEDGE-BASED SYSTEMS THEORY AND APPLICATIONS

3.1 Introduction

In the previous chapter, some aspects of grinding optimization theory were discussed from a mineral processing point of view. In this chapter, an overview of concepts of *K*nowledge-*B*ased *S*ystems (KBSs) or *E*xpert *S*ystems (ES) will be given, with examples related to the grinding domain. KBSs have their roots in *A*rtificial *I*ntelligence (AI) and many theories discussed in AI are indeed fundamental to KBSs. A discussion of AI fundamentals is left to other references [Barr and Feigenbaum 1981, Genesereth and Nilsson 1987], only the general concepts of KBSs required to understand these systems will be explained here. Applications of KBSs in engineering have been discussed in Rychener 1988, Dym and Levitt 1991 and Balachandran 1993.

3.2 The Concept of Knowledge

Knowledge has been defined as the understanding of a subject area [Durkin 1994]. Coyne et al. [1990] have defined knowledge as statements for mapping between facts. In practice, solving engineering problems requires applying different types of knowledge as shown in Table 3.1. [Dym and Levitt 1991, Durkin 1994]. Some of these are of critical importance and will be explained in this chapter.

In AI, knowledge has been divided into deep knowledge and surface knowledge types [Dym 1985]. Deep knowledge refers to reasoning from basic principles. Surface knowledge refers to reasoning just based on heuristic, experiential knowledge coming

from solving many problems - i.e. experience.

Table 3.1 Types of problem-solving knowledge [Durkin 1994]

Type	Example
Declarative Knowledge	Concepts, objects, facts
Procedural Knowledge	Rules, strategies, agendas, procedures
Meta-Knowledge	Knowledge about knowledge
Heuristic Knowledge	Rules of thumb
Structural Knowledge	Rule sets, concept relationships, concept to object relationships

3.3 Engineering Tasks

KBSs are built to perform or solve a range of tasks such as interpretation, diagnosis, monitoring, prediction, planning and design [Stefik et al. 1983]. These categories are explained in Table 3.2.

In AI, engineering tasks have been roughly categorized as analysis (derivation) problems such as interpretation, diagnosis and monitoring; and synthesis (formation) problems such as planning and design [Dym and Levitt 1991]. Whether a problem is of analysis or synthesis type is of significant importance. The size of the solution space and the required search effort are tightly linked to nature of the problem and impose limitations on the choice of inferencing method [Dym 1985].

3.4 Representation of Knowledge

A number of techniques have been devised to represent domain knowledge; for instance propositional logic, predicate calculus, production rules, semantic networks and frames.

Table 3.2 Tasks that KBSs can perform [Hayes-Roth, Waterman and Lenat 1983]

Category	Task
Interpretation	Inferring situation description from data
Prediction	Inferring likely consequences of given situations
Diagnosis	Inferring system malfunctions from observations
Design	Configuring objects under constraint
Planning	Designing actions
Monitoring	Comparing observations to plan vulnerabilities
Debugging	Prescribing remedies for malfunctions
Repair	Executing a plan to administer a prescribed remedy
Instruction	Diagnosing, debugging, and repairing student behaviour
Control	Governing system behaviour to meet specifications

3.4.1 Facts

In addition to referring to declarative knowledge, facts also refer to the knowledge structures which are used to represent this type of knowledge. In other words, facts are structures to assert correct data or information into the working memory (or facts base) about the problem at hand. Facts asserted in the fact base originate from a number of resources such as the user, data bases, spreadsheets, sensors, other programs and inference by the KBS itself. The following examples show some ways to represent facts:

(ballmill diameter 2.5)
(circulating-load high)
(ballmill product fine)
(cyclone cut-size 75)

Object-Attribute-Value (O-A-V) triplets are widely used to describe one aspect of a

physical or conceptual object. For example, the following patterns define some facts (or knowledge) about objects in a grinding circuit:

(ballmill discharge-type overflow)
(hydrocyclone pressure-drop high)
(liner wear-condition low)

Obviously, more than one of O-A-V triplet is normally needed to describe all attributes of an object. If simple structures and O-A-V triplets are inadequate to represent domain knowledge, owing to the complexity of the object, frames can be used. Frame representation will be discussed in Section 3.4.3.

3.4.2 Rules

Rules are *IF...THEN* knowledge structures that relate some known information, antecedents or premises, to other information that can be concluded or referred to be known, consequents or conclusions. The chunks of knowledge contained in rule statements are mainly heuristics (rules of thumb) or simplifications that effectively limit the search space for finding solutions [Waterman 1986]. Consider the following examples:

IF ball mill model does not predict product size well
THEN modify model parameters or model structure

IF the selection function vs. particle size curve is strongly parabolic
THEN ball size is too small

IF short circuiting of water to cyclone underflow is high
THEN add more water to the circuit or decrease the apex diameter of the cyclone

Rules can use other types of knowledge structures on their *l*eft *h*and *s*ide (LHS) and *r*ight *h*and *s*ide (RHS). These structures basically describe facts or declarative knowledge about the problem or procedural knowledge to perform a specific function.

Rules can represent various knowledge types such as interpretation, diagnosis and design which are closely linked to the problem solving paradigm [Durkin 1994]. The structure of information contained in LHS of a rule varies from very simple patterns to very complex objects.

3.4.3 Frames

Minsky [1975] proposed frames as data structures that can accommodate stereotypical knowledge about physical or conceptual objects. A frame describes an object by its attributes (declarative knowledge) and behaviour (procedural knowledge). A frame (also called a schema or object) has a name, a number of slots to describe properties of the object, and fillers (Boolean, symbolic or numeric values) to fill the frame slots:

frame name
slot *filler*
slot *filler*
slot *filler*

Facets are also used in frames to provide additional control on an attribute's value such as constraints.

In a FBS, a class frame is used to represent a set of objects that share the same properties. To refer to a specific object, an instance frame is made using the object class frame. For instance, Figures 3.1 and 3.2 show simple class and instance frames for ball mills and a specific ball mill, respectively. A FBS is established by defining a hierarchy of object classes. A frame in the hierarchy can inherit properties from an upper-level class (single inheritance) or from a number of upper-level classes (multiple inheritance). Figure 3.3 shows an example of a hierarchy for objects used in comminution circuits. The implementation of a frame-based expert system is highly tool-dependent and is limited by capabilities offered by the shell.

frame name	BALLMILL	
properties	slot (attribute)	filler (value)
	length	unknown
	internal diameter	unknown
	media size	unknown
	media consumption	unknown
	liner wear condition	unknown
	liner wear	unknown
methods	GrindFeedMaterial	

Figure 3.1 A class frame representing ball mills as objects in a grinding circuit

frame name	ballmill-1	
class	BALLMILL	
properties	slot (attribute)	(filler) value
	length	3.048 m
	internal diameter	2.743 m
	media size	100 mm
	media consumption	2.5 kg/t
	liner wear condition	high
	liner wear	0.021kg/t
methods	GrindFeedMaterial	

Figure 3.2 An instance frame representing a specific ball mill in a grinding circuit

3.5 Inference Techniques

The inference engines of KBSs are designed based on three reasoning methods: (1) deduction or reasoning from a known principle to an unknown (2) abduction or reasoning from a conclusion to premises (3) induction or reasoning from particular facts or individual cases to general conclusions. While the first method is a sound rule of inference, the last two are unsound rules of inference (not logically correct). Most KBSs have deductive inference engines. KBSs with inductive inference engines are also available. The details of these reasoning methods have been discussed in Genesereth and Nilsson 1987.

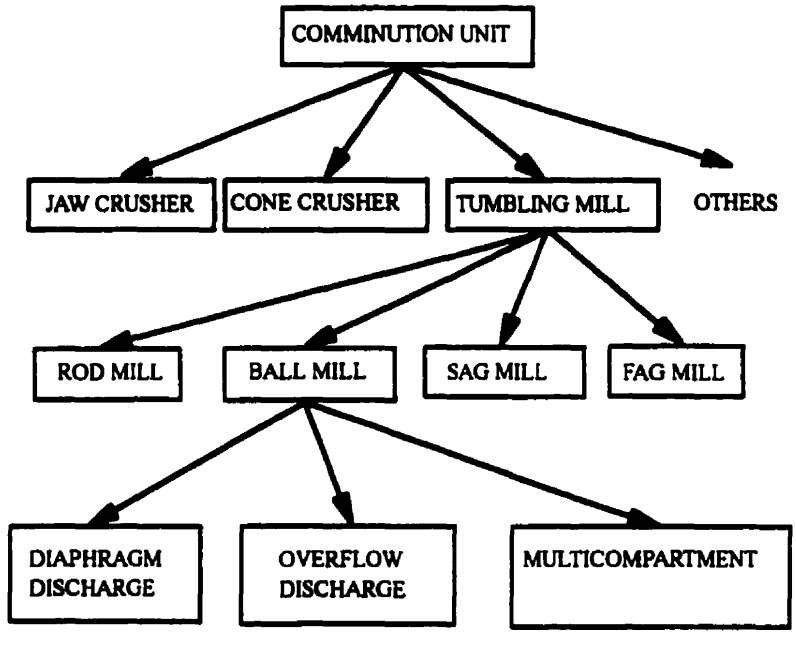


Figure 3.3 A hierarchy of comminution objects

The inference engines of *R*ule-*B*ased *S*ystems (RBSs) find satisfied rules, create a prioritized list of them in an agenda and then execute them [Giarratano and Riley

1989]. RBSs use two main control strategies or inferencing methods, forward chaining and backward chaining [Turban 1988]. The inference engine is designed to use one or both of these reasoning strategies to find matching or satisfied rules. The type of project significantly affects the choice of inference engine; for instance, while diagnostic problems are solved better with backward chaining; prognosis, monitoring and control problems can be solved better by forward chaining. In the following sections the two control strategies, i.e. the forward and backward chaining methods, which are extensively used in RBSs to automate deductive reasoning are explained.

3.5.1 Forward Chaining

Forward chaining is a control strategy that allows a RBS to infer new facts from given facts. Figure 3.4 shows how the reasoning process is started from the initial facts, F and G, and is ended by concluding the final fact, C. A rule is selected for execution when its premises are satisfied. All satisfied rules are placed into the execution list (agenda) based on a conflict resolution scheme. Recursively, after a rule is executed, its conclusions might match the premises of other rules causing them to be fired by the inference engine. This type of inference suits applications that are inherently data-driven such as design, simulation. monitoring and control.

3.5.2 Backward Chaining

Backward chaining control strategy tries to prove a hypothesis (conclusion) by finding supporting evidence in the facts base (Fig. 3.5). Backward chaining is a goal-driven reasoning method and is suitable when the relationships between facts are well known. The backward chaining method has been applied successfully to diagnosis and classification problems.

3.6 Uncertainty Management

In many cases, there are uncertainties associated with information (evidence), rules or both that affect the problem solving process; reasoning under these circumstances

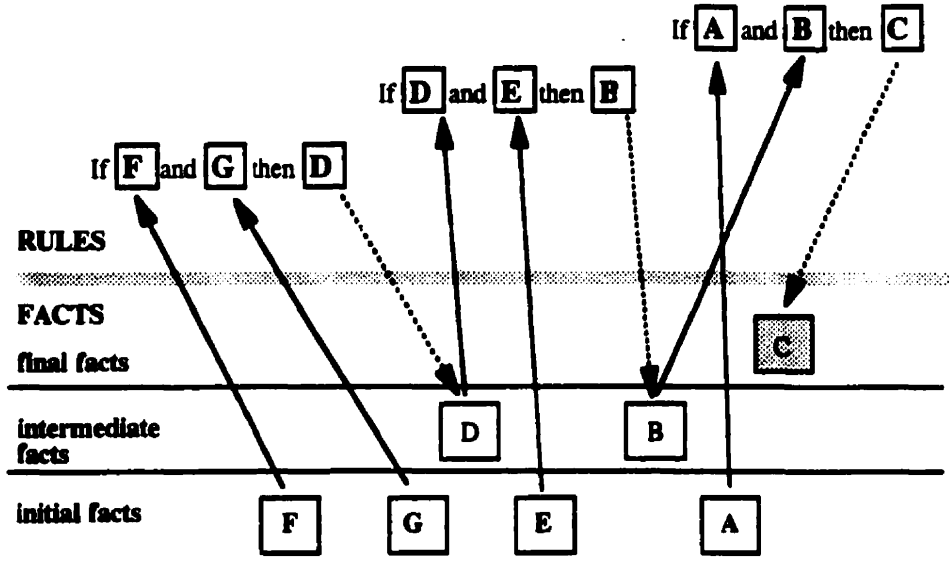


Figure 3.4 Forward chaining inference method [Coyne et al. 1990]

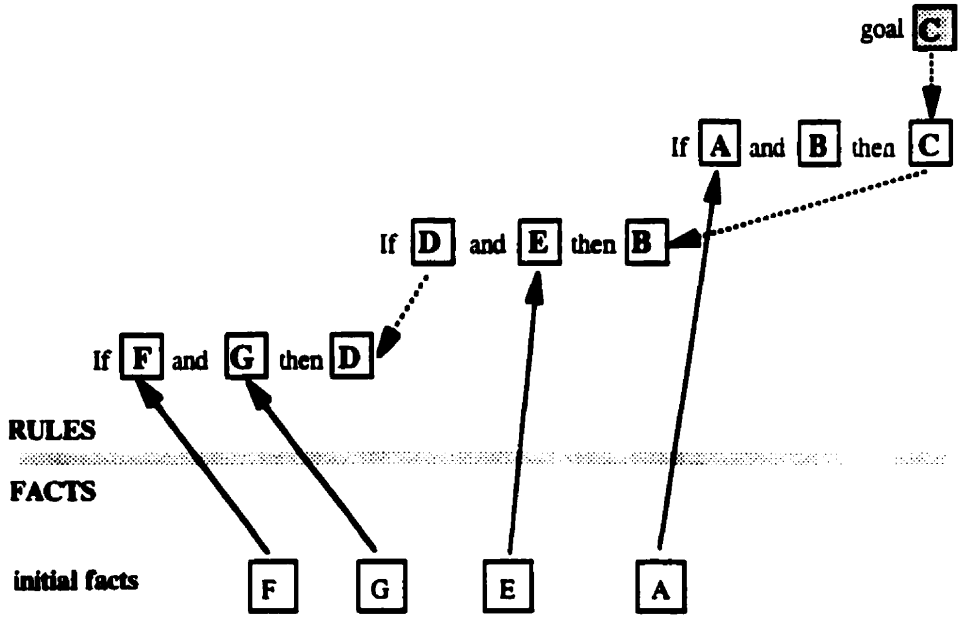


Figure 3.5 Backward chaining inference method [Coyne et al. 1990]

requires uncertainty handling capabilities by the KBS or expert system. Different methods have been proposed to cope with uncertainty such as Bayes's theory, Dempster-Shafer's theory of evidence, certainty factors and fuzzy logic [Giarratano and Riley 1989]. Among these the last two are most frequently used and will be discussed here.

3.6.1 Certainty Factors

A simple approach to handle uncertainty is based on certainty theory [Shortliffe and Buchanan 1975]. A Certainty Factor (CF) is a number assigned to a statement (a piece of evidence or a rule) and represents the degree of belief in that statement. CF is a bounded value as follows:

$$-1 \leq CF \leq 1$$

A certainty value of -1 means definitely false, 0 unknown and 1 definitely true. The propagation of certainty factors in a KBS has been discussed at length in Durkin 1994, Giarratano and Riley 1989 and Gonzalez and Dankel 1993. Briefly, a KBS determines the degree of belief of a hypothesis (derived conclusion) by combining certainty factors associated with facts (evidence) and rules.

$$CF(H|E) = CF(E)*CF(RULE) \quad \textit{for single premise rules}$$

$$\textit{IF } E_1 \wedge E_2 \wedge \ldots \textit{THEN } H \quad CF(RULE) \quad \textit{for multiple premise conjunctive rules}$$
$$CF(H|E_1 \wedge E_2 \wedge \ldots) = \min \{CF(E_i)\} * CF(RULE)$$

$$\textit{IF } E_1 \vee E_2 \vee \ldots \textit{THEN } H \; CF(RULE) \textit{ for multiple premise disjunctive rules}$$
$$CF(H|E_1 \vee E_2 \vee \ldots) = \max \{CF(E_i)\} * CF(RULE)$$

The certainty model is simple to implement and is acceptable in many

applications; however, it lacks a strong statistical basis and relies basically on an expert's or user's judgemental belief concerning rules and facts.

3.6.2 Fuzzy Logic

Another approach is based on the theory of fuzzy sets and fuzzy logic [Zadeh 1965]. Fuzzy sets can represent imprecise concepts (e.g. coarse, heavy, high) that are commonly used by human beings to solve a problem. A membership function (or value) allows to map a quantitatively described concept to a fuzzy set.

For example, the sharpness of separation of a hydrocyclone in a grinding circuit is an imprecise concept. Obviously, this performance indicator can be estimated numerically and represented by a number; however, whether it is undesirable, desirable or highly desirable depends on engineering judgment. Using fuzzy sets, and based on a grinding expert's knowledge, three sets can be defined as UNDESIRABLE, DESIRABLE and HIGHLY DESIRABLE (Fig. 3.6).

Fuzzy rules, once fuzzy sets are defined, are used in KBSs to reason and make decision based on imprecise concepts. The following rule recommends to increase the number of operating cyclones to increase separation sharpness:

IF separation sharpness is low and
recovery of fluid to cyclone underflow is low
THEN put more cyclones in operation

One of the advantages of fuzzy logic is that it allows for uncertainty with data, which can be desirable in many cases. However, in developing any fuzzy KBSs, defining fuzzy sets and membership functions remains as a serious stumbling block to be overcome by the knowledge engineer and domain expert.

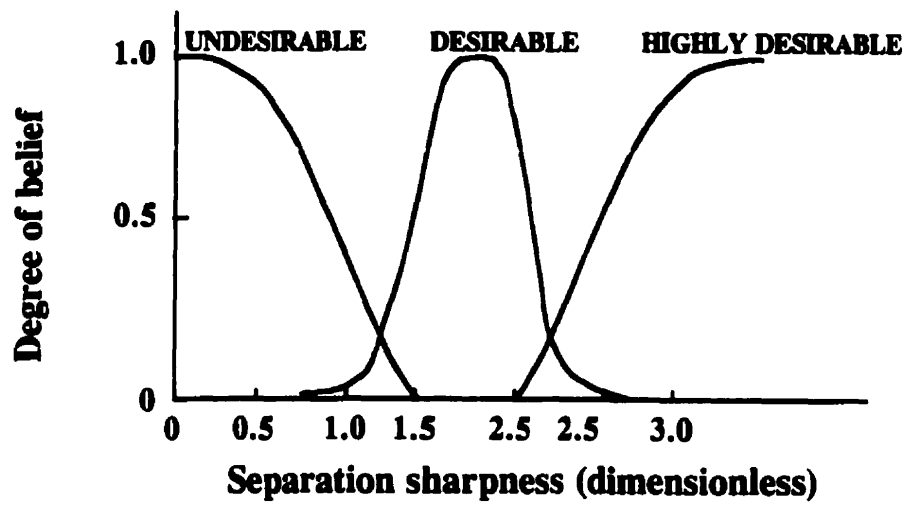


Figure 3.6 Fuzzy sets for a hydrocyclone separation sharpness

Applications of fuzzy set theory and fuzzy logic in mineral processing, with reference to monitoring and control systems, have been discussed at length elsewhere [Harris and Meech 1987; Meech 1992; Inoue and Okaya 1993; Beale, Prisbrey and Demuth 1994]. Fuzzy reasoning in design has been discussed by Coyne et al., 1990.

3.7 Knowledge Engineering Tools

A knowledge engineering tool is a programming environment that allows easier development of a knowledge-based system or expert system. Development tools are primarily assessed in terms of their capabilities for knowledge representation, inferencing and knowledge acquisition. Hence, there is a wide range of KBSs tools available which varies from simple shells to hybrid systems.

KBSs tools are built using programming languages which are either problem-oriented such as FORTRAN, C and PASCAL or symbol-manipulation languages such

as LISP and PROLOG. The reader is referred to [Waterman 1986] for a detailed review of expert system tools.

3.7.1 C Language Integrated Production System (CLIPS)

To construct the *G*rinding *C*ircuits *O*ptimization *S*upervisor (GCOS), CLIPS version 6.0 was selected as the shell and development tool. CLIPS is primarily a forward-chaining rule-based system, developed in C by NASA [Donnell 1994]; however, it allows for *O*bject *O*riented *P*rogramming (OOP). *C*LIPS *O*bject-*O*riented *L*anguage (COOL) supports the creation of classes with multiple inheritance, instances, message passing constructs, and query system [STB 1993, Donnell 1994].

CLIPS has a number of constructs to define rules or classes using a specialized syntax. However, patterns and pattern templates can have free formats. There are pre-defined functions in CLIPS that can be used to perform an action, either at command line or inside rules. CLIPS also has a *T*ruth *M*aintenance *S*ystem (TMS) through the use of logical dependencies. The truth maintenance capability of CLIPS allows its inference engine to retract automatically any fact that loses its logical support [Donnell 1994].

3.8 On-Line Applications of KBSs

KBSs have been applied to grinding and flotation circuits since late eighties; they are mostly known as expert systems [Meech 1990, Bearman and Milne 1992]. Table 3.3 lists expert systems installed in a number of grinding circuits to monitor and control their operations.

Though these systems have targeted operational problems that could be solved on-line, they are not radically different from off-line systems that address problems that need to be solved off-line. While on-line systems can be run in either closed loop (without plant operator's interventions) or open-loop just as a consultative or recommendation

system, their main difference with off-line optimization systems is the type of problems that they can solve. For this reason an overview of these projects is presented here.

Table 3.3 Expert systems applied to mineral grinding circuits

Site	Circuit type	Task	Development tool	Reference
Dome Mine, Ontario, Canada	rod mill/ball	control	Comdale/C	Eggert and Benford 1994
Wabush Mine, Labrador, Canada	autogenous/spiral	monitoring and control	Comdale/C	McDermott et al. 1992
Les Mines Selbaie, Quebec, Canada	semi-autogenous grinding (SAG)	on-line optimization	Comdale/C	Perry and Hall 1994
Brenda Mines Ltd., Canada	rod mill/ball mill	control	Superintendent	Spring and Edwards 1989
Kiruna LKAB Concentrators, Sweden	secondary grinding, pebble mills	control	N/A	Samskog et al. 1995
Mexicana de Cobre, La Caridad Unit, Sonora, Mexico	primary ball mills/hydrocyclones	control	N/A	Herbst et al. 1995

3.8.1 Brenda Mines Ltd.

A real time expert system was implemented by Brenda Process Technology [Spring and Edwards 1989] to control one of their rod mill/ball mill grinding circuits. SUPERINTENDENT, written in Pascal, was used as the expert system shell and is based

on a supportive control package called ONSPEC; both were supplied by the Heuristics Inc. Brenda developed and encoded the knowledge base, GRINDX, which contains rules to control the #2 grinding circuit.

The main goal of this project was to introduce expert systems to mineral processing plants and to explore if they could achieve the performance of traditional supervisory control systems. When running on-line, Superintendent could take supervisory control actions (manipulating PID set points, issuing messages, etc.) according to the rules in GRINDX. The system was found to perform as well as the conventional supervisory control system.

3.8.2 Les Mines Selbaie

An expert system was added to the automatic control system in May 1992 [Perry and Hall 1994] to optimize the A1 closed grinding circuit by manipulating the set-points of existing PI control loops. The knowledge base containing (fuzzy) control rules is run under the Comdale/C shell. The operating expertise, extracted from interviews with plant control personnel, was represented by 188 rules and 69 fuzzy sets.

Figure 3.7 illustrates the structure of the knowledge base and its execution cycle. The knowledge base has four sets of rules: (1) to validate the measured data (2) to verify consistency of the measured and inferred data (3) to identify process problems (4) to correct identified problems by changing related set-points. In each cycle, the system applies rules sequentially to identify and solve seven problems: SAG (*S*emi *A*utogenous *G*rinding) mill overloading, SAG mill under utilization, SAG mill feed rate instability, SAG mill ball addition, screen overloading, flotation feed size off specification and flotation feed rate density off specification. The system was reported to create increases in throughput and reductions in steel consumption.

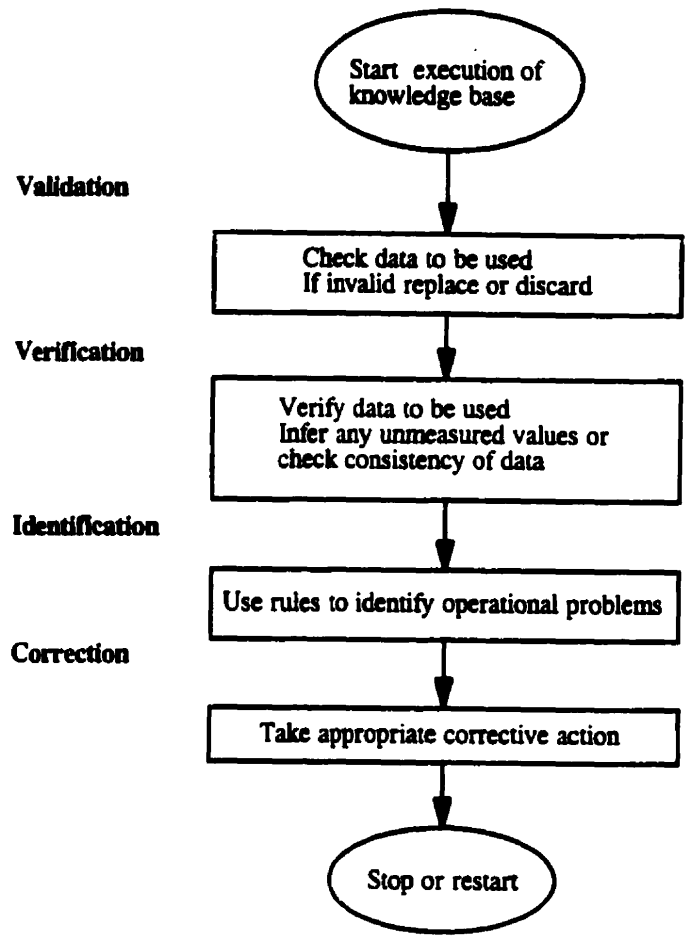


Figure 3.7 Expert control system at Les Mines Selbaie [Perry and Hall 1994]

3.8.3 Wabush Mine

The expert system was developed jointly by Wabush Mines and Comdale Technologies to improve spiral recovery and mill productivity by monitoring and control of No. 4 autogenous mill and spiral circuits [McDermott et al. 1992]. The system was designed (1) to monitor mill power draw and regulate mill feed rate, (2) improve control over the spiral density and feed rate, (3) improve control over the rougher and cleaner wash water, (4) reduce the frequency of pumps plugging and (5) to allow easy modifications to test various operating strategies.

The system was developed using the Comdale/C shell. The knowledge base included 76 rules, 35 fuzzy sets to recognize process states, 19 fuzzy sets for control actions, 25 fuzzy sets to identify trends and 24 variables monitored for time variation. The rules and fuzzy sets embody the operating and control expertise of Wabush Mine personnel. A sample rule which uses fuzzy sets looks like:

IF *mill power draw is high and trending upward fast and*
recirculating density is not too high and not trending upward and
recirculating sump level is too high and trending upward
THEN *reduce mill feed water by small amount*

The knowledge base was tested at each stage of project development. These tests included (1) an off-line evaluation by developers at Comdale's office, (2) an on-line evaluation in monitoring capacity at plant site and (3) a final on-line evaluation in controlling capacity at plant site. The system was reported to produce a 20% higher production performance when compared to the performance of the other five parallel mill lines. The other reported benefits included reduced process down-time caused by plugged pumps and improved quality of spiral concentrate.

3.8.4 Dome Mine

An expert system was developed by Comdale Technologies with the objective of increasing circuit tonnage [Eggert and Benford 1994]. The grinding circuit knowledge base was written using the Comdale/C expert system shell to supervise the *D*istributed *C*ontrol *S*ystem (DCS). The knowledge base consisted of simple rules with an O-A-V structure to represent process information and also fuzzy rules to implement a fuzzy logic control scheme.

The system was evaluated on-line by mill operators and was accepted because of a discernable increase in tonnage [Eggert and Benford 1994]. This was achieved despite claims that circuit operation had already been optimized.

3.8.5 Kiruna LKAB Concentrators

Samskog et al. [1995] have discussed implementation of a *M*odel-*B*ased *E*xpert *C*ontrol *S*ystem (MBEC) in the Kiruna LKAB concentrators, Sweden. MBEC systems have been installed for dynamic optimization of the three old pebble mill circuits and the new concentrator.

Figure 3.8 shows the software structure of pebble mill MBEC system at the KA1 old Kiruna concentrator. The expert system maintains the proportion of <44 μm in hydrocyclone overflow within very narrow limits. It also controls the volumetric feed rate to hydrocyclones based on a process model or rules.

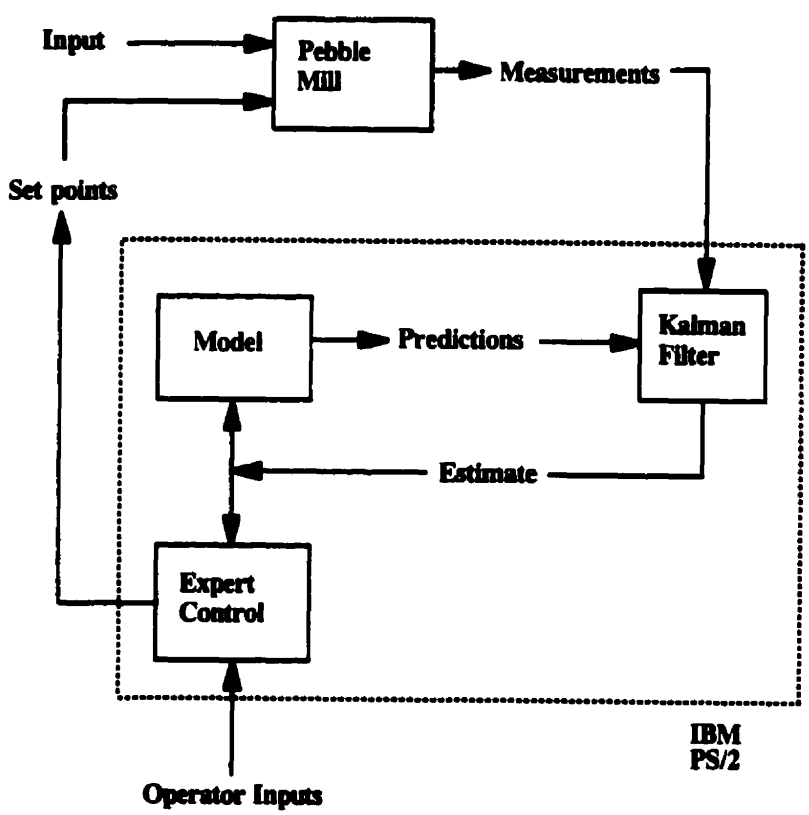


Figure 3.8 Pebble Mill MBEC system at KA1 old Kiruna concentrator [Samskog et al. 1995]

The control system for the new Kiruna plant consists of modelling and expertise modules (Fig. 3.9). The modelling module includes dynamic process models which continuously calculate and predict the state of the process. The expertise module includes sets of rules which use information generated by the modelling module.

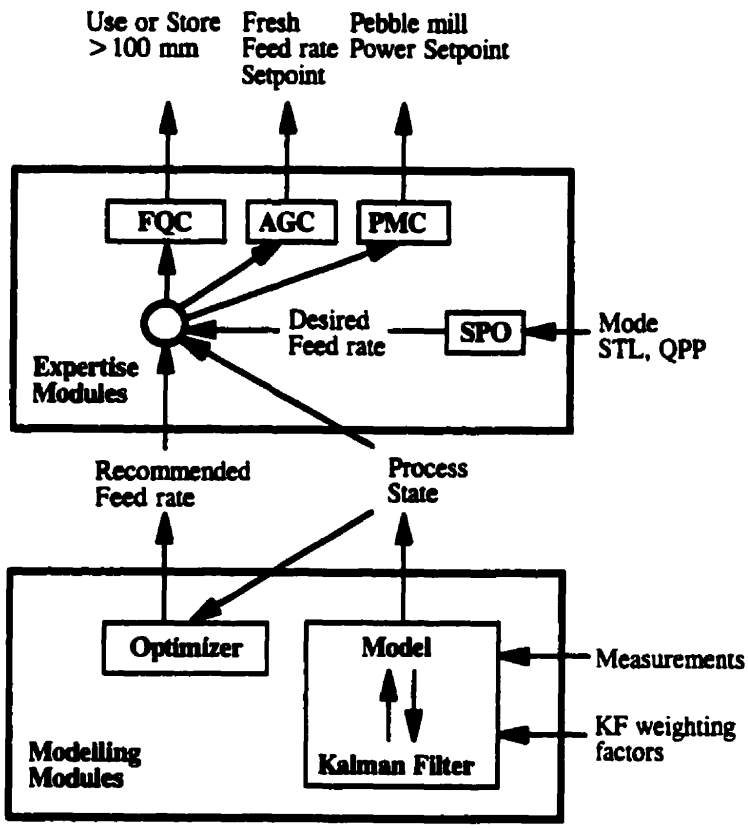


Figure 3.9 MBEC system at Kiruna new plant [Samskog et al. 1995]

3.8.6 Mexicana de Cobre

As reported by Herbst et al. [1995], Mexicana de Cobre in Sonora, Mexico, is one of the mineral processing plants close to achieving a plantwide control and overall plant optimization (i.e. simultaneous manipulation of all significant variables across the whole plant) through implementation of a knowledge-based supervisory control system

(Fig. 3.10). The purpose of the plantwide coordination strategy is to maximize the plant production by optimizing throughput, recovery and concentrate grade.

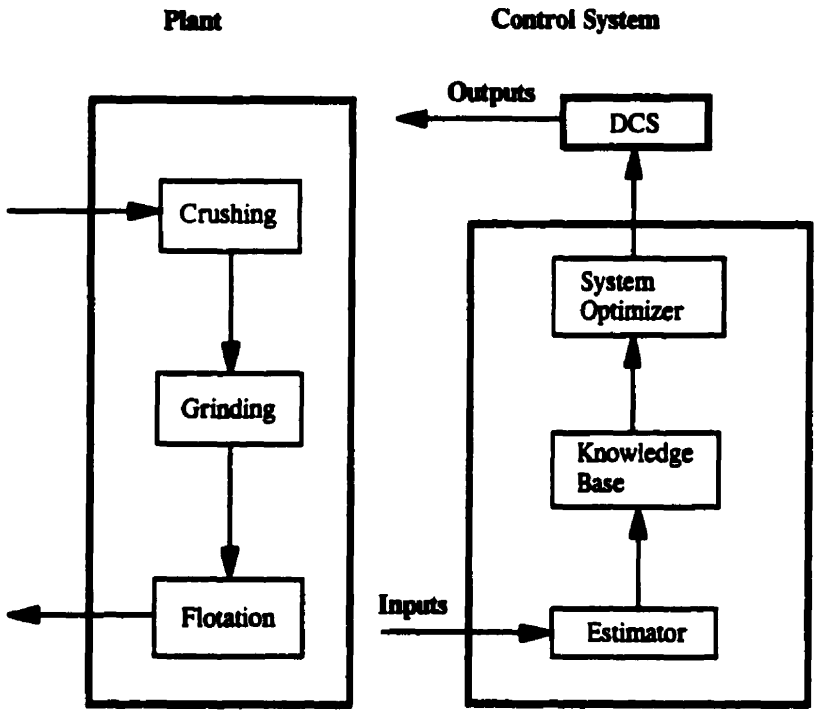


Figure 3.10 Knowledge-based control system at Mexican La Cobre [Herbst et al. 1995]

The control strategy utilizes knowledge and information that originate from a variety of resources: (1) heuristics based on practice of the best operator, (2) process models to estimate variables that cannot be measured on-line and (3) neural networks for processes that cannot be modeled accurately because of their inherent complexity.

3.9 Off-Line Applications of KBSs

There are a few commercial computer programs which have been developed for off-line study of grinding circuits' design, analysis and optimization (Table 3.4).

These programs are a suit of mathematical models of process units (phenomenological or empirical) which allow steady-state or dynamic simulation of most industrial circuits. They have various capabilities in terms of availability of models, accuracy of models, parametric estimation, user interface, etc. Nevertheless, none of them has been reported yet to have integrated KBSs components.

Table 3.4 Mineral processing simulation software

Simulator	Organization	Reference
JKSimMet	JKMRC (Australia)	Napier-Munn and Lynch 1992, Wiseman and Richardson 1991
USIM-PAC	BRGM (France)	Durance et al. 1993
MICROSIM	Univ. of Witwatersrand (South Africa)	Cilliers and King 1987
MetSim	Proware (USA)	Charlesworth 1998
Utah-MODSIM	Univ. of Utah (USA)	Herbst, Schena and Fu 1989
MODSIM	Univ. of Witwatersrand (South Africa)	Ford and King 1984
DYNAFRAG	Univ. of Laval (Canada)	Dubé and Hodouin 1982
SPOC	CANMET (Canada)	Laguitton 1985

Researchers possessing adequate knowledge in mineral processing modelling and simulation have successfully used these types of computer programs to improve the design and operation of industrial grinding circuits. However, plant metallurgists and process engineers with less theoretical background in process modelling and simulation might find it difficult to use these computer programs effectively and on a routine basis. Modelling and simulation is a multi-stage and complicated activity which demands a broad theoretical knowledge and computer skills. Most of the software tools used in modelling and simulation practice provide solely numerical algorithms, either in a single package or separate pieces, to do various computations, but they usually fail to provide

the adequate problem solving knowledge which is required to use them. This type of problem solving knowledge is mostly non-numerical and can be integrated into conventional simulators using KBSs.

3.10 Summary

Theoretically, KBSs are the best currently available tools to model human problem solving expertise. KBSs provide tools to represent and manipulate types of knowledge used by a human expert to solve complex real world problems. KBSs applied to monitor and control grinding circuits have been proved to be able to increase productivity. These systems have been able to improve circuit operation by addressing problems that inherently belong to the subject of on-line optimization. In this thesis, however, the KBS approach is applied to cases which must be solved off-line. These problems include fundamental aspects concerning a grinding circuit such as its layout and its optimal steady-state. The installation of an on-line monitoring and controlling system is considered as the next optimization step. The scope of this work is limited to the main aspects of off-line grinding optimization, namely grinding kinetics and classification performance. To develop a KBS as a tool to assist one in off-line grinding optimization studies, it is crucial to clarify the problem solving knowledge used by a human expert first. In Chapter 4, cases of industrial grinding circuit optimization done during this project to acquire and develop the knowledge base will be discussed.

CHAPTER 4
NUMERICAL* GRINDING OPTIMIZATION TOOLS IN C (NGOTC)

4.1 Introduction

The off-line steady-state optimization of mineral grinding circuits requires the use of special computer programs at each stage of the optimization process. For example, these programs are essential to perform material balance calculations, estimations of parameters of various models and process simulations. The *N*umerical *G*rinding *O*ptimization *T*ools in *C* (NGOTC) provides some of these tools. This program currently contains routines to estimate the selection function of continuous wet-grinding ball mills, scale an estimated selection function according to ball size and simulate continuous wet-grinding ball mills. Other tools that might be needed in off-line optimization studies and are not part of the current version of NGOTC are algorithms to determine the breakage function and residence time distribution. The "BREAK" and "RTDOPEN" programs written in TBASIC and BASIC, respectively, are being used at McGill University to determine these parameters [Laplante 1996].

In this chapter, the main algorithms of NGOTC and their applications in several studies of industrial grinding circuits will be explained. The NGOTC can be effectively used for grinding process analysis, modelling and optimization. The optimization of grinding medium size (normally balls and slugs/truncated cones) uses all three modules. Theoretical aspects of the algorithms will be discussed as well.

---

* The term 'numerical' is emphasizing the numerical nature of these programs, in contrast with symbolic ones.

All grinding circuit simulation programs rely on various mathematical models to predict the performance of the various units under specific operating conditions. The general form of these models must be rendered specific to a real process by replacing model parameters by their numerical values. Although significant progress is being made in estimating these parameters from fundamental principles, the most common approach still relies on back-calculation of these parameters from sampling of full circuits [Rajamani and Herbst 1984]. Parametric estimation and model validation are required pre-simulation stages. A simulation package must normally provide tools to facilitate these stages.

4.2 The Structure of NGOTC

4.2.1 Modules

Figure 4.1 is a simple diagram of the NGOTC internal structure. A modular system was designed which eased program development and reduced efforts for future maintenance and extension. The NGOTC package has been implemented as a C multi-file project. All source code files (files with C01 extension) are separate modules that must be compiled before linking to other compiled (objective) files. After linking the objective files, a single executable file is produced which can be run under MS-DOS environment.

Modules which provide utility functions, spline curve fitting, graphics and memory management are general algorithms, while selection function estimation, selection function scaling and single ball mill simulator are specialized algorithms. The former modules provide functions or sub-routines that are called by the latter modules.

4.2.2 User Interface and Data Entry

The NGOTC has a textual user interface which includes data entry and output screens. The data can be read from existing text files as well. In case of an incorrect input, an error message will be displayed. The output can be sent to a printer or saved

to a text (ASCII) file. The selection function estimation module provides the graphical representations of the mill feed and discharge size distribution data. The selection function scaling module enables the user to see a graphical representation of the estimated selection function. The ball mill simulator module also enables the user to see the feed size distribution and selection function data graphically.

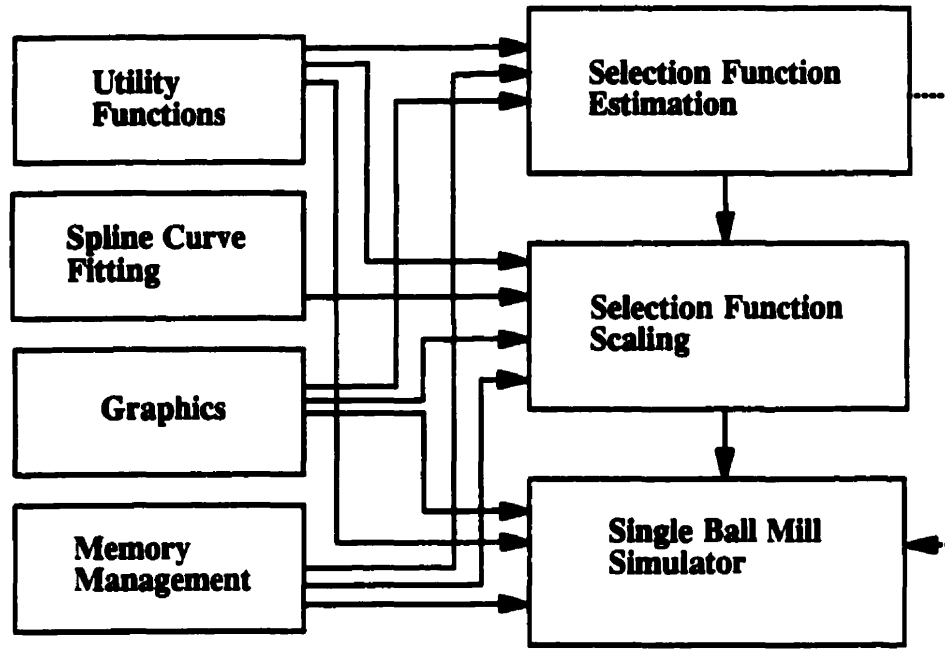


Figure 4.1 Simplified structure of NGOTC

4.3 Modelling of Selection Function Data

Modelling of selection function data means the fitting of a mathematical function to selection function elements determined experimentally or back-calculated from a set of mill data. Often a single polynomial function or multiple polynomials is used to fit the selection function data points. Multiple polynomials are also known as spline

functions.

The quality of the estimated selection function can be upgraded by fitting polynomials or spline functions since they can remove existing noise originating from many contributing sources. A spline curve fitting algorithm was used in NGOTC as part of the scaling procedure.

4.3.1 Polynomial Function Models

Power functions can often fit the estimated selection functions of grinding mills [Spring, Larsen and Mular 1985]:

$$\ln(S_i)=\ln(s_1)+s_2\ln\left(\frac{x_i}{x_0}\right) \tag{4.1}$$

$$\ln(S_i)=\ln(s_1)+s_2\ln\left(\frac{x_i}{x_0}\right)+s_3\left[\ln\left(\frac{x_i}{x_0}\right)\right]^2 \tag{4.2}$$

$$\ln(S_i)=\ln(s_1)+s_2\ln\left(\frac{x_i}{x_0}\right)+s_3\left[\ln\left(\frac{x_i}{x_0}\right)\right]^2+s_4\left[\ln\left(\frac{x_i}{x_0}\right)\right]^3 \tag{4.3}$$

$$S_i=\frac{s_1(x_i/x_0)^{s_2}}{1+\left[\frac{(x_i/x_0)}{s_3}\right]^{s_4}} \tag{4.4}$$

where S_i is the selection function value for size class i and s_1, s_2, s_3 and s_4 are constants, x_i is the geometric mean size of size fraction i; x_0 is the reference size and is equal to 1 mm.

The above functions have been used in the FINDBS module of the SPOC software to estimate selection functions [Laguitton et al. 1985]. To choose the model which optimally estimates the selection function is often a matter of trial and error. The objective function is normally the criterion used to select the best model. A model that generates the minimum value of the objective function is considered the best one. The

objective function is defined as the sum of the squared differences between the measured and simulated size distributions for the circuit product or all streams.

4.3.2 Spline Function Models

In many cases, it has been observed that the estimated mill selection functions using a sequential interval-by-interval method show no obvious trend and are scattered. In these cases the selection function cannot be modelled by a single polynomial function satisfactorily, and the use of spline functions to model data piecewise is preferred.

Rather than simple polynomial functions, spline functions have been used by a number of investigators [Whiten 1971, Whiten 1972, Morrell 1990] to fit selection function data sets. A spline regression is less likely to impose an inappropriate form or relationship onto data. The spline curve fitting algorithm developed by Dierckx was originally in FORTRAN, which was converted to C to be integrated to other modules of the NGOTC. This algorithm has more flexibility than other models to fit most estimated mill selection functions. The theory of spline functions is discussed in De Boor 1972, Cox 1972 and Whiten 1971. The algorithm developed by Dierckx has been briefly explained in Appendix B.1.

This algorithm allows fitting a smooth spline curve to discrete selection function data. When there are several selection function data sets, the average selection function can be used. In such cases, a standard deviation can be associated to each selection function element that will be used for a weighted curve fitting. The degree of smoothness of the curve is controlled by the smoothness factor which is set by the user. However, there is a compromise between the goodness of the fit and smoothness of the curve.

4.4 Estimation of Selection Functions

The selection function parameter, in size-discretized S_i or continuous S(x) forms, can be estimated for industrial or laboratory mills. The selection function of industrial mills, running in continuous wet or dry mode, can be determined indirectly from feed and discharge size distribution and operating data using back-calculation methods. In the laboratory, the selection function of test mills can be determined directly by performing one-size fraction grinding experiments or back-calculated. Although general spreadsheet software such as Excel* or Quattro Pro† can be used for numerical calculations with a higher flexibility, a dedicated program is more desirable for very complex calculations.

The accuracy of calculated particle size distributions and other predictions made by a grinding circuit simulator is highly dependent on the accuracy or quality of parametric estimations used to calibrate the mathematical unit models used. Hence, it is critical to determine the selection function of an industrial grinding mill as accurately as possible in order to obtain reliable simulation results.

4.4.1 Back-Calculation from Continuous Mill Data

The selection function estimation module of the NGOTC consists of an algorithm to back-calculate a set or a vector of selection function elements based on a set of input data. The selection function elements are back-calculated by trial and error using a bi-section search procedure. The selection function elements are back-calculated sequentially, i.e. first the selection function element for the top size class is determined, then using this estimated value, the selection function of the second size class will be estimated. The core of the algorithm is in fact a single ball mill simulator which produces product size distributions. The criterion to stop the iteration process is the difference between the measured and predicted mass of the current size class, which must

*Excel is a trade mark of Microsoft incorporation

†Quattro Pro is a trade mark of Borland incorporation

be within a tolerance interval set by the user.

The estimated selection function elements, then, can be used as input to other modules -- i.e. the selection function scaling and single ball mill simulator.

4.4.2 Analysis of Selection Functions

The selection function of tumbling ball mills has been studied by a number of researchers [Verma and Rajamani 1995]. The shape of the selection function vs. particle size curve is of critical importance in mill performance analysis, which can indicate potential problems regarding grinding media inefficiency. In addition to grinding media other grinding environment parameters such as mill lining can affect grinding kinetics seriously.

In this thesis, only the effect of ball size was studied irrespective of the effects caused by other variables such as the state of mill lining and mill size. This means that for a given mill and ore, the effect of ball size on grinding kinetics can be studied by estimating the selection function and determination of any abnormal breakage. The plot of the selection function vs. particle size data is often a straight or curved line. Due to the strong relationship between the grinding kinetics and grinding media size, the shape of the selection function vs. particle size curve is used as an indicator of grinding media efficiency.

4.5 Ball Size Selection

The size of balls used as the grinding medium strongly affects comminution performance [Austin, Shoji and Luckie 1976] and its optimal selection can improve mill performance and operation costs [Dunn 1989]. A single make-up size or a multiple make-up size is used in charging practices which are often based on a media rationing investigation [McIvor 1997]. The final decision to use a certain ball size is usually taken by trial and error [Wills 1997] and based on economic considerations. There have been

attempts to find the optimal ball mixture [Austin and Klimpel 1985; Vermeulen 1986; Concha, Magne and Austin 1992]. The idea of using a grinding simulator capable of taking into account ball size effects on mill performance has been suggested by some researchers.

Grinding kinetics, represented quantitatively by the selection function, has been used as a performance indicator to investigate the effect of a change in media shape or size on grinding efficiency [Herbst and Lo 1989; Cooper et al. 1993; Cooper, Bazin and Grant 1994; Staples, Cooper and Grant 1997]. To realize a change in grinding performance due to a change in media shape or size, these researchers based their investigation on changes to selection functions after making a real change of media shape or size. This approach however is more tedious and costly than a simulation approach.

Since simulation is cheaper and faster than experimental approach, one objective of this work was to integrate all the necessary tools in one demonstration package. The linkage of Morrell's scaling procedure to selection function estimation and ball mill simulation algorithms makes this possible.

4.5.1 Overall Selection Function for Balls Mixture

The size-by-size selection function elements estimated by the NGOTC are in fact the weighted average of selection function elements for various ball sizes. In other words, they are overall selection function values for a given mixture of ball sizes. Theoretically it can be written [Austin, Shoji and Luckie 1976]:

$$\overline{S_i}=\sum_{k=1}^{m} S_{i,k} m_k \qquad (4.5)$$

where $\bar{S}_i$ is the weighted average selection function of size class i, $S_{i,k}$ is the selection function of size class i with ball size k and m_k is the weighting factor which is fractional

mass of ball size k.

4.5.2 Selection Function Scaling According to Ball Size

The selection function as a general model parameter is significantly dependent on mill design and operating variables. For example, mill speed, mill diameter, ball size and particle size affect the selection function. Their effects are normally represented by empirical relationships when optimizing existing mills, however, one of the most important parameters that must be considered and used is ball size since other parameters such as design variables (mill diameter, liner design) are not easily changed.

The scaling procedure [Morrell 1990, Morrell and Man 1997] to predict the selection function of a mill due to a change in ball size is based on the assumption that two main size reduction mechanisms simultaneously take place in a mill, i.e impact and attrition breakages.

The amount of broken material due to these breakage mechanisms mainly depends on the ball mass and the surface area available. In turn, ball mass and surface area are directly and indirectly proportional to ball diameter as below:

$$ball\ mass \propto d_b^3 \tag{4.6}$$

$$surface\ area \propto d_b^{-1} \tag{4.7}$$

where d_b is grinding ball diameter.

The impact breakage is affected by both ball mass and surface area [Morrell and Man 1997] and therefore by combining proportionalities 4.6 and 4.7 it can be written:

$$Impact\ breakage \propto d_b^2 \tag{4.8}$$

Attrition breakage is considered to be affected only by ball surface area, therefore:

$$Attrition\ breakage \propto d_b^{-1} \tag{4.9}$$

Impact and attrition breakages occur below and above of a certain particle size which is defined as:

$$x_m = Kd_b^2 \tag{4.10}$$

where x_m is the particle size (mm) at which the selection function is maximum and K is the proportionality factor (1/mm) between 10^{-3} and $0.7\text{x}10^{-3}$ [Austin, Shoji and Luckie 1976]. The K factor depends on ore hardness and grindability, with higher values for softer material. d_b is in mm.

In order to scale from a large to small ball size, the following relationships can be written:

$$S_{i,s}=S_{i,l}\left(\frac{d_{b,l}}{d_{b,s}}\right) \qquad if\ \ x_i \le x_{m,s} \tag{4.11}$$

$$S_{i,s}=S_{i,l}\frac{1}{\left(\frac{d_{b,l}}{d_{b,s}}\right)^2} \qquad if\ \ x_i \ge x_{m,l} \tag{4.12}$$

where

$S_{i,s}$ and $S_{i,l}$: selection functions of size class i for small and large ball sizes

$d_{b,s}$ and $d_{b,l}$: diameters of small and large balls

$x_{m,s}$ and $x_{m,l}$: particle sizes corresponding to the maximum selection functions

Since there is no theoretical relationship to directly scale selection function elements at particle sizes between $x_{m,s}$ and $x_{m,l}$, these selection function elements are determined by evaluating the fitted spline function at these particle sizes. Their accuracy depends mainly on the reliability of the spline curve, which decreases with increasing the interval between $x_{m,s}$ and $x_{m,l}$. Since particle sizes corresponding to maximum grinding kinetics are related to hypothetical and current ball sizes, therefore, these selection functions are more accurate when the hypothetical ball size is close to the current one.

The accuracy of mill product size distributions predicted using this procedure has been verified [Morrell 1990] for a limited number of industrial cases which showed a good agreement between predicted size distributions and measured ones for two of three case studies performed.

Figure 4.2 illustrates the selection function scaling algorithm proposed by Morrell. This algorithm was implemented in C and integrated with the selection function estimation module.

4.6 Industrial Applications

NGOTC was tested and validated in a number of process analysis and simulation studies of industrial wet grinding operations. A number of plant data sets were collected from existing grinding circuits to investigate grinding kinetics and optimize ball size. Furthermore, these data were used to verify the reliability of NGOTC computations and to develop the *B*all *M*illing *C*ircuits *S*imulator (BMCS) program and the knowledge base of *G*rinding *C*ircuits *O*ptimization *S*upervisor (GCOS). The BMCS and GCOS programs will be explained in Chapters 5 and 6.

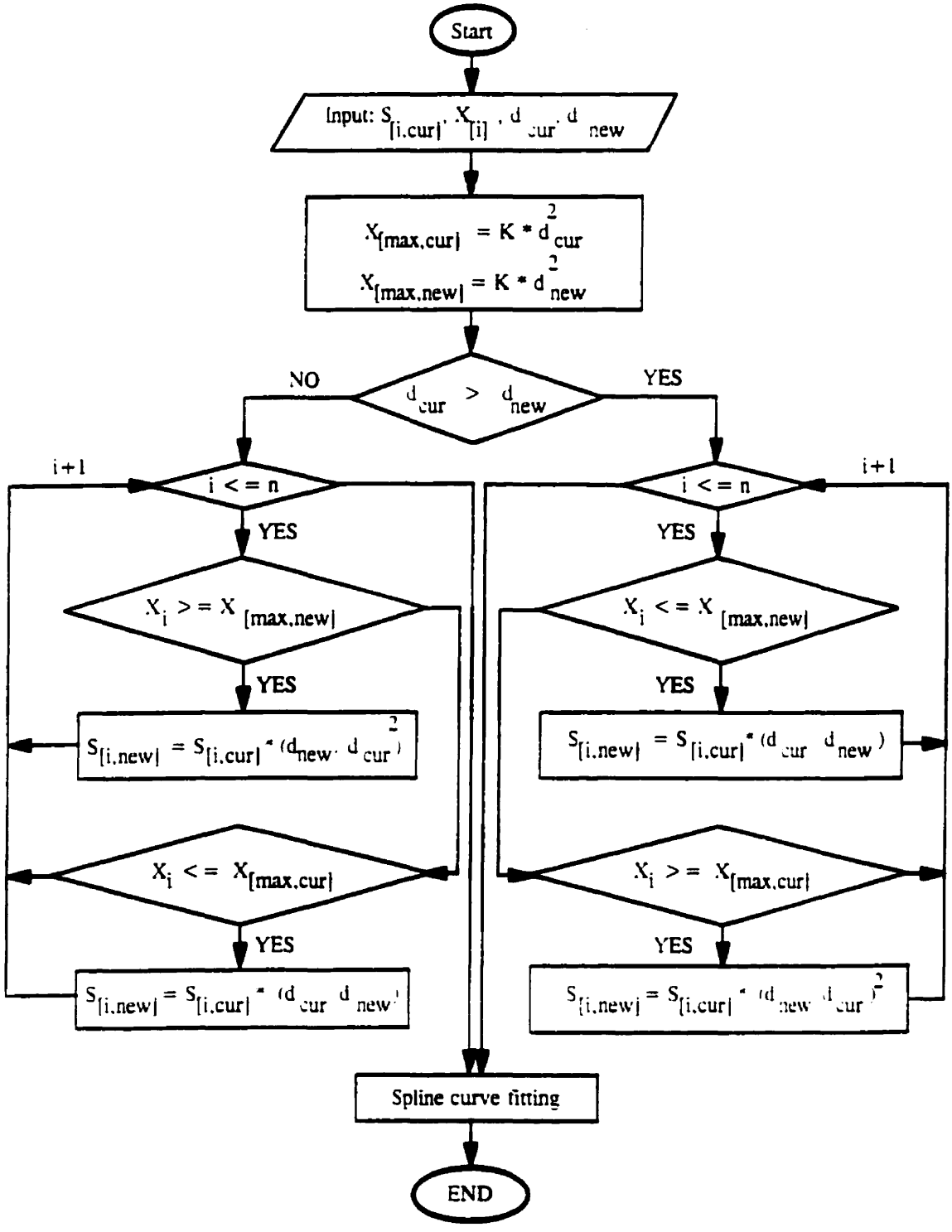


Figure 4.2 The selection function scaling algorithm as implemented in NGOTC

4.6.1 Agnico-Eagle, La Ronde Division

The Agnico-Eagle, La Ronde Division (AELRD) plant is located in Cadillac, Québec, and treats approximately 71 t/h of a gold and copper ore [Laplante et al. 1995]. The AELRD plant has two identical grinding lines consisting of single stage ball mills fed by finely crushed ore and operated in closed circuits with cyclopaks and Knelson Concentrators (KC) units. Currently, two cyclopaks, each consisting of five 25 cm (10") cyclones, are being used, having replaced two single 46 cm (18") hydrocyclones. Some difficulties with classification have been reported after using cyclopaks instead of individual but larger cyclones.

The sampling program included two sampling runs performed on July 13 1996. The slurry samples were taken from various streams of both grinding lines to study grinding kinetics and cyclopak performances. To ensure a high quality sampling campaign, all sampling cutters were first cleaned and examined to detect leaks, damage or incorrect design. A composite sample of twelve increments of slurry was extracted from each pre-identified stream over a two-hour period, once it had been established from the control room and discussions with operators that both circuits were in steady state.

Figure 4.3 shows the simplified flowsheet of both grinding circuits. At the time of sampling, the ball mills were being fed with the ore coming from shaft No. 2 and three cyclones out of five were in operation. A blend of balls with different sizes, 75% 76 mm and 25% 38 mm, was being charged to ball mill of Line 1 as make-up media, and had been for at least two months to reach equilibrium. A make-up charge of balls with the same size (100% 76 mm) was used for ball mill of Line 2. The values of fresh feed rate, circulating load, density of cyclopak feed and amperage for the sampling period were read and recorded every 10 minutes from the monitoring terminal located in the control room; the average values were then used in data analysis.

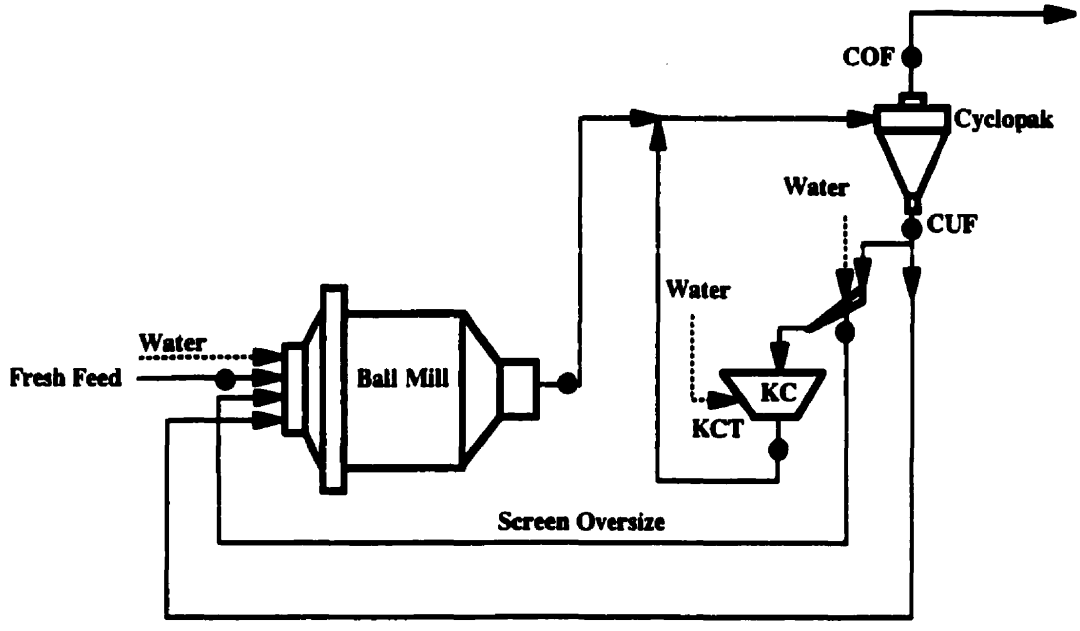


Figure 4.3 Simplified flowsheet of AELRD grinding circuits and sampling points

The raw data were first adjusted using the NORBAL3 software [Spring 1992]. The adjusted data which conform to the mass conservation law are considered to represent the plant state better than raw data and therefore are used in process analysis. The result of mass balancing is given in Appendix B.2. Since the portion of the Knelson Concentrator feed which reports to concentrate stream was relatively small, it was assumed that Knelson tail and feed (or screen undersize) streams had the same flow rate and particle size distribution. This makes it possible to represent the circuit in a nodal form suitable for mass balancing in which the pre-concentration screen and Knelson Concentrator units are considered as a single unit (Fig. 4.4). The adjusted data were then used to evaluate the grinding kinetics of the ball mills and the classification performances of the cyclopaks.

The adjusted particle size distributions of fresh feed to ball mill, cyclopak underflow, screen oversize and ball mill discharge were used to study grinding kinetics

(for both lines). As the final feed to each ball mill is a mixture of three separate streams, their flow rates and particle size distributions were used to calculate the size distribution of the combined feed to each ball mill.

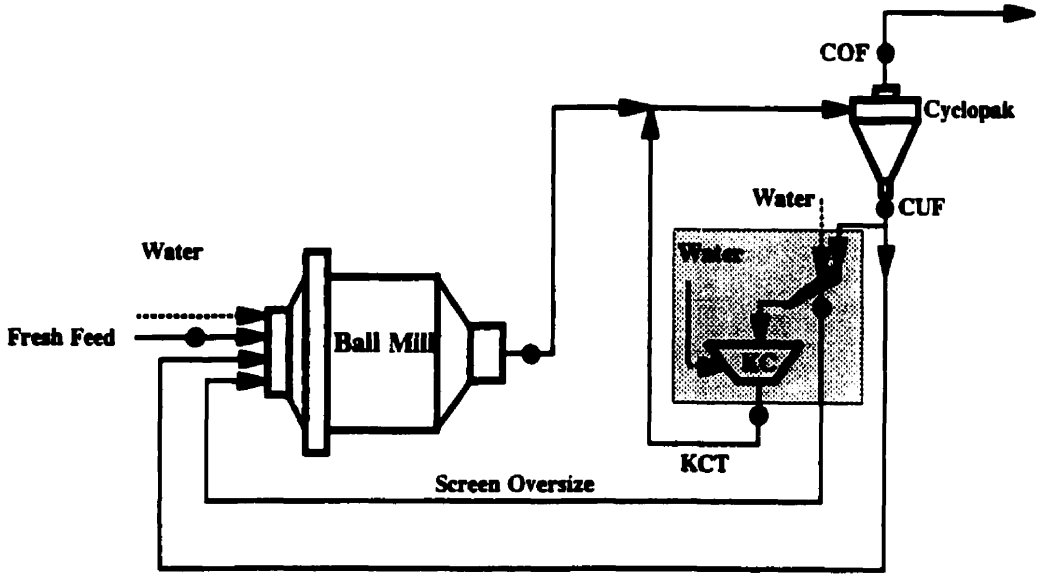


Figure 4.4 The pre-concentration screen and KC unit are considered as a single block for mass balancing

The selection function was back-calculated using the NGOTC [Farzanegan, Laplante and Lowther 1997] for both mills using assumed dimensionless or normalized parameters of Weller's RTD which yield a total mean retention time of unity for mill 1. The impact of using assumed RTD parameters instead of measured ones has been found insignificant if estimates are consistent and inversely proportional to dry feed rate [Laplante, Finch and del Villar 1987 and Laplante et al. 1995]. In order to compare grinding kinetics of the two mills, the dimensionless RTD parameters of mill 2 were obtained using RTD parameters of mill 1 as the reference set [Laplante and Redstone 1984]:

$$(\tau_{pf}^{*},\ \tau_{s}^{*},\ \tau_{l}^{*}) = \frac{Q_{ref.}}{Q_{sim.}} * (\tau_{pf},\ \tau_{s},\ \tau_{l}) \qquad (4.13)$$

where:

$Q_{ref.}$: the reference feed rate at which RTD parameters (τ_{pf}, τ_{s} and τ_{l}) were measured

$Q_{sim.}$: the current feed rate which is simulated

τ^{*}_{pf}, τ^{*}_{s} and τ^{*}_{l}: RTD parameters corresponding to the current feed rate

The results of the selection function estimation using NGOTC are given in Appendix B.3. Figure 4.5 shows the selection function vs. particle size curves for both ball mills. The obvious noise at the coarse end of the curves is attributed to the unreliability of screen analysis data regarding very low mass in these size classes, i.e. less than 1%. The problem of poorly estimated selection functions for coarse size classes has been also observed by Klimpel and Austin [1984]. The curves indicate that both lines have almost the same grinding kinetics without the presence of a desired hump. With this situation the top ball size does not necessarily need to be changed. However, the use of a finer blend charge (50% 75 mm and 50% 38 mm) was proposed, and has since been implemented.

Detailed evaluation of grinding kinetics (i.e. unit performance) should not detract one from overall considerations. The efficiency of the AELRD grinding circuit will always be limited in its present design, in that single stage grinding limits the judicious matching of ball size to the desired product size. This is why Agnico-Eagle should consider as a priority the modification of the present flowsheet in the expanded plant, preferably to SAG - ball milling. This would result in the ability to use finer balls for the energy-intensive ball milling step, thereby increasing grinding efficiency significantly. The present work has shown that using a ball mix cannot fully correct the loss of efficiency associated to single stage grinding.

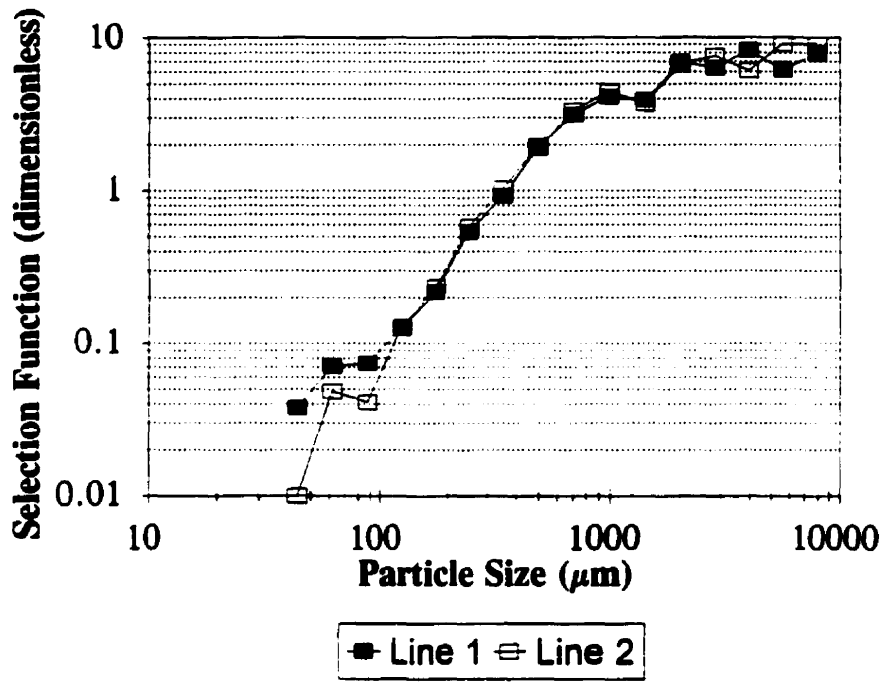


Figure 4.5 Breakage kinetics for both ball mills of AELRD plant (July 13 1996 survey)

4.6.2 Les Mines Selbaie

Les Mines Selbaie (LMS) is located about 80 km north of Joutel in Québec province. The mill produces copper and zinc concentrates and treats approximately 8000 t/d of ore in two separate grinding lines, A1 and B. The results of grinding kinetics analysis of circuit B using data from two surveys, conducted on July 15 1996 and February 9 1998, will be discussed here.

Figure 4.6 shows circuit B which includes a rod mill (2.22 m x 3.37 m), a ball mill (3.07 m x 3.92 m) and a tricone mill (2.87 x 3.23 m) configured in a series. The rod mill discharge is first comminuted in the ball mill circuit. The ball mill circuit

product then goes to the tricone circuit for further grinding. The ball mill and tricone mill are closed with primary and secondary cyclopaks consisting of 51 cm (20") and 38 cm (15") diameter cyclones. In the ball mill, 38 mm (1.5") balls are used as grinding media. In the tricone mill, 25 mm (1") slugs are used. The make-up ball charge practice for the ball mill, however, had been subject to modifications to find the optimum media size. A mix of 50% 25 mm (1") and 50% 50 mm (2") balls was being used at the time of February 9 1998 survey.

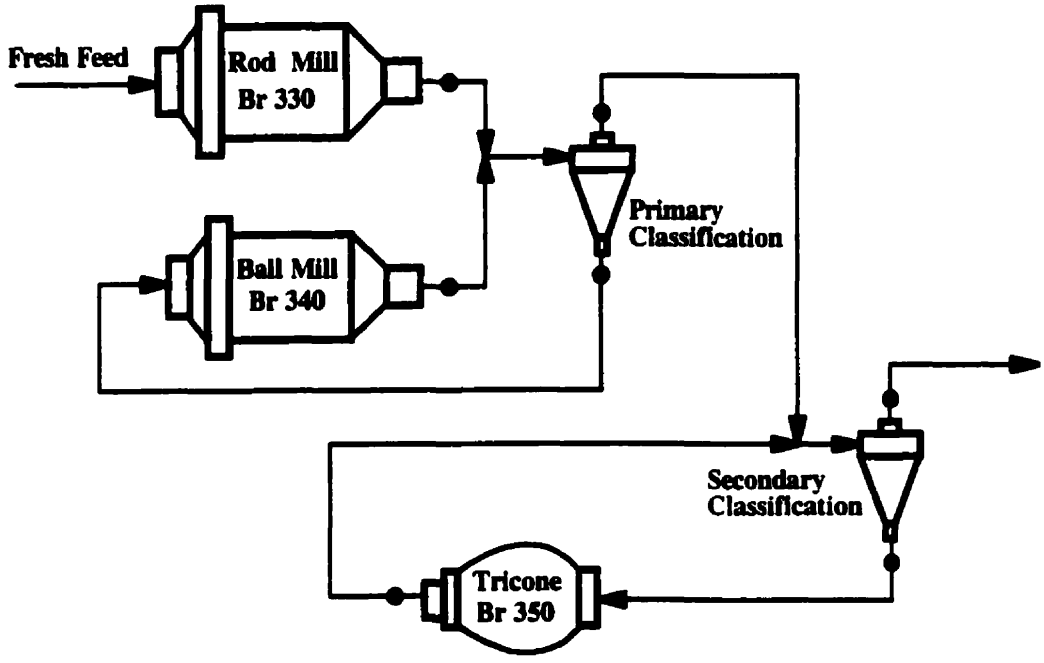


Figure 4.6 Simplified flowsheet of the grinding circuit B of LMS and sampling points

The July 15 1996 sampling campaign was performed by the principal thesis supervisor, candidate and plant personnel. Eight composite slurry samples, twelve increments over a two-hour period, were taken from various streams and sized from 11180 μm (2 mesh) to 25 μm (500 mesh). The February 9 1998 survey data was

supplied by LMS plant.

The size distribution data were adjusted with NORBAL3 [Spring 1992] before using them in various circuit analyses. The result of mass balancing is given in Appendix B.4. NGOTC was then used to estimate the selection function for both surveys (Appendix B.5). Figure 4.7 shows the selection function vs. particle size curve which has a strong hump (strongly parabolic) interpreted as the inadequacy of the grinding media to break the largest ore particles. The estimated selection function for February 9 1998 survey which was performed after using the mixed charge, has shown an obvious decrease in grinding kinetics for the smaller sizes.

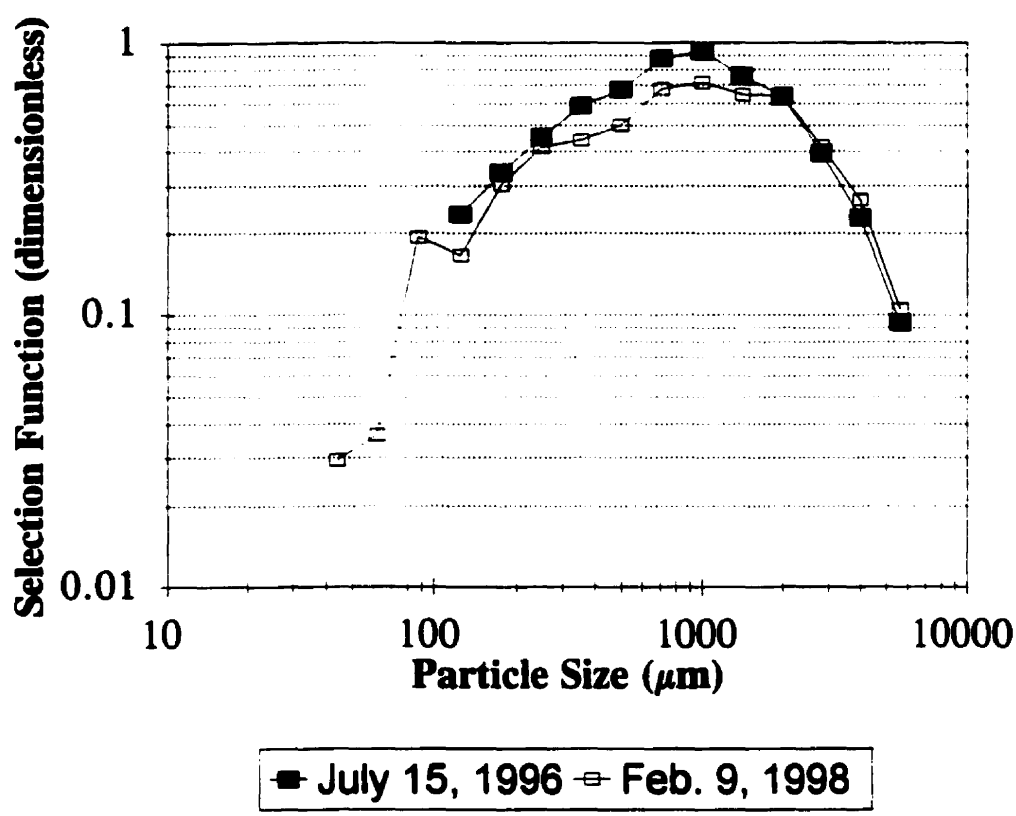


Figure 4.7 Breakage kinetics of the ball mill of grinding circuit B, LMS

Figure 4.8 shows the estimated selection function for the tricone mill for the July 15 1996 and February 9 1998 surveys. In this case while the selection function data shows an increasing trend, it is highly noisy. The sudden increase of grinding kinetics for February 9 1998 survey at a particle size around 300 μm is due to the erratic size distribution data of the coarsest size class of tricone mill discharge stream.

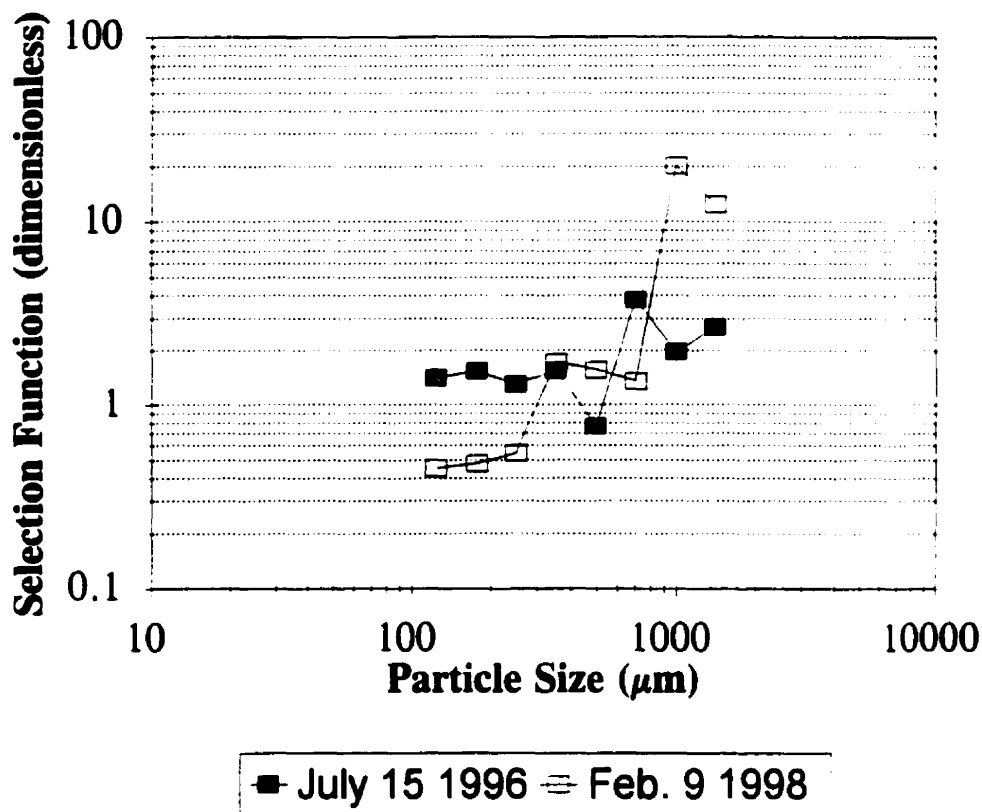


Figure 4.8 Breakage kinetics of the tricone mill of grinding circuit B, LMS

4.6.3 Echo Bay Mine, Lupin

Lupin Mine, operated by Echo Bay Mines, is located at Northwest Territories of Canada. The mill treats 2300 t/d of gold ore. Figure 4.9 shows the circuit configuration. The discharge of a single rod mill is fed to a secondary grinding circuit

consisting of two long ball mills (referred to as tube mills by mill metallurgists) operated in closed circuit with two cyclopak units. Each cyclopak is an assembly of four 38 cm (15") cyclones. The grinding media of ball mills is a mixture of 38, 50 and 64 mm (1.5, 2 and 2.5") balls.

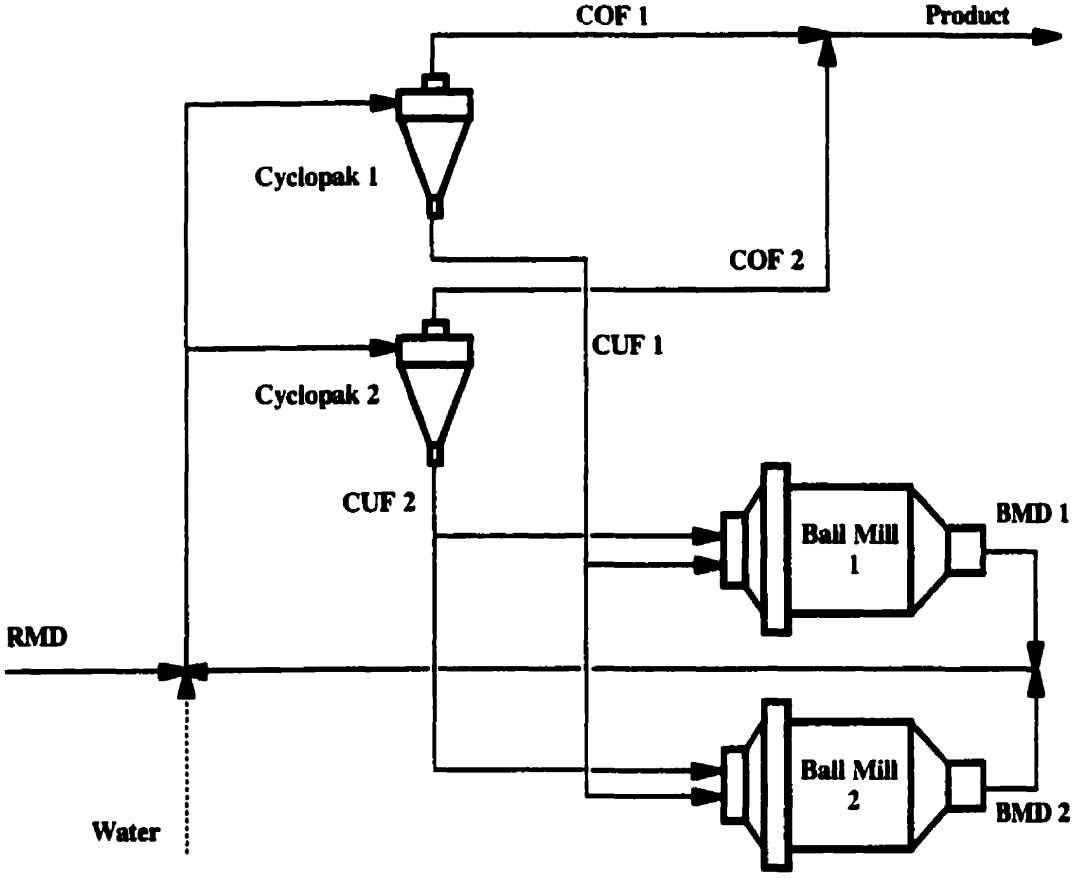


Figure 4.9 Simplified flowsheet of the Lupin grinding circuit and sampling points

Three sets of raw data were supplied to investigate the performance of grinding and classification operations. All size distribution data were first adjusted (Appendix B.6). The adjusted size distribution data were then used to estimate selection functions (Appendix B.7). Figure 4.10 shows selection function vs. particles size curves for the three data sets. The typical breakage function and RTD mean retention times were used to determine selection functions. The shape of all curves shows a maximum; grinding

kinetics falls off after this maximum and indicates non-optimal choice of grinding ball size. The estimated selection functions were used for simulations using BMCS which is explained in Chapter 5.

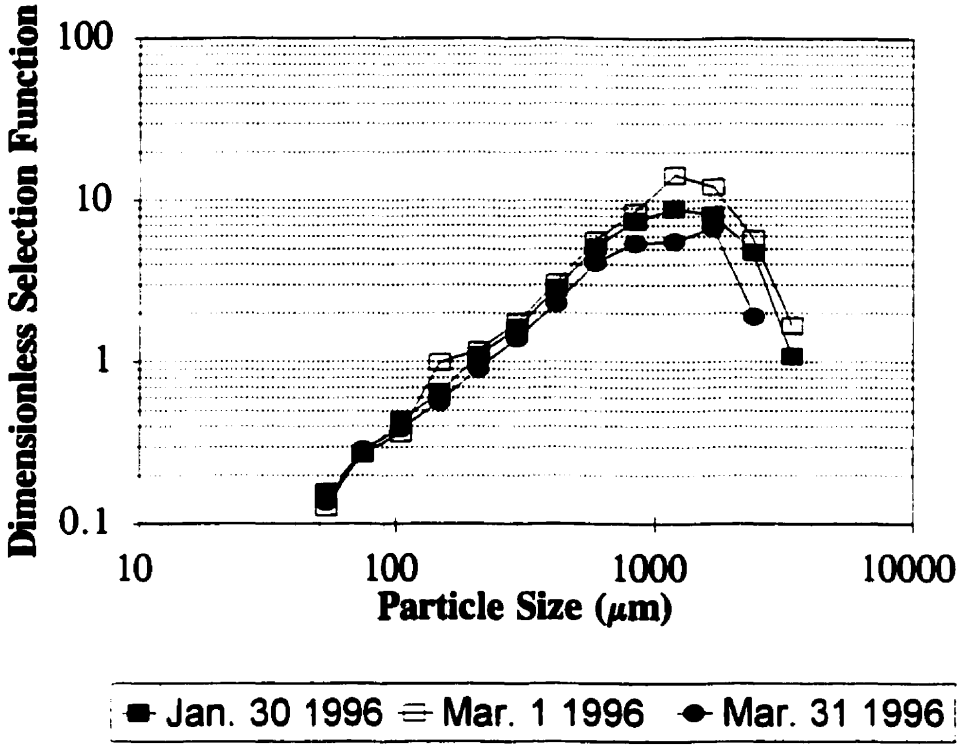


Figure 4.10 Breakage kinetics of Lupin ball mills

4.6.4 Louvicourt Mine

The Louvicourt mine is located at Val d'Or, Québec. Its mill processes approximately 4000 t/d of a complex sulphide ore containing chalcopyrite, sphalerite and pyrite [MacPhail, Racine and Cousins 1997]. The ore has an average grade of 3.8% copper, 1.75% zinc, 15% iron and 0.7 g/t silver. The grinding operation comprises a SAG mill closed with a vibrating screen (outside the scope of this study), a primary ball

mill closed with one 12-hydrocyclone cyclopak and two regrind ball mills, both closed circuit with one 8-hydrocyclone cyclopak. The regrind circuits are part of the downstream copper and zinc flotation circuits. A survey was performed around the primary ball mill and regrind circuits on February 25 1997.

The performance of primary ball mill circuit was evaluated by doing a number of steady-state simulations using the BMCS (in terms of grind fineness and grinding capacity). The NGOTC was used to back-calculate selection function parameter for circuit analysis and subsequent simulations.

For primary ball milling circuit, representative samples were taken from various streams around the ball mill and cyclopak by Louvicourt Mill personnel. Figure 4.11 shows the circuit configuration and sampling locations. The particle size distributions of samples from the vibrating screen underflow (undersize or fine product of the screen), ball mill discharge, cyclopak overflow and underflow streams were determined and provided by plant personnel. A series of screens (with a Tyler geometrical progression of $\sqrt{2}$) was used that adequately covers a particles size range from 25 μm to 12500 μm.

The raw particle size distribution data were adjusted using the NORBAL3 (Appendix B.8). A mill capacity of 4300 t/d was used to calculate flow rates. Adjustments to size fractions were minimal, as would be expected from a sound sampling campaign completed at steady state. The adjusted data were then used to study grinding kinetics.

The adjusted particle size distributions of CUF and BMD samples were used for estimating the selection function of the primary ball mill. Size-by-size grinding kinetics is represented by a selection function element (also known as specific breakage rate or more accurately breakage rate constant) which is back-calculated based on a mathematical model of grinding process. NGOTC was used to compute selection function elements

for all size classes (Appendix B.9).

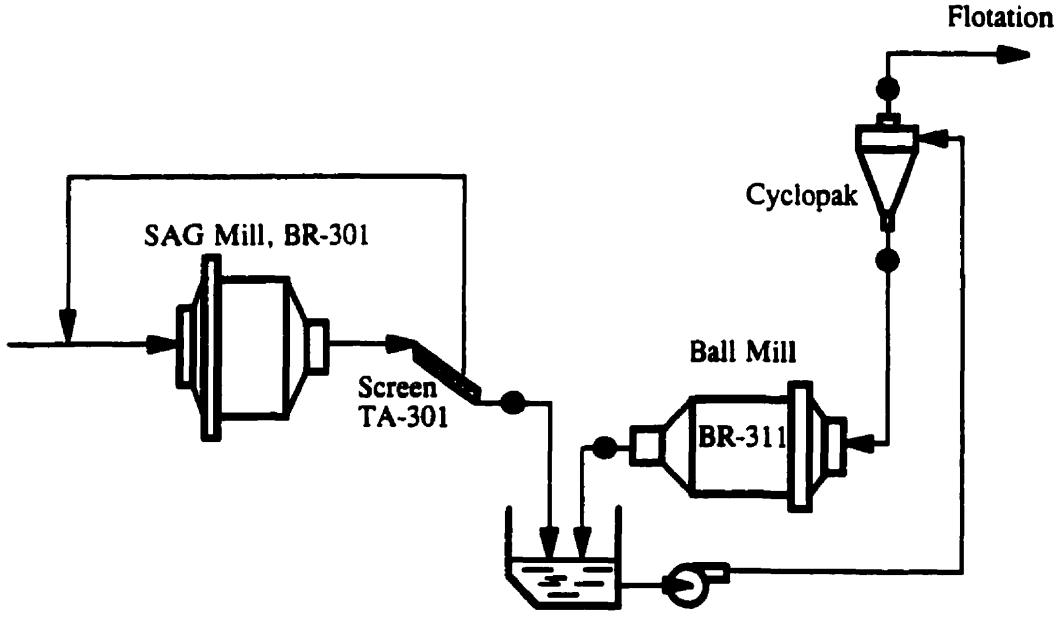


Figure 4.11 Simplified flowsheet of the Louvicourt ball mill/cyclopak grinding circuit and sampling points

Figure 4.12 shows the selection function vs. particle size curve, which confirms the existence of a moderate hump at the coarse end. This eliminates existing doubt about the selection function values of the first two size classes reported by Lacombe based on October 10, 1995. This result might rule out further decreases in ball size recommended in a previous report. However, if achieving a higher tonnage or finer product is sought, a smaller ball size can be tested. The use of a smaller ball size might increase grinding tonnage due to higher grinding kinetics for finer sizes taking into account that around 4.43% of the circulating load has a particle size greater than that of the hump (a very

low percentage). For this reason, the effect of changing ball size from 38 mm (1.5") to 25 mm (1") on the selection function was investigated using the ball size optimization module of NGOTC. The output of this module can be found in Appendix B.10.

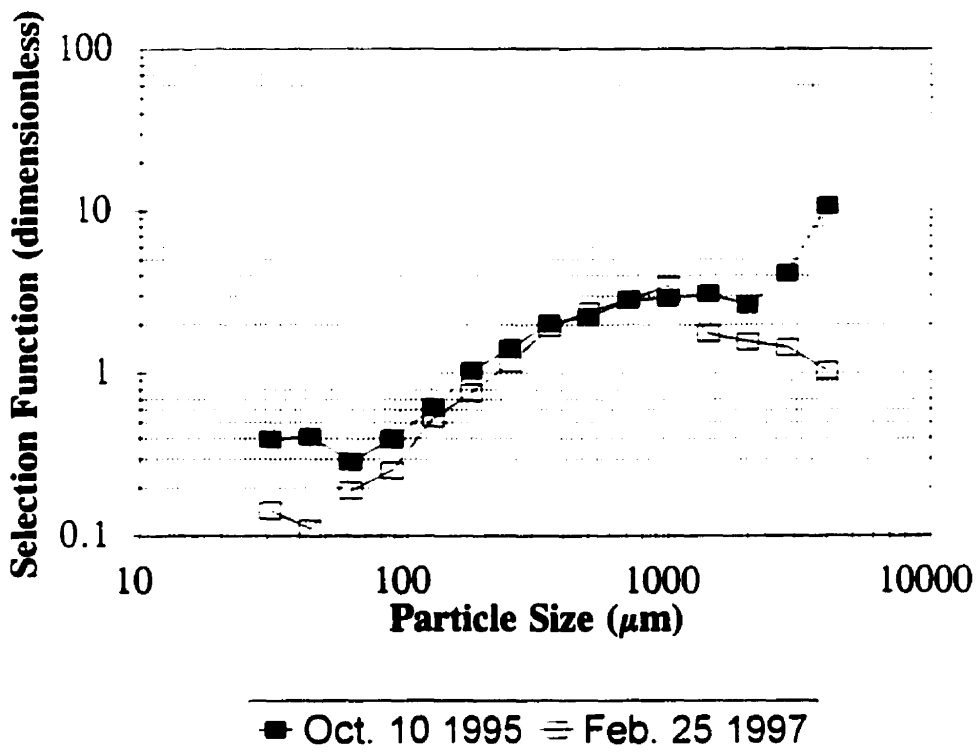


Figure 4.12 Breakage kinetics of the primary ball mill, Louvicourt plant

Figure 4.13 shows dimensionless estimated and scaled selection functions. The lines are the cubic spline fits. The curve fitted to the scaled selection functions (when using a hypothetical ball size of 25 mm (1")) shows that grinding kinetics will be decreased for particle sizes greater than 300 μm. However, grinding kinetics will be increased for particles smaller than 300 μm. A plant test is required to confirm whether the increased grinding kinetics for finer size classes is worth the decrease in breakage rate constant for the coarser size classes, after making the ball size change. Decreasing

screen opening size would minimize the risk of oversize overload in the mill and increase the benefit of smaller grinding balls.

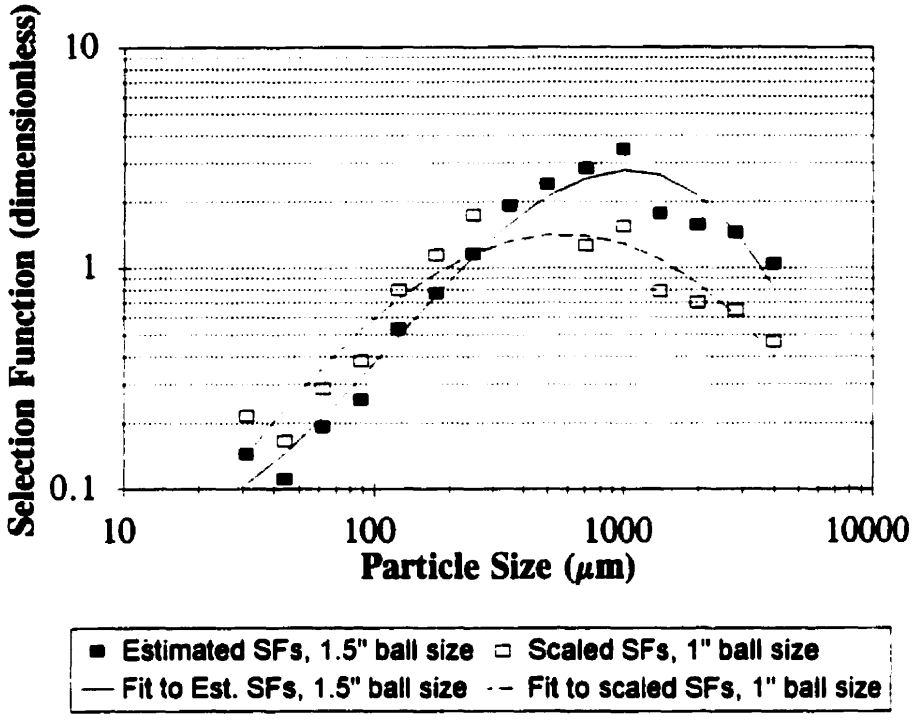


Figure 4.13 Estimated and scaled selection functions of the primary ball mill, Louvicourt plant

To evaluate the performance of the copper regrind circuit, slurry samples were taken from various streams on February 25 1997. The samples were analyzed to determine their particle size distributions using a Tyler series of sieves from 150 μm (100 mesh) down to 20 μm (635 mesh). A simplified flowsheet of the copper regrind circuit including sampling locations is shown in Figure 4.14. The product of the primary ball mill/cyclopak circuit is fed to the copper flotation section which includes a closed regrind circuit. A mixture of cleaner feed, cleaner scavenger concentrate constitutes the fresh feed to the copper regrind circuit. The combined tailings of rougher and cleaner flotation

cells will be treated downstream in the zinc flotation circuit.

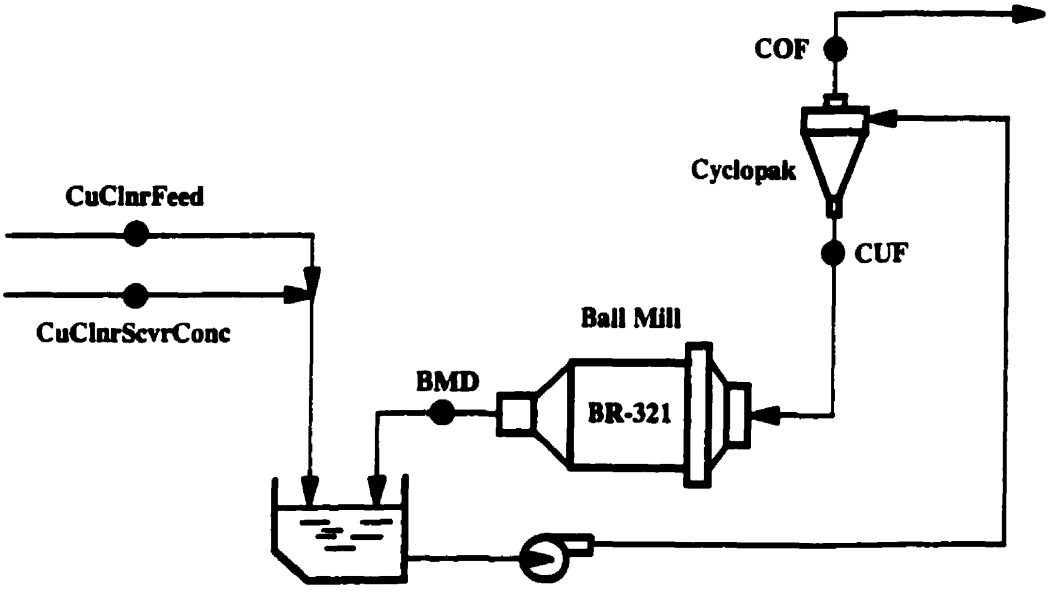


Figure 4.14 Simplified flowsheet of the Cu regrind circuit and sampling points, Louvicourt plant

The measured particle size distributions of slurry samples from cleaner scavenger concentrate, cleaner feed, ball mill discharge, cyclopak underflow and overflow streams were adjusted before further analysis (Appendix B.11). The balanced size distributions of cyclopak underflow and ball mill (BR-321) discharge were used as input into NGOTC to back-calculate the selection function (Appendix B.12). Figure 4.15 shows the selection function vs. particle size curve. The geometric mean of the lower and upper limits of each size class was used as characteristic particle size. The curve indicates a decrease in grinding kinetics for the first size class. The amount of mass in this size class is too low (2.15%) to justify using larger grinding media, and even suggests that smaller balls or slugs could be used to produce a finer mill discharge.

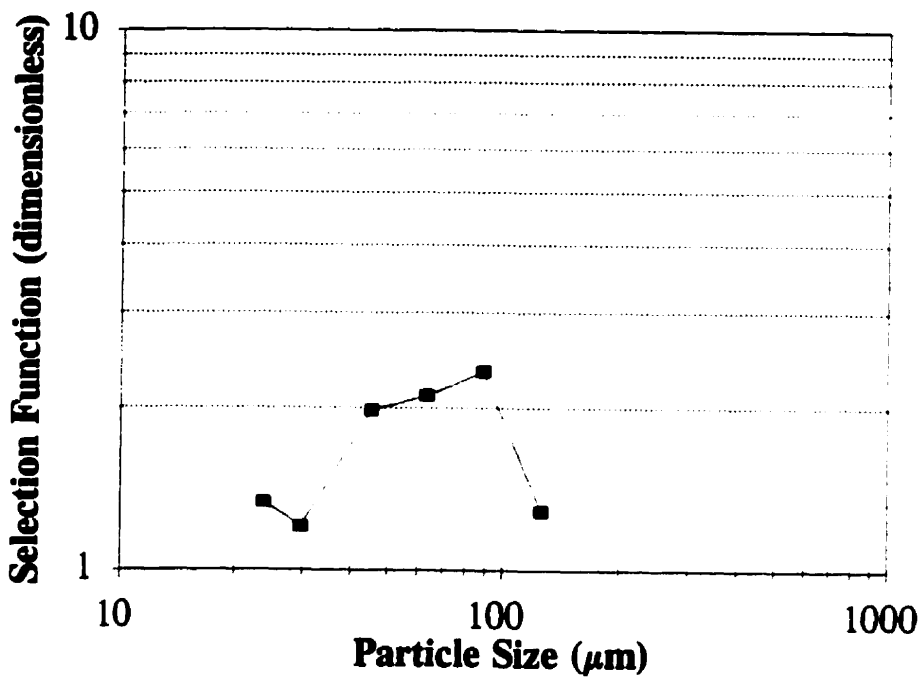


Figure 4.15 Breakage kinetics of the Cu regrind ball mill, Louvicourt plant

For zinc regrind circuit, samples taken on February 25 1997 were analyzed to determine their size distributions using a Tyler series of sieves from 150 μm (100 mesh) down to 20 μm (635 mesh). Norbal3 [Spring 1992] was then used to adjust these data and generate an estimate of the circulating load (Appendix B.13). Figure 4.16 shows the circuit flowsheet. The zinc regrind circuit is a part of zinc flotation circuit which treats the combined tailings of rougher and cleaner copper flotation. A mixture of rougher, scavenger and cleaner concentrates makes the fresh feed of the regrind circuit.

The circulating load was estimated at 285%, which is significantly higher than that of reported for previous sampling campaigns [Lafontaine 1996]. The balanced size distribution data of the cyclopak underflow and ball mill discharge were used to estimate

size-by-size selection function values (Appendix B.14). Figure 4.17 shows the selection function vs. particle size curve and clearly indicates a decrease in grinding kinetics for particle sizes larger than 60 μm. The size distribution of cyclopak underflow (regrind ball mill feed) indicates that only 3.41% of mass reports to the first two size classes. This implies no need to use larger grinding media size, and suggests that a smaller grinding medium could be used to grind finer.

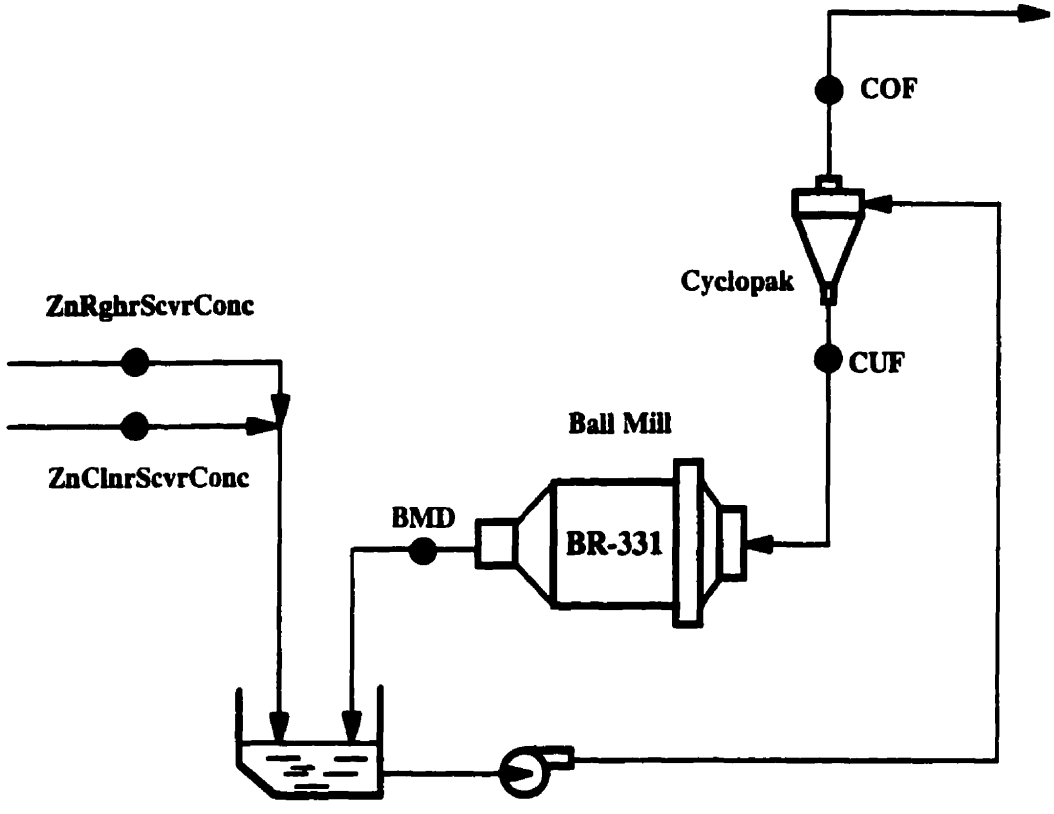


Figure 4.16 Simplified flowsheet of the Zn regrind circuit and sampling points, Louvicourt plant

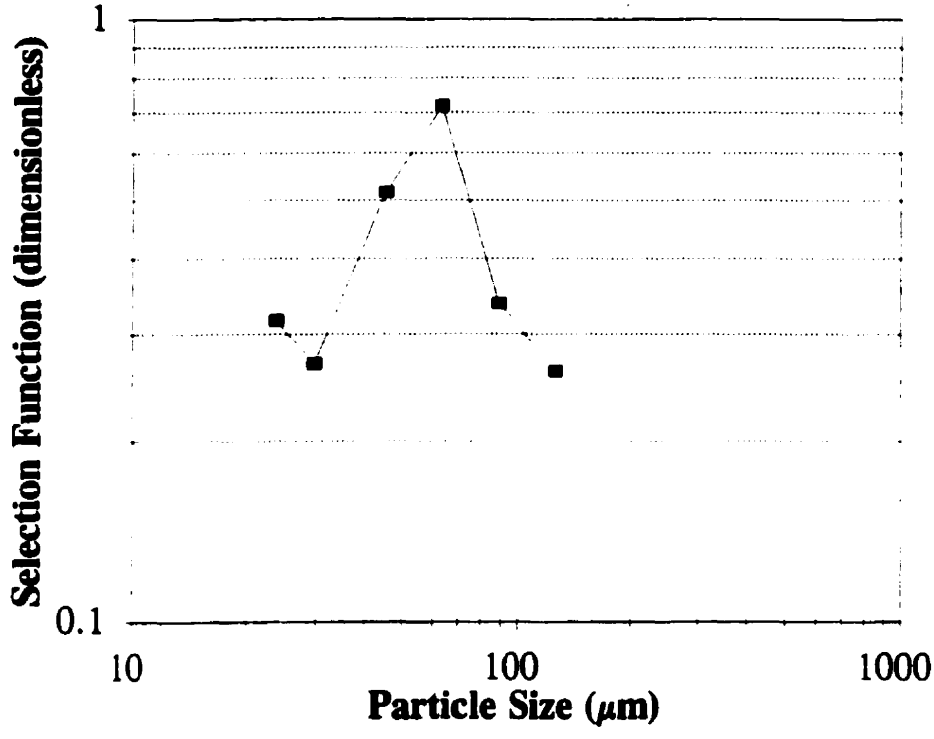


Figure 4.17 Breakage kinetics of the Zn regrind ball mill, Louvicourt plant

4.7 Reliability of Back-Calculated Selection Functions

The reliability of an estimated mill selection function depends on the back-calculation method itself and the accuracy of particle size distribution data. Since the interval-by-interval search algorithm finds selection function elements sequentially by starting from the top-size class, the error then will be propagated to finer size classes. In search methods which use functional forms of the selection function and minimize a defined objective function to find the best selection function elements, the error however will be distributed over selection function elements.

The magnitude of error associated with each selection function element can be

calculated if there are more than one data set available. The most significant errors are attributed to the plant sampling. To minimize sampling errors, it is important that the circuit be near to steady state and samples as representative as possible. While screening to determine particle size distributions of various samples is a reproducible process, errors can be easily introduced if sieves are defective (apertures are blocked or the mesh ruptured). Performing wet screening before dry screening can significantly reduce the error associated with very fine material. Because the spline curve fitting algorithm of NGOTC uses the standard deviation of each selection function element as a weighting factor, it provides a tool for more accurate estimation of the selection function.

4.8 Summary

NGOTC was used to estimate selection functions of several industrial mills. The first benefit of estimating these selection functions was their use as a diagnostic tool for mill performance analysis. Reliability of the program as a process analysis and modelling tool was demonstrated. The diverse applications of NGOTC made it possible to verify the credibility of its computations and predictions. The selection functions obtained by using estimation or scaling modules can be used confidently for modelling and simulation purposes.

The procedure to scale a mill selection function has been verified in a number of cases by Morrell [1990]. The incorporation of this procedure in NGOTC and its integration with other modules provided a tool for ball size optimization studies.

CHAPTER 5
BALL MILLING CIRCUITS SIMULATOR (BMCS)

5.1 Introduction

In this chapter details of *B*all *M*illing *C*ircuits *S*imulator (BMCS) will be presented. This program allows one to simulate ball milling circuits consisting of ball mill and hydrocyclone units. Previous work at McGill University [del Villar and Laplante 1985] includes a series of programs written in Applesoft BASIC such as RTDOPEN and RTDCLOSE to estimate parameters of *R*esidence *T*ime *D*istribution (RTD) models, PSD to simulate ball mills, SELFUNC to estimate selection functions and GRINDSIM to simulate closed grinding circuits. In this chapter various aspects of developing and testing of BMCS, which is partly based on GRINDSIM, but written in the more efficient C language, will be discussed.

Steady-state simulation is increasingly used by mineral processors to design, analyze and optimize various grinding circuits [Austin, Luckie and Wightman 1975; Herbst and Fuerstenau 1980; Hodouin, McMullen and Everell 1980; Finch and Ramirez-Castro 1981; del Villar et al. 1985]. For example, grinding simulators have been used to evaluate process performance under various circuit designs, series vs. parallel grinding [del Villar et al. 1985] or two stage vs. single stage classification [Dahlstrom and Kam 1988], and operating conditions, slurry densities [Laplante and Redstone 1984], ball size [Hartley et al. 1983] or to find the optimum steady-state parameters which meet a required metallurgical performance. The current version of BMCS can be readily used to simulate steady-state circuit performance, given the flow rate, % solids and particle size distribution of new (fresh) feed and the flow rate of water additions to the circuit.

additions to the circuit. However, finding the optimum operating conditions to meet a desired circuit performance requires manual search by the user.

5.2 Program Structure

The modular structure of BMCS involves several executable sub-programs that are run and accessed by the main program "BMCS.EXE" during a simulation trial. This simulation structure provides a high level of flexibility and economy of programming [Austin, Klimpel and Luckie 1984] and needs a minimum of computer memory requirements. In addition, very complex circuits, which include multi-stage grinding and classification systems, can be simulated.

Figure 5.1 shows the various steps of a simulation run. A pre-defined circuit is first selected for simulation. Then the program reads the basic data, i.e. a connectivity matrix and the number of nodes and streams in the circuit nodal representation and starts to compute sequentially the output of each node by calling the related sub-program or module. The sub-programs contain general mathematical models of physical process objects which are components of typical ball milling circuits (ball mills, hydrocyclones) and functions which simulate junction, split and convergence blocks. Simulation data files must be created by the user to provide the required information for simulator calibration and execution.

5.3 Mathematical Models

The current version of BMCS contains two modules for the simulation of ball mill and hydrocyclone units. The implemented model structures are inflexible, i.e. source code changes are required to modify them. The other modules are JUNCTION, SPLIT, CONVERGE, FIXCLASS which are necessary to construct simulation models of closed grinding circuits. All modules are explained in the next sections.

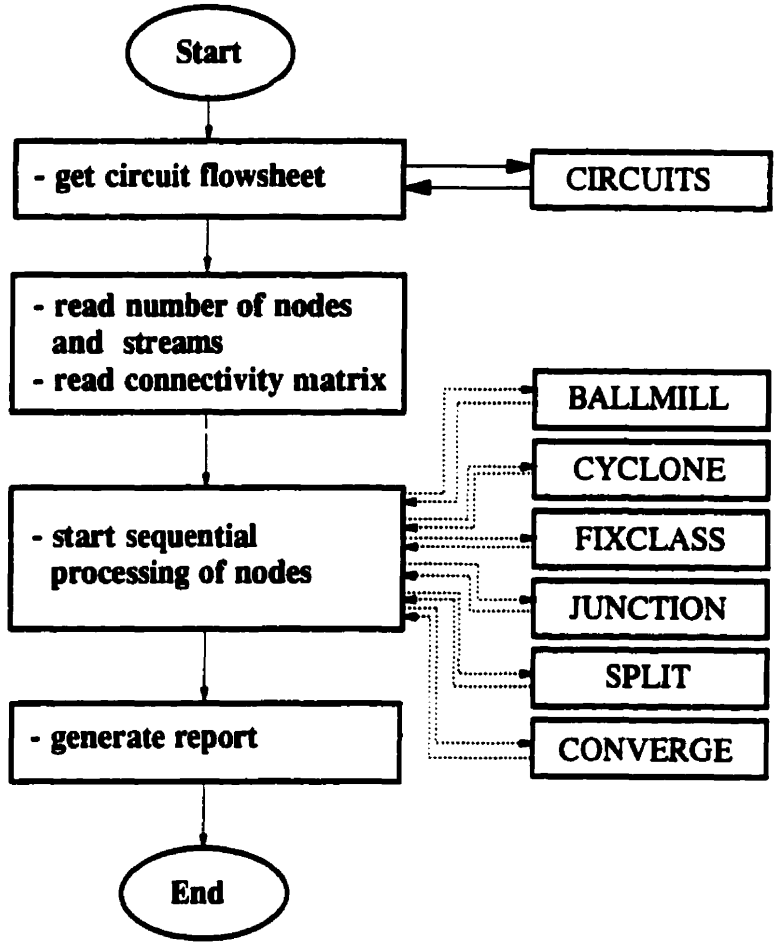


Figure 5.1 Simplified flowchart of a BMCS simulation run

5.3.1 Ball Mill

The classical time-continuous, size-discrete population balance model of tumbling ball mills was implemented in C using BC++ 3.1 compiler. A number of researchers contributed to the introduction of this model [Epstein 1947 and 1948, Sedlatschek and Bass 1953, Broadbent and Callcott 1956] which was later evolved and validated by Gaudin and Meloy, 1962. The model predicts the overall size distribution of ore as a single-component solids phase. Multi-component models have been also developed [Finch and Ramirez-Castro 1981, Weller et al. 1988] which can predict the performance of individual components given size-by-size assay information, but this was deemed unnecessary here because of the focus of the investigation.

The ball mill model requires values for its breakage function, selection function and residence time distribution. The breakage function should be that used to generate the selection function vector with NGOTC. The selection function parameters are estimated using NGOTC from data originating from the circuit to be simulated, operating at steady state and as close as possible to the conditions to be simulated. The residence time distribution model used is that of Weller [1980]. Since the classical ball mill model is generally not numerically sensitive to the RTD parameters and breakage function, it was decided that improvements on the parametric estimation approach of del Villar [1985] was unnecessary.

The source code developed by the author was tested using real plant data with more focus on the verification of computations. The ball mill executable program is called by the main module of BMCS software and cannot be run separately.

5.3.2 Hydrocyclone

The original version of Plitt's hydrocyclone model [1976] was implemented in C to simulate hydrocyclones. The model can simulate the operation of a hydrocyclone under the normal discharge regime. Extensive use of Plitt's model has shown its

capability to predict hydrocyclone performances to a first approximation [Napier-Munn et al. 1996] with no need of plant constants. The accuracy of predictions can be augmented by applying constant correction factors to original equations for the corrected cut size, d_{50c}, the separation sharpness, m, the pressure drop, P, and the volume flow split, S.

5.3.3 Fixed Classification

In this module classification is simply defined by pre-set mass split coefficients for each size into the fine and coarse streams, rather than using a mathematical unit model to derive those coefficients. The classification coefficients for each size class, c_i, can be estimated from plant tests and must be supplied as input by the user. This module could be used for an approximate representation of screen units. The program calculates size distributions of oversize and undersize streams based on the feed size distribution and the classification coefficients.

5.3.4 Junction

In a grinding circuit, there are points where two or more streams mix together and form a single stream. These junctions or mixing points could be either real or conceptual. For example, in closed grinding circuits a ball mill unit may be fed with a fresh feed and circulating load. In simulation, these two streams should be combined to produce a single ball mill feed stream, although in reality the two streams could be added separately to the ball mill and combine only in the mill itself. The junction module calculates the solids flow rate, density and particle size distribution of the combined stream using those of each input stream.

5.3.5 Split

At a split point, a single stream is simply split indiscriminately into two or more streams. In a closed ball mill/hydrocyclone grinding circuit, for example, part of the hydrocyclone underflow stream might be bled for gravity separation and the rest returned

to a ball mill for further grinding.

To simulate a split point or node, it is assumed that there is no change in properties of the stream, namely size distribution and % solids of the stream. However, the flow rate of each output stream is calculated as a percent of input stream which must be defined by the user. The computations made by the split module were also checked and verified.

5.3.6 Convergence

The presence of recycle streams in a grinding circuit makes its simulation more complicated due to the iterative calculations required to calculate the flow rate, % solids and particle size distribution of recycle streams. Conceptually, each recycle stream maps to a convergence block in the nodal representation of the circuit being simulated. A convergence block is in fact a pseudo-module [Richardson, Coles and White 1981] to solve recycle sets.

In this module, the successive substitution approach was implemented to test the convergence of a recycle stream. The particle size distribution and solids flow rate of a recycle stream are set to their initial guess value, zero mass in each size class at the first iteration. The computed values of one iteration are used as initial guesses for the next iteration. After each computation cycle, the solids flow rate of the recycle stream computed in the current and previous iterations are tested for convergence (the difference between the two values must be within a tolerance range set by the user). Another approach developed by Wegstein [1958] remarkably accelerates convergence iterations. However, because of the relative simplicity of the convergence problem in grinding circuits it was decided to use the simpler successive substitution approach.

5.4 Simulation of Closed Ball Milling Circuits

The simulation of a closed ball milling circuit is possible by mathematical models

of participating process units and their linking by an executive or main simulation structure. Since a grinding circuit is indeed a system constructed of interacting sub-systems or units such as ball mills and hydrocyclones, performing circuit simulations is required to optimize operation correctly. Therefore, the final impact of changing a sub-system design or operating parameter (e.g. grinding balls size, fresh feed rate, hydrocyclone apex or vortex diameters) on a full circuit or system can be investigated by such simulations.

Westerberg et al. [1979] have discussed various approaches of process flowsheeting systems. Basically, either the sequential-modular or equation-solving approach is used to construct process simulators or flowsheeting systems. In equation-solving simulators the full circuit is defined as a system of equations that are solved simultaneously. In sequential-modular simulators, computations are done sequentially according to a calculation path (or computational order) based on nodal representation of the circuit.

The sequential-modular approach was chosen to the develop simulation structure due to its ease of implementation and low complexity. In contrast, the equation-solving approach can be extremely complex and requires more computer resources.

5.4.1 Simulation Input Data

Practically, a user must collect a large amount of information to be able to simulate a real grinding circuit. This involves gathering information concerning circuit flowsheet, selection and breakage functions to customize a ball mill model to each ball mill unit and geometrical specification and adjusting factors to customize Plitt's model to each hydrocyclone unit.

In BMCS, circuit flowsheets are described by connectivity matrices which must be placed into a text file named "CONMAT.LIB". The connectivity matrix of a circuit

in fact determines the calculation path for the simulator and must be defined by the user (a number of circuit flowsheets have been already pre-defined and included in "CONMAT.LIB"). The other information must be given in an input text file prepared according to a certain format and named with the correct extension.

5.4.2 Simulation Output

The purpose of using the simulator is to calculate the steady-state particle size distribution, solids flow rate and % solids of all streams under certain operating conditions described by the flow rate of solids fed to the circuit, % solids and water addition flow rates. While simulations are performed in a forward direction, from a pre-defined set of input parameters to a set of output parameters considered as performance space, various simulation trials can be done to find a set of input parameters which produces a desired circuit performance such as fineness of grind or production capacity.

5.5 Simulation of Lupin Mine Grinding Circuit

The ball milling circuit of the Lupin Mine, operated by Echo Bay Mines, consists of two parallel tube ball mills (2.4 m x 7.3 m (8′ x 24′)) fed by the underflow streams of two cyclopaks which are also operated in parallel (Fig. 4.9). The circuit was simulated under the same operating conditions of the sampling campaign of March 31 1996 (solids feed rate 95 t/h) so that a comparison of predicted and measured size distribution could be made. The details of circuit modelling and simulation are explained below.

5.5.1 Modelling Ball Mills

The dimensionless (normalized) selection function was estimated from data of sampling campaign of March 1 1996. using the same normalizable ore breakage function and dimensionless RTD parameters used to back-calculate selection function. The breakage function and RTD parameters were assumed rather than estimated directly from laboratory and plant tests.

5.5.2 Modelling Classification

The accuracy of predictions made by the cyclone module was tested by using data sets from three full grinding circuit surveys performed on January 30 1996, March 1 1996 and March 31 1996. These data had been used perviously to estimate the mill selection functions (see Section 4.6.3). The main purpose of testing was to verify computations performed by the cyclone module and to demonstrate whether or not calibrating Plitt's model could generate more accurate predictions.

The data set obtained from the March 1 1996 grinding survey was selected to calibrate Plitt's model. Figure 5.2 shows predicted and measured particle size distributions of cyclopaks overflow and underflow streams. Agreement between predicted and measured size distributions was found to be reasonable for the underflow stream over the full particle size range, but the overflow distributions clearly differ significantly. A similar problem was found with the January 30 and March 31 1996 data sets (Appendix C.1). Calibration factors were calculated using following relationships:

$$CF_{d_{50c}} = \frac{d_{50c,\ measured}}{d_{50c,\ prediction}} \tag{5.1}$$

$$CF_{m} = \frac{m_{measured}}{m_{prediction}} \tag{5.2}$$

$$CF_{R_f} = \frac{R_{f,\ measured}}{R_{f,\ prediction}} \tag{5.3}$$

where CF_{d50c}, CF_m and CF_{Rf} are calibration factors for the corrected cut size, separation sharpness and recovery of fluid to hydrocyclone underflow, respectively. The factors were then used to adjust equations 2.16, 2.18 and 2.23.

BMCS was then used to simulate cyclopaks performance under two different operating conditions. The simulation results are given in Appendix C.2. The flow rate,

% solids and size distribution of the cyclopak feed were calculated using adjusted cyclopak overflow and underflow data. Figures 5.3 shows predicted and measured size distributions of cyclopaks overflow and underflow streams for the January 30 1996 survey. Plitt's model predictions are remarkably improved. A similar improvement was also observed for the other test data set, March 31 1996, as well as the data set used to estimate the calibration factors, March 1 1996 (Appendix C.3). These simulations also verified the predictions made by the cyclone module of BMCS simulator and the importance of cyclone model calibration to specific applications before simulation.

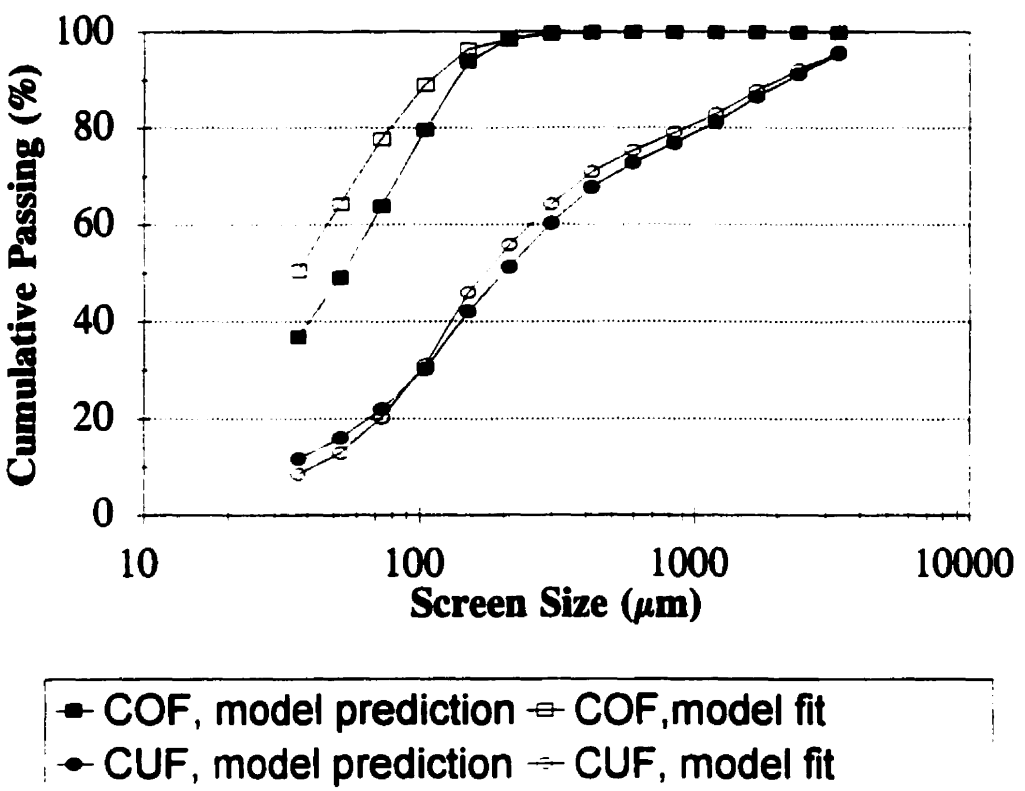


Figure 5.2 Predicted and measured size distributions of cyclopaks overflow and underflow streams, Lupin Mine (March 1 1996 survey)

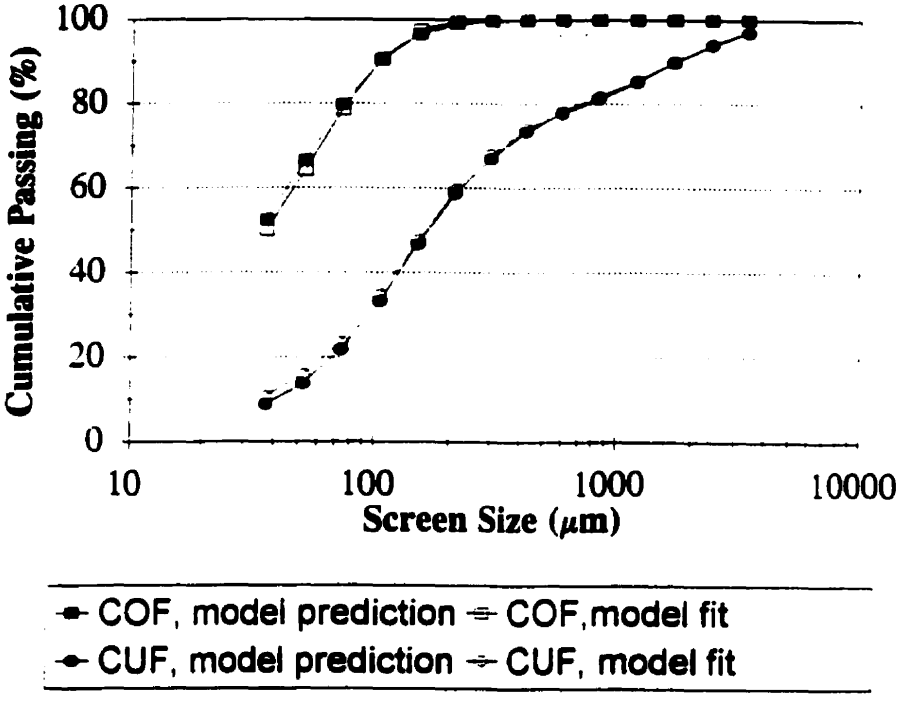


Figure 5.3 Comparison of predicted and measured size distributions of cyclopaks overflow and underflow streams after Plitt's model calibration, Lupin Mine (Jan. 30 1996 survey)

Table 5.1 Comparison of predicted and measured Plitt's model parameters

	January 30, 1996			March 31, 1996		
	Meas.	Pred.*	Pred.†	Meas.	Pred.*	Pred.†
$d_{50c}(\mu m)$	51.54	118	58	56.45	112	55
m	1.24	2.21	1.37	1.38	2.22	1.37
R_f	0.23	0.34	0.19	0.23	0.32	0.18

* and † designate predictions before and after model calibration, respectively.

Table 5.1 compares Plitt's model parameters obtained from measured data and simulations.

5.5.3 Circuit Simulation

After calibrating the ball mill and hydrocyclone models, circuit simulations were performed to test the simulator accuracy in predicting the size distributions of various streams.

The size distributions of various streams predicted by the simulator using the same feed size distribution and flow rate of March 31 1996 data set are compared to the measured size distributions in Figure 5.4. Table 5.2 gives the predicted and measured flow rates and % solids of various streams. It is concluded that the circuit performance can be simulated with an error normally lower than 10%. There are many sources that can contribute to the discrepancy between the simulated and measured performances such as inherent difficulties involved in the calibration of ball mill and hydrocyclone models due to sampling errors and estimating model parameters. Differential mineral behaviour also contributes to the lack of it, but to a lesser extent than for massive sulphide ores.

5.6 Simulation of Louvicourt Mine Grinding Circuit

BMCS was used to simulate the performance of the ball mill cyclopak circuit of the Louvicourt plant under various ball sizes and hydrocyclones apex diameters, to investigate the effect of these parameters on the circuit energy efficiency and product fineness. A detailed study of classification had to be undertaken in order to interpret correctly subsequent simulation results. The study included an investigation of the individual mineral responses to classification.

5.6.1 Modelling the Ball Mill

In this case, only one ball mill unit, 5.0 m x 7.3 m (16.5' x 24'), had to be modelled. The mill selection function was estimated based on February 25 1997 survey

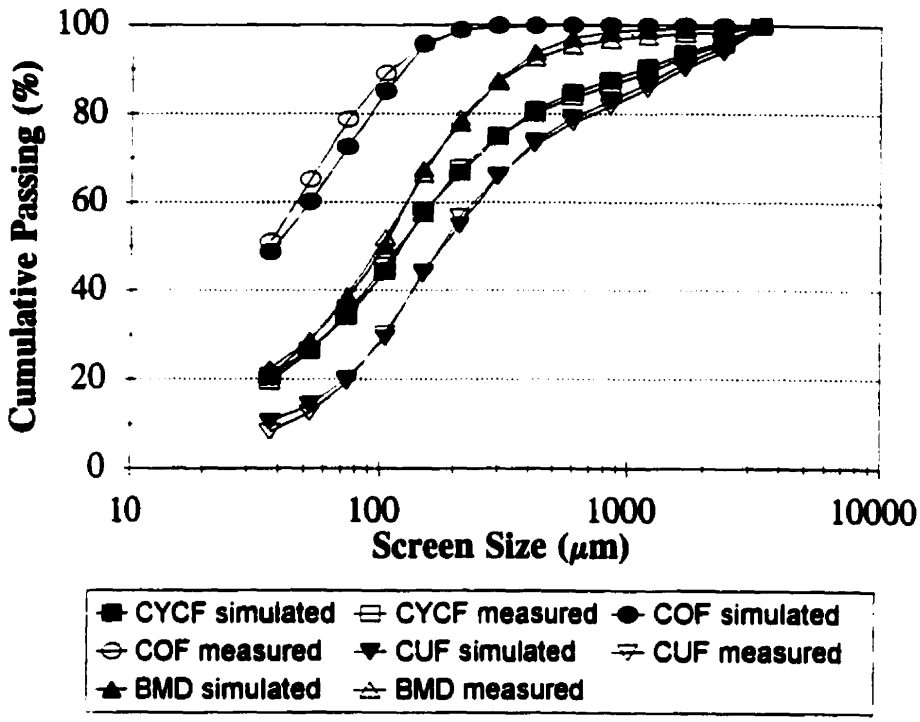


Figure 5.4 Comparison of predicted and measured size distributions of various streams, Lupin Mine (March 31 1996 survey)

Table 5.2 Comparison of predicted and measured flow rates and % solids, Lupin Mine

	CF		COF		CUF	
	Pred.	Meas.	Pred.	Meas.	Pred.	Meas.
flow rate (t/h)	355	363	95	95	260	267
solids (%)	57.1	53.1	30.5	27.7	83.9	78.7

and was smoothed with a cubic spline function to calibrate the simulator when 38 mm (1.5") grinding balls are used. The same ore breakage function and RTD parameters assumed to estimate the selection function. To predict the circuit performance when the ball mill is charged with 25 mm (1") grinding balls, the simulator was calibrated using the smoothed, scaled selection function (see Fig. 4.13).

5.6.2 Modelling Classification

A cyclopak of twelve 38 cm (15") hydrocyclones is operated in closed circuit with the ball mill unit (BR-301). Previous work indicated that there is a plateau in the classification performance curve [Lacombe 1995, Farzanegan and Laplante 1997a]. This was attributed to the multi-component nature of the ore and studied in detail [Farzanegan and Laplante 1997b]. To assess the classification performance of the major minerals, chalcopyrite, sphalerite, pyrite and *non-sulphides gangue* (NSG), individual size classes were analyzed for metal content, using samples extracted on February 25 1997. To estimate the size-by-size recovery of individual minerals to the cyclopak underflow, the balanced size distribution and solids flow rate of the overflow and underflow samples were used. The size distribution of the cyclopak feed was calculated from the size distribution of the overflow and underflow streams. The raw data were adjusted using NORBAL3 [Spring 1992] and were given in Appendix B.8.

The rough percentages of minerals that constitute the Louvicourt ore are given in Table 5.3. Due to the low amount of pyrrhotite, magnetite, ilmenite and galena, they were ignored when size-by-size mineral contents were calculated. The actual amount of each mineral fed to the circuit at sampling time was calculated based on the metal content of each mineral (Table 5.3).

The content of chalcopyrite, sphalerite and pyrite was calculated from the metal assays using the following relationships (assuming that NSG minerals contain no iron):

Table 5.3 Mineralogical composition of Louvicourt ore

Mineral	Formula	Specific Gravity	Metal Content (%)	Percent
Chalcopyrite	$CuFeS_2$	4.1-4.3	Cu-34.6%	9-13
Sphalerite	ZnS	3.9-4.1	Zn-60.6%, Fe-6.5%	3-6
Pyrite	FeS_2	5.0	Fe-46.7%	22-55
Pyrrhotite	Fe_xS_y	4.6	Fe-61.5%*	1-3
Magnetite and ilmenite	$FeOFe_2O_3$ $FeTiO_3$	5.2,4.5-5.0	Fe-72.4% Ti-31.6%	1-2
Galena	PbS	7.4-7.6	Pb-86.6%	<1
Silica and carbonate	SiO_2 $CaCo_3$	2.65-2.66,2.7	-	20-64

*variable

$$m_{i,chalcopyrite} = x_{i,Cu}/0.346$$
$$m_{i,sphalerite} = x_{i,Zn}/0.606$$
$$m_{i,pyrite} = (x_{i,Fe}-0.3*m_{i,chalcopyrite}-0.065*m_{i,sphalerite})/0.467$$

where m_i and x_i are the content of the specified mineral and metal in size class i, respectively. The content of NSG minerals, then, was calculated by:

$$m_{i,NSG} = 100-(m_{i,chalcopyrite}+m_{i,sphalerite}+m_{i,pyrite})$$

The partition curves of the various minerals were then fitted to Plitt's model [1976], and are shown in Figure 5.5. The goodness of fit for sphalerite and pyrite was much better than that of other minerals. The low goodness of fit for chalcopyrite and NSG minerals was linked to the errors of the metal assays for the pan fraction. More specifically, the 4.15% Cu of the cyclopak underflow -25 μm fraction is significantly above that of slightly coarser size classes. If indeed inaccurate, this analysis would have caused the high recovery to the cyclopak underflow for chalcopyrite and low recovery for NSG (which is what was observed).

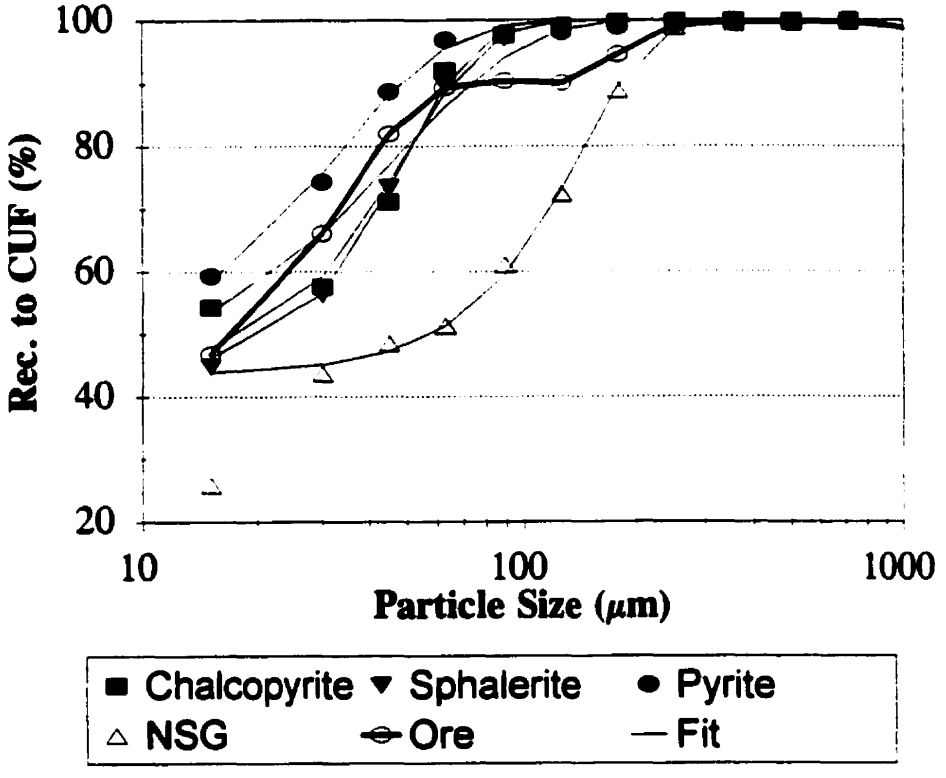


Figure 5.5 Individual classification performance curves of the primary cyclopak, Louvicourt plant

R_f was calculated from % solids data and then was used to fit Plitt's model to the measured individual partition data points. The best fit was found using the non-linear

optimization tool of Quattro Pro* software with a search for two unknown parameters, namely m and d_{50c}. Tables 5.4 and 5.5 show cyclopak operating conditions and Plitt's model parameters estimated for individual components and ore. The separation sharpnesses of chalcopyrite and sphalerite, equal to 2.13 and 2.51, respectively, are higher than that of ore, 1.38.

The individual mineral-by-mineral performance curves explain the plateau in the overall curve for the ore. Chalcopyrite and sphalerite show almost the same classification behaviour due to their very similar specific gravities. Pyrite and NSG minerals show different classification behaviours due to their different specific gravities. For a more detailed discussion, the reader is referred to Laplante and Finch 1984.

These curves confirm that the plateau is due to mineral density effects and that classification sharpness is much better than that of the overall ore would suggests, and imply that sphalerite and chalcopyrite are well liberated from pyrite and NSG, as the performance curves are clearly different. The classification sharpness of pyrite is poor, 1.49, but could well be due to the presence of iron in NSG minerals.

The individual mineral partition curves confirm that the classification performance of primary cyclopak is highly affected by the multi-component nature of the ore. This implies that Plitt's model cannot be used accurately to fit or simulate ore classification data. Even though Plitt's model cannot model the classification of overall ore effectively, the sampling and modelling exercise can still yield informative data.

The circulating load is high, 362%, clearly above the "ideal" of 250%, % solids of the cyclopak underflow is low, 74%, below the ideal grinding % solids of high density ores, 75-79% (possibly more, Laplante and Redstone 1984) and short circuiting to the

*Quattro Pro is a trademark of Borland International

Table 5.4 Primary cyclopak operating conditions, Louvicourt plant (Feb. 25 1997 survey)

	Solids flow rate (t/h)	Solids content (%)	Water flow rate (t/h)
COF	176	38.2	286
CUF	639	74.3	222
CF	815	61.7	508

Table 5.5 Plitt's model parameters estimated for individual minerals and ore, Louvicourt plant (Feb. 25 1997 survey)

	Chalcopyrite	Sphalerite	Pyrite	NSG	Ore
R_f	0.44	0.44	0.44	0.44	0.44
m	2.13	2.51	1.49	2.34	1.38
d_{50c} (μm)	43	44	26	123	38
Lack of fit	70.07	6.81	11.71	336.81	197.98

CUF is high, 44%: all these problems can be corrected by decreasing apex diameters.

5.6.3 Primary Circuit Simulations

To test the recommendation of a smaller ball size, 25 mm (1"), and a smaller apex diameter, 57 mm (2.25"), BMCS was used to simulate the circuit performance under the proposed modifications. Although simulations were performed under some uncertainties regarding model calibration (i.e. the unusual cyclone performance curve), the predicted trends can still guide further optimization decisions.

To build a simulation model of the circuit, unit models were calibrated using data from the October 10 1995 survey. The calibration factors of Plitt's model were calculated using relationships 5.1, 5.2 and 5.3. It must be emphasized that the use of Plitt's equations to model classification performance is subject to interpretation due to the existing plateau in the uncorrected partition curve. Since Plitt's model assumes that the hydrocyclone feed consists of a single solids phase, it cannot model such plateaus precisely. For the purpose of this research it was decided not to pursue individual mineral treatment, which would have complicated significantly the simulator's source code.

After parameters of simulation models were estimated, the simulator was tested by comparing its predictions with the measured data obtained from the survey performed on February 25 1997. The outputs of the simulator for the October 10 1995 survey (data set used for estimating the selection function parameter) and February 25 1997 survey (data set used for testing the simulation model) are given in Appendix C.4. Figure 5.6 shows predicted and measured size distributions of cyclopak overflow stream for the February 25 1997 survey. The predicted mass of material passing a specified size is lower than the corresponding values measured. Nevertheless, the simulator was used to investigate trends rather than produce absolute estimates. The inability of simulator to generate more accurate predictions stems from a number of sources particularly its failure

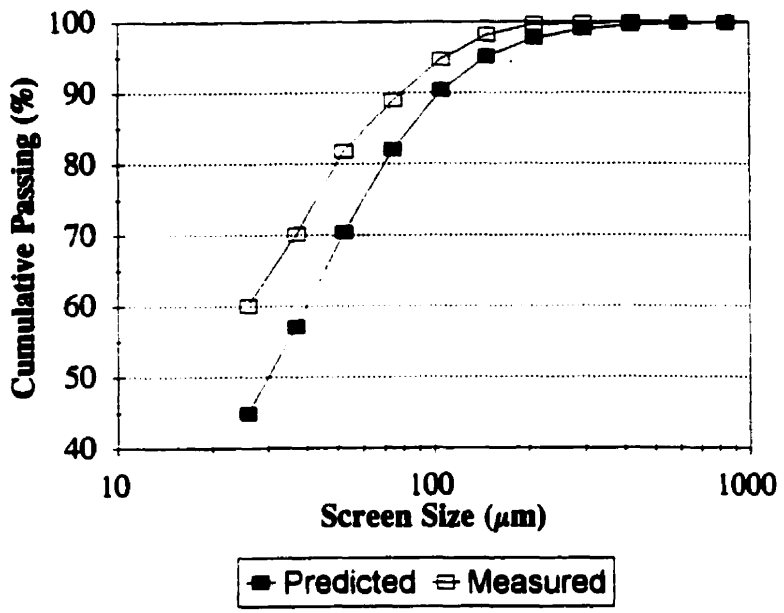


Figure 5.6 Comparison of predicted and measured size distributions of the primary cyclopak overflow stream, Louvicourt plant (Feb. 25 1997 survey)

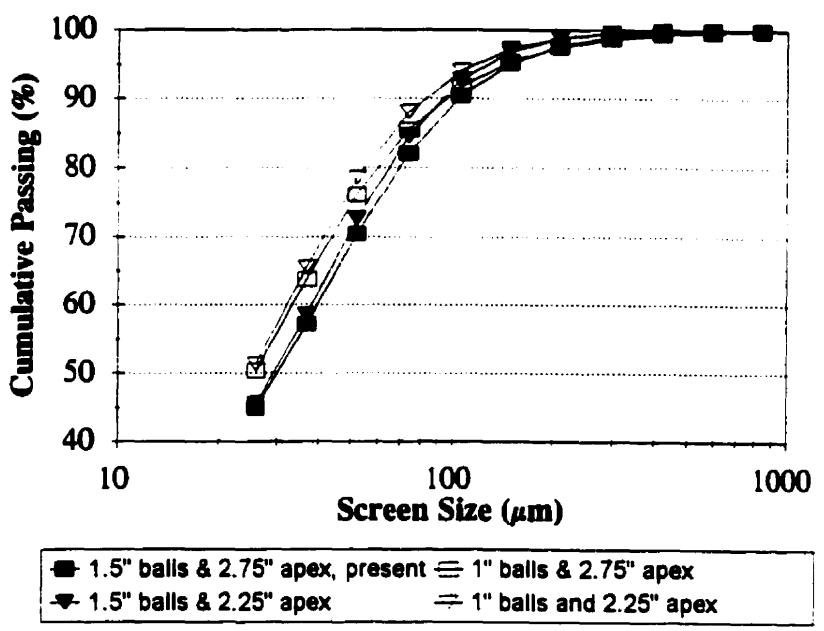


Figure 5.7 Predicted size distribution of Cu flotation feed, Louvicourt plant (Feb. 25 1997 survey)

to simulate classification behaviour of non-homogeneous ore.

The calibrated simulator was used to perform a number of simulation trials. These include trials to predict circuit performance using 25 mm (1") ball size instead of current ball size 38 mm (1.5") and changing hydrocyclones apex diameter to 57 mm (2.25"). Ball size has a significant effect on the % -75 μm of the cyclopak overflow but none on the circulating load and short circuiting value (Table 5.6). Decreasing the apex diameter, d_a, does bring both the circulating load and short-circuiting down. Presumably, the benefit of decreasing d_a can be increased if the circulating load is brought back up to its original value, either by (a) adding more water to the grinding loop, (b) operating with fewer hydrocyclones or (c) changing vortex finder diameter. A simulation was also performed to study circuit performance when both modifications are applied at the same time. The outputs of simulator can be found in Appendix C.5. As the fineness of the circuit product (Cu flotation feed) is of more interest, its predicted size distribution under the proposed modifications is shown in Figure 5.7. The effect of ball size is much more significant than that of apex diameter at 30 μm, but as shown in Table 5.6, a smaller apex diameter can yield a primary cyclone overflow with a larger - 75 μm content.

That the decrease in apex diameter can reduce the mass of +75 μm fraction in the primary cyclopak overflow without increasing the amount of -30 μm significantly is interesting. In applications where overgrinding can be a problem (i.e. flotation with a libration mesh above 30 μm but not significantly above 75 μm), this is clearly desirable. In applications where maximum "grind" is the target (i.e. particle surface produced), such as grinding of pelletization feeds, increased grinding capacity via a better choice of ball size or grinding density is the better option.

To estimate the increase in circuit capacity when 25 mm (1") balls are used, with producing a size distribution close to that of 38 mm (1.5") balls, three simulation runs

were performed with 5, 10 and 20 percent increase in current feed flow rate (Appendix C.6). Figure 5.8 compares the particle size distributions of circuit product (cyclopak overflow) for all simulations. It can be concluded that an increase of at least 20% in circuit capacity can be achieved using 25 mm (1") balls without a decrease in the current fineness of grind.

Table 5.6 Predictions of percent passing 75 μm, circulating load (CL) and recovery of fluid to cyclone underflow (R_f) under various operating conditions

	Make-Up Ball Size					
Apex Diameter	38 mm (1.5")			25 mm (1")		
↓	-75 μm (%)	CL (t/h)	R_f (%)	-75 μm (%)	CL (t/h)	R_f (%)
70 mm (2.75")	76	712	41	81	712	41
57 mm (2.25")	79	461	27	83	456	27

The results of classification performance study presented in this report must be considered cautiously as the measured partition curve indicates the existence of mineral phases with different settling velocities which cannot be readily modeled by Plitt's model. The best estimated value of separation sharpness (by optimal fit of Plitt's model) was equal to 0.83 and in reality will be higher when minerals are considered individually. An accurate assessment of classification performance requires size-by-size assays of minerals present in ore.

Despite some uncertainty associated with the predictions of the calibrated simulators, the results are still very useful for comparing circuit performance under different conditions. The predicted trends are very clear, and suggest substantial

improvements if both grinding kinetics (ball size) and classification performance (short-circuiting) are improved.

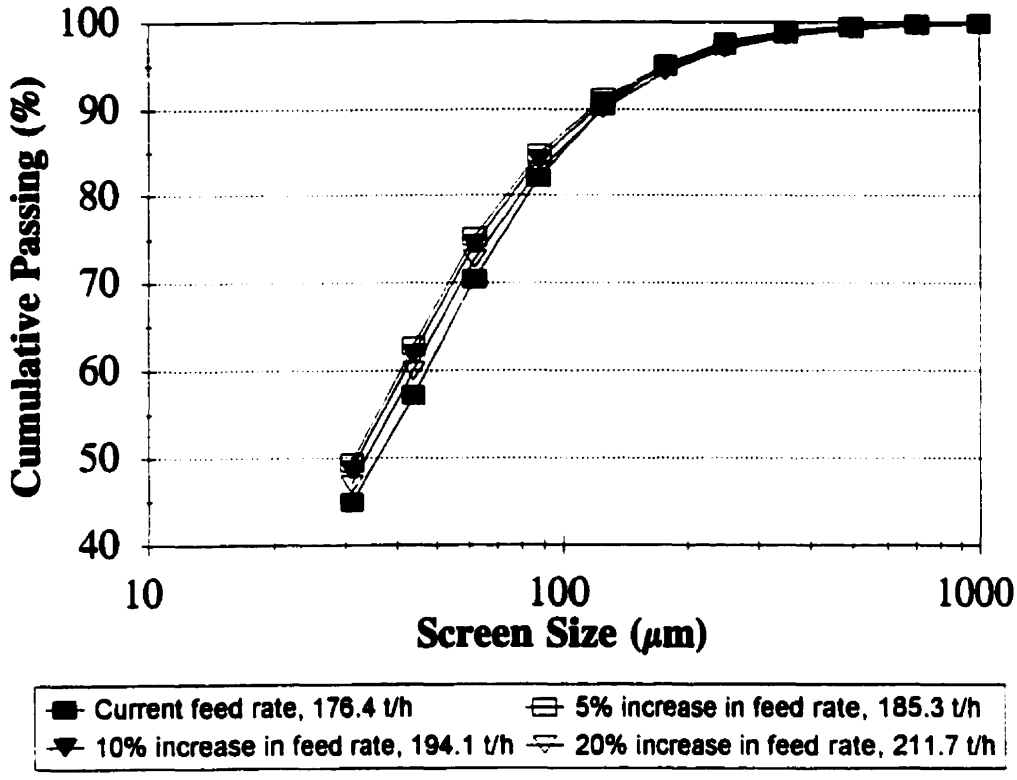


Figure 5.8 Comparison of simulated particle size distributions of circuit product under various feed rates, Louvicourt plant

The uncertainty associated with the behaviour of coarser particles in the ball mill can be significantly reduced by using smaller openings on the SAG discharge screen. A reduction from 4 to 2 mm is suggested as existing screen performance is very good and the reduction rate of 2:1 corresponds to the reduction ratio of the peak size of the selection function when reducing ball size from 38 mm to 25 mm (Fig. 4.13). Since simulations were performed without removal of the +2 mm fraction, the finer feed would also improve in circuit performance - i.e. and increase of 5% in the - 75 μm % passing of the circuit product. Another safety factor would be the ability to increase

circulating load back to the historical level of 350%.

The simulation with scaled selection functions for 25 mm (1") balls resulted in a finer product. However, there was no significant changes in hydrocyclone operation. The simulation of hydrocyclones with a smaller apex diameter showed a lower water recovery to the underflow and a higher separation sharpness, equal to 27% and 0.76, respectively, compared with those predicted for hydrocyclones with current apex diameter. It also showed a large decrease in circulating load.

At the time of writing a number of modifications, recommended followed this research, had been made:

- reducing the SAG mill discharge screen openings from 4 mm to 2 mm
- using a make-up ball charge consisting of 50% 32 mm (1.25") and 50% 25 mm (1") balls instead of 100% 38 mm (1.5")
- reducing the apex diameters of primary hydrocyclones from 70 mm (2.75") to 57 mm (2.25")

The change of make-up ball size took place at the end of May 1998 and has been reported causing P_{80} to decrease from 51 μm to 47 μm. This is in line with the simulation results considering the fact that simulations performed were based on the selection function predicted if a make-up ball size of 100% 1" balls had been used. This means that even a finer grind can be resulted if 1" balls are used. However, the use of a single make-up charge of 25 mm (1") balls has been avoided by the plant operators due to their higher costs when compared with that of bigger balls.

The change to hydrocyclone's apex diameters was implemented at the end of July 1998 and was reported has caused a decrease in the P_{80} of the cyclopak overflow stream but not as significant as that of ball size change. This is very interesting as the lesser

effect of apex diameter change on the product fineness had been predicted by the simulator (see Fig. 5.7).

5.7 Summary

The BMCS was developed and tested as the core of a series of tools for the off-line steady-state optimization of mineral ball milling circuits. The current version of BMCS is capable of simulating grinding circuits consisting of ball mill and hydrocyclone units. All modules were tested using data from various industrial plants and were proven to give reliable results.

The simulation of the Lupin and Louvicourt Mines grinding circuits demonstrated how the simulator must be calibrated and used to represent or approximate real plant performances. Using the Lupin classification data, the cyclone module of BMCS was tested and proven to produce accurate predictions particularly when it was calibrated to measured classification parameters. The performance of primary classification at the Louvicourt Mine could not be precisely simulated by the cyclone module due to the multi-component nature of ore. However, circuit simulations performed to predict possible trends in case of changes to ball size and cyclone apex diameters, taking into account the uncertainties involved due to inadequate modelling of classification process.

A step forward in use of process simulators to analyze and optimize grinding circuits is incorporation of symbolic-oriented metallurgical knowledge required to build simulation models and interpret results. This can be achieved by use of rule-based system programming which will be explained in Chapter 6.

CHAPTER 6
GRINDING CIRCUITS OPTIMIZATION SUPERVISOR (GCOS)

6.1 Introduction

To optimize the design and operation of a mineral grinding circuit, a mineral process engineer has to be knowledgable of theoretical aspects of the underlying processes and computer-based tools required to complete the optimization task at hand. In most off-line optimization exercises, steady-state circuit simulation packages are used as primary tools to investigate proposed new operating conditions or circuit designs or to search for promising ones. Therefore, one must learn how to use programs such as mass balancing, parameter estimation and simulation.

In this chapter the development of *G*rinding *C*ircuits *O*ptimization *S*upervisor (GCOS) will be explained. While *N*umerical *G*rinding *O*ptimization *T*ools in *C* (NGOTC) and *B*all *M*illing *C*ircuits *S*imulator (BMCS) programs, described in Chapters 4 and 5, provide some basic tools, they cannot assist a process engineer in making decisions required in various steps of an off-line optimization study. Hence, a program capable of helping one to make decisions or interpret various information could be useful. The progress in artificial intelligence, particularly in *K*nowledge-*B*ased *S*ystems (KBSs), has allowed development of computer programs that can achieve this goal. GCOS in fact attempts to computerize the problem solving expertise often used in the off-line optimization of mineral grinding circuits. The basic concepts of KBSs were introduced in Chapter 3. The inference engine of CLIPS which controls the execution of the GCOS knowledge base has been discussed elsewhere [STB 1993]. In this chapter,

domain-specific aspects of GCOS and its implementation are presented.

Expert systems have found many applications in mineral processing particularly in the monitoring and control of comminution and flotation circuits [Bradford 1991, Ynchausti and Hales 1992, Reuter and Van Deventer 1992]. These systems are designed to be used in off-line or on-line mode as depicted in Figure 6.1. In the off-line mode, the user provides the input to the system and must analyze its output. In the on-line mode, these systems are configured in open or closed loops [Leiviskä 1991]. Open-loop systems receive measurement data from sensors and advise process operator to take the appropriate control actions. In other words, the control loop is closed by the human operator. In on-line closed-loop systems, the expert system implements actions on itself without human intervention.

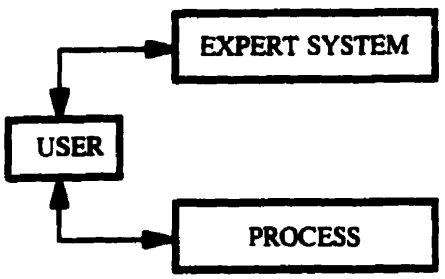
The GCOS knowledge base runs off-line and requires the user to input data. The problems addressed by the knowledge base are not necessarily related to process monitoring and control.

6.2 The Structure of GCOS

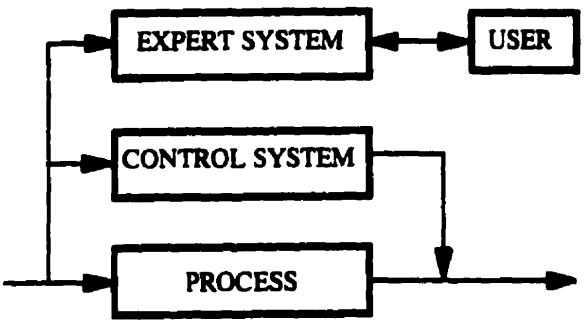
Figure 6.2 shows a simplified view of the GCOS knowledge base which consists of several modules or partitions. The modular design was considered due to the increased level of control that is provided by such structures to execute various rule sets. The modules can be classified into general and specific categories. The general modules, i.e. TEMPLATE, QUERY and FUNCTIONS, provide the primary structural blocks of the knowledge base. The specific modules define the grinding optimization domain and capture the expert knowledge.

6.2.1 General Modules

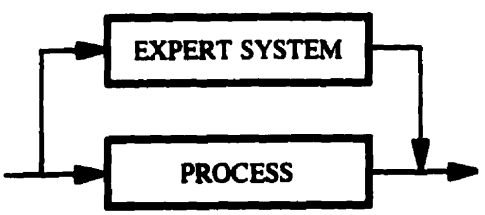
The TEMPLATES module defines the basic template facts which are used in rules. All templates were made exportable to other modules by setting the export slot

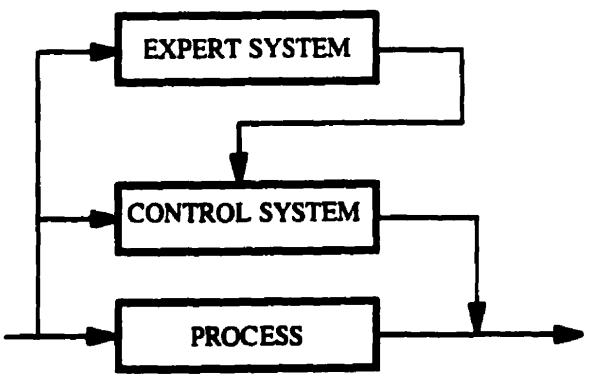


(a) Off-line mode



(b) On-line open-loop advisory system





(c) On-line closed-loop systems

Figure 6.1 Various applications of expert systems

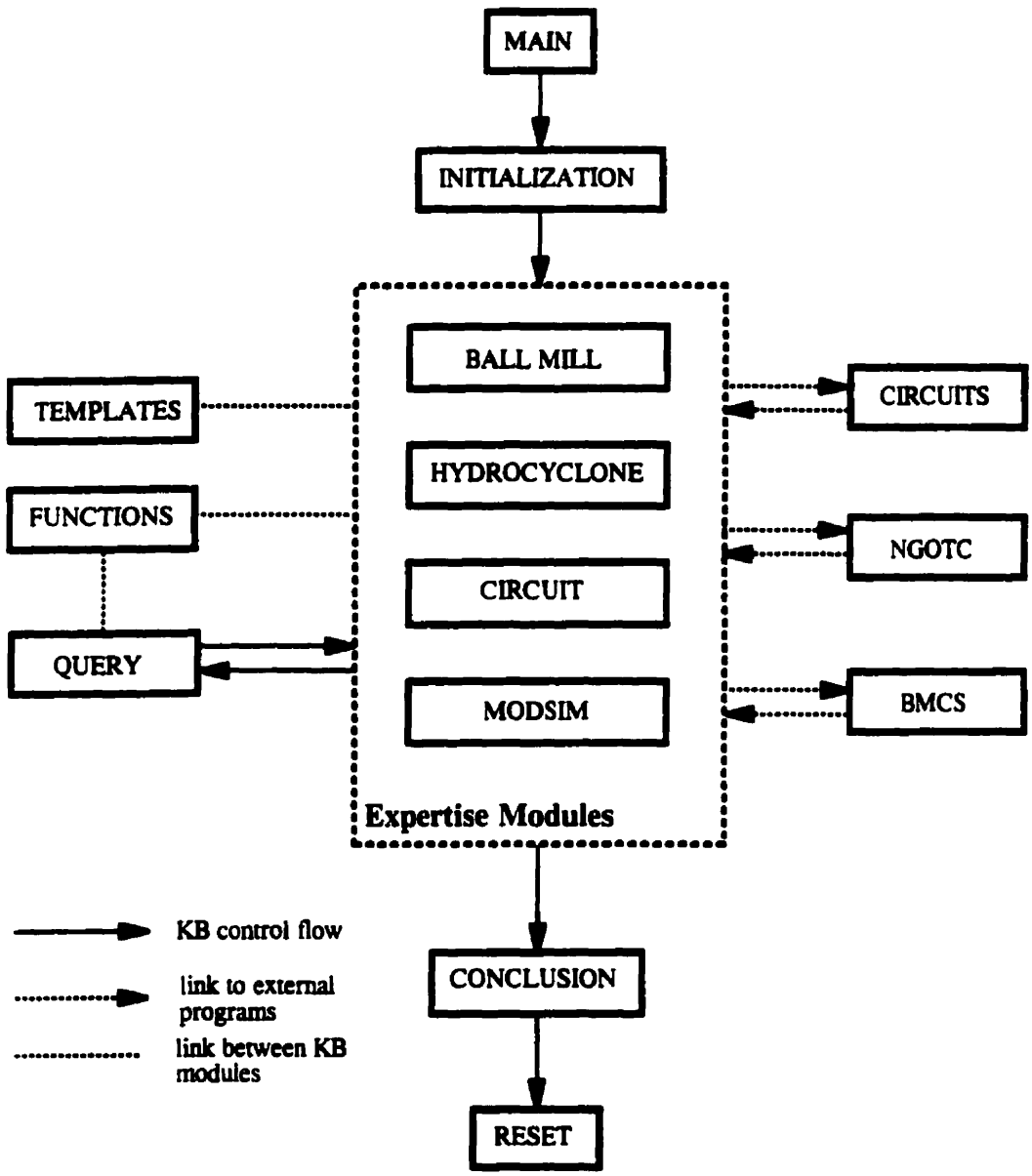


Figure 6.2 Simplified view of GCOS structure

of the defmodule construct which defines the TEMPLATES module.

The QUERY module asks all relevant questions whose answers are needed at each stage of the information processing. It imports all templates and functions defined in TEMPLATES and FUNCTION modules and allows exporting its constructs to other modules. The query rules are triggered when a parameter (numeric or non-numeric) is not bound to a specific value and its "question" non-ordered fact still exists in the knowledge base. For example, the following rule has been written to query the user about non-numeric (symbolic) parameters:

```
(defrule QUERY::ask-non-numeric-question
?f <- (question (parameter ?the-parameter)
                (module ?the-module)
                (the-question ?the-question)
                (precursors)
                (already-asked FALSE)
                (help ?the-help))
    (parameter (name current-module)(value ?the-module))
    (nonnumparam (name ?the-parameter)(menu $?the-menu))
=>
(modify ?f (already-asked TRUE))
(assert (user-response ?the-parameter
        (ask-non-numeric ?the-question ?the-help $?the-menu))))
```

The FUNCTIONS module defines functions used in other modules. The most important functions are those designed to query the user to assign values to numeric or non-numeric parameters. Two separate functions "ask-numeric" and "ask-non-numeric" were defined to ask questions regarding numeric and non-numeric parameters, respectively. These functions are called from the *r*ight *h*and *s*ide (RHS) of query rules defined in the QUERY module.

The knowledge base execution is started by focusing on the MAIN module which welcomes the user and then causes the focus of the knowledge base to move onto the INITIALIZATION module. This module then will set the current module on one of the expertise modules, i.e. BALLMILL, HYDROCYCLONE, CIRCUIT or MODSIM,

depending on the user choice.

6.2.2 Expertise Modules

The expertise modules were designed to cover problems pertaining to the optimization of mineral grinding circuits. The BALLMILL module includes various sets of rules for the optimization of ball size and other operating conditions. The HYDROCYCLONE module contains rules which are used to improve classification efficiency through optimizing a number of critical parameters such as water recovery to the cyclone underflow and separation sharpness. The MODSIM module assists the user in building the unit operation models and simulating circuits. The modelling part of this module helps the user to estimate selection function parameters, in order to analyze the grinding kinetics and model ball mills. The simulation section of this module uses frames to represent unit operations. The rules first verify that all requirements to start a simulation session are satisfied. A simulation trial is then performed by a set of rules. The unit operations involved in a circuit flowsheet are simulated by sending a message to the related object.

6.3 Knowledge Acquisition

A systematic approach was taken to gather various types of optimization knowledge applied to industrial grinding circuits. A major part of this knowledge was obtained from working on a number of real plant optimization cases including Agnico Eagle, La Ronde Division, Les Mine Selbaie, Lupin Mine and Louvicourt Mine. Moreover, the existing grinding literature was consulted to extract additional knowledge. The knowledge-base was developed incrementally by adding new rules to each module after preliminary prototyping. This was possible due to the modularity of the knowledge base and its relatively small size.

6.3.1 Knowledge Types

An experienced mineral processor who does an optimization study uses many

kinds of knowledge to finally come up with recommendations that can improve the operation. The knowledge or expertise applied by the mineral processor can be categorized as that of diagnosis, interpretation, trouble-shooting or optimization. The rules contained in GCOS embody some of this knowledge types particularly interpretation, diagnostic and optimization. Many rules encoded in GCOS are universal and can be applied to any grinding process; others are very specific to the software developed in this thesis.

6.3.2 Conflicting knowledge

During the literature survey and the development of GCOS, conflicting knowledge were noticed and if possible they were resolved. For example, Bond and Azzaroni have proposed rules of thumb relationships to predict make-up or top ball sizes. Using the same data, however, Azzaroni's equation predicts a size much larger than that predicted by Bond's. In such cases, GCOS informs the user of both predictions. Since all rules of GCOS had to be encoded and placed in the knowledge base manually, the consistency of implemented rules were considered during their development. This was possible also due to the relatively small size of the current knowledge base.

6.4 GCOS Implementation

Since CLIPS was selected as the shell, the knowledge-base was implemented according to the CLIPS special syntax in order to define facts, rules and frames. However, the rules constructing the expertise modules were first expressed in English language regardless of the selected knowledge-base shell and then gradually encoded into the knowledge base using CLIPS syntax. The various rule sets were incrementally built by studying various grinding circuit cases and existing literature.

6.4.1 Knowledge-Base Shell and Development Tool

The knowledge elicited from experts and literature was formalized in the form of facts, rules, objects and functions. GCOS is in fact a knowledge base strongly dependant

on the pattern matching ability of CLIPS, due to the extensive use of complicated patterns in the *l*eft *h*and *s*ide (LHS) of rules. Pattern connectives and wildcards, field constraints, mathematical operators and test features have been used in rules.

6.4.2 Knowledge-Representation

The knowledge was basically represented by the ordered facts, non-ordered or template facts, rules and frames (objects). The external executable programs are called from the RHS of rules.

There are constructs in CLIPS which are used to define rules and classes (frames). A rule or frame defined in the knowledge base can be very simple or complex. In CLIPS, a rule is defined by the defrule construct [STB 1993]:

```
(defrule <rule-name> [<comment>]
        [<declaration>]
        <conditional-element>*                  ; LHS
        =>
        <action>*                               ; RHS
```

Rules can have more than one conditional element or action. They can also have a single declaration pattern to define a specific property such as the priority of firing a rule over the other rules in the agenda.

To define a class frame, the defclass construct provided by *C*LIPS *O*bject *O*riented *L*anguage (COOL) must be used. The general syntax of a class frame in CLIPS is as follows:

```
(defclass <name> [<comment>]
        (is-a <superclass-name> +)
        [<role>]
        [<pattern-match-role>]
        <slot>*
        <handler-documentation>*)

        <role> ::= (role concrete | abstract)
        <pattern-match-role> ::= (pattern-match reactive | non-reactive)
```

<slot> ::= (slot <name> <facet>) |*
(single-slot <name> <facet>) |*
(multislot <name> <facet>)*
<handler-documentation> ::= (message-handler <name> [<handler-type>])

A class frame can inherit properties and behaviours from one or more pre-defined superclasses. The new properties and behaviours are directly defined through various slots and message handlers. Facets allow various properties such as default value, access type or constraint attributes to be defined for each slot.

6.5 GCOS Knowledge Modules

The GCOS knowledge base consists of four modules to cover various domain areas: (1) BALLMILL to optimize the performance of ball mill units, (2) HYDROCYCLONE to optimize the performance of hydrocyclone classification units, (3) CIRCUIT to propose alternative circuit configurations and (4) MODSIM to model and simulate full grinding circuits. These modules are discussed below.

6.5.1 Ball Mills

This knowledge module involves facts and rules to describe ball mills and improves the breakage process taking place inside ball mill grinding units. One of the most important parameters is grinding media size. Evaluating grinding kinetics is critical to investigate the efficiency of grinding mills in breaking particles.

Various parameters must be considered during optimization. Table 6.1 shows a listing of parameters or attributes defined in the ball mill module which are used for decision-making. During the execution of the knowledge base, all parameters will be finally bound to specific values which are then considered as known facts. The value of a parameter is either given by the user as an initial fact (evidence) or inferred from other previously known facts as an intermediate or final fact.

Table 6.1 Important design and operating parameters defined in Ball Mill module of GCOS

Parameter	Type	Unit	Derivation	Constraint
Discharge type	Symbolic	-	Initial*	Overflow, diaphragm
Mill length	Numeric	m.	Initial	< = 9 m.
Mill diameter	Numeric	m.	Initial	< = 6 m.
Laboratory work index	Numeric	kWh/t	Initial	> 0
Operating work index	Numeric	kWh/t	Initial	> 0
Critical speed	Numeric	rpm	Intermediate	> 0
Media type	Symbolic	-		Ball, slug
Make-up ball size	Numeric	mm.	Initial	> 0
Liner wear rate	Numeric	kg/kWh	Initial	> 0
Ball wear rate	Numeric	kg/kWh	Initial	> 0
Energy consumption	Symbolic	-	Intermediate	Good, ok, bad
Operation mode	Symbolic	-	Initial	Wet, dry
Circuit type	Symbolic	-	Initial	Open, closed

*An initial derivation requires the user to provide the value of the parameter, while for intermediate ones the system will infer the value of the parameter from known parameters.

All parameters defined in the knowledge base are constrained to specified bounds or values. Numerical parameters may have lower and/or upper numerical bounds. These boundaries fill the min and max slots of the fact template "numparam" defined for numerical parameters.

```
(deftemplate numparam
        (slot name)
        (slot min (default ?) (type NUMBER))
        (slot max (default ?) (type NUMBER)))

(deftemplate nonnumparam
        (slot name)
        (multislot values)
        (multislot menu)
        (slot convert))
```

In the case of non-numerical parameters, they are constrained to specified multiple symbolic values defined in knowledge base. All constraint information is passed to the query functions to validate the input data.

The information obtained following a grinding survey contains very useful facts in the sense that they can be used to infer new information which allows process identification, analysis, diagnostic and optimization. In the ball mill module of GCOS, a number of rules were developed based on the heuristic used in the interpretation of ball mill facts (see Appendix D.1). These included:

- estimation of the make-up or top ball size using the Bond and Azzaroni empirical relationships
- determination of the energy efficiency based on laboratory and operating work indices
- identification of the excessive liner wear condition
- identification of the excessive ball wear condition

In practice a single objective or a combination thereof can be pursued during an optimization exercise. An objective is set by querying the user and binding the appropriate value to the optimization-objective parameter. The optimization-objective parameter has been used as a conditional element in the LHS of all rules and depending on its value, a rule set is selected and applied to the optimization problem. Given a specific optimization objective, the system identifies the potential areas that can be improved using available process information obtained from grinding surveys.

6.5.2 Hydrocyclones

This module consists of rules to analyze the performance of classification systems based on the information obtained from a sampling campaign. Since hydrocyclones are often used to close the grinding circuit, most rules refer specifically to this type of classifier.

The rules defined by this module contain heuristic knowledge on how hydrocyclone variables can be changed to achieve a desired goal (see Appendix D.2). Most often the number of operating cyclones, the apex diameter, the vortex finder diameter are used to modify hydrocyclone operation and performance. Though hydrocyclone diameter and inclination are also important parameters, they cannot be as easily used for optimization purposes. Table 6.2 shows some of the parameters considered in the hydrocyclone module.

The objective of hydrocyclone (classification) optimization could be chosen as one of the following:

- achieving the correct product size and/or density (% solids)
- improving classification efficiency
- increasing capacity

Table 6.2 Important parameters defined in hydrocyclone knowledge source

Parameter	Type	Unit	Derivation	Constraint
Number of operating cyclones	Numeric	-	Initial	> 0
Cyclone pressure	Numeric	kPa	Initial	> 0
Feed flow rate	Numeric	t/h	Initial	> 0
CUF density (% solids)	Numeric	%	Initial	0-100
Corrected cut size, $D_{50}c$	Numeric	μm	Initial	> 0
Separation sharpness, m	Numeric	-	Initial	> 0
Water split, R_f	Numeric	%	Initial	0-100
Underflow discharge type	Symbolic	-	Initial	Spray, semi-rope, rope
Classification arrangement	Symbolic	-	Initial	Single stage, multi-stage
Ore constitution	Symbolic	-	Initial	Single species, multi-species

Figure 6.3 shows a rule-based decision tree that was designed as a part of the HYDROCYCLONE module. Since the inefficiency problems of hydrocyclones operations have been well recognized, they can be solved by a number of corrective actions which must be decided based on the performance indicators and the optimization objectives. In Figure 6.3, the leaf nodes represent various conclusions that can be derived through a series of questions about parameters which are sequentially related to each other.

6.5.3 Circuits

The CIRCUITS module involves rules that apply to a grinding circuit as a single system. Various circuit designs can be found in industrial plants. Grinding circuit design in terms of equipment and configuration has evolved from very simple circuits to more complex ones. The selection of the right equipment and flowsheet for a new plant is normally based on past experience. Furthermore, many existing plants find their equipment constrained to meet new demands such as higher capacity or finer grind. Therefore, drastic modifications to an existing plant design could be expected when current equipment and configuration cannot meet operation objectives.

The rules helps a user consider alternate circuit configurations (see Appendix D.3). All parameters defined in this module (except circuit configuration) are symbolic ones and their values are derived by querying the user. The symbolic parameters having a 'yes' or 'no' value are listed below:

- product size distribution coarseness
- product size distribution width
- overgrinding problem
- sharp classification required
- ball mill density
- fresh feed contains fines

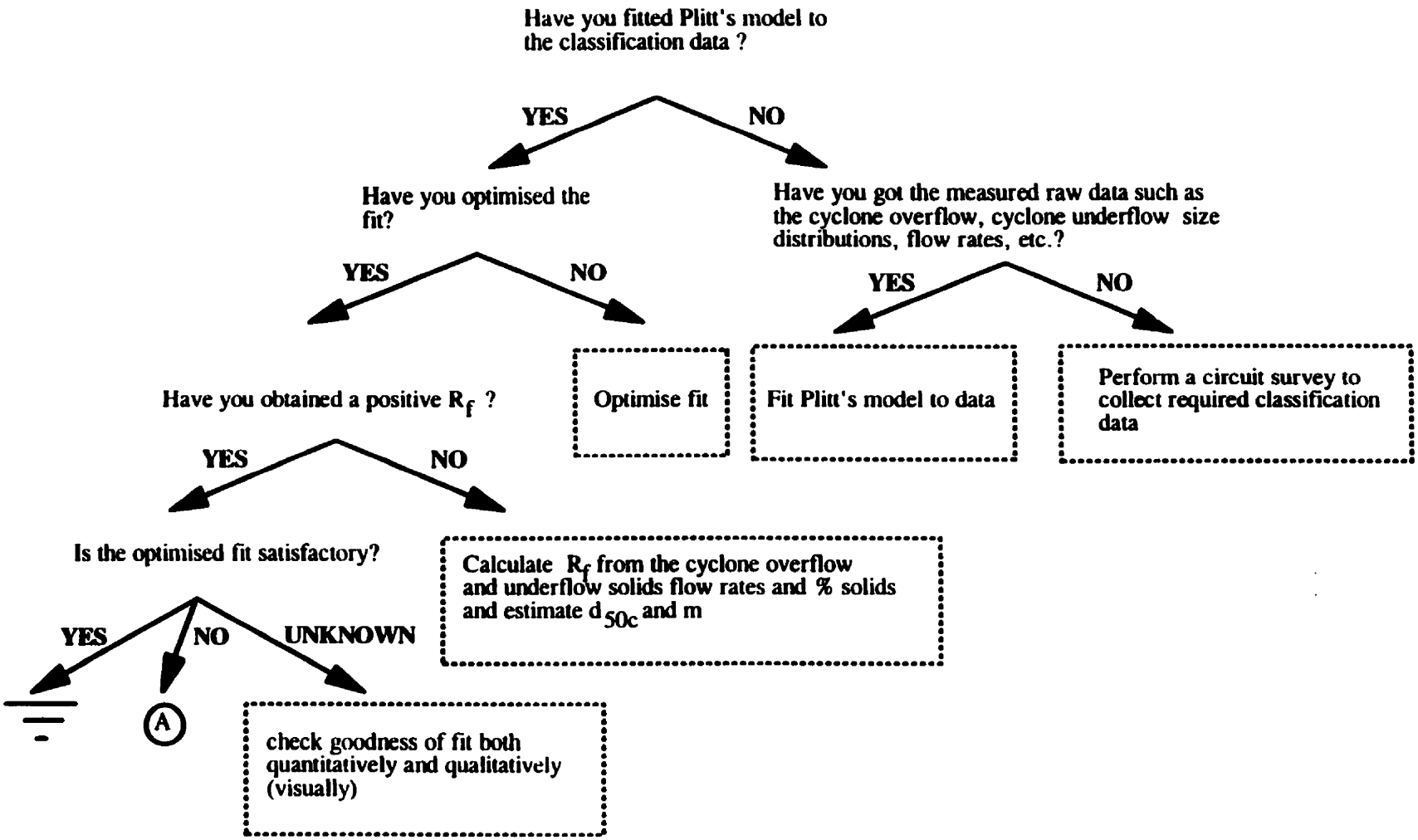


Figure 6.3 The rule-based decision tree implemented as a part of HYDROCYCLONE module

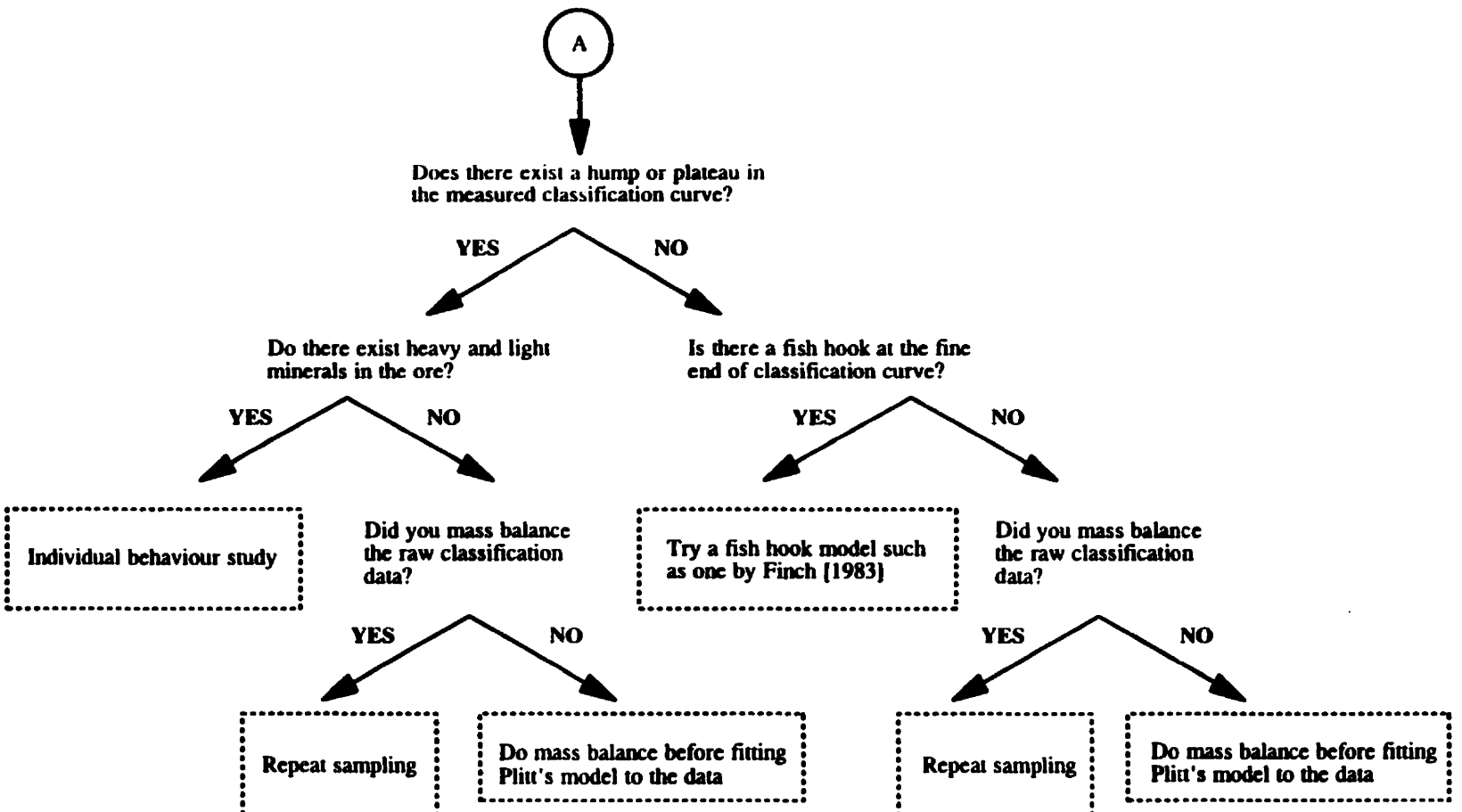


Figure 6.3 The rule-based decision tree implemented as a part of HYDROCYCLONE module (continued)

- fresh feed contains coarse
- higher capacity required
- finer grind required

6.5.4 Modelling and Simulation

This module, MODSIM, involves rules concerning modelling and simulation tasks (see Appendix D.4). Simulation exercise can be divided into more detailed stages including pre-simulation, simulation and post-simulation. The mathematical modelling of unit operations, or simply modelling, is in fact the pre-simulation stage which is very critical in building a grinding circuit simulator. Usually verified models already exist (off-the-shelf models) and only parametric estimation is required. However, when such models are unavailable, they must be developed from scratch. Parametric estimation is also a tool for unit performance analysis. For example, in the ball mill model, the selection function is a measure of grinding kinetics efficiency or in Plitt's model m indicates how sharply a hydrocyclone separates particles above and under the cut size.

Parameters involved in modelling and simulation tasks have been identified and represented by a simple fact template called "parameter". Depending on its type, a parameter can assume either a numerical or a non-numerical value. The value of a parameter is found by rules or simply asking the user. Parameter facts are mostly used in LHS of rules to establish other facts or final conclusions. For example, parameters such as the shape of selection function vs. particle size curve, goodness of fit and noise associated with measured data have been represented in GCOS as symbolic data.

6.5.4.1 Modelling

The two models for ball mill and hydrocyclone units must be calibrated by the user. The modelling module lets the user estimate the selection function or predict selection function values in the case of a ball diameter change.

The following parameters must be known in order to fit the ball mill and Plitt's hydrocyclone models to a specific circuit:

- breakage (distribution) function
- selection function
- residence time distribution (RTD)
- cyclone geometry
- Plitt's model calibration (adjusting) factors

The rules defined in modelling section of MODSIM guides a user in parametric estimation of the selection function and model fitting process. For Plitt's model the performance indexes and calibration factors must be estimated off-line (e.g. with a commercial spreadsheet).

6.5.4.2 Simulation

A grinding circuit simulator is a valuable tool to a mineral processor to investigate the effect of design and operating parameters on relevant performance parameters in a noise-free environment. However, there are many ways for a user to initiate a simulator search for a desired solution. Depending on the objective of a study, rules can guide the user to narrow the search space, and in fact guide simulation trials.

The system first checks if the user has done all the necessary steps or not. The simulator will be activated if all information seems to be available. The rules implemented in this module provide linkage to BMCS. The system can give some general recommendations regarding simulation trials.

The simulation section of the MODSIM partition comprises of the rules that guide the user to simulate a grinding circuit. The circuit objects are first created. The simulation of each node is then performed by sending a message to the corresponding

object. There are five rules that are circularly fired to perform iterations to reach the steady-state.

The purpose of performing simulations is to search for better design and operating states and engineering judgement is very critical to evaluate the results. One should be aware of practical limitations when performing this evaluation. A number of rules were developed to ensure simulation results are acceptable from both technical and practical perspectives.

6.6 Frame-Based Simulation

The frame-based or object-oriented approach was used to represent grinding and classification devices and then to simulate a full grinding circuit by sending messages to these objects to process their input stream. The use of *O*bject-*O*riented *P*rogramming (OOP) in developing mineral processing simulation and control software has been discussed by Reuter and van Deventer [1992], Ynchausti and Hales [1992] and Sastry and Sudhir [1995]. There are various types of comminution devices used in industry including crushers and grinding units. Also, for size classification various devices such as screens, screw classifiers and hydrocyclones are used.

In a frame-based system, two mechanisms can be used to manipulate the knowledge contained in the frames, i.e. pattern matching rules and object communication. Using the former approach, after defining BALLMILL as a class, we need only one rule to calculate the size distribution of all instances of that class:

IF ball mill feed data is available
ball mill model is defined
THEN calculate ballmill output size distribution

In the second approach, however, a message is sent to each instance of the BALLMILL class:

```
IF      ball mill feed data is available
        ball the mill model is defined
THEN SendMessage(ballmill1, CalculateOutputSize)
```

After rule execution, a message is sent to ballmill1 to generate the ball mill output size distribution. It is assumed that we have already defined a procedure attached to the class frame BALLMILL which intercepts the message and calculates the size distribution of the product accordingly. In this example, the message was sent from the RHS of a rule. Messages can be sent also from an object (defined in its class frame) to another object.

The frames defined in GCOS include a hierarchy of object classes for comminution and classification devices. Only the BALLMILL and HYDROCYCLONE frames will be explained here. These objects are called as corresponding modules to do their function as explained in Chapter 5.

6.6.1 COMMINUTION

The hierarchy of comminution objects shown in Figure 6.4 was implemented by using the defclass construct of COOL. The defined classes however cover one type of comminution device (see Fig. 3.3) used in fine grinding, due to the limited scope of the thesis. The only concrete frame class is OVERFLOW-DISCHARGE-BALLMILL which is used for creating instances of ball mills. A concrete frame permits instantiation of objects.

```
(defclass COMMINUTION-UNIT (is-a USER)
        (role abstract)
        (slot manufacturer-name)
        (slot identification-number)
        (slot installation-year)
        (slot electrical-power)
        (slot capacity)
        (slot net-id (create-accessor read-write))
        (slot node (create-accessor read-write)))

(defclass TUMBLING-MILL (is-a COMMINUTION-UNIT)
        (role abstract)
```

```
        (slot length)
        (slot diameter))

(defclass BALLMILL (is-a TUMBLING-MILL)
        (role abstract)
        (slot media-type)
        (slot media-size)
        (message-handler grind))

(defmessage-handler BALLMILL grind()
        (system ballmill.exe))

(defclass OVERFLOW-DISCHARGE-BALLMILL (is-a BALLMILL)
        (role concrete)
        (pattern-match reactive))
```

The message handler grind() has been defined to call the BALLMILL program in response to the message sent to the created instances of OVERFLOW-DISCHARGE-BALLMILL object. As a user-defined class COMMINUTION-UNIT directly inherits a number of functions defined in USER such as the system message handlers for initialization and deletion actions. The USER is a pre-defined class of COOL. The other classes, such as BALLMILL, inherit these functions indirectly.

6.6.2 HYDROCYCLONE

Hydrocyclones are special classification devices that are widely used in industrial grinding plants. The hydrocyclone frame has slots to represent knowledge that is significant such as separation sharpness, cut size, and pressure. The message handler 'classify' simulates the classification operation of a hydrocyclone.

```
(defclass HYDROCYCLONE (is-a USER)
        (role concrete)
        (slot manufacturer-name)
        (slot identification-number)
        (slot installation-year)
        (slot separation-sharpness)
        (slot cut-size)
        (slot pressure-drop)
        (slot net-id (create-accessor read-write))
        (slot node (create-accessor read-write))
        (pattern-match reactive))
```

```
(defmessage-handler HYDROCYCLONE classify()
        (system cyclone.exe))
```

In practice there are certain steps prior to the steady-state simulation of a grinding circuit. A model of the circuit must be built and validated. The GCOS rule-based system has a number of rules that check if the user has done this step or not. If not, the GCOS gives the user an appropriate recommendation. Once all the information is available, the GCOS runs the pre-simulation program.

Based on the selected circuit, instances of different objects that represent each node in the circuit are constructed. The required instances are made after firing the following rule:

```
(defrule create-instances
        (circuit number ?cirno)
        (circuit ?cirno ?node ?nodetype ?id)
=>
        (if (eq ?nodetype 1) then (make-instance (sym-cat "BALLMILL-" ?id) of
                OVERFLOW-DISCHARGE-BALLMILL (node ?node) (net-id ?id)))
        (if (eq ?nodetype 2) then (make-instance (sym-cat "HYDROCYCLONE-" ?id) of
                HYDROCYCLONE (node ?node) (net-id ?id)))
        (if (eq ?nodetype 3) then (make-instance (sym-cat "JUNCTION-" ?id) of
                JUNCTION (node ?node) (net-id ?id)))
        (if (eq ?nodetype 4) then (make-instance (sym-cat "SPLIT-" ?id) of SPLIT (node
                ?node) (net-id ?id)))
        (if (eq ?nodetype 5) then (make-instance (sym-cat "FIXCLASS-" ?id) of
                FIXCLASS (node ?node) (net-id ?id)))
        (if (eq ?nodetype 100) then (make-instance (sym-cat "CONVERGENCE-" ?id)
                of CONVERGENCE (node ?node) (net-id ?id))))
```

All object classes have slots for the node number and its network identification number.

The following rule checks whether a calculation pass (or an iteration cycle) is completed or not. When the current node is not the last node, the fact defining the current node is updated to specify the next node to be processed. If the current node is the last one, a new fact is asserted to trigger the rule which then checks convergence and

simulation status.

If convergence is not reached within the tolerance given by the user, then another iteration will be started.

```
(defrule check-convergence
        ?f1 <- (check convergence)
        ?f2 <- (current-iteration ?iter)
=>
        (retract ?f1 ?f2)
        (system simcont.exe)
        (open simstat.lst sim "r")
        (bind ?status (read sim))
        (if (eq ?status completed) then (assert (simulation completed))(system
        repgen.exe)
        else (assert (current-iteration =(+ ?iter 1))))
        (close sim))
```

When the rule 'check-convergence' is fired, the matching facts f1 and f2 are retracted from the knowledge base by the retract command. Then the simulation status is read from the 'simstat.lst' file. Whether or not the steady state is reached is indicated by the convergence objects. If the simulation is completed, a report is generated and displayed; otherwise, a new current-iteration fact is asserted into the knowledge base which triggers another rule to start the next simulation iteration.

6.7 Summary

The implementation of GCOS demonstrated the effectiveness of rule-based systems to represent (symbolic or textual) grinding optimization knowledge and therefore their capability to improve the user interface of these types of software in terms of assisting the user in making decisions during off-line optimization practices.

The conclusions drawn by GCOS assist a process engineer to have a better understanding of the process and make optimization decisions. Using process knowledge expressed in the form of simple facts and rules, GCOS quickly infers all possible new

information that helps identify operation problems in terms of energy utilization, metal consumption due to liner and/or ball wear and appropriate make-up ball size. The required data by GCOS are very basic and are normally collected during a grinding survey.

The classification rules generate recommendations regarding how to increase or decrease classification cut size, reduce the recovery of fluid (water) to cyclone underflow and increase separation sharpness. When no classification objective is set by the user, the system tries to establish one by asking the user a series of questions related to data analysis and classification efficiency parameters.

Rules defined in the knowledge base of GCOS are either general such as those for recommending an advice for improving a grinding circuit operation or very specific such as those for performing parametric estimation or simulation by executing external programs, NGOTC and BMCS. The testing of GCOS will be presented in Chapter 8.

CHAPTER 7
TESTING OF GRINDING CIRCUITS OPTIMIZATION SUPERVISOR

7.1 Introduction

The performance of a *K*nowledge-*B*ased *S*ystem (KBS) must be assessed by its testing using various data sets, preferably realistic ones. A knowledge base, even with a small number of rules, can generate unexpected performance if it is not tested for error detection and removal. In rule-based systems, however, debugging differs from that of traditional computer programs conceptually and practically. For example, a knowledge base must be checked for removing redundant, conflicting, subsumed, circular and unnecessary rules in order to be consistent and efficient [Gonzalez and Dankel 1993]. The knowledge base must be also checked for dead-end, missing and unreachable rules. Comprehensive knowledge base testing includes two parts, i.e. verification and validation. In verification the knowledge base is examined to be right in terms of its consistency and completeness. In validation the knowledge base is examined to be adequate in terms of addressing specified domain problems by providing correct answers. Therefore, a KBS may be verified, but not validated. A system must be first verified in order to be validated.

*G*rinding *C*ircuits *O*ptimization *S*upervisor (GCOS), explained in Chapter 6, was tested using a number of prepared analysis and optimization cases of real grinding circuits. In this chapter example runs of GCOS using these data are presented. The aim of the testing was to make sure that GCOS performs correctly as expected by the system design. For example, focusing on the relevant module, firing of relevant rules, asking

questions when they are necessary and unredundant and providing context sensitive help are among the issues that were considered during system testing.

7.2 Testing GCOS

All modules of GCOS have been tested using various data. Since it was impractical to present all test runs, some of them are given below as examples. These example runs demonstrate how a novice user can use GCOS as a consulting system and also show how the system responds to the user input. The user input and system output in consultation sessions were saved into a text file using the "dribble-on" command of CLIPS (*C Language Integrated Production System*).

7.2.1 Agnico Eagle, La Ronde Division

The selection function estimation results for the two ball mills were discussed in Section 4.6.1. Table 7.1 shows general circuit design and operating data.

Table 7.1 Design and operating data of ball milling circuits (AELRD)

Circuit type	Closed
Grinding operation mode	Wet
Discharge mechanism	Overflow
Diameter inside liners (m)	$\simeq 3.5$
Length inside liners (m)	$\simeq 5.2$
Mill speed (%CS)	N/A
Ball top size (mm)	76
Ball material	Steel
Ore specific gravity	N/A
Laboratory work index (kWh/t)	N/A
Operating work index (kWh/t)	N/A
F_{80} (μm)	$\simeq 212$
Installed motor power (kW)	N/A
Power draw (kW)	N/A

The two grinding lines have the same design and are operated similarly. The shape of the selection function vs. particle size curves show a linear trend without a hump. This situation may indicate that grinding balls are too large. This information was used to test a set of rules regarding the selection function shape and ball size. The consultation sessions are provided in below*:

```
**********************************************************
* Grinding Circuits Optimization Supervisor (GCOS)       *
* August 1998                                            *
* McGill University                                      *
* Mining and Metallurgical Engineering Department        *
* Mineral Processing Group                               *
**********************************************************

GCOS is a knowledge-based expert system to assist
a mineral processing engineer to optimize a ball
milling circuit. The system will ask a series of
questions to reach a conclusion or a number of
conclusions.

Please press any key to continue ...

GCOS> Please choose one of the following topics?
      1 ball mill
      2 hydrocyclone
      3 circuit
      4 modelling and simulation
      ==> 1
GCOS> What is your optimization objective?
      I increase throughput or grind fineness
      D decrease operating costs
      U unknown
      ==> i
GCOS> What shape does the selection function vs. particle size curve have?
      L linear
      SH straight line with a small hump at coarse end
      LH large hump
      ==> l
GCOS> What mode of grinding operation is used?
      W wet
      D dry
      ==> w
GCOS> What material are balls made of?
```

*In all consultation sessions provided, questions were made bold. The text of title screen was omitted in next ones for brevity.

S steel
I silica
==> s

GCOS> What make-up ball size is used (in millimeters)?
==> 76

GCOS> Please enter the internal diameter of the mill, measured inside the liner (in meters)?
==> 3.5

GCOS> What type of discharge mechanism does the mill have?
O overflow
D diaphragm
==> o

GCOS> What circuit type is used?
O open
C closed
==> c

GCOS> At what speed is the mill running (in rpm)?
==> u

GCOS> Can the ball mill feed get much coarser? (y/n)
==> y

****

CONCLUSIONS

****

No conclusion was found.
Please press any key to continue ...

The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> y

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 1

GCOS> What is your optimization objective?
I increase throughput or grind fineness
D decrease operating costs
U unknown
==> i

GCOS> What shape does the selection function vs. particle size curve have?
L linear
SH straight line with a small hump at coarse end
LH large hump
==> l

GCOS> What mode of grinding operation is used?
W wet
D dry
==> w

GCOS> What material are balls made of?
S steel

I silica
==> s
GCOS> What make-up ball size is used (in millimeters)?
==> 76
GCOS> Please enter the internal diameter of the mill, measured inside the liner (in meters)?
==> 3.5
GCOS> What type of discharge mechanism does the mill have?
O overflow
D diaphragm
==> o
GCOS> What circuit type is used?
O open
C closed
==> c
GCOS> At what speed is the mill running (in rpm)?
==> u
GCOS> Can the ball mill feed get much coarser? (y/n)
==> n
GCOS> Was ball size increased in the past to improve throughput or grind? (y/n)
==> n

****

CONCLUSIONS

****

The system reached to the following conclusions:

- Decrease make-up or top ball size by 13 mm (0.5 inch). This can be achieved using a blend of make-up balls. Test the effect of this change by NGOTC before real plant exercise.

==>
The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> n

Figures 7.1 and 7.2 show the classification performance curves calculated based on the same data sets for Lines 1 and 2, respectively. To fit the model, R_f was calculated using measured percent solids and two other parameters, d_{50c} and m, were estimated using the optimizer tool of the Quattro Pro* software (a non-linear optimization package which uses the Newton or conjugate method to minimize the value of a target cell. The target cell in this case is the sum of the squared of the difference between actual and calculated recoveries). The blocks of the spreadsheet file containing the results of the last iteration for both lines are given in Appendices E.1 and E.2. It should

*Quattro Pro is a trademark of Borland International

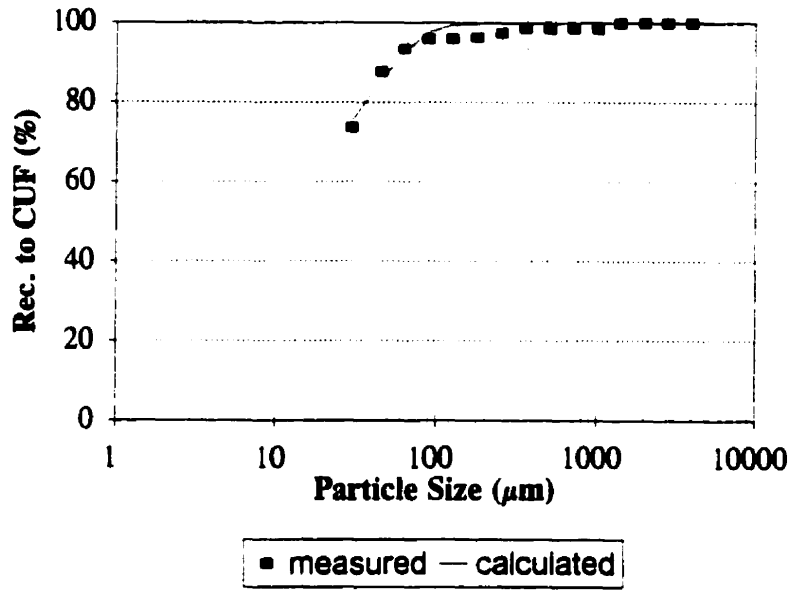


Figure 7.1 Classification performance curve of the cyclopak of Line 1, AELRD (July 13 1996 survey)

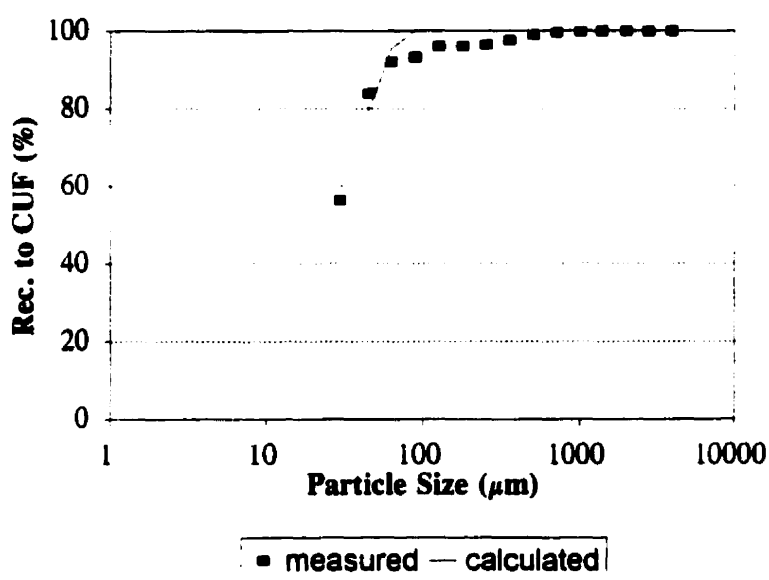


Figure 7.2 Classification performance curve of the cyclopak of Line 2, AELRD (July 13 1996 survey)

be noted that R_f could not be estimated by this method, as there were too few data points below the cut-size.

R_f, 50% and 43% for grinding lines 1 and 2 respectively, is too high and indicates a situation far from optimal conditions. The estimated value of *m* for Line 1 is equal to 1.32, which is very low; showing a poor sharpness of separation. For Line 2, the value of *m* is equal to 2.73, which is considered acceptable. However, there is uncertainty regarding the d_{50c} and m values, because relatively few points are used in the calculation of the sum of square and all are above the cut-size. Further, there is a distinctive hump in both performance curves which is probably associated with the transition from fine sulphides to coarse silicates in the cyclone feed (with increasing particle size).

As a result, it is not certain that classification sharpness needs improvement. Rather, it is the decrease of short-circuiting which is the most important corrective action, by the decreasing of the apex diameter. One would suggest that the underflow density of the individual cyclones be checked and that the apex of the most dilute ones be changed for smaller ones. This should assist in decreasing circulating load significantly, which in turn will benefit residence time in the mills.

A number of consultation sessions were made using the information concerning classification:

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 2
GCOS> What is your classification objective?
I increase cut size, d50
D decrease d50
R reduce water recovery to the cyclone underflow, Rf
S increase separation sharpness, m
U unknown
==> u

GCOS> Have you fitted Plitt's model to the classification data? (y/n)
==> y
GCOS> Have you optimized the fit? (y/n)
==> y
GCOS> Have you obtained a positive Rf? (y/n)
==> n

****

CONCLUSIONS

****

The system reached to the following conclusions:

- It is recommended to use a wider screening size range in next circuit survey so that Rf can be estimated when model fitting is optimized.

==>

- When the fit is optimized, the final values of estimated parameters, Rf, d50c and m must be positive. If not, this can be due to incomplete size distribution information of cyclone streams for fine size classes. To solve this problem, Rf can be calculated from cyclone overflow and underflow solids flow rate and percent solids information. Then, the other two parameters can be estimated using the optimization tool.

==>

The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> y

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 2
GCOS> What is your classification objective?
I increase cut size, d50
D decrease d50
R reduce water recovery to the cyclone underflow, Rf
S increase separation sharpness, m
U unknown
==> r

****

CONCLUSIONS

****

The system reached to the following conclusions:

- Water recovery to the cyclone underflow, Rf, can be reduced by using larger vortex finder diameter. It is recommended to use the BMCS to assess the impact of this change on full circuit performance.

==>

- Water recovery to the cyclone underflow, Rf, can be reduced by using smaller apex diameter. It is recommended to use the BMCS to assess the impact of this change on full circuit performance.

==>

The current consultation session is terminated.

Would you like to start a new session? (y/n)

==> y

GCOS> Please choose one of the following topics?

1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation

==> 2

GCOS> What is your classification objective?

I increase cut size, d50
D decrease d50
R reduce water recovery to the cyclone underflow, Rf
S increase separation sharpness, m
U unknown

==> s

****

CONCLUSIONS

****

The system reached to the following conclusions:

- In case of excessively high feed solids concentration or high slimes concentrations, it is recommended to dilute the feed to reduce the viscosity of the fluid. This can be led to improved separation sharpness.

==>

- The Plitt's separation sharpness can be increased by modifications that decrease water recovery to the cyclone underflow or short circuiting. The separation sharpness can be improved by adding the number of operating hydrocyclones or increasing pressure drop if it is too low.

==>

The current consultation session is terminated.

Would you like to start a new session? (y/n)

==> n

The above examples show some of the conclusions that are offered by CLASSIFICATION module. One of the problems addressed in the knowledge base is how to correctly fit Plitt's model to the measured classification data. For example, having a negative R_f parameter after optimizing the fit is a problem for which the knowledge base can offer a solution. More examples related to testing other sets of rules in this module will be given in Sections 7.2.4 and 7.2.5.

7.2.2 Les Mines Selbaie

The selection function analysis of Les Mines Selbaie case was discussed in Section 4.6.2. Table 7.2 lists some of the important design and operating data of circuit B. The shapes of the selection function vs. particle size curves for the ball mill and tricone mill were given in Figures 4.7 and 4.8.

Table 7.2 Design and operating data of the circuit B (Les Mines Selbaie)

Parameter	Ball mill	Tricone mill
Circuit type	Closed	Closed
Grinding operation mode	Wet	Wet
Discharge mechanism	Overflow	Overflow
Diameter inside liners (m)	≈3.07	≈2.87
Length inside liners (m)	≈3.92	≈3.23
Mill speed (%CS)	76	73
Ball top size (mm)	38	25 (slugs)
Ball material	Steel	Steel
Ore specific gravity	2.9	2.9
Laboratory work index (kWh/t)	15.00	15.00
Operating work index	11.59	12.29
F_{80} (μm)	≈1058	≈153
Installed motor power (kW)	N/A	N/A
Power draw (kW)	≈563	≈354

The following is the example run made by using the information given in Table 7.2 and the selection function estimation results (see Fig. 4.7).

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 1

GCOS> What is your optimization objective?
I increase throughput or grind fineness
D decrease operating costs
U unknown
==> i
GCOS> What shape does the selection function vs. particle size curve have?
L linear
SH straight line with a small hump at coarse end
LH large hump
==> lh
GCOS> What mode of grinding operation is used?
W wet
D dry
==> w
GCOS> What material are balls made of?
S steel
I silica
==> s
GCOS> What make-up ball size is used (in millimeters)?
==> 38
GCOS> Please enter the internal diameter of the mill, measured inside the liner (in meters)?
==> 3.07
GCOS> What type of discharge mechanism does the mill have?
O overflow
D diaphragm
==> o
GCOS> What circuit type is used?
O open
C closed
==> c
GCOS> At what speed is the mill running (in rpm)?
==> 18
GCOS> What is the value of laboratory Bond Work index (in kWh/t)?
==> 15
GCOS> Please enter d80 of the ball mill feed (in microns)?
==> 1058
GCOS> What specific gravity does ore have (in g per cubic cm)?
==> 2.9
GCOS> What percent of the ball mill discharge has a size larger than that of hump?
==> 12
GCOS> Enter the maximum selection function value?
==> 1
GCOS> Enter the top size class selection function?
==> 0.03

****

CONCLUSIONS

****

The system reached to the following conclusions:

- The recommended make-up or top ball size for single size recharge is:

51.44 mm (based on Azzaroni's relationship)
25.25 mm (based on Bond's relationship)

==>
The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> n

Since only 12% (less than 20%) of the ball mill discharge has a size larger than that of hump, using a larger ball size was not recommended by the system, despite the pronounced hump shown in Figure 4.7. The system, however, has given the make-up ball sizes calculated using Azzaroni and Bond relationships. That used in the mill is intermediate, 38 mm.

The following two sections of the example run focus on testing the modelling rules regarding the validity of the ball mill and tricone mill estimated selection functions, respectively.

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 4
GCOS> What task do you want to do?
M modelling
S simulation
==> m
GCOS> What type of study are you doing now?
P preliminary
D detailed
==> p
GCOS> Have you determined parameters of Weller's model that describe residence time distribution (RTD) of solid material flowing through the mill? (y/n)
==> y
GCOS> Is the breakage function of the ore available? (y/n)
==> y
GCOS> Have you estimated the value of selection function for each size class? (y/n)
==> y
GCOS> Does the selection function vs. particle size show a clear trend? (y/n)
==> y
GCOS> Is there noise in the selection function vs. particle size data? (y/n)
==> n

****
CONCLUSIONS
****

The system reached to the following conclusions:

- The estimated selection function seems to be valid. Normally, if selection function vs. particle size curve has a clear trend and there is no significant noise in data particularly at coarse size classes, it indicates a reliable estimated selection function.

==>

The current consultation session is terminated.

Would you like to start a new session? (y/n)

==> n

The estimated selection functions for the tricone mill show increasing trends, but they exhibit a high level of noise at coarse size classes. These two parameters, trend and noise, can help to indicate the validity of the estimated selection function. The below example run shows the system conclusion in such cases.

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 4

GCOS> What task do you want to do?
M modelling
S simulation
==> m

GCOS> What type of study are you doing now?
P preliminary
D detailed
==> p

GCOS> Have you determined parameters of Weller's model that describe residence time distribution (RTD) of solid material flowing through the mill? (y/n)
==> y

GCOS> Is the breakage function of the ore available? (y/n)
==> y

GCOS> Have you estimated the value of selection function for each size class? (y/n)
==> y

GCOS> Does the selection function vs. particle size show a clear trend? (y/n)
==> y

GCOS> Is there noise in the selection function vs. particle size data? (y/n)
==> y

GCOS> Does the noise exist in the top size classes? (y/n)
==> y

```
**********************************************************************
                          CONCLUSIONS
**********************************************************************
```

The system reached to the following conclusions:

- The estimated selection function is valid for fine size classes. However, for the top size classes, the selection function values may be uncertain and erratic due to screening errors, if there is very little mass in top size classes.

==>

The current consultation session is terminated.

Would you like to start a new session? (y/n)

==> n

Table 7.3 Design and operating data of ball mill circuit (Lupin Mine)

Circuit type	Closed
Grinding operation mode	Wet
Discharge mechanism	Overflow
Diameter inside liners (m)	≈2.44
Length inside liners (m)	≈7.32
Mill speed (%CS)	N/A
Ball top size (mm)	51
Ball material	Steel
Ore specific gravity	N/A
Laboratory Work Index (kWh/t)	15.40
Operating work index (kWh/t)	N/A
F_{80} (μm)	≈850
Installed motor power (kW)	N/A
Power drawn (kW)	513

7.2.3 Echo Bay Mine, Lupin

Table 7.3 shows the information about the ball mill circuit at Lupin. In this case, only the selection function estimation results were used in testing. The following consultation session was made using data given in Table 7.3 and the selection function estimation results shown in Figure 4.10.

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 1
GCOS> What is your optimization objective?
I increase throughput or grind fineness
D decrease operating costs
U unknown
==> i
GCOS> What shape does the selection function vs. particle size curve have?
L linear
SH straight line with a small hump at coarse end
LH large hump
==> sh
GCOS> What mode of grinding operation is used?
W wet
D dry
==> w
GCOS> What material are balls made of?
S steel
I silica
==> s
GCOS> What make-up ball size is used (in millimeters)?
==> 51
GCOS> Please enter the internal diameter of the mill, measured inside the liner (in meters)?
==> 2.44
GCOS> What type of discharge mechanism does the mill have?
O overflow
D diaphragm
==> o
GCOS> What circuit type is used?
O open
C closed
==> c
GCOS> At what speed is the mill running (in rpm)?
==> u

****

CONCLUSIONS

****

The system reached to the following conclusions:

- There is no serious need to change the make-up ball size. However, the impact of any change to the make-up ball size can only be indicated by simulation.

==>

The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> n

In the above example run, after a series of questions the system concluded that the ball size does not need to be changed. The query process was terminated as soon as the last question was answered unknown.

7.2.4 Louvicourt Mine

Table 7.4 shows the information about the primary ball mill circuit.

Table 7.4 Design and operating data of the primary ball mill circuit (Louvicourt Mine)

Circuit type	Closed
Grinding operation mode	Wet
Discharge mechanism	Overflow
Diameter inside liners (m)	≃5.00
Length inside liners (m)	≃7.30
Mill speed (%CS)	N/A
Ball top size (mm)	76
Ball material	Steel
Ore specific gravity	N/A
Laboratory work index (kWh/t)	N/A
Operating work index (kWh/t)	N/A
F_{80} (μm)	N/A
Installed motor power (kW)	≃2980
Power draw (kW)	≃2700

The example run in this case demonstrates how the system responded to the classification information of Louvicourt Mine:

GCOS > Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 2

GCOS> What is your classification objective?
I increase cut size, d50
D decrease d50
R reduce water recovery to the cyclone underflow, Rf
S increase separation sharpness, m
U unknown
==> u
GCOS> Have you fitted Plitt's model to the classification data? (y/n)
==> y
GCOS> Have you optimized the fit? (y/n)
==> y
GCOS> Have you obtained a positive Rf? (y/n)
==> y
GCOS> Is the optimized fit satisfactory? (y/n/u)
==> n
GCOS> Does the partition curve have a hump or plateau in intermediate size range? (y/n)
==> y
GCOS> Does the cyclone feed contain significant heavy and light mineral phases? (y/n)
==> y

****

CONCLUSIONS

****

The system reached to the following conclusions:

- Individual mineral classification behaviour is the possible cause of the lack of fit. The Plitt hydrocyclone model can only be used for trending.

==>
The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> n

Besides the individual mineral classification behaviour, the fish hook phenomenon can also cause the unsatisfactory fit of Plitt's model. This problem was observed in none of the studied cases. Nevertheless, there are rules in the knowledge base which address this problem (see Appendix D.2).

The following session shows the system conclusions when the classification performance information for a single mineral, chalcopyrite, is input:

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation

==> 2

GCOS> What is your classification objective?

I increase cut size, d50
D decrease d50
R reduce water recovery to the cyclone underflow, Rf
S increase separation sharpness, m
U unknown

==> u

GCOS> Have you fitted Plitt's model to the classification data? (y/n)

==> y

GCOS> Have you optimized the fit? (y/n)

==> y

GCOS> Have you obtained a positive Rf? (y/n)

==> y

GCOS> Is the optimized fit satisfactory? (y/n/u)

==> y

GCOS> What is the estimated value of the water recovery to the cyclone underflow (in %)?

==> 44

GCOS> What is the estimated value of the separation sharpness?

==> 2.13

****

CONCLUSIONS

****

The system reached to the following conclusions:

- The efficiency of the cyclone operation in terms of the amount of water recovered to the cyclone underflow is poor. It is recommended to reduce water recovery to the cyclone underflow.

==>

- The cyclone separation sharpness is normal.

==>

The current consultation session is terminated.

Would you like to start a new session? (y/n)

==> n

The system further consulted for finding the ways to reduce water recovery to the cyclone underflow:

GCOS> Please choose one of the following topics?

1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation

==> 2

GCOS> What is your classification objective?

I increase cut size, d50
D decrease d50
R reduce water recovery to the cyclone underflow, Rf

S increase separation sharpness, m
U unknown
==> r

****

CONCLUSIONS

****

The system reached to the following conclusions:

- Water recovery to the cyclone underflow, Rf, can be reduced by using larger vortex finder diameter. It is recommended to use the BMCS to assess the impact of this change on full circuit performance.

==>

- Water recovery to the cyclone underflow, Rf, can be reduced by using smaller apex diameter. It is recommended to use the BMCS to assess the impact of this change on full circuit performance.

==>

The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> n

The above example illustrates how the present set of rules may give a choice of course of action. This could be eventually resolved by adding rules. For example, decreasing R_f can be achieved by increasing the vortex finder diameter or decreasing apex diameter. The first action will also decrease circulating load significantly, as will the second to a lesser extent. If the circulating load is already low, neither actions are advisable. Rather, decreasing both apex and vortex finder diameters or adding more water to the grinding circuit would achieve the objective of increased circulating load and decreased R_f.

The above discussion is highly relevant to the Louvicourt Mine case. As recommended by BMCS, smaller balls were added to the ball mill, and resulted in a finer grind. However, when apex diameters were reduced to lower R_f, the drop of circulating load from 280% to 200% caused the grind to coarsen to its original value. Clearly a new set of rules must be enacted to simultaneously adjust R_f and the circulating load to desirable values.

7.2.5 Dome Mine

Dome Mine, one of the Placer Dome Inc. operation sites, is located at Timmins, Ontario. The mill treats approximately 10600 t/d of gold ore, although recently throughput has averaged above 13000 t/d. The comminution process includes crushing and grinding. Grinding consists of circuits A and B. Only B circuit grinding surveys were used for the testing which is presented here.. It must be noted that the Dome Mine is an independent case from those used to develop the knowledge base.

The B circuit has a nominal capacity of 10000 t/d and consists of single stage ball milling and cycloning. A cyclopak of 10 38 cm (15") hydrocyclones is used for sizing of the ball mill discharge. A portion of the cyclopak underflow is bled and then is concentrated by a *K*nelson *C*oncentrator (KC) unit. Two data sets from two surveys performed on September 23 and November 18 of 1997 were supplied for circuit analysis. The November 18 1997 data set, however, was not used in the analysis due to the erratic size distribution data.

The raw data includes the laboratory work index of the ore fed to the circuit, particle size distributions and % solids of samples of various streams. The assessment of grinding kinetics and classification performance was considered in data analysis. In the mass balancing of raw data, the KC unit was ignored due to the relatively small mass of material which is processed.

Table 7.5 lists the information about the circuit B. The GCOS was consulted after the selection function had been estimated and plotted vs. particle size. The raw data were first mass balanced using NORBAL3 [Spring 1992], see Appendix E.3. Then, the balanced data were used to estimate the ball mill selection function and cyclopak classification performance (Appendix E.4). Figure 7.3 shows the selection function vs. particle size curve. The data exhibits a clear and increasing trend between 212 and 1800 μm and then a noisy decreasing trend above 5000 μm.

Table 7.5 Design and operating data of the circuit B (Dome Mine)

Circuit type	Closed
Grinding operation mode	Wet
Discharge mechanism	Overflow
Diameter inside liners (m)	N/A
Length inside liners (m)	N/A
Mill speed (%CS)	N/A
Ball top size (mm)	76
Ball material	Steel
Ore specific gravity	2.8
Laboratory work index (kWh/t)	12.8
Operating work index (kWh/t)	N/A
F_{80} (μm)	N/A
Installed motor power (kW)	N/A
Power draw (kW)	N/A

Figure 7.4 shows the measured and corrected classification performance curves. The classification efficiency indicated by Plitt's model to the measured data shows a poor performance in terms of separation sharpness. The information obtained from selection function estimation and classification performance analysis was used in a consultation session with GCOS as below:

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 1
GCOS> What is your optimization objective?
I increase throughput or grind fineness
D decrease operating costs
U unknown
==> i

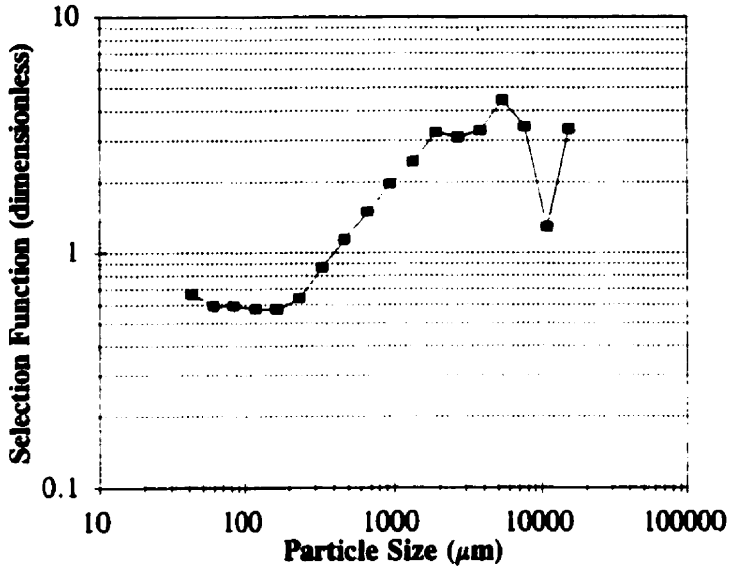


Figure 7.3 Grinding kinetics of the circuit B ball mill, Dome Mine (September 23 1997 survey)

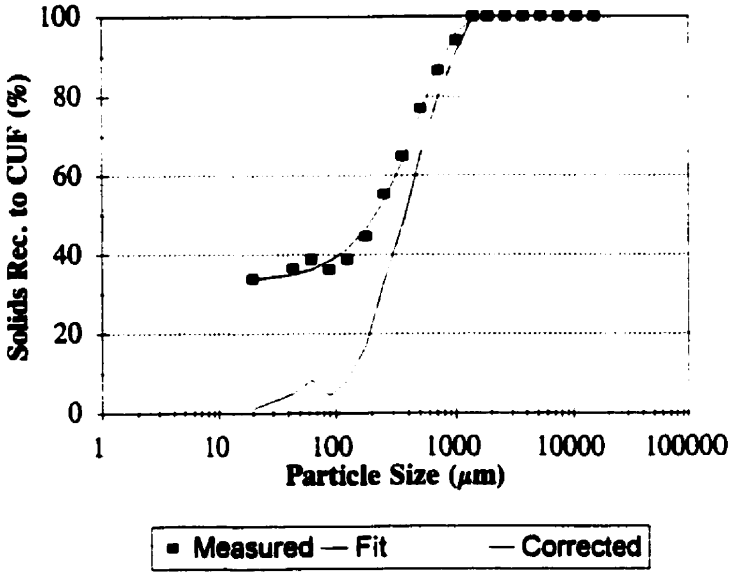


Figure 7.4 Classification performance curves of the circuit B cyclopak, Dome Mine (September 23 1997 survey)

GCOS> What shape does the selection function vs. particle size curve have?
L linear
SH straight line with a small hump at coarse end
LH large hump
==> h

GCOS HELP

Please look at the shape of selection function vs. particle size curve and then choose one.

You can quit any time by typing quit, bye or system. If you do not know the answer just type <unknown> or <u>.

Please press any key to continue ...

GCOS> What shape does the selection function vs. particle size curve have?
L linear
SH straight line with a small hump at coarse end
LH large hump
==> sh
GCOS> What mode of grinding operation is used?
W wet
D dry
==> w
GCOS> What material are balls made of?
S steel
I silica
==> s
GCOS> What make-up ball size is used (in millimeters)?
==> 76
GCOS> Please enter the internal diameter of the mill, measured inside the liner (in meters)?
==> u
GCOS> What type of discharge mechanism does the mill have?
O overflow
D diaphragm
==> o
GCOS> What circuit type is used?
O open
C closed
==> c

****

CONCLUSIONS

****

No conclusion was found.
Please press any key to continue ...

The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> y

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 2
GCOS> What is your classification objective?
I increase cut size, d50
D decrease d50
R reduce water recovery to the cyclone underflow, Rf
S increase separation sharpness, m
U unknown
==> u
GCOS> Have you fitted Plitt's model to the classification data? (y/n)
==> y
GCOS> Have you optimized the fit? (y/n)
==> y
GCOS> Have you obtained a positive Rf? (y/n)
==> y
GCOS> Is the optimized fit satisfactory? (y/n/u)
==> y
GCOS> What is the estimated value of the water recovery to the cyclone underflow (in %)?
==> 33
GCOS> What is the estimated value of the separation sharpness?
==> 1.46
GCOS> Does the partition curve have a hump or plateau in intermediate size range? (y/n)
==> n

****

CONCLUSIONS

****

The system reached to the following conclusions:

- The efficiency of the cyclone operation in terms of the amount of water recovered to the cyclone underflow is reasonable.

==>

- The cyclone separation sharpness is poor.

==>

The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> n

GCOS> Please choose one of the following topics?
1 ball mill
2 hydrocyclone
3 circuit
4 modelling and simulation
==> 4
GCOS> What task do you want to do?
M modelling
S simulation

==> m

GCOS> What type of study are you doing now?

P preliminary

D detailed

==> p

GCOS> Have you determined parameters of Weller's model that describe residence time distribution (RTD) of solid material flowing through the mill? (y/n)

==> n

GCOS> Is the breakage function of the ore available? (y/n)

==> n

****

CONCLUSIONS

****

The system reached to the following conclusions:

- It is recommended to use typical values for Weller's model parameters such as tau PF=0.1, tau SPM=0.1 and tau LPM=0.7 at a standard ball mill feed rate.

==>

- Although the actual RTD parameters are unavailable, it is still possible to estimate the selection function using typical values. For a detailed study, however, you may need to measure RTD.

==>

- Although the actual ore breakage function is unavailable, it is still possible to estimate the selection function using typical ore breakage functions. For a detailed study, however, you may need to measure the breakage function.

==>

- It is recommended to use the breakage function of a similar ore for a preliminary work. For very accurate simulations, you would better off to determine the breakage function of the ore using representative samples of the ore.

==>

The current consultation session is terminated.

Would you like to start a new session? (y/n)

==> y

GCOS> Please choose one of the following topics?

1 ball mill

2 hydrocyclone

3 circuit

4 modelling and simulation

==> 4

GCOS> What task do you want to do?

M modelling

S simulation

==> m

GCOS> What type of study are you doing now?

P preliminary

D detailed

==> d
GCOS> Have you determined parameters of Weller's model that describe residence time distribution (RTD) of solid material flowing through the mill? (y/n)
==> y
GCOS> Is the breakage function of the ore available? (y/n)
==> y
GCOS> Have you estimated the value of selection function for each size class? (y/n)
==> y
GCOS> Does the selection function vs. particle size show a clear trend? (y/n)
==> y
GCOS> Is there noise in the selection function vs. particle size data? (y/n)
==> y
GCOS> Does the noise exist in the top size classes? (y/n)
==> y
GCOS> Which method did you use to estimate the mill selection function?
S sequential interval-by-interval search
F use of functional forms of selection functions
==> s

****

CONCLUSIONS

****

The system reached to the following conclusions:

- The estimated selection function is valid for fine size classes. However, for the top size classes, the selection function values may be uncertain and erratic due to screening error, if there is very little mass in top size classes.

==>

- As this is a detailed study, it is recommended to use other selection function estimation methods as well. For example, (1) use more than one data set (2) methods based on assumed functional forms of selection functions can be used. The best selection function vector then can be found by the analysis of results from various methods.

==>

The current consultation session is terminated.
Would you like to start a new session? (y/n)
==> n

7.3 Summary

The example runs of GCOS using information from a number of real grinding circuit case studies partially demonstrated how it could offer a novice mineral processor conclusions or recommendations to optimize the circuit performance. Nevertheless, if one wants to explore all situations covered by the knowledge base, one needs to present more cases with different parameter values. Therefore, a diverse base of test cases must

be used to check the knowledge base performance. While this has been done during the system development, it was not fully presented in this chapter for briefness.

The example runs demonstrate how the system queries the user to collect the relevant information to reach conclusions. When in doubt on how to answer a specific question, the user can access the context sensitive help which has been designed as an integrated part of the query module. The rule that triggers the help function is fired whenever the user answers a question by typing help or <h>. The help string corresponding to the question asked is provided by the help slot of the related question fact.

The example runs presented in this chapter demonstrate the functionality of some of the rules encoded in GCOS. The correct firing of rules that cause the execution of external programs such as NGOTC (*N*umerical *G*rinding *O*ptimization *T*ools in *C*) and BMCS (*B*all *M*illing *C*ircuits *S*imulator) for parametric estimation and simulation, has also been tested and verified. Although the current knowledge base was fully verified and validated under the limited scope of the thesis, it can be deepened and broadened in future work to address domain problems for more details and to a larger extent.

CHAPTER 8
SUMMARY AND CONCLUSIONS

8.1 Summary

Mineral processors have to conduct off-line optimization studies on a regular basis in order to keep the plant performance as high as possible. There are many design and operating parameters which must be considered in circuit optimization such as the process flowsheet, the grinding medium size and the fresh feed flow rate. It is well recognized that off-line optimization is a complex task which includes various stages such as plant sampling, mass balancing of raw data, parametric estimation and simulation. In this thesis the theoretical aspects and industrial applications of only some of these tasks such as the selection function parameter estimation and its prediction based on the ball size, classification performance analysis and circuit simulations were considered and discussed in detail. The other aspects such as sampling and mass balancing theories were overlooked due to the limited scope of the thesis.

In most off-line optimization cases, a number of computer programs such as those developed for mass balancing, parametric estimation and simulation must be used. In order to complete an optimization study, one inevitably has to be knowledgeable about the various optimization methodologies and required tools. In order to assist a mineral processor in the optimization process, a knowledge-based system was developed which provides expertise extracted from experienced people and resources.

The thesis was mainly focused on the following areas:

1. developing a rule, frame based system to supervise grinding circuit optimization studies
2. developing a tool to scale mill selection functions given the mill selection function vector estimated for the current ball size
3. developing a sequential-modular program to simulate ball milling circuits.
4. integration of various numerical programs via a knowledge-based system approach

The use of an integrated representation of rules and frames for the circuit simulation was demonstrated. The frames were used to represent circuit objects which are more consistent with real grinding circuits. There are tasks that must be done before and after any simulation trial that can be computerized through the rule-based programming. It must be stressed that testing is an integral part of developing computer programs. The plant work was required for both verification and validation of the software. The real optimization studies served various purposes such as knowledge acquisition, verification and validation.

8.2 Conclusions

The following conclusions can be drawn from this research work:

1. While not one of the direct objectives of this thesis, the use of simulation approach in off-line grinding optimization was shown to be a very useful approach to investigate alternative modifications to a plant practice.
2. The case studies showed that successful modelling and simulation depends on how knowledgeable the user is in the field and how skilful he/she is in using these tools.
3. Rule based system programming can be effectively used to guide the optimization processes.

4. Frames or objects are very useful representation tools to develop an optimization knowledge-base.
5. Integration of rules and frames and numerical procedures can provide a single computational environment.
6. The calibration of a simulator is a very critical step in optimization process, which requires a lot of effort. Perhaps due to the difficulty of the calibration procedure, simulators are not being used on a regular basis by mineral processors to optimize industrial circuits.
7. GCOS (*G*rinding *C*ircuits *O*ptimization *S*upervisor) is an interpretation, diagnostic and optimization rule and frame based system that infers critical facts. The modular knowledge base provides ball milling, classification, modelling and simulation. Each module uses simple facts and template facts to represent knowledge about important parameters or attributes that define the state of design and operating conditions. In the simulation module, an object-oriented approach was taken to represent process objects by frames or objects. The simulation is done using messages sent by objects via rules that govern the simulation process.
8. The contribution of GCOS to grinding simulation lies in its ability to accommodate optimization knowledge in a very effective way using facts, objects and rules. In other words, many rules in the knowledge base are heuristic used by mineral processors to evaluate circuit operation and to make changes to operation to meet a specific goal. This type of program can remove difficulties that have prevented extensive use of grinding simulators by plant metallurgists. Previously such tools have been used only by developers and researchers.
9. The rules do different tasks such as interpretation, diagnosis and optimization.

8.3 Future Work

The knowledge base developed here can be implemented more effectively using more powerful commercial shells that provide knowledge acquisition tools and graphical user interfaces. Some of the conclusions or recommendations offered by the current knowledge base are very general which can be made more specific or accurate by defining new parameters and rule sets. Also, the other aspects of off-line grinding circuits optimization such as plant sampling and mass balancing must be eventually covered by future knowledge-based systems.

Most benefits of the knowledge-based systems however will be realised when they are fully integrated into the conventional grinding simulation software packages. To accomplish this task, much work remains to be done -- mainly the formalization of engineering knowledge so that it can be readily programmed into the computer software. Rule-based formalization of various mineral processing knowledge such as grinding and flotation is a required step to develop successful (commercial) software.

REFERENCES

[Austin, Shoji and Luckie 1976] Austin, L.G., Shoji, K. and Luckie, P.T. 1976. The Effect of Ball Size on Mill Performance. *Powder Technology*. Vol. 14: 71-79.

[Austin, Luckie and Wightman 1975] Austin, L.G., Luckie, P.T. and Wightman, D. 1975. Steady-State Simulation of A Cement-Milling Circuit. *Intl. Jour. of Miner. Process*. Vol. 2: 127-150.

[Austin, Luckie and Ateya 1971] Austin, L.G., Luckie, P.T. and Ateya, B.G. 1971. Residence Time Distribution in Mills. *Cement and Concrete Research*. Vol. 1: 241-255.

[Austin and Weller 1982] Austin, L.G. and Weller, K.R., 1982. Simulation and Scale-Up of Wet Ball Milling. *Proc. of 14th Intl. Miner. Process. Cong.* CIM. Toronto, Canada, Oct. 17-23: I-8.1 - I-8.24.

[Austin 1971] Austin, L.G. 1971. A Review Introduction to the Mathematical Description of Grinding as a Rate Process. *Powder Technology*. Vol. 5: 1-17.

[Austin, Klimpel and Luckie 1984] Austin, L.G., Klimpel, R.R. and Luckie, P.T. 1984. *Process Engineering of Size Reduction: Ball Milling*. New York, New York: SME of the AIME, Inc.

[Austin 1973] Austin, L.G. 1973. Understanding Ball Mill Sizing. *Ind. Eng. Chem. Process Des. Develop*. Vol. 12, No. 2: 121-129.

[Austin and Klimpel 1985] Austin, L.G. and Klimpel, R.R. 1985. Ball Wear and Ball Size Distributions in Tumbling Ball Mills. *Powder Technology*. Vol. 41: 279-286.

[Azzaroni 1981] Azzaroni, E. 1981. Grinding Media Size and Multiple Recharge Practice. *Proc. of Second Asian Symposium on Grinding*. Manila, Philippines.

[Balachandran 1993] Balachandran, M. 1993. *Knowledge-Based Optimum Design*. Topics in Engineering, Ed. C.A. Brebbia and J.J. Connor Vol. 10. Southampton, UK: Computational Mechanics Pub.

[Banisi, Laplante and Marois 1991] Banisi, S., Laplante, A.R. and Marois, J. 1991.

The Behaviour of Gold in Hemlo Mines Ltd. Grinding Circuit. *Proc. of 23rd Ann. Meeting of Canadian Mineral Processors*. Ottawa, Canada. Jan. 22-24.

[Barr and Feigenbaum 1981] Barr, A. and Feigenbaum, E.A. 1981. *The Handbook of Artificial Intelligence*. Los Altos, CA: William Kaufmann, Inc.

[Bazin and Hodouin 1988] Bazin, C. and Hodouin, D. 1988. Residence Time Distribution Modelling for Mineral Processes Evaluation. *CIM Bull.* Vol. 81, No. 911: 115-125.

[Beale, Prisbrey and Demuth 1994] Beale, M., Prisbrey, K. and Demuth, H. 1994. Comminution Control Example Using the Fuzzy Systems and Neural Network Toolboxes for MATLAB and SIMULINK. *Proc. of SME Annual Meeting*. Albuquerque, New Mexico.

[Bearman and Milne 1992] Bearman, R.A. and Milne, R.W. 1992. Expert Systems: Opportunities in the Minerals Industry. *Minerals Engineering*. Vol. 5, Nos. 10/12: 1307-1323.

[Bond 1958] Bond, F.C. 1958. Grinding Ball Size Selection. *Mining Engineering, Trans. AIME*. Vol. 10. (May): 592-595.

[Bond 1952] Bond, F.C. 1952. The Third Theory of Comminution. *Mining Engineering, Trans. AIME*. Vol. 193. (May): 484-494.

[Bradford 1991] Bradford, S.H. 1991. An Expert System for Control of a SAG/BALL Mill Circuit. *Proc. of IFAC on Expert Systems in Mineral and Metal Processing*. Espoo, Finland: 1-6.

[Broadbent and Callcott 1956] Broadbent, S.R., Callcott, T.G. 1956. Coal Breakage Processes: I. A New Analysis of Coal Breakage Processes. *Jour. of the Inst. of Fuel*. 29, (Dec.): 524-528.

[Charlesworth 1998] Charlesworth, K. 1998. METSIM: A Computer Program for Complex Chemical, Metallurgical and Environmental Processes. in *http://www.midcoast.com.au/~charles/index.htm*.

[Cilliers and King 1987] Cilliers, J.J. and King, R.P. 1987. Steady-State Simulation and Microcomputers: Meeting A Challenge. *Proc. 20th APCOM*. 2 SAIMM, Johannesburg: 63-70.

[Concha, Magne and Austin 1992] Concha, F., Magne, L. and Austin, L.G. 1992. Optimization of the Make-Up Ball Charge in A Grinding Mill. *Intl. Jour. of Miner.*

Process. Vol. 34: 231-241.

[Cooper, Bazin and Grant 1994] Cooper, M., Bazin, C., Grant, R. 1994. Grinding Media Evaluation at Brunswick Mining. *CIM Bull.* Vol. 87, No. 985 (Nov.-Dec.): 73-75.

[Cooper et al. 1993] Cooper, M., Bazin, C., Grant, R. and Tessier, R. 1993. Grinding Media Evaluation at Brunswick Mining. *Proc. of 25th Ann. Meeting of the Canadian Mineral Processors.* A Div. of CIM, Ottawa, Jan. 19-21.

[Cox 1972] Cox, M.G. 1972. The Numerical Evaluation of B-Splines. *Jour. Inst. Maths Applics.* Vol. 10: 134-149.

[Coyne et al. 1990] Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M. and Gero, J.S. 1990. *Knowledge-Based Design Systems.* Reading, Massachusetts: Addison-Wesley Publishing Company.

[Dahlstrom and Kam 1988] Dahlstrom, D.A. and Kam, W.-P. 1988. Potential Energy Savings in Comminution by Two-Stage Classification. *Intl. Jour. of Miner. Process.* Vol. 22: 239-250.

[Dahlstrom and Kam 1992] Dahlstrom, D.A. and Kam, W.-P. 1992. Classification for Energy's Sake. In *Comminution-Theory and Practice.* Ed. S.K. Kawatra. 249-260. Ann Arbor, MI: SME, Inc.

[De Boor 1972] De Boor C. 1972. On Calculating with B-Splines. *Jour. of Approximation Theory.* Vol. 6: 50-61.

[del Villar et al. 1985] del Villar, R., Finch, J.A., Laplante, A.R. and Nesset, J.E. 1985. Computer Simulation of A Brunswick Mining Grinding Circuit. *Extr. and Process. Metall. in Complex Sulphides.* AIME & CIM San Diego Nov. 10-13. 1985. 317-331.

[Del Villar and Laplante 1985] Del Villar, R. and Laplante, A.R. 1985. Grinding Simulation in Applesoft Basic. *CIM Bull.* Vol. 78, No. 883 (Nov.): 62-65.

[del Villar 1985] del Villar, R. 1985. *Modelling and Simulation of Brunswick Mining Grinding Circuit.* Ph.D. Thesis. McGill University, Montréal.

[Dierckx 1982] Dierckx, P. 1982. A Fast Algorithm for Smoothing Data on A Rectangular Grid While using Spline Functions. *SIAM Jour. Numer. Anal.* Vol. 19, No.6 (Dec.): 1286-1304.

[Donnell 1994] Donnell, B.L. 1994. Objects/Rule Integration in CLIPS. *Expert Systems*. Vol. 11, No. 1: 29-45.

[Dunn 1989] Dunn, D.J. 1989. Design of Grinding Balls. *Mining Engineering*. (Sept.): 951-955.

[Durance et al. 1993] Durance, M.-V., Guillaneau, J.-C., Villeneuve, J., Fourniguet, G. and Brochot, S. 1993. Computer Simulation of Mineral and Hydrometallurgical Processes: USIM PAC 2, A Single Software from Design to Optimization. *Proc. of Intl. Symp. on Modelling, Simulation and Control of Hydrometallurgical Processes*. Ed. V.G. Papangelakis and G.P. Demopoulos. Montreal, Québec, Canada. Aug. 24 to Sept. 24: 109-121.

[Durkin 1994] Durkin, J. 1994. *Expert Systems: Design and Development*. New York, New York: Macmillan Publishing Company.

[Dym and Levitt 1991] Dym, C.L. and Levitt, R.E. 1991. Toward the Integration of Knowledge for Engineering Modelling and Computation. *Engineering with Computers*. 7: 209-224.

[Dym 1985] Dym, C.L. 1985. Expert Systems: New Approaches to Computer-Aided Engineering. *Engineering with Computers*. 1: 9-25.

[Eggert and Benford 1994] Eggert, J. and Benford, P. 1994. Development and Implementation of An Expert System for Grinding Control at the Dome Mine. *Proc. of 26th Ann. Meeting of the Canadian Mineral Processors*. Ottawa: CIM, Jan.: paper No. 18.

[Epstein 1947] Epstein, B. 1947. The Mathematical Description of Certain Breakage Mechanisms Leading to the Logarithmico-Normal Distribution. *Jour. of Franklin Inst.* Vol. 244: 471-477.

[Epstein 1948] Epstein, B. 1948. Logaritmico-Normal Distribution in Breakage of Solids. *Industrial and Engineering Chemistry*. Vol. 40, No. 12 (Dec.): 2289-2291.

[Farzanegan, Laplante and Lowther 1995] Farzanegan, A., Laplante, A.R. and Lowther, D.A. 1995. Application of Knowledge-Based Systems in Optimization of Mineral Grinding Circuits. *Proc. of CAMI'95: 3rd Canadian Conference on Computer Applications in the Mineral Industry*. Montreal, Québec, Canada, Oct. 22-25: 725-732.

[Farzanegan, Laplante and Lowther 1997] Farzanegan, A., Laplante, A.R. and Lowther, D.A. 1997. A Knowledge-Based System for Off-Line Optimization of Mineral Grinding Circuits. *Proc. of 29th of CMP*. Ottawa, Ontario, Canada. Jan. 21-23, paper No. 13.

[Farzanegan and Laplante 1997b] Farzanegan, A. and Laplante, R.A. 1997. *Audit of Louvicourt Mill's Grinding Circuits: Individual Mineral Classification Performance Curves*. A report to Louvicourt Mine. (25 June): 1-4.

[Farzanegan and Laplante 1997a] Farzanegan, A. and Laplante, R.A. 1997. *Audit of Louvicourt Mill's Grinding Circuits: Evaluation and Simulations of Ball Mill and Primary Cyclopak Performance*. A report to Louvicourt Mine. (9 May): 1-11.

[Fikes and Kehler 1985] Fikes, R. and Kehler, T. 1985. The Role of Frame-Based Representation in Reasoning. *Communications of ACM*. Vol. 28, No. 9. (Sept.): 904-20.

[Finch and Ramirez-Castro 1981] Finch, J.A. and Ramirez-Castro, J. 1981. Modelling Mineral Size Reduction in the Closed-Circuit Ball Mill at the Pine Point Mines Concentrator. *Intl. Jour. of Miner. Processing*. Vol. 8: 61-78.

[Finch and Matwijenko 1977] Finch, J.A. and Matwijenko, O. 1977. Individual Mineral Behaviour in A Closed Grinding Circuit. *CIM Bull*. Vol. 70, No. 878: 164-172.

[Finch 1983] Finch, J.A. 1983. Modelling A Fish-Hook in Hydrocyclone Selectivity Curves. *Powder Technology*. Vol. 36: 127-129.

[Flintoff, Plitt and Turak 1987] Flintoff, B.C., Plitt, L.R. and Turak, A.A. 1987. Cyclone Modelling: A Review of Present Technology. *CIM Bull*. Vol. 80, No. 905. (Sept.): 39-50.

[Ford and King 1984] Ford, M.A. and King, R.P. 1984. The Simulation of Ore-Dressing Plants. *Intl. Jour. of Miner. Process*. Vol. 12: 285-304.

[Gaudin and Meloy 1962] Gaudin, A.M. and Meloy, T.P. 1962. Model and A Comminution Distribution Equation for Repeated Fracture. *Tran. SME*. 223 (Mar.): 43-50.

[Genesereth and Nilsson 1987] Genesereth, M.R. and Nilsson, N.J. 1987. *Logical Foundation of Artificial Intelligence*. Palo Alto, CA: Morgan Kaufmann Publishers, Inc.

[Giarratano and Riley 1989] Giarratano, J. and Riley, G. 1989. *Expert Systems: Principles and Programming*. Boston, Massachusetts: PWS-KENT Publishing Company.

[Gonzalez and Dankel 1993] Gonzalez, A.J. and Dankel, D.D. 1993. *The Engineering of Knowledge-Based Systems: Theory and Practice*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

[Harris and Meech 1990] Harris, C.A. and Meech, J.A. 1990. Improving Productivity: Justification for Use of Expert Systems in the Process Industries. *Mineral Comminution Systems, Professional Development Seminars*. Montreal, PQ: McGill Univ., Dept. of Min. and Metall. Eng. (MAY): 1-11.

[Harris and Meech 1987] Harris, C.A. and Meech, J.A. 1987. Fuzzy Logic: A Potential Control Technique for Mineral Processing. *CIM Bull.* Vol. 80, No. 905. (Sept.): 51-59.

[Hartley et al. 1983] Hartley, D.G., Hayward, P.C., Sterns, U.J. and Weller, K.R. The Use of Mathematical Models of Grinding and Classification to Optimize Grinding Circuits at the Mt. Lyell Copper Concentrator, Tasmania. *Mining Engineering*. (Jan.): 48-55.

[Hayes-Roth, Waterman and Lenat 1983] Hayes-Roth, F., Waterman, D.A. and Lenat, D.B. eds. 1983. *Building Expert Systems*. Teknowledge Series in Knowledge Engineering. Reading, Massachusetts: Addison-Wesley Publishing Company.

[Hayes-Roth 1985] Hayes-Roth, F. 1985. Rule-Based Systems. *Communications of ACM*. Vol. 28, No. 9. (Sept.): 921-32.

[Herbst and Lo 1989] Herbst, J. and Lo, Y.C. 1989. Grinding Efficiency with Balls or Cones as Media. *Intl. Jour. of Miner. Process*. Vol. 26: 141-151.

[Herbst and Mular 1979] Herbst, J.A. and Mular A.L. 1979. Modelling and Simulation of Mineral Processing Unit Operations. *Computer Methods for the 80's*. 823-836f.

[Herbst et al. 1995] Herbst, J.A. and Pate, W.T., Flores, R.T. and Zarate, H.A. 1995. Plantwide Control: The Next Step in Mineral Processing Plant Optimization. In *Comminution and Simulation and Control, Volume 1: Proc. of the XIX IMPC*. 211-215. Littleton, Colorado: SME.

[Herbst et al. 1989] Herbst, J.A., Hales, L.B., Pate, W.T. and Sepulveda, J.E. 1989. Report on Actual Benefits Arising from the Application of Expert Control Systems in Industrial Semi-Autogenous Grinding Circuits. In *Proc. of 6th IFAC on Automation in Mining, Mineral and Metal Processing*. Buenos Aires, Argentina. Sept. 4-8: 53-60.

[Herbst, Schena and Fu 1989] Herbst, J.A., Schena, G.D. and Fu, L.S. 1989. Incorporating State of the Art Models into A Mineral Processing Plant Simulator. *Trans. IMM, Sec. C*. Vol. 98: C1-C11.

[Herbst, Grandy and Fuerstenau 1973] Herbst, J.A., Grandy, G.A. and Fuerstenau,

D.W. 1973. Population Balance Models for the Design of Continuous Grinding Mills. *Proc. of 10th Intl. Miner. Process. Cong.* London, UK. (Apr.): 23-45.

[Herbst and Fuerstenau 1980] Herbst, J.A. and Fuerstenau, D.W. 1980. Scale-Up Procedure for Continuous Grinding Mill Design Using Population Balance Models. *Intl. Jour. of Miner. Process.* Vol. 7: 1-31.

[Herbst and Fuerstenau 1968] Herbst, J.A. and Fuerstenau, D.W. 1968. The Zero Order Production of Fine Sizes in Comminution and Its Implications in Simulation. *Trans. SME, AIME.* Vol. 241: 538-548.

[Herbst, Alba, Pate and Oblad 1988] Herbst, J.A., Alba J. F., Pate, W.T. and Oblad, A.E. 1988. Optimal Control of Comminution Operations. *Intl. Jour. of Miner. Process.* Vol. 22: 275-296.

[Hodouin, McMullen and Everell 1980] Hodouin, D., McMullen, J. and Everell, M.D. 1980. Mathematical Simulation of the Operation of a Three-Stage Grinding Circuit for a Fine Grained Zn/Pb/Cu Ore. *Proc. of European Symp. on Particle Technology.* Amsterdam, June, 1980.

[Hodouin, Bérubé and Everell 1978] Hodouin, D., Bérubé, M.A. and Everell, M.D. 1978. Modelling Industrial Grinding Circuits and Application in Design. *CIM Bull.* Vol. 71, No. 797. (Sept.): 138-146.

[Inoue and Okaya 1993] Inoue, T. and Okaya, K. 1993. Factors Affecting the Performance of the Fuzzy Control System for Mineral Processing - A Simulation Study. *Proc. of XVIII Intl. Miner. Process. Cong.* Sydney, (May): 335-341.

[Kelly and Spottiswood 1990] Kelly, E.G. and Spottiswood, D.J. 1990. The Breakage Function; What Is It Really? *Minerals Engineering.* Vol. 3, No. 5: 405-414.

[Kelly and Spottiswood 1982] Kelly, E.G. and Spottiswood, D.J. 1982. *Introduction to Mineral Processing.* New York: John Wiley and Sons.

[Klimpel and Austin 1970] Klimpel, R.R. and Austin, L.G. 1970. Determination of Selection for Breakage Functions in the Batch Grinding Equation by Nonlinear Optimization. *Ind. Eng. Chem. Fundam.* Vol. 9, No. 2: 230-37.

[Klimpel and Austin 1984] Klimpel, R.R. and Austin, L.G. 1984. The Back-Calculation of Specific Rates of Breakage from Continuous Mill Data. *Powder Technology.* Vol. 38: 77-91.

[Lacombe 1995] Lacombe, P. 1995. *Échantillonnage du circuit de broyage le 10*

October 1995. A report to Louvicourt Mine. Nov. 20, 1995.

[Lafontaine 1996] Lafontaine, M. 1996. *Rebroyeur de Zinc*. Report to Louvicourt Mine, June 27th.

[Laguitton et al. 1984] Laguitton, D., Flament, F., Spring, R., Gupta, V.K. and Hodouin, D. Documented Fortran Programs for Ball-Mill Modelling. *CIM Bull*. Vol. 77, No. 870. (Oct.): 71-77.

[Laguitton 1982] Laguitton, D. 1982. Methodology Transfer for the Simulation of Mineral and Coal Processing plants. *CIM Bull*. Vol. 75, No. 840. (Apr.): 166-170.

[Laguitton et al. 1985] Laguitton, D., leung, J, Gupta, V.K., Hodouin, D. and Spring, R. 1985 FINDBS-Program for Breakage and Selection Functions Determination in the Kinetic Model of Ball Mills. *The SPOC Manual*. Ed. D. Laguitton.

[Laguitton 1985] Laguitton, D. Ed. *The SPOC Manual* Canmet, Canada (1985).

[Laplante et al. 1995] Laplante, R.A., Woodcock, F. and Noaparast, M. 1995. Predicting Gravity Separation Gold Recoveries. *Miner. and Metall. Process*. (MAY): 74-79.

[Laplante and Finch 1984] Laplante, A.R., Finch, J.A. 1984. The Origin of Unusual Cyclone Performance Curves. *Intl. Jour. of Miner. Process*. Vol. 13: 1-11.

[Laplante 1996] Laplante, A.R. 1996. Course 306-442a: Modelling and Control of Mineral Processing Systems. *McGill Course Pack Development Service*.

[Laplante, Finch and del Villar 1987] Laplante, A.R., Finch, J.A., and del Villar, R. 1987. Simplification of the Grinding Equation for Plant Simulation. *Trans. IMM, Sec. C*. 96: 108-112.

[Laplante and Redstone 1984] Laplante, R.A. and Redstone, J. 1984. Modelling of Grinding Kinetics at the Sidbec-Normines Port-Cartier Pelletizing Plant. *Miner. and Metall. Process*. (Aug.): 143-149.

[Leiviskä 1991] Leiviskä, K. 1991. Application Viewpoints of Expert Systems in Mineral and Metal Processing. *Proc. of IFAC on Expert Systems in Mineral and Metal Processing*. Espoo, Finland: 191-195.

[Levin 1992] Levin, J. 1992. Indicators of Grindability and Grinding Efficiency. *Jour. S. Afr. Min. Metall*. Vol. 92, No. 10 (Oct.): 283-290.

[Lowrison 1974] Lowrison, G.C. 1974. *Crushing and Grinding: The Size Reduction of Solid Materials*. Ohio: CRC Press, INC.

[Lynch et al. 1977] Lynch, A.J. 1977. *Mineral Crushing and Grinding Circuits: Their Simulation, Optimisation, Design and Control*. Amsterdam, The Netherlands: Elsevier Scientific Publishing Company. 105-121.

[Lynch and Rao 1975] Lynch, A.J. and Rao, T.C. 1975. Modelling and Scale-Up of Hydrocyclone Classifiers. *Proc. of 11th Intl. Miner. Process. Cong.* Cagliari, Italy: Universita di Cagliari. (Apr.): 20-26.

[MacPhail, Racine and Cousins 1997] MacPhail, R.S., Racine, L. and Cousins, B.G. 1997. Improved Zinc Flotation Kinetics at the Louvicourt Mill Using FLEX 31. *CIM Bull.* Vol. 90, No. 1014. (Oct.): 81-84.

[Marchand, Hodouin and Everell 1980] Marchand, J.C., Hodouin, D. and Everell, M.D. 1980. Residence Time Distribution and Mass Transport Characteristics of Large Industrial Grinding Mills. *Automation in Mining, Mineral and Metal Processing,* Proc. of IFAC Symp. Edited by J. O'Shea and M. Polis. Montreal, Canada, Aug. 18-20: 295-302.

[McDermott et al. 1992] McDermott, K., Cleyle, P., Hall, M. and Harris, C.A. 1992. Expert System for Control of No.4 Autogenous Mill Circuit at Wabush Mines. *Proc. of Ann. Meeting of Canadian Mineral Processors.* Ottawa, Jan.

[McIvor et al. 1990] McIvor, R.E., Lavallee, M.L., Wood, K.R., Blythe, P.M. and Finch, J.A. 1990. Functional Performance Characteristics of Ball Milling. *Mining Engineering.* (Mar.): 269-276.

[McIvor 1997] McIvor, R.E. 1997. Effect of Media Sizing on Ball Milling Efficiency. In *Comminution Practices, Volume 1.* Ed. S.K. Kawatra. 279-292. Littleton, CO: SME.

[Meech 1992] Meech, J.A. 1992. Managing Uncertainty in Expert Systems: A Fuzzy Logic Approach. in *Artificial Intelligence in Materials Processing.* Ed. S.A. Argyropoulos and G. Carayannis. Proc. of the intl. Symp. on Artificial Intelligence in Materials Process. 77-85. Edmonton, Alberta, Aug. 23-27.

[Meech 1990] Meech, J.A. 1990. Expert Systems for Teaching and Training in the Mineral Industry. *Minerals Engineering.* Vol. 3, No. 1/2: 129-136.

[Minsky 1975] Minsky, M. 1975. A Framework for Representing Knowledge. *The Psychology of Computer Vision.* Ed. Patrick H. Winston. New York: McGraw-Hill

Book Company.

[Morrell 1990] Morrell, S. 1990. *Effect of Ball Size on Ball Mill Breakage Rates.* Rept., Queensland, Australia: Julius Kruttschnitt Mineral Research Center (JKMRC).

[Morrell and Man 1997] Morrell S. and Man Y.T. 1997. Using Modelling and Simulation for the Design of Full Scale Ball Mill Circuits. *Minerals Engineering.* Vol. 10, No. 12: 1311-1327.

[Motard, Shacham and Rosen 1975] Motard, R.L., Shacham, M. and Rosen, E.M. 1975. Steady-State Chemical Process Simulation. *AIChE Jour.* Vol. 21, No. 3 (May): 417-36.

[Mular 1972] Mular, A.L. 1972. Empirical Modelling and Optimization of Mineral Processes. *Mineral Sci. Engng.* Vol. 4, No. 3: 30-42.

[Napier-Munn et al. 1996] Napier-Munn, T.J., Morrell, S., Morrison, R.D. and Kojovic, T. *Mineral Comminution Circuits: Their Operation and Optimization.* Queensland, Australia: Julius Kruttschnitt Mineral Research Centre, The University of Queensland.

[Napier-Munn and Lynch 1992] Napier-Munn, T.J. and Lynch A.J. 1992. The Modelling and Computer Simulation of Mineral Treatment Processes - Current Status and Future Trends. *Minerals Engineering.* Vol. 5, No. 2: 143-167.

[Perry and Hall 1994] Perry, R. and Hall, M. 1994. Les Mines Selbaie: Semi-Autogenous Grinding Circuit Expert System. *Proc. of 26^{h} Ann. Meeting of the Canadian Mineral Processors.* Ottawa: CIM, Jan. Paper No. 19.

[Plitt, Finch and Flintoff 1980] Plitt, L.R., Finch, J.A. and Flintoff, B.C. 1980. Modelling the Hydrocyclone Classifier. *Proc. of European Symp. Particle Technology.* Amsterdam, Netherlands: 790-304.

[Plitt 1976] Plitt, L.R. 1976. A Mathematical Model of the Hydrocyclone Classifier. *CIM Bull.* Vol. 69, No. 776 (Dec.): 114-23.

[Plitt, Flintoff and Neale 1986] Plitt, L.R., Flintoff, B.C. and Neale, A.J. 1986. Hydro Cyclone Operations under Unusual Conditions. *Proc. of 18th Ann. Meeting of the Canadian Mineral Processors.* Ottawa, Canada. Jan. 21-23: 549-570.

[Plitt, Flintoff and Stuffco 1987] Plitt, L.R., Flintoff, B.C. and Stuffco, T.J. 1987. Roping in Hydrocyclones. *Proc. of 3rd Intl. Conf. on Hydrocyclones.* Oxford, England. Sept. 30 - Oct. 2. 21-33.

[Plitt, Conil and Broussaud 1990] Plitt, L.R., Conil, P. and Broussaud, A. 1990. Technical Note: An Improved Method of Calculating the Water-Split in Hydrocyclones. *Minerals Engineering*. Vol. 3, No. 5: 533-535.

[Rajamani and Herbst 1984] Rajamani, K. and Herbst, J.A. 1984. Simultaneous Estimation of Selection and Breakage Functions from Batch and Continuous Grinding Data. *Trans. Instn Min. Metall. (Sect. C: Mineral Processing. Extr. Metall.)*: C74-C85.

[Reuter and van Deventer 1992] Reuter, M.A. and van Deventer, J.S.J. 1992. The Simulation and Identification of Flotation Processes by Use of a Knowledge Based Model. *Intl. Jour. of Miner. Process.* Vol. 35: 13-49.

[Richardson, Coles and White 1981] Richardson, J.M., Coles, D.R. and White, J.W. 1981. FLEXMET: A Computer-Aided and Flexible Metallurgical Technique for Steady-State Flowsheet Analysis. *E&MJ* (Oct.): 88-97.

[Rowland 1973] Rowland, C.A. 1973. Comparison of Work Indices Calculated from Operating Data with Those from Laboratory Test Data. *Proc. of 10th Intl. Miner. Process. Cong.* London, UK. (Apr.): 47-61.

[Rychener 1988] Rychener, M.D. 1988. *Expert Systems for Engineering Design.* Academic Press, Inc. San Diego, CA.

[Samskog et al. 1995] Samskog, P.O., Björkman, J, Söderman, P., Broussaud A. and Guyot, O. 1995. Model-Based and Expert Supervisory Control at Kiruna LKAB Concentrators - Sweden. *in Comminution and Simulation and Control: Proc. of the XIX IMPC, Vol. 1*: 211-15.

[Sastry and Sudhir 1995] Sastry, Kal V.S. and Sudhir, G.S. 1995. Software Modelling of Grinding Circuits. *Mining Engineering*. Sept.: 835-837.

[Sedlatschek and Bass 1953] Sedlatschek, K. and Bass, L. Contribution to the Theory of Milling Processes. *Powder Metallurgy Bulletin*. Vol. 6, No. 5: 148-153.

[Shortliffe and Buchanan 1975] Shortliffe, E.H. and Buchanan, B.G. 1975. A Model of Inexact Reasoning in Medicine. *Mathematical Biosciences*. 23: 351-379.

[Spring, Larsen and Mular 1985] Spring, R., Larsen, C. and Mular, A. 1985. SPOC Manual: Chapter 4.1: Industrial Ball Mill Modelling: Documented Application of the Kinetic Model. *Report SP85-1/4.1E, CANMET, Energy, Mines and Resources Canada.*

[Spring 1992] Spring, R. 1992. *NORBAL3: Software for Material Balance Reconciliation*. Centre de Recherche Noranda, Point-Claire, Québec.

[Spring and Edwards 1989] Spring, R. and Edwards, R. 1989. Real-Time Expert System Control of the Brenda Mines Grinding Circuit. *Proc. of AIME Meeting*. Las Vegas, Jan.

[Staples, Cooper and Grant 1997] Staples, P., Cooper, M. and Grant, R. Evaluation of Grinding Media Shape and Size in A Pilot Plant Ball Mill. *Proc. of the 29th Ann. Meeting of the Canadian Mineral Processors*. A Div. of CIM. Ottawa, Jan. 21-23.

[STB 1993] Software Technology Branch, NASA. 1993. *CLIPS Reference Manual*. Lyndon B. Johnson Space Center.

[Stefik et al. 1983] Stefik, M., Aikins, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F. and Sacerdoti, E. 1983. Basic Concepts for Building Expert Systems. *Building Expert Systems*. Hayes-Roth, F, Waterman, D.A. and Lenat, D.B. Eds, Reading, Massachusetts, Addison-Wesley Publishing Company, Inc.: 59-86.

[Turban 1988] Turban, E. 1988. *Decision Support and Expert Systems: Managerial Perspectives*. New York, New York: Macmillan Publishing Company.

[Verma and Rajamani 1995] Verma, R. and Rajamani, R.K. 1995. Environment-dependent breakage rates in ball milling. *Powder Technology*. Vol. 84: 127-137.

[Vermeulen 1986] Vermeulen, L.A. 1986. A Contribution to 'Ball Wear and Ball Size Distribution in Tumbling Ball Mills'. *Powder Technology*. Vol. 46: 281-285.

[Walker et al. 1937] Walker, W.H., Lewis, W.K., McAdams, W.H., Gilliland, E.R. 1937. *Principles of Chemicals Engineering*. New York: McGraw-Hill.

[Waterman 1986] Waterman, D.A. 1986. *A Guide to Expert Systems*. Reading, Massachusetts: Addison-Wesley Publishing Company.

[Wegstein 1958] Wegstein, J.H. 1958. Accelerating Convergence of Iterative Processes. *Commun. Assoc. Comput. Mach.* Vol. 1: 9-13.

[Weller 1980] Weller, K.R. 1980. Hold-Up and Residence Time Characteristics of Full Scale Grinding Circuits. In *Automation in Mining, Mineral and Metal Processing*, Proc. of 3rd IFAC Symp. Edited by J. O'Shea and M. Polis. Montreal, Canada, Aug. 18-20: 303-309.

[Weller et al. 1988] Weller, K.R., Sterns, U.J., Artone, E. and Bruckard, W.J. 1988. Multicomponent Models of Grinding and Classification for scale-up from Continuous Small or Pilot Scale Circuits. *Intl. Jour. of Miner. Process.* Vol. 22: 119-147.

[Westerberg et al. 1979] Westerberg, A.W., Hutchison, H.P., Motard, R.L. and Winter, P. *Process Flowsheeting*. London: Cambridge University Press.

[Whiten 1972] Whiten, W.J. 1972. The Simulation of Crushing Plants with Models Developed using Multiple Spline Regression. *Jour. of the S. Afr. Inst. of Min. and Metall.* (May): 257-264.

[Whiten 1971] Whiten, W.J. 1971. The Use of Multi-Dimensional Cubic Spline Functions for Regression and Smoothing. *The Australian Computer Jour.* Vol. 3, No. 2 (May): 81-88.

[Wills 1997] Wills, B.A. 1997. *Mineral Processing Technology*. Oxford, Butterworth-Heinemann.

[Wiseman and Richardson 1991] Wiseman, D.M. and Richardson, J.M. 1991. JKSimMet - The Mineral Processing Simulator. *Proc. of 2nd CAMI.* Quebec, Canada, Mar. 7.

[Ynchausti and Hales 1992] Ynchausti, R.A. and Hales, L.B. 1992. Real-Time Supervisory Control of Grinding and Flotation Circuits using Object Oriented Expert Systems. in *Artificial Intelligence in Materials Processing*. Ed. S.A. Argyropoulos and G. Carayannis. Proc. of the intl. Symp. on Artificial Intelligence in Materials Process. 53-61. Edmonton, Alberta, Aug.

[Zadeh 1965] Zadeh, L. A. 1965. Fuzzy Sets. *Information and Control.* Vol. 8, No. 3: 338-353

APPENDIX A

GLOSSARY OF KNOWLEDGE-BASED SYSTEMS

Agenda A prioritized list of rules created by the inference engine, whose patterns are satisfied by facts in working memory

Antecedent A condition in if part (left hand side) of a rule

Attribute A particular aspect of an object

Backward chaining An inference strategy in rule-based systems that begins with a hypothesis and tries to prove it to be true by proving the premises of a rule that contains the hypothesis as its conclusion

Cardinality A facet in frame-based systems that restricts the number of values a slot can have

Class A collection of objects that share common properties

Conflict resolution Technique of resolving the problem of multiple matches in a rule-based system during inferencing. When more than one rule can be fired during a cycle, a conflict arises and a decision must be made on which rule will allowed to fire

Consequent A conclusion or a number of conclusions in then part (right hand side) of a rule

Consistency A property of a system of rules where all deductions are logically in agreement

Declarative knowledge Descriptive or factual knowledge

Deep knowledge The basic knowledge that comes from first principles or physical laws of the domain

Domain The area which a knowledge-based system is designed to solve problems related to that specific area

Domain expert A person who possesses the skill and knowledge to solve a specific problem in a manner superior than others

Expert system A computer program designed to model the problem-solving behaviour of a human expert

Facet Extended knowledge about a frame's property, such as its type, rake or what procedures to execute if the value is needed or changed

Forward chaining An inference strategy in rule-based systems where conclusions are drawn by first looking at the facts or data on the problem

Frame A knowledge representation method that associates an object with a collection of features. Each feature or attribute is stored in a slot with a corresponding attribute value, or method for acquiring the value

Frame-based systems A system which uses frame as the main data structure to represent the domain knowledge

Granularity The level of detail of knowledge in a rule or frame

Heuristic search A search method that uses heuristic to guide search process

Heuristic Knowledge, often expressed as a rule of thumb, that guides the search process

Hybrid system An expert system that uses various types of knowledge structures to represent domain knowledge

Inference The process of deriving new information from known information

Instance A specific object from a class of objects

Knowledge A collection of facts, rules, and concepts used in reasoning process

Knowledge base A part of a knowledge-based system which contains the domain knowledge

Knowledge-based systems system whose performance depends on encoded knowledge

Meta-knowledge Knowledge about knowledge

Method is a function that defines a behaviour of an object

Object A physical or conceptual being that has a collection of related attributes that describe it

Premise A condition on left hand side of a rule

Problem space A tree or graph containing nodes and branches used for searching for a solution to a given problem. The nodes represent possible problem states and the branches possible paths between states

Problem solving The process of seeking a soltion to a given problem

Production system A model of human problem solving where problem situations contained in the short-term memory are combined with productions in the long-term memory to infer new information

Rule A knowledge representation method consisted of one or more premises and conclusions expressed in IF ...THEN...

Rule of tumb A rule based on good judgment, gained from experience rather than first principal. Often called shallow knowledge

Rule-based systems A computer program that processes problem-specific information contained in the working memory with a set of rules contained in the knowledge base, using an inference engine to infer new information

Shallow knowledge knowledge based on good judgment rather than first principles. Often called a heuristic

Slot A component of a frame. Describes a particular attribute or relationship of the frame

Symbol An alphanumeric pattern that represents some object characteristic of event of a problem

Symbolic programming Manipulating symbols that represent objects and their relationships

Uncertainty The level of belief in a given fact either given by the user of an expert system or derived by the system

APPENDIX B.1
SPLINE CURVE FITTING

The problem of curve or data fitting is finding a function which matches a set of observed, measured or simulated values at a set of given points. In interpolation, the smoothing function must exactly match the given values at any point, while in the least-square, due to the errors or noise in the given values, the smoothing function is not required to match the given values exactly.

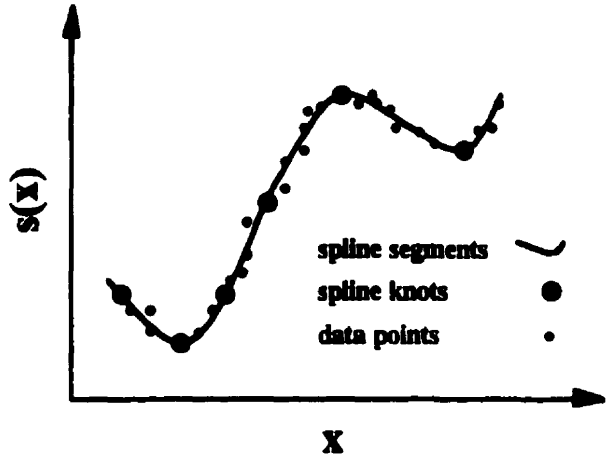


Figure A.1 A spline curve fitted to a number of data points

A spline curve is composed of segments of normal polynomial curves fitted between spline knots, Figure A.1. A knot is a point at which two segments of a spline curve are joined together, smoothly. The shape assumed by the spline between two adjacent knots depends on the degree of spline curve. Normally, a third degree

(cubic) polynomial curve is used. The smoothness of a curve means being free from or proceeding without abrupt curves, bends, etc. and can be described mathematically using derivatives.

Spline Function in One Variable

Considering the strictly increasing sequence of real numbers:

$$a=\lambda_0<\lambda_1<...<\lambda_g<\lambda_{g+1}=b$$

The function s(x) is defined as a spline of degree k on [a,b] with knots λ_i, i=1,2,3, ..,g, if it satisfies the following conditions:

In each interval $[\lambda_i, \lambda_{i+1}]$, i=0, 1, 2, ..., g, s(x) is given by some polynomial of degree k or less.

s(x) and its derivatives of orders 1, 2, 3, ..., k-1 are continuous everywhere in [a,b].

If the below additional knots are considered:

$$\lambda_{-k}=\lambda_{-k+1}=...=\lambda_{-1}=a$$
$$b=\lambda_{g+2}=...=\lambda_{g+k}=\lambda_{g+k+1}$$

then the spline can be represented by a linear combination of B-splines as below:

$$s(x)=\sum_{i=-k}^{g} c_i M_{i,k+1}(x)$$

where c_is are constant coefficients and $M_{i,k+1}(x)$ are B-splines. B-splines are more convenient representation in terms of numerical computations having the following

property:

$$M_{i,k+1}(x)=0 \qquad if\ x \le \lambda_i\ or\ x \ge \lambda_{i+k+1}$$

Spline Function Determination

Given the function values fq at the points x_q, q=1,2,3, ..., m1, Xq < Xq+1, the spline function, s(x) is determined by the following approximation criterion:

Minimize the objective function:

$$\sum_{q=1}^{m_1}\left(f_q - \sum_{i=-k}^{g} c_i M_{i,k+1}(X_q)\right)^2$$

under the constraint:

$$\sum_{q=1}^{g}\left(\sum_{i=-k}^{g} c_i a_{i,q}\right)^2 \le S$$

while

$$a_{i,q} = M_{i,k+1}^{(k)}(\lambda_q+0) - M_{i,k+1}^{(k)}(\lambda_q-0)$$

S is a non-negative constant which is called the smoothing factor. S sets the smoothness of the spline curve and can vary between zero and infinity.

Spline Function Evaluation

Too small value of S results in overfitting and too large S values results in an underfitting of the data [Dirckx 1982]. In extreme cases, the algorithms returns the least-square polynomial for very large S and interpolating spline for S=0. The graphical interface of NGOTC (*N*umerical *G*rinding *O*ptimization *T*ools in *C*) allows

the user to visually examine the fit before accepting it as satisfactory.

The Advantages of Spline Functions

The most important advantages of using spline functions to fit data can be enumerated as follows, Whiten [1971]:

- Spline functions provide a means of finding an adequate fit when there is no indication of particular analytic form.
- The smoothed data by spline fit can help indicating an analytical form in further data analysis.
- The fit is local, i.e. the behaviour of spline curve of a specified segment has little effect on the spline curve for segments few knots away.
- A piecewise cubic spline function can be rapidly evaluated.

Appendix B.2 Mass Balancing Results of AELRD

Residual sum of squares: 58.87399 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	FF	35.02	35.0	35.0	1.0	-0.0
2	CUFR	254.67	100.0	254.7	100.0	154.7*
3	BMD	319.14		319.1		
4	KCT	51.48	50.0	51.5	20.0	1.5
5	SCROS	29.45		29.5		
6	COF	35.02		35.0		
7	CUF	335.60		335.6		
8	SCRF	80.93	70.0	80.9	30.0	10.9

	Stream	Relative Solids Flowrate
1	FF	100.00
2	CUFR	727.26
3	BMD	911.36
4	KCT	147.00
5	SCROS	84.10
6	COF	100.00
7	CUF	958.36
8	SCRF	231.10

Fractional Size Distribution Data

Size	CUFR Meas	Calc	SD.	Adj.	BMD Meas	Calc	SD.	Adj.
6 MESH	0.10	0.11	0.1	0.0	0.15	0.12	0.1	-0.0
8 MESH	0.20	0.21	0.1	0.0	0.26	0.22	0.2	-0.0
10 MESH	0.16	0.18	0.1	0.0	0.38	0.20	0.2	-0.2
14 MESH	0.41	0.46	0.2	0.1	0.82	0.46	0.4	-0.4
20 MESH	0.38	0.43	0.2	0.0	1.11	0.46	0.5	-0.7*
30 MESH	0.69	0.70	0.4	0.0	0.67	0.65	0.4	-0.0
40 MESH	1.30	1.26	0.5	-0.0	1.06	1.15	0.5	0.1
50 MESH	2.94	2.95	0.5	0.0	2.65	2.64	0.5	-0.0
70 MESH	5.19	5.14	0.5	-0.1	4.56	4.68	0.5	0.1
100 MESH	12.07	11.87	0.5	-0.2	10.53	10.97	0.5	0.4
140 MESH	17.34	17.19	0.5	-0.2	15.69	16.00	0.5	0.3
200 MESH	21.68	22.39	0.5	0.7*	22.49	20.85	0.5	-1.6*
270 MESH	16.77	15.96	0.5	-0.8*	13.51	15.34	0.5	1.8*
400 MESH	10.02	9.84	0.5	-0.2	9.81	10.20	0.5	0.4
500 MESH	4.20	4.02	0.5	-0.2	4.73	5.12	0.5	0.4

KCT	SCROS

Size	Meas	Calc	SD.	Adj.	Meas	Calc	SD.	Adj.
6 MESH	0.00	-0.01	0.1	-0.0	0.47	0.36	0.3	-0.1
8 MESH	0.00	-0.00	0.1	-0.0	0.55	0.53	0.3	-0.0
10 MESH	0.00	-0.01	0.1	-0.0	0.71	0.58	0.4	-0.1
14 MESH	0.07	0.07	0.1	-0.0	1.01	0.99	0.5	-0.0
20 MESH	0.17	0.13	0.1	-0.0	1.46	1.21	0.5	-0.2
30 MESH	0.58	0.58	0.3	-0.0	0.85	0.85	0.4	0.0
40 MESH	1.15	1.17	0.5	0.0	1.43	1.43	0.5	0.0
50 MESH	3.00	3.03	0.5	0.0	2.68	2.70	0.5	0.0
70 MESH	5.24	5.30	0.5	0.1	4.81	4.84	0.5	0.0
100 MESH	12.16	12.31	0.5	0.1	11.15	11.21	0.5	0.1
140 MESH	17.68	17.69	0.5	0.0	16.84	16.82	0.5	-0.0
200 MESH	24.16	23.37	0.5	-0.8*	21.92	21.62	0.5	-0.3
270 MESH	15.18	15.87	0.5	0.7*	15.45	15.70	0.5	0.2
400 MESH	10.05	10.13	0.5	0.1	9.59	9.61	0.5	0.0
500 MESH	3.97	4.07	0.5	0.1	4.07	4.09	0.5	0.0

	COF				CUF			
Size	Meas	Calc	SD.	Adj.	Meas	Calc	SD.	Adj.
6 MESH	0.00	0.00	0.1	0.0	0.10	0.11	0.1	0.0
8 MESH	0.00	0.00	0.1	0.0	0.20	0.21	0.1	0.0
10 MESH	0.00	0.00	0.1	0.0	0.16	0.19	0.1	0.0
14 MESH	0.00	0.00	0.1	0.0	0.41	0.46	0.2	0.0
20 MESH	0.06	0.06	0.1	0.0	0.38	0.45	0.2	0.1
30 MESH	0.09	0.09	0.1	0.0	0.69	0.70	0.4	0.0
40 MESH	0.19	0.19	0.1	-0.0	1.30	1.26	0.5	-0.0
50 MESH	0.39	0.39	0.2	0.0	2.94	2.94	0.5	-0.0
70 MESH	1.29	1.28	0.5	-0.0	5.19	5.13	0.5	-0.1
100 MESH	4.37	4.32	0.5	-0.0	12.07	11.87	0.5	-0.2
140 MESH	6.86	6.83	0.5	-0.0	17.34	17.22	0.5	-0.1
200 MESH	8.89	9.07	0.5	0.2	21.68	22.46	0.5	0.8*
270 MESH	10.78	10.58	0.5	-0.2	16.77	15.93	0.5	-0.8*
400 MESH	13.32	13.28	0.5	-0.0	10.02	9.86	0.5	-0.2
500 MESH	13.84	13.80	0.5	-0.0	4.20	4.03	0.5	-0.2

	SCRF			
Size	Meas	Calc	SD.	Adj.
6 MESH	0.10	0.14	0.1	0.0
8 MESH	0.20	0.21	0.1	0.0
10 MESH	0.16	0.21	0.1	0.1
14 MESH	0.41	0.44	0.2	0.0
20 MESH	0.38	0.54	0.2	0.2
30 MESH	0.69	0.69	0.4	-0.0
40 MESH	1.30	1.28	0.5	-0.0
50 MESH	2.94	2.90	0.5	-0.0
70 MESH	5.19	5.12	0.5	-0.1
100 MESH	12.07	11.90	0.5	-0.2
140 MESH	17.34	17.34	0.5	0.0
200 MESH	21.68	22.73	0.5	1.1*
270 MESH	16.77	15.83	0.5	-0.9*
400 MESH	10.02	9.92	0.5	-0.1
500 MESH	4.20	4.08	0.5	-0.1

Residual sum of squares: 34.24823 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	FF	34.78	35.0	34.8	1.0	-0.2
2	CUTR	208.98	100.0	209.0	100.0	109.0*
3	BMD	247.95		247.9		
4	KCT	57.76	50.0	57.8	20.0	7.8
5	SCROS	4.18		4.2		
6	COF	34.78		34.8		
7	CUF	270.92		270.9		
8	SCRF	61.94	70.0	61.9	30.0	-8.1

	Stream	Relative Solids Flowrate
1	FF	100.00
2	CUTR	600.85
3	BMD	712.87
4	KCT	166.06
5	SCROS	12.02
6	COF	100.00
7	CUF	778.93
8	SCRF	178.09

Fractional Size Distribution Data

Size	CUTR Meas	Calc	SD.	Adj.	BMD Meas	Calc	SD.	Adj.
6 MESH	0.12	0.13	0.1	0.0	0.16	0.13	0.1	-0.0
8 MESH	0.09	0.11	0.1	0.0	0.39	0.12	0.2	-0.3*
10 MESH	0.13	0.15	0.1	0.0	0.48	0.17	0.3	-0.3*
14 MESH	0.38	0.45	0.2	0.1	1.00	0.43	0.5	-0.6*
20 MESH	0.34	0.40	0.2	0.1	0.75	0.37	0.4	-0.4
30 MESH	0.61	0.62	0.3	0.0	0.56	0.55	0.3	-0.0
40 MESH	1.26	1.20	0.5	-0.1	0.92	1.05	0.5	0.1
50 MESH	2.48	2.53	0.5	0.1	2.34	2.24	0.5	-0.1
70 MESH	4.67	4.64	0.5	-0.0	4.08	4.16	0.5	0.1
100 MESH	11.42	11.29	0.5	-0.1	9.89	10.17	0.5	0.3
140 MESH	18.23	17.97	0.5	-0.3	15.61	16.20	0.5	0.6*
200 MESH	25.13	25.42	0.5	0.3	24.45	23.82	0.5	-0.6*
270 MESH	17.26	16.56	0.5	-0.7*	14.52	15.82	0.5	1.3*
400 MESH	9.01	8.92	0.5	-0.1	9.53	9.59	0.5	0.1
500 MESH	3.35	2.82	0.5	-0.5*	3.93	4.92	0.5	1.0*

Size	KCT	SCROS

Size	Meas	Calc	SD.	Adj.	Meas	Calc	SD.	Adj.
6 MESH	0.00	0.01	0.1	0.0	1.22	1.27	0.5	0.0
8 MESH	0.00	-0.01	0.1	-0.0	1.77	1.73	0.5	-0.0
10 MESH	0.00	-0.01	0.1	-0.0	2.44	2.40	0.5	-0.0
14 MESH	0.08	0.09	0.1	0.0	2.98	3.02	0.5	0.0
20 MESH	0.21	0.21	0.1	-0.0	1.95	1.96	0.5	0.0
30 MESH	0.49	0.51	0.3	0.0	1.49	1.50	0.5	0.0
40 MESH	0.92	1.07	0.5	0.1	1.49	1.50	0.5	0.0
50 MESH	2.57	2.53	0.5	-0.0	2.17	2.17	0.5	-0.0
70 MESH	4.61	4.68	0.5	0.1	3.52	3.52	0.5	0.0
100 MESH	11.33	11.50	0.5	0.2	8.26	8.27	0.5	0.0
140 MESH	17.57	18.08	0.5	0.5*	13.48	13.51	0.5	0.0
200 MESH	26.25	25.84	0.5	-0.4	20.95	20.93	0.5	-0.0
270 MESH	17.45	17.46	0.5	0.0	16.29	16.27	0.5	-0.0
400 MESH	10.05	9.54	0.5	-0.5*	9.77	9.73	0.5	-0.0
500 MESH	3.37	3.39	0.5	0.0	3.68	3.66	0.5	-0.0

	COF				CUF			
Size	Meas	Calc	SD.	Adj.	Meas	Calc	SD.	Adj.
6 MESH	0.00	0.00	0.1	0.0	0.12	0.12	0.1	0.0
8 MESH	0.00	0.00	0.1	0.0	0.09	0.11	0.1	0.0
10 MESH	0.00	0.00	0.1	0.0	0.13	0.16	0.1	0.0
14 MESH	0.00	0.00	0.1	0.0	0.38	0.41	0.2	0.0
20 MESH	0.01	0.01	0.1	0.0	0.34	0.38	0.2	0.0
30 MESH	0.02	0.02	0.1	0.0	0.61	0.61	0.3	-0.0
40 MESH	0.10	0.10	0.1	-0.0	1.26	1.18	0.5	-0.1
50 MESH	0.48	0.48	0.3	0.0	2.48	2.53	0.5	0.0
70 MESH	1.35	1.34	0.5	-0.0	4.67	4.63	0.5	-0.0
100 MESH	3.67	3.63	0.5	-0.0	11.42	11.29	0.5	-0.1
140 MESH	5.93	5.85	0.5	-0.1	18.23	17.92	0.5	-0.3
200 MESH	14.43	14.52	0.5	0.1	25.13	25.44	0.5	0.3
270 MESH	11.53	11.35	0.5	-0.2	17.26	16.74	0.5	-0.5*
400 MESH	13.66	13.65	0.5	-0.0	9.01	9.06	0.5	0.1
500 MESH	17.92	17.78	0.5	-0.1	3.35	2.95	0.5	-0.4

	SCRF			
Size	Meas	Calc	SD.	Adj.
6 MESH	0.12	0.09	0.1	-0.0
8 MESH	0.09	0.11	0.1	0.0
10 MESH	0.13	0.16	0.1	0.0
14 MESH	0.38	0.30	0.2	-0.1
20 MESH	0.34	0.33	0.2	-0.0
30 MESH	0.61	0.58	0.3	-0.0
40 MESH	1.26	1.10	0.5	-0.2
50 MESH	2.48	2.51	0.5	0.0
70 MESH	4.67	4.60	0.5	-0.1
100 MESH	11.42	11.27	0.5	-0.1
140 MESH	18.23	17.76	0.5	-0.5
200 MESH	25.13	25.50	0.5	0.4
270 MESH	17.26	17.37	0.5	0.1
400 MESH	9.01	9.55	0.5	0.5*
500 MESH	3.35	3.41	0.5	0.1

Appendix B.3 Selection Function Estimation Results of AELRD

****
Selection Function Estimation Results
****

Agnico Eagle, Line #1
Date: 4/29/1997
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Breakage Function Matrix

0.00														
0.44	0.00													
0.19	0.44	0.00												
0.09	0.19	0.44	0.00											
0.05	0.09	0.19	0.44	0.00										
0.03	0.05	0.09	0.19	0.44	0.00									
0.03	0.03	0.05	0.09	0.19	0.44	0.00								
0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00							
0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00						
0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00					
0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00				
0,01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00			
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00		
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00
0.00	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44
0.00														
0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19
0.44	0.00													

CLASS	SIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	6700	2.35	0.05	0.05	7.9606
2	4738	1.52	0.14	0.14	6.1482
3	3350	1.21	0.12	0.12	8.2834
4	2369	0.98	0.21	0.21	6.3275
5	1675	0.80	0.21	0.21	6.9363
6	1184	0.95	0.47	0.47	3.8964

7	837	0.79	0.46	0.46	4.1202
8	592	0.95	0.65	0.65	3.1249
9	419	1.47	1.16	1.16	1.8993
10	296	3.03	2.64	2.64	0.9209
11	209	4.99	4.68	4.68	0.5357
12	148	11.06	10.97	10.97	0.2184
13	105	15.73	16.01	16.01	0.1258
14	74	20.27	20.85	20.85	0.0742
15	52	14.57	15.34	15.34	0.0710
16	37	9.04	10.19	10.19	0.0378
17	26	3.67	5.10	4.74	0.0000

****
Selection Function Estimation Results
****

Agnico Eagle, Line #2
Date: 4/29/1997
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Breakage Function Matrix

0.00														
0.44	0.00													
0.19	0.44	0.00												
0.09	0.19	0.44	0.00											
0.05	0.09	0.19	0.44	0.00										
0.03	0.05	0.09	0.19	0.44	0.00									
0.03	0.03	0.05	0.09	0.19	0.44	0.00								
0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00							
0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00						
0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00					
0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00				
0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00			
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00		
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00
0.00	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44
0.00														
0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19
0.44	0.00													

CLASS	SEIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	6700	2.78	0.02	0.02	9.0808
2	4738	1.83	0.04	0.04	9.1470
3	3350	1.52	0.13	0.13	6.1096
4	2369	1.10	0.12	0.12	7.6238
5	1675	0.94	0.17	0.17	6.7042
6	1184	1.05	0.43	0.43	3.6876

7	837	0.80	0.37	0.37	4.4350
8	592	0.94	0.55	0.55	3.3189
9	419	1.47	1.05	1.05	1.9692
10	296	2.72	2.24	2.24	1.0343
11	209	4.53	4.16	4.16	0.5810
12	148	10.37	10.17	10.17	0.2360
13	105	15.97	16.20	16.20	0.1292
14	74	22.31	23.82	23.82	0.0413
15	52	14.73	15.82	15.82	0.0485
16	37	8.07	9.59	9.59	0.0101
17	26	2.53	4.93	3.60	0.0000

Appendix B.4 Mass Balancing Results of LMS

Les Mines Selbaie, Grinding Circuit Survey, Zone B, July 15, 1996

Residual sum of squares: 9.358769 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	RMD	75.87	75.9	75.9	0.5	-0.1
2	BMD	418.06		418.1		
3	CUF1	418.06	250.0	418.1	250.0	168.1
4	COF1	75.87		75.9		
5	CUF2	56.67	250.0	56.7	250.0	-193.3
6	TriconeD	56.67		56.7		
7	COF2	75.87		75.9		

	Stream	Relative Solids Flowrate
1	RMD	100.00
2	BMD	551.02
3	CUF1	551.02
4	COF1	100.00
5	CUF2	74.70
6	TriconeD	74.70
7	COF2	100.00

Fractional Size Distribution Data

Size	RMD Meas	Calc	SD.	Adj.	BMD Meas	Calc	SD.	Adj.
2 MESH	0.00	0.00	0.1	0.0	0.00	0.02	0.1	0.0
2.5 MESH	0.10	0.10	0.1	-0.0	0.70	0.56	0.4	-0.1
3 MESH	0.20	0.21	0.1	0.0	0.40	0.50	0.2	0.1
4 MESH	1.10	1.13	0.5	0.0	2.20	2.35	0.5	0.1
6 MESH	4.60	4.65	0.5	0.1	3.70	3.98	0.5	0.3
8 MESH	9.30	9.32	0.5	0.0	5.00	5.10	0.5	0.1
10 MESH	9.80	9.80	0.5	0.0	4.20	4.21	0.5	0.0
14 MESH	11.80	11.77	0.5	-0.0	4.90	4.73	0.5	-0.2
20 MESH	8.80	8.76	0.5	-0.0	4.30	4.10	0.5	-0.2
28 MESH	7.30	7.28	0.5	-0.0	4.70	4.59	0.5	-0.1
35 MESH	6.60	6.57	0.5	-0.0	6.50	6.36	0.5	-0.1
50 MESH	4.60	4.60	0.5	-0.0	7.10	7.08	0.5	-0.0
65 MESH	3.90	3.86	0.5	-0.0	8.60	8.35	0.5	-0.2
100 MESH	3.50	3.46	0.5	-0.0	8.60	8.36	0.5	-0.2
140 MESH	3.40	3.37	0.5	-0.0	7.70	7.54	0.5	-0.2

Size	CUF1 Meas	Calc	SD.	Adj.	COF1 Meas	Calc	SD.	Adj.
2 MESH	0.20	0.02	0.1	-0.2*	0.00	-0.00	0.1	-0.0
2.5 MESH	0.50	0.58	0.3	0.1	0.00	0.00	0.1	0.0
3 MESH	1.00	0.54	0.5	-0.5	0.00	-0.00	0.1	-0.0
4 MESH	2.70	2.55	0.5	-0.1	0.00	-0.00	0.1	-0.0
6 MESH	5.10	4.82	0.5	-0.3	0.00	-0.00	0.1	-0.0

8 MESH	6.90	6.80	0.5	-0.1	0.00	-0.00	0.1	-0.0
10 MESH	6.00	5.99	0.5	-0.0	0.00	-0.00	0.1	-0.0
14 MESH	6.70	6.87	0.5	0.2	0.00	0.02	0.1	0.0
20 MESH	5.50	5.70	0.5	0.2	0.00	0.00	0.1	0.0
28 MESH	5.80	5.91	0.5	0.1	0.00	0.02	0.1	0.0
35 MESH	7.40	7.54	0.5	0.1	0.10	0.10	0.1	-0.0
50 MESH	7.80	7.82	0.5	0.0	0.60	0.54	0.3	-0.1
65 MESH	8.30	8.55	0.5	0.2	3.10	2.79	0.5	-0.3
100 MESH	7.40	7.64	0.5	0.2	7.40	7.45	0.5	0.1
140 MESH	6.10	6.26	0.5	0.2	10.20	10.37	0.5	0.2

	CUF2				TriconeD			
Size	Meas	Calc	SD.	Adj.	Meas	Calc	SD.	Adj.
2 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
2.5 MESH	0.00	0.00	0.1	0.0	0.00	-0.00	0.1	-0.0
3 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
4 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
6 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
8 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
10 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
14 MESH	0.10	0.06	0.1	-0.0	0.00	0.01	0.1	0.0
20 MESH	0.00	0.00	0.1	0.0	0.00	-0.00	0.1	-0.0
28 MESH	0.10	0.06	0.1	-0.0	0.00	0.01	0.1	0.0
35 MESH	0.30	0.32	0.2	0.0	0.20	0.19	0.1	-0.0
50 MESH	1.00	1.10	0.5	0.1	0.40	0.38	0.2	-0.0
65 MESH	5.40	5.67	0.5	0.3	2.50	2.23	0.5	-0.3
100 MESH	13.90	13.89	0.5	-0.0	5.10	5.11	0.5	0.0
140 MESH	19.20	19.09	0.5	-0.1	8.80	8.91	0.5	0.1

	COF2			
Size	Meas	Calc	SD.	Adj.
2 MESH	0.00	-0.00	0.1	-0.0
2.5 MESH	0.00	0.00	0.1	0.0
3 MESH	0.00	-0.00	0.1	-0.0
4 MESH	0.00	-0.00	0.1	-0.0
6 MESH	0.00	-0.00	0.1	-0.0
8 MESH	0.00	-0.00	0.1	-0.0
10 MESH	0.00	-0.00	0.1	-0.0
14 MESH	0.00	-0.02	0.1	-0.0
20 MESH	0.00	0.00	0.1	0.0
28 MESH	0.00	-0.02	0.1	-0.0
35 MESH	0.00	0.00	0.1	0.0
50 MESH	0.00	0.00	0.1	0.0
65 MESH	0.20	0.23	0.1	0.0
100 MESH	0.90	0.89	0.5	-0.0
140 MESH	2.90	2.76	0.5	-0.1

Les Mines Selbaie, Grinding Circuit Survey, Zone B, Feb 9, 1998

Residual sum of squares: 32.71175 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	RMD	81.00	81.0	81.0	0.0	0.0
2	BMD	307.06		307.1		
3	CUF1	307.06	250.0	307.1	250.0	57.1
4	COF1	81.00		81.0		
5	CUF2	51.60	250.0	51.6	250.0	-198.4
6	TriconeD	51.60		51.6		
7	COF2	81.00		81.0		

	Stream	Relative Solids Flowrate
1	RMD	100.00
2	BMD	379.09
3	CUF1	379.09
4	COF1	100.00
5	CUF2	63.71
6	TriconeD	63.71
7	COF2	100.00

Fractional Size Distribution Data

Size	RMD Meas	Calc	SD.	Adj.	BMD Meas	Calc	SD.	Adj.
4 MESH	1.60	1.52	0.5	-0.1	3.00	2.70	0.5	-0.3
6 MESH	5.30	5.40	0.5	0.1	3.70	4.09	0.5	0.4
8 MESH	10.90	11.01	0.5	0.1	5.40	5.80	0.5	0.4
10 MESH	11.30	11.38	0.5	0.1	4.70	5.00	0.5	0.3
14 MESH	12.40	12.36	0.5	-0.0	6.60	6.43	0.5	-0.2
20 MESH	8.70	8.63	0.5	-0.1	6.30	6.02	0.5	-0.3
28 MESH	7.00	6.92	0.5	-0.1	6.90	6.60	0.5	-0.3
35 MESH	6.10	6.01	0.5	-0.1	9.30	8.96	0.5	-0.3
50 MESH	4.30	4.24	0.5	-0.1	8.90	8.67	0.5	-0.2
65 MESH	3.40	3.36	0.5	-0.0	7.30	7.13	0.5	-0.2
100 MESH	3.20	3.20	0.5	0.0	6.30	6.30	0.5	0.0
140 MESH	2.90	2.95	0.5	0.1	5.30	5.49	0.5	0.2
200 MESH	2.50	2.48	0.5	-0.0	3.70	3.64	0.5	-0.1
270 MESH	2.60	2.62	0.5	0.0	3.20	3.29	0.5	0.1
400 MESH	1.90	1.89	0.5	-0.0	2.40	2.34	0.5	-0.1

Size	CUF1 Meas	Calc	SD.	Adj.	COF1 Meas	Calc	SD.	Adj.
4 MESH	2.80	3.10	0.5	0.3	0.00	0.00	0.1	0.0
6 MESH	5.90	5.51	0.5	-0.4	0.00	-0.00	0.1	-0.0
8 MESH	9.10	8.70	0.5	-0.4	0.00	-0.00	0.1	-0.0
10 MESH	8.30	8.00	0.5	-0.3	0.00	-0.00	0.1	-0.0
14 MESH	9.50	9.67	0.5	0.2	0.10	0.08	0.1	-0.0

20 MESH	8.00	8.28	0.5	0.3	0.10	0.08	0.1	-0.0	
28 MESH	8.10	8.40	0.5	0.3	0.10	0.13	0.1	0.0	
35 MESH	9.90	10.24	0.5	0.3	0.90	1.17	0.5	0.3	
50 MESH	8.50	8.73	0.5	0.2	3.60	4.00	0.5	0.4	
65 MESH	6.10	6.27	0.5	0.2	7.40	6.64	0.5	-0.8*	
100 MESH	4.60	4.60	0.5	-0.0	10.00	9.66	0.5	-0.3	
140 MESH	3.60	3.41	0.5	-0.2	10.20	10.87	0.5	0.7*	
200 MESH	2.20	2.26	0.5	0.1	8.60	7.74	0.5	-0.9*	
270 MESH	1.80	1.71	0.5	-0.1	8.60	8.63	0.5	0.0	
400 MESH	1.20	1.26	0.5	0.1	6.30	6.01	0.5	-0.3	

	CUF2				TriconeD			
Size	Meas	Calc	SD.	Adj.	Meas	Calc	SD.	Adj.
4 MESH	0.00	0.00	0.1	0.0	0.00	-0.00	0.1	-0.0
6 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
8 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
10 MESH	0.00	-0.00	0.1	-0.0	0.00	0.00	0.1	0.0
14 MESH	0.10	0.11	0.1	0.0	0.00	-0.00	0.1	-0.0
20 MESH	0.10	0.11	0.1	0.0	0.00	-0.00	0.1	-0.0
28 MESH	0.30	0.23	0.2	-0.1	0.00	0.01	0.1	0.0
35 MESH	2.00	1.85	0.5	-0.2	0.00	0.00	0.1	0.0
50 MESH	6.50	6.28	0.5	-0.2	0.00	0.00	0.1	0.0
65 MESH	13.60	14.11	0.5	0.5*	6.30	5.79	0.5	-0.5*
100 MESH	17.10	17.31	0.5	0.2	7.60	7.39	0.5	-0.2
140 MESH	12.90	12.44	0.5	-0.5	6.50	6.96	0.5	0.5
200 MESH	9.60	10.16	0.5	0.6*	14.40	13.84	0.5	-0.6*
270 MESH	8.40	8.37	0.5	-0.0	11.50	11.53	0.5	0.0
400 MESH	4.90	5.09	0.5	0.2	8.40	8.21	0.5	-0.2

	COF2			
Size	Meas	Calc	SD.	Adj.
4 MESH	0.00	0.00	0.1	0.0
6 MESH	0.00	-0.00	0.1	-0.0
8 MESH	0.00	-0.00	0.1	-0.0
10 MESH	0.00	-0.00	0.1	-0.0
14 MESH	0.00	0.01	0.1	0.0
20 MESH	0.00	0.01	0.1	0.0
28 MESH	0.00	-0.01	0.1	-0.0
35 MESH	0.00	-0.00	0.1	-0.0
50 MESH	0.00	-0.00	0.1	-0.0
65 MESH	0.80	1.34	0.4	0.5*
100 MESH	3.00	3.34	0.5	0.3
140 MESH	8.10	7.38	0.5	-0.7*
200 MESH	9.20	10.08	0.5	0.9*
270 MESH	10.70	10.65	0.5	-0.1
400 MESH	7.70	8.00	0.5	0.3

Appendix B.5 Selection Function Estimation Results of LMS

****
Selection Function Estimation Results
****

Les Mines Selbaie, ball mill, July 15,1996
Date: 12/8/1996
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Breakage Function Matrix

0.00										
0.44	0.00									
0.19	0.44	0.00								
0.09	0.19	0.44	0.00							
0.05	0.09	0.19	0.44	0.00						
0.03	0.05	0.09	0.19	0.44	0.00					
0.03	0.03	0.05	0.09	0.19	0.44	0.00				
0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00			
0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00		
0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00
0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44
0.00										
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19
0.44	0.00									
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09
0.19	0.44	0.00								
0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05
0.09	0.19	0.44	0.00							

CLASS	SEIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	11180	0.02	0.02	0.02	0.0000
2	7905	0.58	0.56	0.56	0.0354
3	5590	0.54	0.50	0.50	0.0957
4	3953	2.55	2.35	2.35	0.0939
5	2795	4.82	3.98	3.98	0.2273
6	1976	6.80	5.10	5.10	0.3940
7	1398	5.99	4.21	4.21	0.6356
8	988	6.87	4.73	4.73	0.7548
9	699	5.70	4.10	4.10	0.9258
10	494	5.91	4.59	4.59	0.8760
11	349	7.54	6.36	6.36	0.6703
12	247	7.82	7.08	7.08	0.5868
13	175	8.55	8.35	8.35	0.4508
14	124	7.64	8.36	8.36	0.3344
15	87	6.26	7.54	7.54	0.2350

****
Selection Function Estimation Results
****

LMS, ball mill, Feb. 9, 98
Date: 5/10/1998
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Reference feed flow rate = 307.1 t/h
Current feed flow rate = 307.1 t/h

Breakage Function Matrix

0.00										
0.44	0.00									
0.19	0.44	0.00								
0.09	0.19	0.44	0.00							
0.05	0.09	0.19	0.44	0.00						
0.03	0.05	0.09	0.19	0.44	0.00					
0.02	0.03	0.05	0.09	0.19	0.44	0.00				
0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00			
0.02	0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00		
0.01	0.02	0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.01	0.02	0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.05	0.09	0.19	0.44
0.00										
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.05	0.09	0.19
0.44	0.00									
0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.05	0.09
0.19	0.44	0.00								
0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.05
0.09	0.19	0.44	0.00							

CLASS	SEIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	4750	3.10	2.70	2.70	0.1431
2	3359	5.51	4.09	4.09	0.3609
3	2375	8.70	5.80	5.80	0.5689
4	1679	8.00	5.00	5.00	0.8730
5	1188	9.67	6.43	6.43	0.8751
6	840	8.28	6.02	6.02	0.9737
7	594	8.40	6.60	6.60	0.9214
8	420	10.24	8.96	8.96	0.6792
9	297	8.73	8.67	8.67	0.5989
10	210	6.27	7.13	7.13	0.5663
11	148	4.60	6.30	6.30	0.4107
12	105	3.41	5.49	5.49	0.2257
13	74	2.26	3.64	3.64	0.2635
14	52	1.71	3.29	3.29	0.0493
15	37	1.26	2.34	2.34	0.0400

****
Selection Function Estimation Results
****

LMS, ball mill, Ref. July 96, Feb. 9, 98
Date: 5/10/1998
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Reference feed flow rate = 418.1 t/h
Current feed flow rate = 307.1 t/h

Breakage Function Matrix

0.00										
0.44	0.00									
0.19	0.44	0.00								
0.09	0.19	0.44	0.00							
0.05	0.09	0.19	0.44	0.00						
0.03	0.05	0.09	0.19	0.44	0.00					
0.02	0.03	0.05	0.09	0.19	0.44	0.00				
0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00			
0.02	0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00		
0.01	0.02	0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.01	0.02	0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.05	0.09	0.19	0.44
0.00										
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.05	0.09	0.19
0.44	0.00									
0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.05	0.09
0.19	0.44	0.00								
0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.05
0.09	0.19	0.44	0.00							

CLASS	SEIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	4750	3.10	2.70	2.70	0.1051
2	3359	5.51	4.09	4.09	0.2651
3	2375	8.70	5.80	5.80	0.4178
4	1679	8.00	5.00	5.00	0.6412
5	1188	9.67	6.43	6.43	0.6427
6	840	8.28	6.02	6.02	0.7151
7	594	8.40	6.60	6.60	0.6768
8	420	10.24	8.96	8.96	0.4989
9	297	8.73	8.67	8.67	0.4399
10	210	6.27	7.13	7.13	0.4159
11	148	4.60	6.30	6.30	0.3017
12	105	3.41	5.49	5.49	0.1658
13	74	2.26	3.64	3.64	0.1935
14	52	1.71	3.29	3.29	0.0362
15	37	1.26	2.34	2.34	0.0294

****
Selection Function Estimation Results
****

LMS, Tricone SF, July 96
Date: 5/11/1998
Tau Plug Flow = 0.00 Tau Small PM = 0.10 Tau Large PM = 0.70

Reference feed flow rate = 56.7 t/h
Current feed flow rate = 56.7 t/h

Breakage Function Matrix

0.00							
0.44	0.00						
0.19	0.44	0.00					
0.09	0.19	0.44	0.00				
0.05	0.09	0.19	0.44	0.00			
0.03	0.05	0.09	0.19	0.44	0.00		
0.02	0.03	0.05	0.09	0.19	0.44	0.00	
0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00

CLASS	SEIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	1180	0.06	0.01	0.01	2.6650
2	834	0.01	0.01	0.01	1.9524
3	590	0.06	0.01	0.01	3.7829
4	417	0.32	0.19	0.19	0.7651
5	295	1.10	0.38	0.38	1.5423
6	209	5.67	2.23	2.23	1.2991
7	148	13.89	5.11	5.11	1.5363
8	104	19.09	8.91	8.91	1.4042

****
Selection Function Estimation Results
****

LMS, Tricone SF, Feb. 98, Ref. July 1996
Date: 5/11/1998
Tau Plug Flow = 0.00 Tau Small PM = 0.10 Tau Large PM = 0.70

Reference feed flow rate = 56.6 t/h
Current feed flow rate = 51.6 t/h

Breakage Function Matrix

0.00							
0.44	0.00						
0.19	0.44	0.00					
0.09	0.19	0.44	0.00				
0.05	0.09	0.19	0.44	0.00			
0.03	0.05	0.09	0.19	0.44	0.00		
0.02	0.03	0.05	0.09	0.19	0.44	0.00	
0.02	0.02	0.03	0.05	0.09	0.19	0.44	0.00

CLASS	SEIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	417	2.30	0.01	0.01	12.4148
2	295	6.28	0.01	0.01	20.1900
3	208	14.11	5.79	5.79	1.3525
4	147	17.31	7.39	7.39	1.5653
5	104	12.44	6.96	6.96	1.7213
6	74	10.16	13.84	13.84	0.5453
7	52	8.37	11.53	11.53	0.4778
8	37	5.09	8.21	8.21	0.4515

Appendix B.6 Mass Balancing Results of Lupin

ECHO BAY MINES LTD., Lupin Operation, Grinding Survey, Jan. 30/1996

Residual sum of squares: 8.503008 User Abort

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	RMD	84.16	84.0	84.2	0.5	0.1
2	CUF	237.30	250.0	237.3	250.0	-12.7
3	BMD	237.30		237.3		
4	COF	84.16		84.2		

	Stream	Relative Solids Flowrate
1	RMD	100.00
2	CUF	281.98
3	BMD	281.98
4	COF	100.00

Cumulative Size Distribution Data

Size	RMD Meas	RMD Calc	RMD SD.	RMD Adj.	CUF Meas	CUF Calc	CUF SD.	CUF Adj.
4 MESH	100.0	100.0	0.0	0.0	100.0	100.0	0.0	0.0
6 MESH	95.45	95.37	0.5	-0.1	96.94	97.17	0.5	0.2
8 MESH	87.60	87.45	0.5	-0.1	93.61	94.02	0.5	0.4
10 MESH	76.75	76.60	0.5	-0.1	89.51	89.92	0.5	0.4
14 MESH	64.20	64.09	0.5	-0.1	84.88	85.19	0.5	0.3
20 MESH	54.70	54.62	0.5	-0.1	81.13	81.34	0.5	0.2
28 MESH	47.45	47.39	0.5	-0.1	77.64	77.82	0.5	0.2
35 MESH	41.70	41.68	0.5	-0.0	73.46	73.52	0.5	0.1
48 MESH	36.45	36.53	0.5	0.1	67.28	67.06	0.5	-0.2
65 MESH	32.55	32.67	0.5	0.1	59.13	58.78	0.5	-0.4
100 MESH	28.75	28.81	0.5	0.1	46.91	46.73	0.5	-0.2
150 MESH	25.15	25.16	0.5	0.0	33.49	33.47	0.5	-0.0
200 MESH	21.65	21.60	0.5	-0.1	21.66	21.81	0.5	0.1
270 MESH	18.00	17.90	0.5	-0.1	13.49	13.78	0.5	0.3
400 MESH	14.45	14.34	0.5	-0.1	8.59	8.91	0.5	0.3

Size	BMD Meas	BMD Calc	BMD SD.	BMD Adj.	COF Meas	COF Calc	COF SD.	COF Adj.
4 MESH	100.0	100.0	0.0	0.0	100.0	100.0	0.0	0.0
6 MESH	99.03	98.81	0.5	-0.2	100.0	100.0	0.0	0.0
8 MESH	98.88	98.47	0.5	-0.4	100.0	100.0	0.0	0.0
10 MESH	98.63	98.22	0.5	-0.4	100.0	100.0	0.0	0.0
14 MESH	98.23	97.92	0.5	-0.3	100.0	100.0	0.0	0.0
20 MESH	97.65	97.44	0.5	-0.2	100.0	100.0	0.0	0.0
28 MESH	96.65	96.47	0.5	-0.2	100.0	100.0	0.0	0.0
35 MESH	94.25	94.19	0.5	-0.1	99.95	99.95	0.1	0.0
48 MESH	89.30	89.52	0.5	0.2	99.85	99.85	0.1	-0.0
65 MESH	82.03	82.38	0.5	0.4	99.30	99.23	0.4	-0.1
100 MESH	70.63	70.81	0.5	0.2	96.75	96.69	0.5	-0.1

150 MESH	56.63	56.65	0.5	0.0	90.55	90.54	0.5	-0.0
200 MESH	42.58	42.43	0.5	-0.1	79.70	79.75	0.5	0.1
270 MESH	31.30	31.01	0.5	-0.3	66.40	66.50	0.5	0.1
400 MESH	22.78	22.46	0.5	-0.3	52.45	52.56	0.5	0.1

ECHO BAY MINES LTD., Lupin Operation, Grinding Survey, Mar. 1,1996

Residual sum of squares: 40.18189 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	RMD	89.80	89.8	89.8	0.5	-0.0
2	CUF	215.03	250.0	215.0	250.0	-35.0
3	BMD	215.03		215.0		
4	COF	89.80		89.8		

	Stream	Relative Solids Flowrate
1	RMD	100.00
2	CUF	239.46
3	BMD	239.46
4	COF	100.00

Cumulative Size Distribution Data

Size	RMD Meas	Calc	SD.	Adj.	CUF Meas	Calc	SD.	Adj.
4 MESH	100.0	100.0	0.0	0.0	100.0	100.0	0.0	0.0
6 MESH	93.05	92.96	0.5	-0.1	95.68	95.89	0.5	0.2
8 MESH	84.85	84.68	0.5	-0.2	91.69	92.09	0.5	0.4
10 MESH	74.90	74.77	0.5	-0.1	87.48	87.79	0.5	0.3
14 MESH	63.70	63.68	0.5	-0.0	82.95	83.01	0.5	0.1
20 MESH	55.20	55.25	0.5	0.0	79.18	79.07	0.5	-0.1
28 MESH	48.50	48.59	0.5	0.1	75.59	75.38	0.5	-0.2
35 MESH	43.05	43.18	0.5	0.1	71.21	70.89	0.5	-0.3
48 MESH	38.00	38.22	0.5	0.2	64.68	64.16	0.5	-0.5*
65 MESH	34.35	34.56	0.5	0.2	56.30	55.80	0.5	-0.5
100 MESH	30.75	30.00	0.5	-0.7*	44.23	46.02	0.5	1.8*
150 MESH	27.15	27.26	0.5	0.1	31.33	31.08	0.5	-0.3
200 MESH	23.65	23.72	0.5	0.1	20.34	20.16	0.5	-0.2
270 MESH	19.80	19.86	0.5	0.1	13.01	12.87	0.5	-0.1
400 MESH	15.90	15.96	0.5	0.1	8.61	8.47	0.5	-0.1

Size	BMD Meas	Calc	SD.	Adj.	COF Meas	Calc	SD.	Adj.
4 MESH	100.0	100.0	0.0	0.0	100.0	100.0	0.0	0.0
6 MESH	99.03	98.83	0.5	-0.2	100.0	100.0	0.0	0.0
8 MESH	98.88	98.48	0.5	-0.4	100.0	100.0	0.0	0.0
10 MESH	98.63	98.32	0.5	-0.3	100.0	100.0	0.0	0.0
14 MESH	98.23	98.17	0.5	-0.1	100.0	100.0	0.0	0.0
20 MESH	97.65	97.76	0.5	0.1	100.0	100.0	0.0	-0.0
28 MESH	96.65	96.86	0.5	0.2	100.0	100.0	0.0	-0.0
35 MESH	94.25	94.57	0.5	0.3	99.90	99.90	0.1	-0.0
48 MESH	89.30	89.82	0.5	0.5*	99.70	99.67	0.2	-0.0
65 MESH	82.03	82.53	0.5	0.5	98.75	98.54	0.5	-0.2
100 MESH	75.63	73.84	0.5	-1.8*	95.85	96.60	0.5	0.7*

150 MESH	56.63	56.88	0.5	0.3	89.15	89.04	0.5	-0.1
200 MESH	42.58	42.76	0.5	0.2	77.90	77.83	0.5	-0.1
270 MESH	31.30	31.44	0.5	0.1	64.40	64.34	0.5	-0.1
400 MESH	22.78	22.92	0.5	0.1	50.60	50.54	0.5	-0.1

Lupin Operation, grinding survey, Mar. 31/96

Residual sum of squares: 28.38126 — Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	RMF	95.13	95.1	95.1	0.1	-0.0
2	RMD	95.13		95.1		
3	CF	362.67	300.0	362.7	300.0	62.7
4	CUF	267.54		267.5		
5	BMD	267.54		267.5		
6	COF	95.13		95.1		

	Stream	Relative Solids Flowrate
1	RMF	100.00
2	RMD	100.00
3	CF	381.23
4	CUF	281.23
5	BMD	281.23
6	COF	100.00

Cumulative Size Distribution Data

Size	RMD Meas	RMD Calc	RMD SD.	RMD Adj.	CF Meas	CF Calc	CF SD.	CF Adj.
6 MESH	100.0	100.0	0.0	0.0	100.0	100.0	0.0	0.0
8 MESH	87.85	87.70	0.5	-0.1	95.30	95.68	0.5	0.4
10 MESH	78.25	78.12	0.5	-0.1	92.65	92.92	0.5	0.3
14 MESH	66.80	66.70	0.5	-0.1	89.20	89.46	0.5	0.3
20 MESH	57.75	57.67	0.5	-0.1	86.25	86.50	0.5	0.3
28 MESH	51.00	50.91	0.5	-0.1	83.40	83.76	0.5	0.4
35 MESH	45.20	45.19	0.5	-0.0	79.80	80.14	0.5	0.3
50 MESH	40.15	40.23	0.5	0.1	74.55	74.95	0.5	0.4
70 MESH	35.90	36.02	0.5	0.1	67.35	67.76	0.5	0.4
100 MESH	31.85	31.86	0.5	0.0	56.90	57.42	0.5	0.5*
150 MESH	28.00	27.96	0.5	-0.0	45.00	45.71	0.5	0.7*
200 MESH	24.10	23.89	0.5	-0.2	34.30	34.96	0.5	0.7*
270 MESH	20.35	20.10	0.5	-0.2	25.85	26.36	0.5	0.5*
400 MESH	16.20	15.94	0.5	-0.3	19.00	19.42	0.5	0.4
500 MESH	12.70	12.49	0.5	-0.2	13.60	14.12	0.5	0.5*

Size	CUF Meas	CUF Calc	CUF SD.	CUF Adj.	BMD Meas	BMD Calc	BMD SD.	BMD Adj.
6 MESH	100.0	100.0	0.0	0.0	100.0	100.0	0.0	0.0
8 MESH	94.01	94.14	0.5	0.1	98.93	98.52	0.5	-0.4
10 MESH	90.24	90.40	0.5	0.2	98.53	98.18	0.5	-0.4
14 MESH	85.61	85.71	0.5	0.1	97.83	97.55	0.5	-0.3
20 MESH	81.66	81.70	0.5	0.0	96.98	96.75	0.5	-0.2
28 MESH	78.01	77.99	0.5	-0.0	95.68	95.44	0.5	-0.2
35 MESH	73.34	73.12	0.5	-0.2	92.60	92.57	0.5	-0.0

50 MESH	66.64	66.12	0.5	-0.5*	87.08	87.30	0.5	0.2
70 MESH	57.34	56.70	0.5	-0.6*	78.70	79.04	0.5	0.3
100 MESH	44.15	43.75	0.5	-0.4	66.48	66.50	0.5	0.0
150 MESH	30.71	30.30	0.5	-0.4	52.15	52.03	0.5	-0.1
200 MESH	19.30	19.40	0.5	0.1	39.48	38.90	0.5	-0.6*
270 MESH	12.23	12.56	0.5	0.3	29.28	28.58	0.5	-0.7*
400 MESH	7.79	8.20	0.5	0.4	21.38	20.66	0.5	-0.7*
500 MESH	5.30	5.52	0.5	0.2	15.30	14.70	0.5	-0.6*

	COF			
Size	Meas	Calc	SD.	Adj.
6 MESH	100.0	100.0	0.0	0.0
8 MESH	100.0	100.0	0.0	0.0
10 MESH	100.0	100.0	0.0	0.0
14 MESH	100.0	100.0	0.0	0.0
20 MESH	100.0	100.0	0.0	0.0
28 MESH	100.0	100.0	0.0	-0.0
35 MESH	99.90	99.90	0.1	-0.0
50 MESH	99.80	99.79	0.1	-0.0
70 MESH	99.05	98.84	0.5	-0.2
100 MESH	96.00	95.86	0.5	-0.1
150 MESH	89.20	89.06	0.5	-0.1
200 MESH	78.70	78.73	0.5	0.0
270 MESH	65.05	65.17	0.5	0.1
400 MESH	50.85	50.99	0.5	0.1
500 MESH	38.20	38.28	0.5	0.1

Appendix B.7 Selection Function Estimation Results of Lupin

****
Selection Function Estimation Results
****

Lupin Operation, survey Jan. 30/96
Date: 8/31/1998
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Reference feed flow rate = 100.0 t/h
Current feed flow rate = 100.0 t/h

Breakage Function Matrix

0.00										
0.44	0.00									
0.19	0.44	0.00								
0.09	0.19	0.44	0.00							
0.05	0.09	0.19	0.44	0.00						
0.03	0.05	0.09	0.19	0.44	0.00					
0.03	0.03	0.05	0.09	0.19	0.44	0.00				
0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00			
0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00		
0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00
0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44
0.00										
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19
0.44	0.00									
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09
0.19	0.44	0.00								

CLASS	SIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	3360	2.83	1.18	1.18	1.0780
2	2380	3.16	0.34	0.34	4.7181
3	1683	4.11	0.24	0.24	8.0812
4	1190	4.74	0.29	0.29	8.8867
5	841	3.85	0.48	0.48	7.4194
6	595	3.53	0.96	0.96	5.1212
7	421	4.30	2.28	2.28	2.8287
8	298	6.46	4.67	4.67	1.6092
9	210	8.29	7.13	7.13	1.0838
10	149	12.04	11.58	11.58	0.6422
11	105	13.27	14.15	14.15	0.4340
12	74	11.65	14.23	14.23	0.2732
13	53	8.03	11.42	11.42	0.1564
14	37	4.86	8.56	8.50	0.0000

****
Selection Function Estimation Results
****

Lupin Operation, survey Mar. 1/96
Date: 8/31/1998
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Reference feed flow rate = 100.0 t/h
Current feed flow rate = 100.0 t/h

Breakage Function Matrix

0.00										
0.44	0.00									
0.19	0.44	0.00								
0.09	0.19	0.44	0.00							
0.05	0.09	0.19	0.44	0.00						
0.03	0.05	0.09	0.19	0.44	0.00					
0.03	0.03	0.05	0.09	0.19	0.44	0.00				
0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00			
0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00		
0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00
0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44
0.00										
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19
0.44	0.00									
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09
0.19	0.44	0.00								

CLASS	SIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	3360	4.11	1.17	1.17	1.6806
2	2380	3.80	0.35	0.35	5.7459
3	1683	4.30	0.16	0.16	12.1031
4	1190	4.78	0.15	0.15	14.3849
5	841	3.94	0.41	0.41	8.4174
6	595	3.69	0.90	0.90	5.6399
7	421	4.49	2.29	2.29	3.0494
8	298	6.73	4.75	4.75	1.7486
9	210	8.36	7.29	7.29	1.1628
10	149	9.78	8.69	8.69	0.9952
11	105	14.94	16.96	16.96	0.3636
12	74	10.92	14.12	14.12	0.2723
13	53	7.29	11.32	11.32	0.1271
14	37	4.40	8.52	8.12	0.0000

****
Selection Function Estimation Results
****

Lupin Operation, survey Mar. 31/96
Date: 8/30/1998
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Reference feed flow rate = 100.0 t/h
Current feed flow rate = 100.0 t/h

Breakage Function Matrix

0.00										
0.44	0.00									
0.19	0.44	0.00								
0.09	0.19	0.44	0.00							
0.05	0.09	0.19	0.44	0.00						
0.03	0.05	0.09	0.19	0.44	0.00					
0.03	0.03	0.05	0.09	0.19	0.44	0.00				
0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00			
0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00		
0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00
0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44
0.00										
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19
0.44	0.00									
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09
0.19	0.44	0.00								

CLASS	SIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	2380	5.86	1.48	1.48	1.8865
2	1683	3.74	0.34	0.34	6.7293
3	1190	4.69	0.63	0.63	5.5160
4	841	4.01	0.80	0.80	5.3844
5	595	3.71	1.31	1.31	4.0762
6	421	4.87	2.87	2.87	2.2824
7	298	7.00	5.27	5.27	1.3876
8	210	9.42	8.26	8.26	0.9023
9	149	12.95	12.54	12.54	0.5624
10	105	13.45	14.47	14.47	0.3890
11	74	10.90	13.13	13.13	0.2882
12	53	6.84	10.32	10.32	0.1367
13	37	4.36	7.92	7.67	0.0000
14	26	2.68	5.96	4.79	0.0000

Appendix B.8 Mass Balancing Results of Louvicourt Mine

Louvicourt Mine, Grinding Circuit Survey, Ball Mill and Primary Cyclopa

Residual sum of squares: 12.31187 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	ScrUF	176.44	180.0	176.4	20.0	-3.6
2	BMD	639.39	630.0	639.4	65.0	9.4
3	COF	176.44		176.4		
4	CUF	639.39		639.4		

	Stream	Relative Solids Flowrate
1	ScrUF	100.00
2	BMD	362.39
3	COF	100.00
4	CUF	362.39

Fractional Size Distribution Data

Size	ScrUF Meas	Calc	SD.	Adj.	BMD Meas	Calc	SD.	Adj.
6 MESH	1.63	1.68	0.5	0.0	0.28	0.30	0.2	0.0
8 MESH	2.30	2.32	0.5	0.0	0.38	0.39	0.2	0.0
10 MESH	2.43	2.43	0.5	0.0	0.50	0.50	0.3	0.0
14 MESH	3.14	3.14	0.5	-0.0	0.61	0.61	0.3	-0.0
20 MESH	3.62	3.64	0.5	0.0	0.33	0.34	0.2	0.0
28 MESH	4.43	4.44	0.5	0.0	0.55	0.56	0.3	0.0
35 MESH	6.07	6.08	0.5	0.0	0.88	0.92	0.4	0.0
50 MESH	6.09	6.13	0.5	0.0	1.35	1.48	0.5	0.1
65 MESH	6.88	6.92	0.5	0.0	3.03	3.16	0.5	0.1
100 MESH	7.18	7.19	0.5	0.0	5.39	5.43	0.5	0.0
150 MESH	6.80	6.80	0.5	-0.0	7.73	7.73	0.5	-0.0
200 MESH	6.86	6.70	0.5	-0.2	15.35	14.77	0.5	-0.6*
270 MESH	6.04	6.31	0.5	0.3	15.82	16.81	0.5	1.0*
400 MESH	5.71	5.76	0.5	0.1	15.83	16.02	0.5	0.2
500 MESH	4.05	4.07	0.5	0.0	6.99	7.05	0.5	0.1

Size	COF Meas	Calc	SD.	Adj.	CUF Meas	Calc	SD.	Adj.
6 MESH	0.00	-0.00	0.1	-0.0	0.90	0.76	0.5	-0.1
8 MESH	0.00	-0.00	0.1	-0.0	1.09	1.03	0.5	-0.1
10 MESH	0.00	-0.00	0.1	-0.0	1.18	1.17	0.5	-0.0
14 MESH	0.00	0.00	0.1	0.0	1.46	1.47	0.5	0.0

20 MESH	0.06	0.06	0.1	-0.0	1.40	1.33	0.5	-0.1
28 MESH	0.01	0.01	0.1	-0.0	1.81	1.78	0.5	-0.0
35 MESH	0.03	0.03	0.1	-0.0	2.63	2.59	0.5	-0.0
50 MESH	0.04	0.04	0.1	-0.0	3.29	3.16	0.5	-0.1
65 MESH	0.15	0.15	0.1	-0.0	5.16	5.03	0.5	-0.1
100 MESH	1.49	1.48	0.5	-0.0	7.04	7.00	0.5	-0.0
150 MESH	3.48	3.48	0.5	0.0	8.64	8.64	0.5	0.0
200 MESH	5.58	5.74	0.5	0.2	14.45	15.03	0.5	0.6*
270 MESH	7.49	7.22	0.5	-0.3	17.56	16.57	0.5	-1.0*
400 MESH	11.60	11.55	0.5	-0.1	14.62	14.43	0.5	-0.2
500 MESH	10.10	10.08	0.5	-0.0	5.46	5.40	0.5	-0.1

Appendix B.9 Selection Function Estimation Results of Louvicourt Mine

****
Selection Function Estimation Results
****

Louvicourt Mine, Ball Mill, Feb. 25, 97
Date: 4/25/1997
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Breakage Function Matrix

0.00														
0.44	0.00													
0.19	0.44	0.00												
0.09	0.19	0.44	0.00											
0.05	0.09	0.19	0.44	0.00										
0.03	0.05	0.09	0.19	0.44	0.00									
0.03	0.03	0.05	0.09	0.19	0.44	0.00								
0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00							
0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00						
0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00					
0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00				
0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00			
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00		
0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00	
0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00

CLASS	SEIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	3350	0.76	0.30	0.30	1.0413
2	2369	1.03	0.39	0.39	1.4444
3	1675	1.17	0.50	0.50	1.5695
4	1184	1.47	0.61	0.61	1.7614
5	837	1.33	0.34	0.34	3.4604
6	592	1.78	0.56	0.56	2.8361
7	419	2.59	0.92	0.92	2.4023
8	296	3.16	1.48	1.48	1.9129
9	209	5.03	3.16	3.16	1.1637
10	148	7.00	5.43	5.43	0.7651

11	105	8.64	7.73	7.73	0.5322
12	74	15.03	14.77	14.77	0.2540
13	52	16.57	16.81	16.81	0.1913
14	37	14.43	16.02	16.02	0.1112
15	26	5.40	7.05	7.05	0.1441

Appendix B.10 Scaling Selection Function of Louvicourt Mine

****

Ball Size Optimization Program Data

****

Louvicourt Mine, Ball Mill, Feb. 25, 97
Date: 4/27/1997
Current ball size = 38.00 mm.
New ball size = 25.40 mm.
K = 0.000440 (1/mm.)

CLASS	SCREEN SIZE	PARTICLE SIZE	EST.SEL.FUNC.	STD.DEV.	SCALED SEL.FUNC.
1	3350	3984	1.0413	1.0000	0.4653
2	2369	2817	1.4444	1.0000	0.6454
3	1675	1992	1.5695	1.0000	0.7012
4	1184	1409	1.7614	1.0000	0.7870
5	837	996	3.4604	1.0000	1.5461
6	592	704	2.8361	1.0000	1.2671
7	419	498	2.4023	1.0000	0.0001
8	296	352	1.9129	1.0000	0.0001
9	209	249	1.1637	1.0000	1.7410
10	148	176	0.7651	1.0000	1.1446
11	105	124	0.5322	1.0000	0.7962
12	74	88	0.2540	1.0000	0.3800
13	52	62	0.1913	1.0000	0.2862
14	37	44	0.1112	1.0000	0.1663
15	26	31	0.1441	1.0000	0.2155

****

Ball Size Optimization Program Results

****

Weighted SSR for the first curve: 0.125143
Weighted SSR for the second curve: 0.162549

CLASS	SCREEN SIZE	PARTICLE SIZE	CALC.EST.SEL.FUNC.	CALC.SCALED SEL.FUNC.
1	3350	3984	0.8165	0.3997
2	2369	2817	1.4599	0.6096
3	1675	1992	2.1424	0.8504
4	1184	1409	2.6359	1.0887
5	837	996	2.7783	1.2832
6	592	704	2.5629	1.3969
7	419	498	2.1140	1.4090
8	296	352	1.5929	1.3213
9	209	249	1.1204	1.1554
10	148	176	0.7514	0.9454
11	105	124	0.4910	0.7261
12	74	88	0.3193	0.5251
13	52	62	0.2112	0.3588
14	37	44	0.1452	0.2323
15	26	31	0.1059	0.1431

Appendix B.11 Mass Balancing Results of Cu Regrind Circuit of Louvicourt Mine

Louvicourt Mine, Survey of Cu Regrind Circuit, February 25 1997

Residual sum of squares: 1.75509 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	ClnrF	100.00	100.0	100.0	0.0	0.0
2	ClnrScvrCo	76.38	80.0	76.4	80.0	-3.6
3	FreshFeed					
4	BMD	156.08	200.0	156.1	200.0	-43.9
5	CUF	156.08		156.1		
6	COF	176.38		176.4		

	Stream	Relative Solids Flowrate
1	ClnrF	100.00
2	ClnrScvrCo	76.38
4	BMD	156.08
5	CUF	156.08
6	COF	176.38

Fractional Size Distribution Data

Size	ClnrF Meas	Calc	SD.	Adj.	ClnrScvrCo Meas	Calc	SD.	Adj.
100 MESH	0.00	0.00	0.1	0.0	0.00	0.00	0.1	0.0
150 MESH	2.19	2.16	0.5	-0.0	0.05	0.05	0.1	-0.0
200 MESH	5.19	5.17	0.5	-0.0	0.32	0.32	0.2	-0.0
270 MESH	10.79	10.68	0.5	-0.1	0.99	0.90	0.5	-0.1
400 MESH	14.41	14.46	0.5	0.0	1.74	1.78	0.5	0.0
500 MESH	15.64	15.73	0.5	0.1	4.64	4.71	0.5	0.1
635 MESH	8.42	8.26	0.5	-0.2	5.60	5.48	0.5	-0.1

Size	BMD Meas	Calc	SD.	Adj.	CUF Meas	Calc	SD.	Adj.
100 MESH	0.00	0.00	0.1	0.0	0.00	0.00	0.1	0.0
150 MESH	0.82	0.78	0.4	-0.0	2.10	2.15	0.5	0.1
200 MESH	1.11	1.07	0.5	-0.0	4.40	4.44	0.5	0.0
270 MESH	3.05	2.87	0.5	-0.2	9.67	9.85	0.5	0.2
400 MESH	5.19	5.27	0.5	0.1	14.95	14.87	0.5	-0.1
500 MESH	12.74	12.88	0.5	0.1	22.96	22.82	0.5	-0.1
635 MESH	10.71	10.46	0.5	-0.2	13.71	13.96	0.5	0.2

Size	COF Meas	Calc	SD.	Adj.
100 MESH	0.00	0.00	0.1	0.0
150 MESH	0.03	0.03	0.1	0.0
200 MESH	0.09	0.09	0.1	0.0
270 MESH	0.25	0.27	0.2	0.0
400 MESH	0.49	0.46	0.3	-0.0
500 MESH	2.31	2.15	0.5	-0.2
635 MESH	3.69	3.97	0.5	0.3

Appendix B.12 Selection Function Estimation Results of Cu Regrind Circuit of Louvicourt Mine

****
Selection Function Estimation Results
****

LM, Cu regrind circuit, Feb. 25,97
Date: 6/15/1997
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Breakage Function Matrix

0.00					
0.44	0.00				
0.19	0.44	0.00			
0.09	0.19	0.44	0.00		
0.05	0.09	0.19	0.44	0.00	
0.04	0.05	0.09	0.19	0.44	0.00

CLASS	SEIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	106	2.15	0.78	0.78	1.2882
2	75	4.44	1.07	1.07	2.3322
3	53	9.85	2.87	2.87	2.1077
4	37	14.87	5.27	5.27	1.9784
5	26	22.82	12.88	12.88	1.2044
6	19	13.96	10.46	10.46	1.3384

Appendix B.13 Mass Balancing Results of Zn Regrind Circuit of Louvicourt Mine

Louvicourt Mine, Survey of Zn Regrind Circuit, February 25 1997

Residual sum of squares: 3.902023 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	RghrScvrCo	100.00	100.0	100.0	0.0	0.0
2	ClnrScvrCo	16.05	20.0	16.0	20.0	-4.0
3	FreshFeed					
4	BMD	330.51	200.0	330.5	200.0	130.5
5	CUF	330.51		330.5		
6	COF	116.05		116.0		

	Stream	Relative Solids Flowrate
1	RghrScvrCo	100.00
2	ClnrConc	16.05
4	BMD	330.51
5	CUF	330.51
6	COF	116.05

Fractional Size Distribution Data

Size	RghrScvrCo Meas	Calc	SD.	Adj.	ClnrConc Meas	Calc	SD.	Adj.
100 MESH	0.00	0.00	0.1	0.0	0.00	0.00	0.1	0.0
150 MESH	1.00	0.97	0.5	-0.0	0.04	0.04	0.1	-0.0
200 MESH	1.79	1.81	0.5	0.0	0.40	0.40	0.2	0.0
270 MESH	7.02	6.87	0.5	-0.2	0.81	0.79	0.4	-0.0
400 MESH	10.40	10.45	0.5	0.0	0.71	0.71	0.4	0.0
500 MESH	12.55	12.53	0.5	-0.0	2.87	2.87	0.5	-0.0
635 MESH	8.74	8.83	0.5	0.1	2.69	2.71	0.5	0.0

Size	BMD Meas	Calc	SD.	Adj.	CUF Meas	Calc	SD.	Adj.
100 MESH	0.00	0.00	0.1	0.0	0.00	0.00	0.1	0.0
150 MESH	1.17	1.08	0.5	-0.1	1.29	1.38	0.5	0.1
200 MESH	1.52	1.59	0.5	0.1	2.10	2.03	0.5	-0.1
270 MESH	3.16	2.65	0.5	-0.5*	4.04	4.55	0.5	0.5*
400 MESH	6.80	6.96	0.5	0.2	9.95	9.79	0.5	-0.2
500 MESH	17.48	17.40	0.5	-0.1	19.95	20.03	0.5	0.1
635 MESH	12.07	12.38	0.5	0.3	13.57	13.26	0.5	-0.3

Size	COF Meas	Calc	SD.	Adj.
100 MESH	0.00	0.00	0.1	0.0
150 MESH	0.01	0.01	0.1	0.0
200 MESH	0.34	0.34	0.2	-0.0
270 MESH	0.57	0.64	0.3	0.1
400 MESH	1.11	1.05	0.5	-0.1
500 MESH	3.70	3.73	0.5	0.0
635 MESH	5.60	5.49	0.5	-0.1

Appendix B.14 Selection Function Estimation Results of Zn Regrind Circuit of Louvicourt Mine

****
Selection Function Estimation Results
****

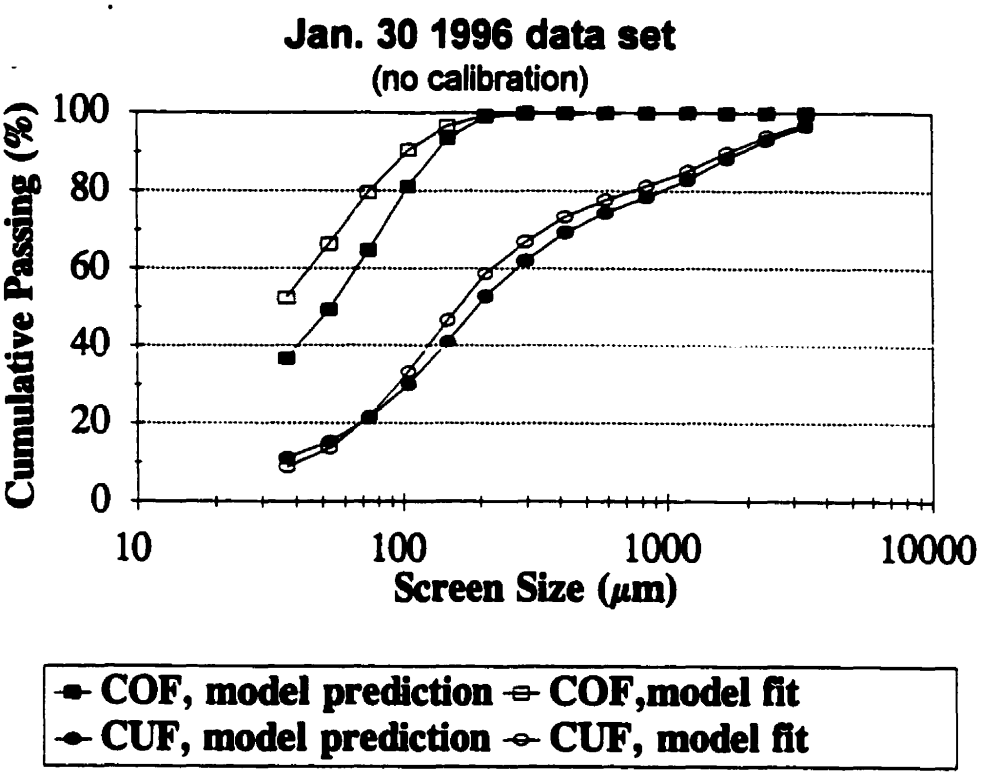
LM, Zinc circuit, Feb. 25, 97
Date: 6/15/1997
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

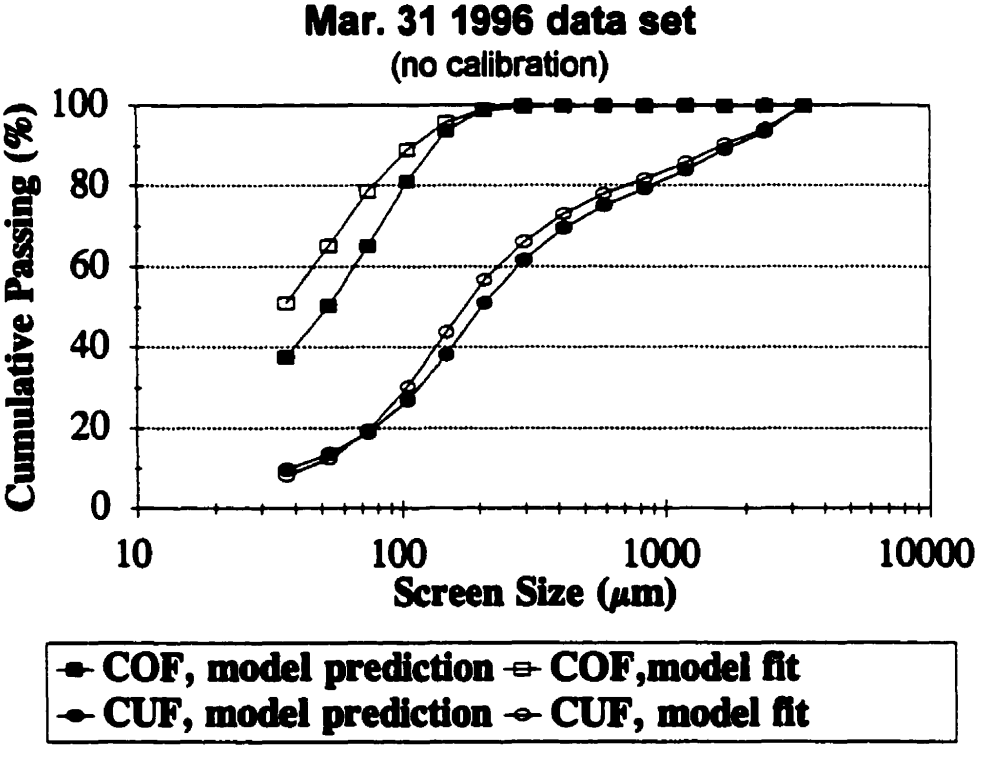
Breakage Function Matrix

0.00					
0.44	0.00				
0.19	0.44	0.00			
0.09	0.19	0.44	0.00		
0.05	0.09	0.19	0.44	0.00	
0.04	0.05	0.09	0.19	0.44	0.00

CLASS	SIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	106	1.38	1.08	1.08	0.2607
2	75	2.03	1.59	1.59	0.3373
3	53	4.55	2.65	2.65	0.7179
4	38	9.79	6.96	6.96	0.5157
5	25	20.03	17.40	17.40	0.2682
6	20	13.26	12.38	12.38	0.3161

Appendix C.1 Predicted and Measured Size Distribution of Cyclopak Overflow and Underflow Streams of Lupin Mine


Jan. 30 1996 data set
(no calibration)
Cumulative Passing (%)
100
80
60
40
20
0
10
100
1000
10000
Screen Size (μm)
COF, model prediction
COF,model fit
CUF, model prediction
CUF, model fit


Mar. 31 1996 data set
(no calibration)
Cumulative Passing (%)
100
80
60
40
20
0
10
100
1000
10000
Screen Size (μm)
COF, model prediction
COF,model fit
CUF, model prediction
CUF, model fit

Appendix C.2 Simulation Results of Cyclopaks at Lupin

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Simulation of Lupin cyclopak data, Jan. 30, 1996, Circuit #2
  NO CALIBRATION

[KNOWN-STREAMS]
  1
  1     320.15      53.01      283.79

[KNOWN-SIZE-DISTRIBUTIONS]
  15
  1
  1
  1     3360      2.09
  2     2380      2.32
  3     1680      3.03
  4     1190      3.5
  5      841      2.84
  6      595      2.61
  7      420      3.18
  8      297      4.79
  9      210      6.28
 10      149      9.55
 11      105      11.40
 12      74       11.42
 13      53        9.39
 14      37        7.24
 15      25       20.36


[CYCLOPAK-1]
  38.1
  9.525
  10.16
  6.98
  119.38
  3.2
  4
  1     1       1       1       1
  100
```

Connectivity Matrix

```
   1    2    1    1   -2   -3
```

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	CYCF	320.150	53.010	283.790
2	COF	114.144	37.970	186.450
3	CUF	206.006	67.910	97.340

Size Distributions

CLASS	SIZE	CYCF	COF	CUF
1	3360	2.09	0.00	3.25
2	2380	2.32	0.00	3.61
3	1680	3.03	0.00	4.71
4	1190	3.50	0.00	5.44
5	841	2.84	0.00	4.41
6	595	2.61	0.00	4.06
7	420	3.18	0.00	4.94
8	297	4.79	0.04	7.42
9	210	6.28	0.97	9.22
10	149	9.55	5.51	11.79
11	105	11.40	12.31	10.90
12	74	11.42	16.45	8.63
13	53	9.39	15.38	6.07
14	37	7.24	12.65	4.24
15	25	20.36	36.69	11.31

Plitt's Model Parameters

CYCLOPAK-1

d50c = 118 µm
P = 36.650 kPa
S = 0.730
m = 2.210
Rf = 0.340

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Simulation of Lupin cyclopak data, Jan. 30, 1996, Circuit #2
  WITH CALIBRATION

[KNOWN-STREAMS]
  1
  1      320.15       53.01      283.79

[KNOWN-SIZE-DISTRIBUTIONS]
  15
  1
  1
  1      3360      2.09
  2      2380      2.32
  3      1680      3.03
  4      1190      3.5
  5       841      2.84
  6       595      2.61
  7       420      3.18
  8       297      4.79
  9       210      6.28
 10       149      9.55
 11       105      11.40
 12       74       11.42
 13       53        9.39
 14       37        7.24
 15       25        20.36


[CYCLOPAK-1]
  38.1
  9.525
  10.16
  6.98
  119.38
  3.2
  4
  .437    1        1        .56        .594
  100
```

--

Connectivity Matrix

--

```
   1    2    1    1   -2   -3
```

--

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	CYCF	320.150	53.010	283.790
2	COF	70.776	23.220	234.020
3	CUF	249.374	83.360	49.770

Size Distributions

CLASS	SIZE	CYCF	COF	CUF
1	3360	2.09	0.00	2.68
2	2380	2.32	0.00	2.98
3	1680	3.03	0.00	3.89
4	1190	3.50	0.00	4.49
5	841	2.84	0.00	3.65
6	595	2.61	0.00	3.35
7	420	3.18	0.00	4.08
8	297	4.79	0.04	6.14
9	210	6.28	0.38	7.95
10	149	9.55	2.30	11.61
11	105	11.40	6.82	12.70
12	74	11.42	12.31	11.17
13	53	9.39	14.62	7.91
14	37	7.24	14.59	5.16
15	25	20.36	48.94	12.25

Plitt's Model Parameters

CYCLOPAK-1

d50c = 52 μm
P = 36.650 kPa
S = 0.730
m = 1.240
Rf = 0.180

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Simulation of Lupin cyclopak data, Mar. 1, 1996, Circuit #2
  NO CALIBRATION

[KNOWN-STREAMS]
  1
  1     304.83       53.64       263.56

[KNOWN-SIZE-DISTRIBUTIONS]
  15
  1
  1
  1      3360      2.9
  2      2380      2.68
  3      1680      3.03
  4      1190      3.37
  5       841      2.78
  6       595      2.6
  7       420      3.20
  8       297      4.82
  9       210      6.23
 10       149      7.47
 11       105      12.77
 12       74       11.01
 13       53        9.12
 14       37        7.17
 15       25        20.85


[CYCLOPAK-1]
  38.1
  9.525
  10.16
  6.98
  119.38
  3.2
  4
  1    1        1        1        1
  100
```

Connectivity Matrix

```
   1    2    1    1   -2   -3
```

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	CYCF	304.830	53.640	263.560
2	COF	110.091	39.190	170.840
3	CUF	194.739	67.740	92.720

Size Distributions

CLASS	SIZE	CYCF	COF	CUF
1	3360	2.90	0.00	4.54
2	2380	2.68	0.00	4.20
3	1680	3.03	0.00	4.74
4	1190	3.37	0.00	5.28
5	841	2.78	0.00	4.35
6	595	2.60	0.00	4.07
7	420	3.20	0.00	5.01
8	297	4.82	0.08	7.50
9	210	6.23	1.25	9.04
10	149	7.47	4.81	8.98
11	105	12.77	14.28	11.92
12	74	11.01	15.88	8.26
13	53	9.12	14.74	5.94
14	37	7.17	12.27	4.28
15	25	20.85	36.68	11.90

Plitt's Model Parameters

CYCLOPAK-1

d50c = 125 μm
P = 33.280 kPa
S = 0.750
m = 2.210
Rf = 0.350

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Simulation of Lupin cyclopak data, Mar. 1, 1996, Circuit #2
  WITH CALIBRATION

[KNOWN-STREAMS]
  1
  1      304.83       53.64       263.56

[KNOWN-SIZE-DISTRIBUTIONS]
  15
  1
  1
  1      3360      2.9
  2      2380      2.68
  3      1680      3.03
  4      1190      3.37
  5       841      2.78
  6       595      2.6
  7       420      3.20
  8       297      4.82
  9       210      6.23
 10       149      7.47
 11       105      12.77
 12       74       11.01
 13       53        9.12
 14       37        7.17
 15       25        20.85


[CYCLOPAK-1]
  38.1
  9.525
  10.16
  6.98
  119.38
  3.2
  4
  .437    1        1        .56       .594
  100
```

--

Connectivity Matrix

--

```
   1    2    1    1   -2   -3
```

--

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	CYCF	304.830	53.640	263.560
2	COF	69.785	24.420	215.960
3	CUF	235.045	83.160	47.600

Size Distributions

CLASS	SIZE	CYCF	COF	CUF
1	3360	2.90	0.00	3.76
2	2380	2.68	0.00	3.48
3	1680	3.03	0.00	3.93
4	1190	3.37	0.00	4.37
5	841	2.78	0.00	3.61
6	595	2.60	0.00	3.37
7	420	3.20	0.00	4.15
8	297	4.82	0.05	6.24
9	210	6.23	0.48	7.94
10	149	7.47	2.05	9.08
11	105	12.77	8.17	14.14
12	74	11.01	12.18	10.66
13	53	9.12	14.20	7.61
14	37	7.17	14.20	5.08
15	25	20.85	48.68	12.59

Plitt's Model Parameters

CYCLOPAK-1

d50c = 55 μm
P = 33.280 kPa
S = 0.750
m = 1.240
Rf = 0.180

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Simulation of Lupin cyclopak data, Mar. 31, 1996, Circuit #2
  NO CALIBRATION

[KNOWN-STREAMS]
  1
  1      362.67      53.11      320.2

[KNOWN-SIZE-DISTRIBUTIONS]
  16
  1
  1
  1      3360      0
  2      2380      4.32
  3      1680      2.76
  4      1190      3.46
  5       841      2.96
  6       595      2.74
  7       420      3.62
  8       297      5.19
  9       210      7.20
 10       149      10.33
 11       105      11.71
 12       74       10.75
 13       53        8.6
 14       37        6.94
 15       25        5.31
 16       13        14.11

[CYCLOPAK-1]
  38.1
  9.525
  10.16
  6.98
  119.38
  3.2
  4
  1    1       1       1       1
  100
```

--

Connectivity Matrix

--

```
   1    2    1    1   -2   -3
```

--

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	CYCF	362.670	53.110	320.200
2	COF	125.788	36.690	217.070
3	CUF	236.882	69.670	103.130

Size Distributions

CLASS	SIZE	CYCF	COF	CUF
1	3360	0.00	0.00	0.00
2	2380	4.32	0.00	6.61
3	1680	2.76	0.00	4.23
4	1190	3.46	0.00	5.30
5	841	2.96	0.00	4.53
6	595	2.74	0.00	4.19
7	420	3.62	0.00	5.54
8	297	5.19	0.03	7.93
9	210	7.20	0.87	10.56
10	149	10.33	5.50	12.89
11	105	11.71	12.58	11.25
12	74	10.75	15.95	7.99
13	53	8.60	14.74	5.34
14	37	6.94	12.78	3.84
15	25	5.31	10.12	2.75
16	13	14.11	27.42	7.04

Plitt's Model Parameters

CYCLOPAK-1

d50c = 112 μm
P = 45.690 kPa
S = 0.690
m = 2.220
Rf = 0.320

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Simulation of Lupin cyclopak data, Mar. 31, 1996, Circuit #2
  WITH CALIBRATION

[KNOWN-STREAMS]
  1
  1      362.67      53.11       320.2

[KNOWN-SIZE-DISTRIBUTIONS]
  16
  1
  1
  1      3360      0
  2      2380      4.32
  3      1680      2.76
  4      1190      3.46
  5       841      2.96
  6       595      2.74
  7       420      3.62
  8       297      5.19
  9       210      7.20
 10       149      10.33
 11       105      11.71
 12       74       10.75
 13       53        8.6
 14       37        6.94
 15       25        5.31
 16       13        14.11

[CYCLOPAK-1]
  38.1
  9.525
  10.16
  6.98
  119.38
  3.2
  4
  .437     1        1        .56        .594
  100
```

--

Connectivity Matrix

--

```
   1    2    1    1   -2   -3
```

--

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	CYCF	362.670	53.110	320.200
2	COF	80.337	23.100	267.380
3	CUF	282.333	84.240	52.820

Size Distributions

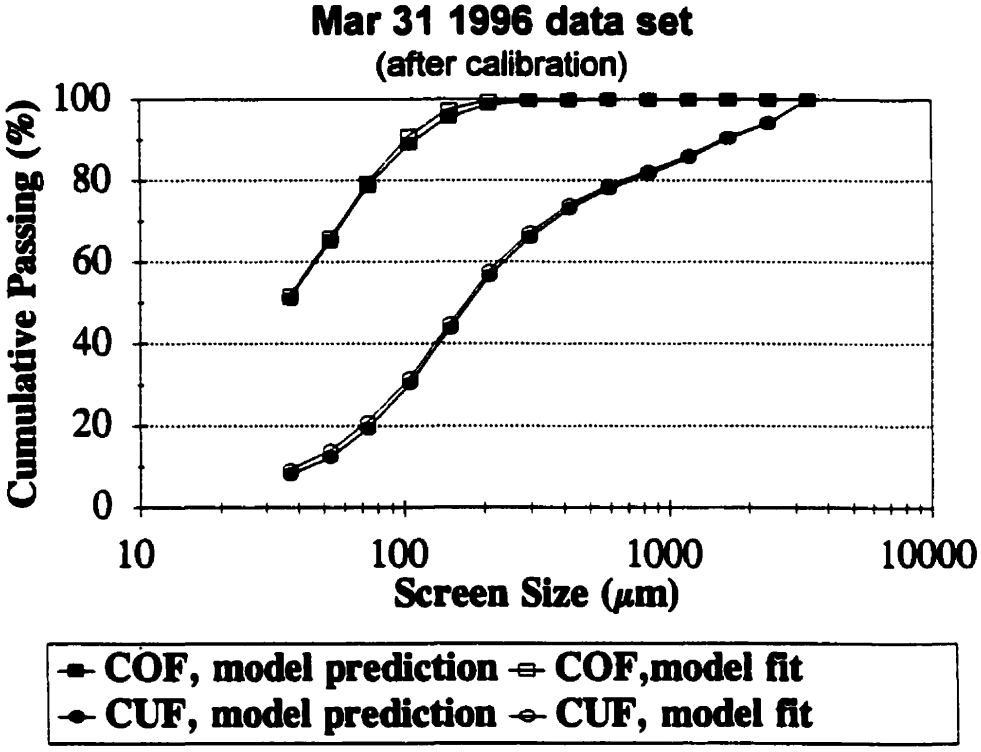
CLASS	SIZE	CYCF	COF	CUF
1	3360	0.00	0.00	0.00
2	2380	4.32	0.00	5.55
3	1680	2.76	0.00	3.55
4	1190	3.46	0.00	4.44
5	841	2.96	0.00	3.80
6	595	2.74	0.00	3.52
7	420	3.62	0.00	4.65
8	297	5.19	0.03	6.66
9	210	7.20	0.34	9.15
10	149	10.33	2.14	12.66
11	105	11.71	6.41	13.22
12	74	10.75	11.04	10.67
13	53	8.60	13.07	7.33
14	37	6.94	13.88	4.96
15	25	5.31	12.82	3.17
16	13	14.11	40.27	6.67

Plitt's Model Parameters

CYCLOPAK-1

d50c = 49 µm
P = 45.690 kPa
S = 0.690
m = 1.240
Rf = 0.160

Appendix C.3 Improved Predictions of Cyclopaks Overflow and Underflow at Lupin Mine


Mar 31 1996 data set
(after calibration)
Cumulative Passing (%)
100
80
60
40
20
0
10
100
1000
10000
Screen Size (μm)
COF, model prediction
COF,model fit
CUF, model prediction
CUF, model fit

Appendix C.4 Circuit Simulation Results of Louvicourt Mine

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Louvicourt Mill's Grinding Circuit, Simulation using Oct. 10, 95 data set.
  Circuit Type: #6

[KNOWN-STREAMS]
  2
  1     217.36      41.4       307.7
  2      0        0             80

[KNOWN-SIZE-DISTRIBUTIONS]
  16
  1
  1
  1     3984     4.03
  2     2807     2.54
  3     2022     2.80
  4     1403     3.69
  5     1011     4.40
  6     714      4.77
  7     505      5.59
  8     357      5.71
  9     252      7.35
  10    178      6.39
  11    126      6.15
  12    89       6.34
  13    63       5.60
  14    45       2.65
  15    30       2.61
  16    15       29.38

[BALLMILL-1]
  10.6292
  4.1295
  2.6330
  3.0827
  2.8999
  2.8324
  2.2264
  2.0337
  1.4206
  1.0444
  0.6199
  0.3950
  0.2866
  0.4083
  0.3936
```

```
  0.0000
  0.4380 0.0000
  0.1920 0.4360  0.0000
  0.0860 0.1750  0.4050  0.0000
  0.0530 0.0940  0.1920  0.4080  0.0000
  0.0340 0.0540  0.0950  0.1790  0.3830  0.0000
  0.0280 0.0400  0.0630  0.1050  0.1980  0.3920  0.0000
  0.0220 0.0300  0.0430  0.0640  0.1060  0.1880  0.3770  0.0000
  0.0200 0.0260  0.0350  0.0490  0.0740  0.1180  0.2110  0.3980  0.0000
  0.0170 0.0210  0.0280  0.0370  0.0520  0.0760  0.1230  0.2100  0.4080
0.0000
  0.0140 0.0170  0.0220  0.0280  0.0380  0.0520  0.0770  0.1220  0.2160
0.4220  0.0000
  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850  0.1380
0.2470  0.4890  0.0000
  0.0120 0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850
0.1380  0.2470  0.4890  0.0000
  0.0110 0.0120  0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000
  0.0080 0.0110  0.0120  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000

  711.88  0.1  0.1  0.7

[CYCLOPAK-1]
  38.1
  9.525
  7.62
  6.98
  119.38
  3.55
  10
  1     1     1       1       1
  100

[CONVERGE-1]
  0.01
```

Connectivity Matrix

1	3	1	1	1000	-1	0	0	0	1
2	2	1	0	0	1	-2	-3	0	0
3	1	1	0	0	0	0	1	-1	0
4	100	1	0	0	0	0	0	1	-1

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)

1	FF	217.360	41.400	307.700
3	CF	940.116	52.910	836.590
4	COF	217.305	35.920	387.730
5	CUF	722.811	61.690	448.860
6	BMD	722.811	61.690	448.860

Size Distributions

CLASS	SIZE	FF	CF	COF	CUF	BMD
1	3984	4.03	0.94	0.00	1.22	0.01
2	2807	2.54	0.70	0.00	0.91	0.15
3	2022	2.80	0.97	0.00	1.26	0.42
4	1403	3.69	1.21	0.00	1.57	0.46
5	1011	4.40	1.52	0.00	1.98	0.65
6	714	4.77	1.72	0.00	2.24	0.80
7	505	5.59	2.34	0.00	3.04	1.36
8	357	5.71	2.66	0.00	3.46	1.74
9	252	7.35	4.31	0.01	5.60	3.40
10	178	6.39	5.59	0.26	7.19	5.35
11	126	6.15	8.89	2.14	10.92	9.71
12	89	6.34	12.82	7.80	14.33	14.77
13	63	5.60	12.73	13.02	12.64	14.88
14	45	2.65	7.42	10.12	6.61	8.85
15	30	2.61	6.25	10.29	5.04	7.35
16	15	29.38	29.93	56.37	21.98	30.09

Plitt's Model Parameters

CYCLOPAK-1

d50c = 64 µm
P = 62.760 kPa
S = 1.450
m = 1.660
Rf = 0.540

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

[PROJECT-TITLE]
Louvicourt Mill's Grinding Circuit, testing simulator using Feb. 25, 97 data set.
Circuit Type: #6

[KNOWN-STREAMS]
2
1 176.44 49.47 180.22
2 0 0 80

[KNOWN-SIZE-DISTRIBUTIONS]
16
1
1
1 3984 1.68
2 2807 2.32
3 2022 2.43
4 1403 3.14
5 1011 3.64
6 714 4.44
7 505 6.08
8 357 6.13
9 252 6.92
10 178 7.19
11 126 6.80
12 89 6.70
13 63 6.31
14 45 5.76
15 30 4.07
16 15 26.39

[BALLMILL-1]
0.8165
1.4599
2.1424
2.6359
2.7783
2.5629
2.1140
1.5929
1.1204
0.7514
0.4910
0.3193
0.2112
0.1452
0.1059

```
  OFF

  0.0000
  0.4380 0.0000
  0.1920 0.4360  0.0000
  0.0860 0.1750  0.4050  0.0000
  0.0530 0.0940  0.1920  0.4080  0.0000
  0.0340 0.0540  0.0950  0.1790  0.3830  0.0000
  0.0280 0.0400  0.0630  0.1050  0.1980  0.3920  0.0000
  0.0220 0.0300  0.0430  0.0640  0.1060  0.1880  0.3770  0.0000
  0.0200 0.0260  0.0350  0.0490  0.0740  0.1180  0.2110  0.3980  0.0000
  0.0170 0.0210  0.0280  0.0370  0.0520  0.0760  0.1230  0.2100  0.4080
0.0000
  0.0140 0.0170  0.0220  0.0280  0.0380  0.0520  0.0770  0.1220  0.2160
0.4220  0.0000
  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850  0.1380
0.2470  0.4890  0.0000
  0.0120 0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850
0.1380  0.2470  0.4890  0.0000
  0.0110 0.0120  0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000
  0.0080 0.0110  0.0120  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000

  711.88  0.1  0.1  0.7

[CYCLOPAK-1]
  38.1
  9.525
  7.62
  6.98
  119.38
  3.55
  10
  0.274     1     1        0.355        0.759
  100

[CONVERGE-1]
  0.01
```

Connectivity Matrix

```
   1    3    1    1 1000   -1    0    0    0    1
   2    2    1    0    0    1   -2   -3    0    0
   3    1    1    0    0    0    0    1   -1    0
   4  100    1    0    0    0    0    0    1   -1
```

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	FF	176.440	49.470	180.220
3	CF	888.818	66.710	443.470
4	COF	176.383	40.400	260.230
5	CUF	712.435	79.540	183.240
6	BMD	712.435	79.540	183.240

Size Distributions

CLASS	SIZE	FF	CF	COF	CUF	BMD
1	3984	1.68	0.66	0.00	0.82	0.41
2	2807	2.32	0.77	0.00	0.96	0.39
3	2022	2.43	0.76	0.00	0.95	0.35
4	1403	3.14	0.91	0.01	1.13	0.36
5	1011	3.64	1.08	0.04	1.34	0.44
6	714	4.44	1.36	0.10	1.67	0.60
7	505	6.08	2.04	0.27	2.48	1.04
8	357	6.13	2.56	0.56	3.05	1.68
9	252	6.92	3.79	1.27	4.41	3.02
10	178	7.19	5.51	2.61	6.23	5.09
11	126	6.80	7.59	4.75	8.29	7.78
12	89	6.70	10.59	8.37	11.14	11.55
13	63	6.31	12.14	11.60	12.27	13.58
14	45	5.76	11.91	13.25	11.58	13.43
15	30	4.07	9.50	12.26	8.82	10.85
16	15	26.39	28.82	44.86	24.85	29.42

Plitt's Model Parameters

CYCLOPAK-1

d50c = 42 μm
P = 44.500 kPa
S = 1.760
m = 0.570
Rf = 0.410

Appendix C.5 Simulation of Proposed Modifications, Louvicourt Mine

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Louvicourt Mill's Grinding Circuit, Predictions with 1" balls.
  Circuit Type: #6

[KNOWN-STREAMS]
  2
  1      176.44      49.47       180.22
  2       0         0            80

[KNOWN-SIZE-DISTRIBUTIONS]
  16
  1
  1
  1      3984     1.68
  2      2807     2.32
  3      2022     2.43
  4      1403     3.14
  5      1011     3.64
  6      714      4.44
  7      505      6.08
  8      357      6.13
  9      252      6.92
  10     178      7.19
  11     126      6.80
  12     89       6.70
  13     63       6.31
  14     45       5.76
  15     30       4.07
  16     15       26.39

[BALLMILL-1]
  0.3997
  0.6096
  0.8504
  1.0887
  1.2832
  1.3969
  1.4090
  1.3213
  1.1554
  0.9454
  0.7261
  0.5251
  0.3588
  0.2323
  0.1431
```

```
  0.0000
  0.4380 0.0000
  0.1920 0.4360  0.0000
  0.0860 0.1750  0.4050  0.0000
  0.0530 0.0940  0.1920  0.4080  0.0000
  0.0340 0.0540  0.0950  0.1790  0.3830  0.0000
  0.0280 0.0400  0.0630  0.1050  0.1980  0.3920  0.0000
  0.0220 0.0300  0.0430  0.0640  0.1060  0.1880  0.3770  0.0000
  0.0200 0.0260  0.0350  0.0490  0.0740  0.1180  0.2110  0.3980  0.0000
  0.0170 0.0210  0.0280  0.0370  0.0520  0.0760  0.1230  0.2100  0.4080
0.0000
  0.0140 0.0170  0.0220  0.0280  0.0380  0.0520  0.0770  0.1220  0.2160
0.4220  0.0000
  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850  0.1380
0.2470  0.4890  0.0000
  0.0120 0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850
0.1380  0.2470  0.4890  0.0000
  0.0110 0.0120  0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000
  0.0080 0.0110  0.0120  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000

  711.88  0.1  0.1  0.7

[CYCLOPAK-1]
  38.1
  9.525
  7.62
  6.98
  119.38
  3.55
  10
  0.274     1     1       0.355       0.759
  100

[CONVERGE-1]
  0.01
```

Connectivity Matrix

1	3	1	1	1000	-1	0	0	0	1
2	2	1	0	0	1	-2	-3	0	0
3	1	1	0	0	0	0	1	-1	0
4	100	1	0	0	0	0	0	1	-1

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)

1	FF	176.440	49.470	180.220
3	CF	888.226	66.690	443.550
4	COF	176.374	40.400	260.220
5	CUF	711.852	79.520	183.330
6	BMD	711.852	79.520	183.330

Size Distributions

CLASS	SIZE	FF	CF	COF	CUF	BMD
1	3984	1.68	1.07	0.00	1.34	0.93
2	2807	2.32	1.34	0.00	1.67	1.10
3	2022	2.43	1.32	0.01	1.65	1.05
4	1403	3.14	1.47	0.02	1.83	1.06
5	1011	3.64	1.64	0.05	2.03	1.14
6	714	4.44	1.91	0.14	2.35	1.28
7	505	6.08	2.57	0.34	3.12	1.70
8	357	6.13	2.89	0.63	3.45	2.09
9	252	6.92	3.72	1.24	4.33	2.93
10	178	7.19	4.68	2.21	5.29	4.06
11	126	6.80	5.81	3.63	6.35	5.57
12	89	6.70	7.92	6.25	8.33	8.22
13	63	6.31	9.80	9.35	9.91	10.67
14	45	5.76	11.09	12.32	10.79	12.42
15	30	4.07	10.38	13.38	9.64	11.95
16	15	26.39	32.37	50.32	27.92	33.84

Plitt's Model Parameters

CYCLOPAK-1

d50c = 42 µm
P = 44.460 kPa
S = 1.760
m = 0.570
Rf = 0.410

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Louvicourt Mill's Grinding Circuit, testing simulator using smaller apex.
  Circuit Type: #6

[KNOWN-STREAMS]
  2
  1      176.44      49.47       180.22
  2       0          0           80

[KNOWN-SIZE-DISTRIBUTIONS]
  16
  1
  1
  1      3984     1.68
  2      2807     2.32
  3      2022     2.43
  4      1403     3.14
  5      1011     3.64
  6      714      4.44
  7      505      6.08
  8      357      6.13
  9      252      6.92
  10     178      7.19
  11     126      6.80
  12     89       6.70
  13     63       6.31
  14     45       5.76
  15     30       4.07
  16     15       26.39

[BALLMILL-1]
  0.8165
  1.4599
  2.1424
  2.6359
  2.7783
  2.5629
  2.1140
  1.5929
  1.1204
  0.7514
  0.4910
  0.3193
  0.2112
  0.1452
  0.1059
```

```
  0.0000
  0.4380 0.0000
  0.1920 0.4360  0.0000
  0.0860 0.1750  0.4050  0.0000
  0.0530 0.0940  0.1920  0.4080  0.0000
  0.0340 0.0540  0.0950  0.1790  0.3830  0.0000
  0.0280 0.0400  0.0630  0.1050  0.1980  0.3920  0.0000
  0.0220 0.0300  0.0430  0.0640  0.1060  0.1880  0.3770  0.0000
  0.0200 0.0260  0.0350  0.0490  0.0740  0.1180  0.2110  0.3980  0.0000
  0.0170 0.0210  0.0280  0.0370  0.0520  0.0760  0.1230  0.2100  0.4080
0.0000
  0.0140 0.0170  0.0220  0.0280  0.0380  0.0520  0.0770  0.1220  0.2160
0.4220  0.0000
  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850  0.1380
0.2470  0.4890  0.0000
  0.0120 0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850
0.1380  0.2470  0.4890  0.0000
  0.0110 0.0120  0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000
  0.0080 0.0110  0.0120  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000

  711.88  0.1  0.1  0.7

[CYCLOPAK-1]
  38.1
  9.525
  7.62
  5.715
  119.38
  3.55
  10
  0.274     1     1       0.355       0.759
  100

[CONVERGE-1]
  0.01
```

Connectivity Matrix

```
   1    3    1    1 1000   -1    0    0    0    1
   2    2    1    0    0    1   -2   -3    0    0
   3    1    1    0    0    0    0    1   -1    0
   4  100    1    0    0    0    0    0    1   -1
```

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	FF	176.440	49.470	180.220
3	CF	637.026	64.100	356.760
4	COF	176.408	40.400	260.230
5	CUF	460.618	82.670	96.530
6	BMD	460.618	82.670	96.530

Size Distributions

CLASS	SIZE	FF	CF	COF	CUF	BMD
1	3984	1.68	0.73	0.00	1.01	0.37
2	2807	2.32	0.89	0.00	1.23	0.34
3	2022	2.43	0.89	0.00	1.23	0.30
4	1403	3.14	1.09	0.00	1.51	0.30
5	1011	3.64	1.28	0.00	1.77	0.37
6	714	4.44	1.61	0.01	2.22	0.52
7	505	6.08	2.34	0.08	3.21	0.91
8	357	6.13	2.83	0.25	3.82	1.56
9	252	6.92	4.05	0.77	5.31	2.95
10	178	7.19	5.76	1.95	7.22	5.21
11	126	6.80	7.86	4.16	9.28	8.27
12	89	6.70	10.92	8.15	11.98	12.54
13	63	6.31	12.37	12.00	12.51	14.69
14	45	5.76	11.88	14.04	11.05	14.22
15	30	4.07	9.23	13.05	7.77	11.20
16	15	26.39	26.27	45.54	18.89	26.22

Plitt's Model Parameters

CYCLOPAK-1

d50c = 46 µm
P = 29.760 kPa
S = 0.920
m = 0.760
Rf = 0.270

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Louvicourt Mill's Grinding Circuit, Predictions with 1" balls
  and smaller apex diameter (2.25").
  Circuit Type: #6

[KNOWN-STREAMS]
  2
  1      176.44      49.47       180.22
  2       0          0           80

[KNOWN-SIZE-DISTRIBUTIONS]
  16
  1
  1
  1      3984     1.68
  2      2807     2.32
  3      2022     2.43
  4      1403     3.14
  5      1011     3.64
  6      714      4.44
  7      505      6.08
  8      357      6.13
  9      252      6.92
  10     178      7.19
  11     126      6.80
  12     89       6.70
  13     63       6.31
  14     45       5.76
  15     30       4.07
  16     15       26.39

[BALLMILL-1]
  0.3997
  0.6096
  0.8504
  1.0887
  1.2832
  1.3969
  1.4090
  1.3213
  1.1554
  0.9454
  0.7261
  0.5251
  0.3588
  0.2323
  0.1431
```

```
  OFF

  0.0000
  0.4380 0.0000
  0.1920 0.4360  0.0000
  0.0860 0.1750  0.4050  0.0000
  0.0530 0.0940  0.1920  0.4080  0.0000
  0.0340 0.0540  0.0950  0.1790  0.3830  0.0000
  0.0280 0.0400  0.0630  0.1050  0.1980  0.3920  0.0000
  0.0220 0.0300  0.0430  0.0640  0.1060  0.1880  0.3770  0.0000
  0.0200 0.0260  0.0350  0.0490  0.0740  0.1180  0.2110  0.3980  0.0000
  0.0170 0.0210  0.0280  0.0370  0.0520  0.0760  0.1230  0.2100  0.4080
0.0000
  0.0140 0.0170  0.0220  0.0280  0.0380  0.0520  0.0770  0.1220  0.2160
0.4220  0.0000
  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850  0.1380
0.2470  0.4890  0.0000
  0.0120 0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850
0.1380  0.2470  0.4890  0.0000
  0.0110 0.0120  0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000
  0.0080 0.0110  0.0120  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000

  711.88  0.1  0.1  0.7

[CYCLOPAK-1]
  38.1
  9.525
  7.62
  5.715
  119.38
  3.55
  10
  0.274     1     1       0.355       0.759
  100

[CONVERGE-1]
  0.01
```

Connectivity Matrix

1	3	1	1	1000	-1	0	0	0	1
2	2	1	0	0	1	-2	-3	0	0
3	1	1	0	0	0	0	1	-1	0
4	100	1	0	0	0	0	0	1	-1

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	FF	176.440	49.470	180.220
3	CF	632.777	63.900	357.540
4	COF	176.408	40.400	260.230
5	CUF	456.369	82.420	97.310
6	BMD	456.369	82.420	97.310

Size Distributions

CLASS	SIZE	FF	CF	COF	CUF	BMD
1	3984	1.68	1.11	0.00	1.54	0.89
2	2807	2.32	1.40	0.00	1.94	1.04
3	2022	2.43	1.39	0.00	1.93	0.99
4	1403	3.14	1.59	0.00	2.20	0.99
5	1011	3.64	1.78	0.00	2.47	1.06
6	714	4.44	2.10	0.02	2.90	1.19
7	505	6.08	2.84	0.09	3.90	1.59
8	357	6.13	3.15	0.27	4.26	2.00
9	252	6.92	4.01	0.73	5.28	2.88
10	178	7.19	4.98	1.64	6.27	4.12
11	126	6.80	6.09	3.15	7.23	5.81
12	89	6.70	8.19	5.99	9.04	8.77
13	63	6.31	10.02	9.56	10.20	11.46
14	45	5.76	11.17	13.00	10.46	13.26
15	30	4.07	10.19	14.21	8.64	12.56
16	15	26.39	30.00	51.37	21.74	31.39

Plitt's Model Parameters

CYCLOPAK-1

d50c = 46 μm
P = 29.510 kPa
S = 0.920
m = 0.760
Rf = 0.270

Appendix C.6 The Effect of Proposed Modifications on Circuit Capacity, Louvicourt Mine

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Louvicourt Mill's Grinding Circuit, Predictions with 1" balls.
  Impact on circuit capacity
  Circuit Type: #6

[KNOWN-STREAMS]
  2
  1      185.262     49.47       189.231
  2       0          0           80

[KNOWN-SIZE-DISTRIBUTIONS]
  16
  1
  1
  1      3984     1.68
  2      2807     2.32
  3      2022     2.43
  4      1403     3.14
  5      1011     3.64
  6      714      4.44
  7      505      6.08
  8      357      6.13
  9      252      6.92
  10     178      7.19
  11     126      6.80
  12     89       6.70
  13     63       6.31
  14     45       5.76
  15     30       4.07
  16     15       26.39

[BALLMILL-1]
  0.3997
  0.6096
  0.8504
  1.0887
  1.2832
  1.3969
  1.4090
  1.3213
  1.1554
  0.9454
  0.7261
  0.5251
  0.3588
  0.2323
  0.1431
```

```
  OFF

  0.0000
  0.4380 0.0000
  0.1920 0.4360  0.0000
  0.0860 0.1750  0.4050  0.0000
  0.0530 0.0940  0.1920  0.4080  0.0000
  0.0340 0.0540  0.0950  0.1790  0.3830  0.0000
  0.0280 0.0400  0.0630  0.1050  0.1980  0.3920  0.0000
  0.0220 0.0300  0.0430  0.0640  0.1060  0.1880  0.3770  0.0000
  0.0200 0.0260  0.0350  0.0490  0.0740  0.1180  0.2110  0.3980  0.0000
  0.0170 0.0210  0.0280  0.0370  0.0520  0.0760  0.1230  0.2100  0.4080
0.0000
  0.0140 0.0170  0.0220  0.0280  0.0380  0.0520  0.0770  0.1220  0.2160
0.4220  0.0000
  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850  0.1380
0.2470  0.4890  0.0000
  0.0120 0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850
0.1380  0.2470  0.4890  0.0000
  0.0110 0.0120  0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000
  0.0080 0.0110  0.0120  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000

  711.88  0.1  0.1  0.7

[CYCLOPAK-1]
  38.1
  9.525
  7.62
  6.98
  119.38
  3.55
  10
  0.274     1     1       0.355       0.759
  100

[CONVERGE-1]
  0.01

-------------------------------------------------------------------------
                          Connectivity Matrix
-------------------------------------------------------------------------

   1    3    1    1 1000   -1    0    0    0    1
   2    2    1    0    0    1   -2   -3    0    0
   3    1    1    0    0    0    0    1   -1    0
   4  100    1    0    0    0    0    0    1   -1


-------------------------------------------------------------------------
                Solids, Percent Solids and Water
```

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	FF	185.262	49.470	189.230
3	CF	928.627	67.100	455.280
4	COF	185.188	40.750	269.230
5	CUF	743.439	79.980	186.050
6	BMD	743.439	79.980	186.050

Size Distributions

CLASS	SIZE	FF	CF	COF	CUF	BMD
1	3984	1.68	1.10	0.00	1.37	0.96
2	2807	2.32	1.38	0.00	1.72	1.15
3	2022	2.43	1.37	0.01	1.71	1.10
4	1403	3.14	1.52	0.02	1.89	1.12
5	1011	3.64	1.69	0.06	2.10	1.21
6	714	4.44	1.97	0.14	2.42	1.35
7	505	6.08	2.65	0.36	3.22	1.79
8	357	6.13	2.98	0.67	3.56	2.20
9	252	6.92	3.84	1.31	4.47	3.07
10	178	7.19	4.83	2.32	5.45	4.24
11	126	6.80	5.98	3.80	6.52	5.77
12	89	6.70	8.09	6.48	8.49	8.44
13	63	6.31	9.90	9.58	9.98	10.80
14	45	5.76	11.06	12.45	10.71	12.38
15	30	4.07	10.19	13.30	9.42	11.72
16	15	26.39	31.43	49.40	26.95	32.68

Plitt's Model Parameters

CYCLOPAK-1

d50c = 42 µm
P = 47.760 kPa
S = 1.730
m = 0.570
Rf = 0.410

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

```
[PROJECT-TITLE]
  Louvicourt Mill's Grinding Circuit, Predictions with 1" balls.
  Impact on circuit capacity
  Circuit Type: #6

[KNOWN-STREAMS]
  2
  1      194.1      49.47      198.26
  2       0         0           80

[KNOWN-SIZE-DISTRIBUTIONS]
  16
  1
  1
  1      3984     1.68
  2      2807     2.32
  3      2022     2.43
  4      1403     3.14
  5      1011     3.64
  6      714      4.44
  7      505      6.08
  8      357      6.13
  9      252      6.92
  10     178      7.19
  11     126      6.80
  12     89       6.70
  13     63       6.31
  14     45       5.76
  15     30       4.07
  16     15       26.39

[BALLMILL-1]
  0.3997
  0.6096
  0.8504
  1.0887
  1.2832
  1.3969
  1.4090
  1.3213
  1.1554
  0.9454
  0.7261
  0.5251
  0.3588
  0.2323
  0.1431
```

```
  OFF

  0.0000
  0.4380 0.0000
  0.1920 0.4360  0.0000
  0.0860 0.1750  0.4050  0.0000
  0.0530 0.0940  0.1920  0.4080  0.0000
  0.0340 0.0540  0.0950  0.1790  0.3830  0.0000
  0.0280 0.0400  0.0630  0.1050  0.1980  0.3920  0.0000
  0.0220 0.0300  0.0430  0.0640  0.1060  0.1880  0.3770  0.0000
  0.0200 0.0260  0.0350  0.0490  0.0740  0.1180  0.2110  0.3980  0.0000
  0.0170 0.0210  0.0280  0.0370  0.0520  0.0760  0.1230  0.2100  0.4080
0.0000
  0.0140 0.0170  0.0220  0.0280  0.0380  0.0520  0.0770  0.1220  0.2160
0.4220  0.0000
  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850  0.1380
0.2470  0.4890  0.0000
  0.0120 0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850
0.1380  0.2470  0.4890  0.0000
  0.0110 0.0120  0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000
  0.0080 0.0110  0.0120  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000

  711.88  0.1  0.1  0.7

[CYCLOPAK-1]
  38.1
  9.525
  7.62
  6.98
  119.38
  3.55
  10
  0.274     1     1       0.355       0.759
  100

[CONVERGE-1]
  0.01
```

Connectivity Matrix

1	3	1	1	1000	-1	0	0	0	1
2	2	1	0	0	1	-2	-3	0	0
3	1	1	0	0	0	0	1	-1	0
4	100	1	0	0	0	0	0	1	-1

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	FF	194.100	49.470	198.260
3	CF	969.080	67.480	466.940
4	COF	194.156	41.100	278.220
5	CUF	774.924	80.420	188.720
6	BMD	774.924	80.420	188.720

Size Distributions

CLASS	SIZE	FF	CF	COF	CUF	BMD
1	3984	1.68	1.16	0.00	1.45	1.03
2	2807	2.32	1.44	0.00	1.80	1.22
3	2022	2.43	1.41	0.01	1.76	1.16
4	1403	3.14	1.57	0.02	1.96	1.18
5	1011	3.64	1.74	0.06	2.16	1.27
6	714	4.44	2.02	0.15	2.49	1.42
7	505	6.08	2.73	0.38	3.32	1.89
8	357	6.13	3.07	0.71	3.66	2.30
9	252	6.92	3.95	1.38	4.60	3.21
10	178	7.19	4.97	2.43	5.61	4.42
11	126	6.80	6.14	3.96	6.69	5.98
12	89	6.70	8.25	6.70	8.64	8.65
13	63	6.31	10.00	9.79	10.05	10.93
14	45	5.76	11.01	12.54	10.63	12.34
15	30	4.07	10.00	13.19	9.20	11.49
16	15	26.39	30.48	48.38	26.00	31.49

Plitt's Model Parameters

CYCLOPAK-1

d50c = 42 μm
P = 51.150 kPa
S = 1.710
m = 0.570
Rf = 0.400

***** BALL MILLING CIRCUITS SIMULATION PROGRAM OUTPUT *****

Simulation Data

[PROJECT-TITLE]
Louvicourt Mill's Grinding Circuit, Predictions with 1" balls.
Impact on circuit capacity
Circuit Type: #6

[KNOWN-STREAMS]
2

1	211.728	49.47	216.27
2	0	0	80

[KNOWN-SIZE-DISTRIBUTIONS]
16
1
1

1	3984	1.68
2	2807	2.32
3	2022	2.43
4	1403	3.14
5	1011	3.64
6	714	4.44
7	505	6.08
8	357	6.13
9	252	6.92
10	178	7.19
11	126	6.80
12	89	6.70
13	63	6.31
14	45	5.76
15	30	4.07
16	15	26.39

[BALLMILL-1]
0.3997
0.6096
0.8504
1.0887
1.2832
1.3969
1.4090
1.3213
1.1554
0.9454
0.7261
0.5251
0.3588
0.2323
0.1431

```
  OFF

  0.0000
  0.4380 0.0000
  0.1920 0.4360  0.0000
  0.0860 0.1750  0.4050  0.0000
  0.0530 0.0940  0.1920  0.4080  0.0000
  0.0340 0.0540  0.0950  0.1790  0.3830  0.0000
  0.0280 0.0400  0.0630  0.1050  0.1980  0.3920  0.0000
  0.0220 0.0300  0.0430  0.0640  0.1060  0.1880  0.3770  0.0000
  0.0200 0.0260  0.0350  0.0490  0.0740  0.1180  0.2110  0.3980  0.0000
  0.0170 0.0210  0.0280  0.0370  0.0520  0.0760  0.1230  0.2100  0.4080
0.0000
  0.0140 0.0170  0.0220  0.0280  0.0380  0.0520  0.0770  0.1220  0.2160
0.4220  0.0000
  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850  0.1380
0.2470  0.4890  0.0000
  0.0120 0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580  0.0850
0.1380  0.2470  0.4890  0.0000
  0.0110 0.0120  0.0130  0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000
  0.0080 0.0110  0.0120  0.0130 0.0160  0.0200  0.0250  0.0320  0.0420  0.0580
0.0850  0.1380  0.2470  0.4890  0.0000

  711.88  0.1  0.1  0.7

[CYCLOPAK-1]
  38.1
  9.525
  7.62
  6.98
  119.38
  3.55
  10
  0.274     1     1       0.355       0.759
  100

[CONVERGE-1]
  0.01
```

Connectivity Matrix

```
   1    3    1    1 1000   -1    0    0    0    1
   2    2    1    0    0    1   -2   -3    0    0
   3    1    1    0    0    0    0    1   -1    0
   4  100    1    0    0    0    0    0    1   -1
```

Solids, Percent Solids and Water

STREAM NO.	STREAM NAME	SOLIDS(t/h)	%SOLIDS	WATER(t/h)
1	FF	211.728	~ 49.470	216.270
3	CF	1050.344	68.190	489.870
4	COF	211.664	41.670	296.270
5	CUF	838.680	81.250	193.600
6	BMD	838.680	81.250	193.600

Size Distributions

CLASS	SIZE	FF	CF	COF	CUF	BMD
1	3984	1.68	1.24	0.00	1.55	1.13
2	2807	2.32	1.55	0.00	1.94	1.35
3	2022	2.43	1.50	0.01	1.88	1.28
4	1403	3.14	1.67	0.03	2.08	1.30
5	1011	3.64	1.85	0.07	2.30	1.40
6	714	4.44	2.14	0.17	2.64	1.56
7	505	6.08	2.88	0.42	3.50	2.07
8	357	6.13	3.25	0.78	3.87	2.52
9	252	6.92	4.17	1.51	4.84	3.48
10	178	7.19	5.23	2.65	5.88	4.74
11	126	6.80	6.42	4.28	6.96	6.33
12	89	6.70	8.54	7.14	8.89	9.00
13	63	6.31	10.14	10.21	10.12	11.11
14	45	5.76	10.91	12.74	10.45	12.21
15	30	4.07	9.63	13.01	8.78	11.03
16	15	26.39	28.85	46.82	24.31	29.47

Plitt's Model Parameters

CYCLOPAK-1

d50c = 43 μm
P = 58.240 kPa
S = 1.670
m = 0.570
Rf = 0.400

Appendix D.1 BALLMILL Module Rules

BALLMILL MODULE

Optimization objective: to increase throughput or grind fineness

IF the mill diameter is D_m m
THEN
the critical-speed is equal to:

$$\frac{42.3}{\sqrt{D_m}} \quad (rpm)$$

IF the percent of critical speed is greater than or equal to 82%
THEN
the mill speed is apparently too high (above 82%)

IF the percent of critical speed is less than or equal to 65%
THEN
the mill speed is apparently too low (below 65%)

IF steel balls are used and
the operation mode is wet and
the discharge mechanism is overflow
THEN
the wet grinding constant is equal to 350

IF steel balls are used and
the operation mode is wet and
the discharge mechanism is diaphragm
THEN
the wet grinding constant is equal to 330

IF steel balls are used and
the operation mode is dry and
the discharge mechanism is diaphragm
THEN
the wet grinding constant is equal to 335

IF silica balls are used and
the operation mode is wet and
the discharge mechanism is diaphragm

THEN
the wet grinding constant is equal to 170

IF silica balls are used and
the operation mode is dry and
the discharge mechanism is diaphragm
THEN
the wet grinding constant is equal to 175

IF the mill diameter is D_m m and
the percent of critical speed is Cs % and
the laboratory work index is W_i kWh/t and
the feed 80% passing size is F_{80} μm and
the ore specific gravity is S_g and
the wet grinding constant is K
THEN
the Bond make-up ball size is equal to:

$$\left(\frac{F_{80}}{K}\right)^{0.5}\left(\frac{S_g W_i}{C_s D_m^{0.5}}\right)^{0.33} \quad (mm)$$

IF the mill diameter is D_m m
the mill speed is N rpm
the laboratory work index is W_i kWh/t
the feed 80% passing size is equal to F_{80} μm
THEN
the Azzaroni make-up ball size is equal to:

$$5.8\ \frac{(F_{80})^{0.29}\,(W_i)^{0.4}}{(ND_m)^{0.25}} \quad (mm)$$

IF the optimization objective is to increase throughput or grind fineness and
the selection function curve is a straight line

THEN
the ball size is too large and must be reduced, if it was not increased in the past due other reasons.

IF the optimization objective is to increase throughput or grind fineness and
the selection function curve has a large hump
THEN
the percent of the ball mill discharge having a size larger than

that of hump, the maximum selection function value and the top size class selection function must be considered to determine if the ball size must be increased.

IF the optimization objective is to increase throughput or grind fineness and
the selection function curve has a large hump and
the percent of discharge material coarser than hump particle size is greater than or equal to 20% and
the top size selection function is less than 0.2 of hump selection function

THEN
the ball size is too small and must be increased, if it was not decreased in the past to for other reasons.

IF the optimization objective is to increase throughput or grind fineness and
the selection function curve has a small hump
THEN
the ball size is near optimum

IF the ball size is too large and
the make-up ball size is greater than or equal to 38 mm (1.5") and
the ball size was not increased in the past*
THEN
decrease the make-up or top ball size by 13 mm (0.5"). Test the effect of this change by NGOTC before the real plant exercise.

IF the ball size is too large and
the make-up ball size is less than 38 mm (1.5") and
the ball size was not increased in the past
THEN
decrease make-up or top ball size by 6 mm (0.25"). This can be achieved using a blend of make-up balls. Test the effect of this change by NGOTC before the real plant exercise.

IF the optimization objective is to increase throughput or grind fineness and
the ball size is too small and
the make-up ball size is less than 51 mm (2") and
the ball size was not decreased in the past
THEN
increase make-up or top ball size by 13 mm (0.5"). Test the effect

*To avoid unnecessary oscillations in make-up ball size

of this change by NGOTC before the real plant exercise.

IF the optimization objective is to increase throughput grind fineness and
the ball size is too small and
the make-up ball size is greater than or equal 51 mm (2") and
the ball size was not decreased in the past
THEN
increase make-up or top ball size by 25 mm (1"). Test the effect
of this change with NGOTC before the real plant exercise.

IF the optimization objective is to increase throughput or grind fineness and
the ball size is too small and
the ball size was decreased in the past
THEN
it seems that the make-up ball size is too small.
Since ball size was recently decreased, it is likely
that the optimum ball size is between the previous
and existing one.

IF the optimization objective is to increase throughput or grind fineness and
the ball size is too large and
the ball size was increased in the past
THEN
it seems that the make-up ball size is too large. However, since there has been an attempt to increase the ball size before, decreasing it again is not likely to improve grinding efficiency, unless feed to the mill has become finer or softer

Optimization objective: to decrease operating costs

IF the optimization objective is to decrease operating costs and
the mill liner condition has not been checked
THEN
check the liner condition.

IF the optimization objective is to decrease operating costs and
the operation mode is wet and
the liner wear is greater than or equal to 0.044 kg/kWh
THEN
it seems that liner wear is too high for this operation. This needs to be checked, any way.

IF the optimization objective is to decrease operating costs and
the operation mode is dry and
the liner wear is greater than or equal to 0.006 kg/kWh
THEN
it seems that liner wear is too high for this operation. This needs to be checked, any way.

IF the optimization objective is to decrease operating costs and
the operating work index to the laboratory work index ratio is between 0.8 and 1.05
THEN
the energy consumption is good

IF the optimization objective is to decrease operating costs and
the operating work index to the laboratory work index ratio is between 1.05 and 1.2
THEN
the energy consumption is ok

IF the optimization objective is to decrease operating costs and
the operating work index to the laboratory work index ratio is greater than 1.2
THEN
the energy consumption is bad

IF the optimization objective is to decrease operating costs and
the energy consumption is good
THEN
the grinding performance, in terms of energy consumption, seems very good.

IF the optimization objective is to decrease operating costs and
the energy consumption is bad
THEN
the grinding performance, in terms of energy consumption, seems not good. You may need to optimize operation in this respect.

IF the optimization objective is to decrease-operating-costs and
the energy consumption ok
THEN
the grinding performance, in terms of energy consumption, seems to be okay.

IF the laboratory work index is unknown

the operating work index is unknown
THEN
the laboratory and operating work indices are recommended to be determined.

IF the optimization objective is to decrease operating costs and
the mill diameter is D_m m and
the charge to roof distance is H m
THEN
the mill-filling is equal to:

$$113 - 126 * \frac{H}{D_m} \quad (\%)$$

IF the optimization objective is to decrease operating costs and
the discharge mechanism is overflow and
the mill filling is greater than or equal to 45%

THEN
the mill filling seems to be high and must be checked. You might be loosing balls at discharge due to the high mill filling.

IF the optimization objective is to decrease operating costs and
the discharge mechanism is overflow and
the mill filling is less than or equal to 30%
THEN
the mill filling seems to be low and must be checked.

Optimization objective: is unknown

IF the optimization objective is unknown
THEN
having clear plant optimization objectives is important since it affects decisions about plant operating changes and guides optimization process.

Appendix D.2 HYDROCYCLONE Module Rules

HYDROCYCLONE MODULE

IF the classification objective is to increase cut size
THEN

the cut size can be increased by reducing the apex diameter of the cyclone(s). However, it is recommended to use the BMCS to assess the impact of using a smaller apex on the classification and full circuit performance.

The cut size can be increased by installing larger cyclones. This option, however, is only practical at the design stage. For an existing circuit, the BMCS can be used to assess the impact of using larger cyclones on the classification and full circuit performance.

The cut size can be increased by increasing the inclination of cyclones to the vertical to 45 degrees or more. This option is normally practical only at the design stage, and when the number of cyclones is small.

Increasing the vortex finder diameter can increase the cut size. The BMCS program can be used to assess the impact of using a larger vortex finder diameter on the circuit performance.

IF the classification objective is to decrease cut size
the number of operating cyclones is greater than 1
THEN

the cut size can be reduced by switching off a cyclone at constant total feed flow rate. It is recommended to use the BMCS to assess the impact of this change on the full circuit performance.

IF the classification objective is to decrease cut size
THEN

the cut size can be reduced by diluting the feed slurry. It is recommended to use the BMCS to assess the impact of this change on the full circuit performance.

IF the classification objective is to reduce water recovery and
the number of operating cyclones is greater than 3 and
the pressure drop is too low
THEN

the water recovery to the cyclone underflow, R_f, can be reduced by switching off one cyclone at constant total feed flow rate. It is recommended to use the BMCS to assess the impact of this change on full circuit performance.

IF the classification objective is to reduce the water recovery
THEN

the water recovery to the cyclone underflow, R_f, can be reduced by using smaller

apex diameter. It is recommended to use the BMCS to assess the impact of this change on full circuit performance.

IF the classification objective is to reduce water recovery and
the cut size is too low and
the circulating load is too high
THEN
the water recovery to the cyclone underflow, R_f, can be reduced by using larger vortex finder diameter. It is recommended to use the BMCS to assess the impact of this change on full circuit performance.

IF the classification objective is to increase the separation sharpness
THEN
the separation sharpness can be increased by modifications that decrease water recovery to the cyclone underflow or short circuiting.

IF the classification objective is to increase the separation sharpness
THEN
in case of excessively high feed solids concentration or high slimes concentrations, it is recommended to dilute the feed to reduce the viscosity of the fluid. This can be led to improved separation sharpness.

IF the classification data is available
THEN
use a spreadsheet software such as QuattroPro* or Excel† to fit Plitt's model to data.

IF the classification data is not available
THEN
do a circuit survey around the cyclone(s).

IF Plitt's model was fitted to the classification data and
Plitt's model fit was not optimized
THEN
the fit must be optimized using a non-linear optimization tool. Spreadsheet softwares normally include this function.

IF Plitt's model was fitted and
the fit was optimized and

---

*QuattroPro is a trademark of Borland International

†Excel is a trademark of Microsoft company

R_f was not positive

THEN

when the fit is optimized, the final values of estimated parameters, R_f, d_{50c} and m must be positive. d_{50c} and m normally are. If R_f is not positive, this can be due to incomplete size distribution information of cyclone streams for fine size classes. To solve this problem, R_f can be calculated from cyclone overflow and underflow solids flow rate and percent solids information. Then, the other two parameters can be estimated using the optimization tool.

IF Plitt's model was fitted and
the fit was optimized and
R_f was not positive

THEN

It is recommended to use a wider screening size range in next circuit survey so that Rf can be estimated when model fitting is optimized.

IF Plitt's model was fitted and
the fit was optimized and
the R_f was positive and
the user does not know if the optimal fit was satisfactory or not

THEN

to check if the optimal fit is satisfactory or not, you can examine the goodness of fit (or the lack of fit) criterion and also visually evaluate how close is the fitted curve to the measured data. R_f from the circulating load, cyclone underflow, overflow and % solids should be close to R_f fitted.

IF Plitt's model was fitted and
the fit was optimized and
the R_f was positive and
the optimized fit is not satisfactory and
there is a hump or plateau in the classification curve and
the ore consists of significant heavy and light phases

THEN

the individual mineral classification behaviour is the cause of lack-of-fit. The grinding model can only be used for predicting trends.

IF Plitt's model was fitted and
the fit was optimized and
the R_f was positive and
the optimized fit is not satisfactory and
there is a hump or plateau in the classification curve and
there is no significant heavy and light phases and
the data was mass balanced

THEN

It is recommended to repeat the sampling.

IF Plitt's model was fitted and
the fit was optimized and
the R_f was positive and
the optimized fit is not satisfactory and
there is a hump or plateau in the classification curve and
there is no significant heavy and light phases and
the data was not mass balanced

THEN

It is recommended to do mass balancing before data analysis.

IF Plitt's model was fitted and
the fit was optimized and
the R_f was positive and
the optimized fit is not satisfactory and
there exist no hump or plateau in the classification curve and
there exist no fish hook at fine end of the classification curve and
the data was mass balanced

THEN

it is recommended to check the validity of data used for the analysis. It might be necessary to redo sampling tests to obtain reliable classification data.

IF Plitt's model was fitted and
the fit was optimized and
the R_f was positive and
the optimized fit is not satisfactory and
there exist no hump or plateau in the classification curve and
there exist no fish hook at fine end of the classification curve and
the data was not mass balanced

THEN

It is recommended to do mass balancing before data analysis.

IF Plitt's model was fitted and
the fit was optimized and
the R_f was positive and
the optimized fit is not satisfactory and
there exist no hump or plateau in the middle of the classification curve and
there exist a fish hook

THEN

it is recommended to use a fish hook model such as the one proposed by Finch [1983].

IF the water recovery to the cyclone underflow is R_f and
the R_f is greater than 50%

THEN

the efficiency of the cyclone operation in terms of the amount of water recovered to the cyclone underflow is very poor. It is recommended to significantly reduce water recovery to the cyclone underflow.

IF the water recovery to the cyclone underflow is R_f and
the R_f is less than or equal to 50% and is greater than 40%

THEN

the efficiency of the cyclone operation in terms of the amount of water recovered to the cyclone underflow is poor. It is recommended to reduce water recovery to the cyclone underflow.

IF the water recovery to the cyclone underflow is R_f and
the R_f is less than or equal to 40% and is greater than 30%

THEN

the efficiency of the cyclone operation in terms of the amount of water recovered to the cyclone underflow is reasonable.

IF the water recovery to the cyclone underflow is R_f and
the R_f is less than or equal to 30% and is greater than 20%

THEN

the efficiency of the cyclone operation in terms of the amount of water recovered to the cyclone underflow is good.

IF the water recovery to the cyclone underflow is R_f and
the R_f is less than or equal to 20% and is greater than or equal to 10%

THEN

the amount of water recovered to the cyclone underflow is too low. The cyclone operation may be subjected to underflow roping. This can be checked visually.

IF the water recovery to the cyclone underflow is R_f and
the R_f is less than 10%

THEN

the amount of water recovered to the cyclone underflow is extremely low. This is very unusual with normal cyclone operations, and is normally achievable only with an underflow valve for producing a product for conveying or stockpiling. We recommend that you check the reliability of the data

IF the separation sharpness (m) is greater than or equal to 3
THEN
the cyclone separation sharpness is excellent.

IF the separation sharpness (m) is less than or equal to 2 and
hydrocyclones are primary
THEN
the cyclone separation sharpness is poor.

IF the separation sharpness (m) is between 2 and 3
THEN
the cyclone separation sharpness is normal.

IF the separation sharpness is poor and
there are significant heavy and light phases
THEN
the separation sharpness is poor because of heavy and light phases

Appendix D.3 CIRCUIT Module Rules

CIRCUIT MODULE

IF circuit flowsheet number 1 is used and
the ball mill discharge size distribution is too coarse and
closed circuit grinding can be used
THEN
circuits 5 and 6 are proposed

IF circuit flowsheet number 1 is used and
the ball mill discharge size distribution is too wide and
closed circuit grinding can be used
THEN
circuits 5 and 6 are proposed

IF circuit flowsheet number 1 is used and
a higher capacity or a finer grind is required and
closed circuit grinding can be used
THEN
circuits 5 and 6 are proposed

IF circuit flowsheet number 1 is used and
circuits 5 and 6 are proposed and
the ball mill discharge density is too low and
the density control is a problem
THEN
circuit 6 is proposed as the alternative to the current circuit.

IF circuit flowsheet number 1 is used and
circuits 5 and 6 are proposed and
the fresh feed is coarse and
the fresh feed contains few fines
THEN
circuit 5 is proposed as the alternative to the current circuit.

IF circuit flowsheet number 1 is used and
circuits 5 and 6 are proposed and
the fresh feed is not coarse and
the fresh feed contains significant fines
THEN
circuit 6 is proposed as an alternative to the current circuit.

IF circuit flowsheet number 1 is used and
a closed circuit grinding cannot be used and
a high ball mill discharge density is required and

closed circuit grinding cannot be used
THEN
it is recommended to consider adding grinding aids to the circuit.

IF circuit flowsheet number 1 is used and
a high ball mill discharge temperature is required
THEN
it is recommended to consider adding grinding aids to the circuit.

IF circuit flowsheet number 5 is used and
fresh feed contains significant fines and
a very sharp classification is not required
THEN
circuit 6 is proposed as an alternative to the current circuit. Since the fresh feed is very fine a pre-classification configuration is preferred.

IF circuit flowsheet number 5 is used and
a very sharp classification is required and
overgrinding is not a problem and
coarse material is not a problem in downstream process
THEN
circuits 7, 8, and 9 are proposed as alternatives to the current circuit.

IF circuit flowsheet number 5 is used and
a very sharp classification is required and
overgrinding is a problem
THEN
circuits 7, 8, and 9 are proposed as alternatives to the current circuit. However, due to the overgrinding problem circuit 11 is preferred.

IF circuit flowsheet number 5 is used and
a very sharp classification is required and
coarse material is a problem in the downstream process
THEN
circuits 7, 8, and 9 are proposed as alternatives to the current circuit. However, to minimize coarse material content in circuit product, circuit 7 is preferred.

IF circuit flowsheet number 6 is used and
a very sharp classification is not required and
the fresh feed contains significant oversize
THEN
circuit 5 is proposed as an alternative to the current circuit.

IF circuit flowsheet number 6 is used and

a very sharp classification is required and
overgrinding is not a problem

THEN

circuits 7 and 8 are proposed as alternative to the current circuit.

IF circuit flowsheet number 6 is used and
a very sharp classification is required and
overgrinding is a problem

THEN

circuits 7 and 8 are proposed as alternative to the current circuit. However, due to the overgrinding problem, circuit 8 would be preferred.

IF circuit flowsheet number 7 is used and
overgrinding is a problem and
the fresh feed does not contains significant fines

THEN

circuit 8 is proposed as an alternative to the current circuit to solve the over grinding problem.

IF circuit flowsheet number 7 is used and
the ball mill density is too low and
the secondary cyclone underflow density is low

THEN

circuit 8 is proposed as an alternative to the current circuit to solve the density problems.

IF circuit flowsheet number 7 is used and
the primary cyclone efficiency is low and
the primary cyclone feed density is high

THEN

circuit 8 is proposed as an alternative to the current circuit to solve primary classification efficiency and density problems.

IF circuit flowsheet number 7 is used and
the fresh feed contains significant fines and
overgrinding is a problem

THEN

circuit 11 is proposed as an alternative to the current circuit.

IF a circuit flowsheet other than 1, 5, 6, 7 and 8 is used

THEN

currently, there are no rules in the knowledge base that can be applied to the selected circuit.

Appendix D.4 MODSIM Module Rules

MODSIM MODULE

Task: modelling

IF the task to do is modelling and
the breakage function is not known and
the study is at a preliminary phase
THEN
it is recommended to use the breakage function of a similar ore for a preliminary work. For very accurate simulations, you would be better off to determine the breakage function of the ore using a representative sample.

IF the task to do is modelling and
the breakage function is not known and
the study is at a preliminary phase
THEN
although the actual ore breakage function is unavailable, it is still possible to estimate the selection function using typical ore breakage functions. For a detailed study, however, you may need to determine the breakage function.

IF the task to do is modelling and
the RTD model parameters are not known and
the study is at a preliminary phase
THEN
although the actual RTD parameters are unavailable, it is still possible to estimate the selection function using typical values. For a detailed study, however, you may need to measure RTD.

IF the task to do is modelling and
the breakage function is not known and
the study is at a detailed phase
THEN
it is recommended to determine the breakage function of the ore using representative samples of the ore.

IF the task to do is modelling and
the selection function is not known and
the data set is not mass balanced
THEN
it is recommended to use mass balanced data for selection function estimation. You must use mass balance software to first adjust raw data.

IF the task to do is modelling and
the breakage function is known and

the RTD model parameters are known and
the data set is mass balanced and
the selection function is not known

THEN

the NGOTC program must be run to back-calculate the selection function based on the available data set. The user, however, must be familiar with the program and can consult NGOTC manual for assistance.

IF the task to do is modelling and
the RTD model parameters are not known and
the study is at a preliminary phase

THEN

it is recommended to use typical values for Weller's model parameters such as tau PF=0.1, tau SPM=0.1 and tau LPM=0.7 at a standard ball mill feed rate.

IF the task to do is modelling and
the RTD model parameters are not known and
the study is at a detailed phase

THEN

it is recommended to do plant tracer tests to determine RTD model parameters.

IF NGOTC must be run

THEN

make a call to operating system to run ngotc.exe

IF the task to do is modelling and
the selection function does not show a clear trend

THEN

the estimated selection function may not be valid. Normally, a selection function vs. particle size curve shows a linear trend at fine size range followed by a non-linear trend at coarse sizes. It is recommended to check the validity of the selection function before using it for the circuit simulations.

IF the task to do is modelling and
the selection function shows a clear trend and
the selection function does not show noise

THEN

the estimated selection function seems to be valid. Normally, if the selection function vs. particle size curve has a clear trend and there is no significant noise in data particularly for coarser size classes it indicates a reliable estimate of the selection function.

IF the task to do is modelling and
the selection function shows a clear trend and

the selection function is noisy and
the level of noise is pronounced in top size classes
THEN
the estimated selection function is valid for fine size classes. However, for the top size classes, the selection function values may be uncertain and erratic due to screening errors, if there is very little mass in top size classes.

IF the task to do is modelling and
the selection function shows a trend and
the selection function shows noise and
the noise is not at top size classes
THEN
if the noise level is low and distributed over the full size range, the estimated selection function is still valid. However, for simulation purposes, it is better to smooth the selection function values by the spline curve fitting algorithm.

IF the task to do is modelling and
the selection function was estimated by sequential interval-by- interval search and
the study is at a detailed phase
THEN
as this is a detailed study, it is recommended to use other selection function estimation methods as well. For example, (1) use more than one data sets (2) methods based on assumed functional forms of selection functions can be used. The best selection function vector then can be found by the analysis of results from various methods.

IF the task to do is modelling and
the selection function was estimated using functional forms and
the study is at a detailed phase
THEN
as this is a detailed study, it is recommended to use other selection function methods as well. For example, (1) use more than one data set (2) sequential interval-by-interval search method can be used. The best selection function vector then can be found by the analysis of results from various methods.

Task: simulation

IF modelling has not been done
THEN
you need to build a model of the circuit by calibrating and validating BMCS simulator. In the modelling step, various model parameters (breakage function, selection function, RTD, cyclone(s) geometry and calibration factors) must be estimated.

IF modelling has been done
THEN

the BMCS program can be run now to simulate the grinding circuit. It is expected that the user be familiar with the program and be able to prepare the correct data file.

IF the calibration and validation of BMCS have not been done and
the two data sets are not available
THEN

to build a grinding circuit model at least two data sets are needed. One data set (from a detailed circuit survey) is required for the estimation of unit model parameters and another one (independent data set) is needed to validate (test) the model built based on the first data set. It is recommended that plant sampling campaigns to be done for calibrating and validating the BMCS simulator.

IF the calibrating and validating of BMCS have not been done and
two data sets are available and
the selection function is not known
THEN

the NGOTC program must be run to back-calculate selection function based on an available data set. The user, however, must be familiar with the program.

Appendix E.1 Classification Performance Calculation of Line 1, AELRD

Classification Curve, Line #1

Screen (MESH)	Size (µm)	Geo Mean (µm)	COF (% ret.)	CUF (%ret.)	CF (%ret.)	Actual Rec. (%)	Calc. Rec. (%)	SD
6	3350	3984	0.00	0.11	0.10	99.94	100.00	0.00
8	2360	2807	0.00	0.21	0.19	99.98	100.00	0.00
10	1700	2022	0.00	0.19	0.17	99.94	100.00	0.00
14	1180	1403	0.00	0.45	0.41	99.99	100.00	0.00
20	850	1011	0.06	0.45	0.41	98.59	100.00	1.98
30	600	714	0.09	0.70	0.64	98.67	100.00	1.77
40	425	505	0.19	1.26	1.16	98.46	100.00	2.38
50	300	357	0.39	2.94	2.70	98.63	100.00	1.87
70	212	252	1.28	5.13	4.77	97.47	100.00	6.40
100	150	178	4.32	11.87	11.16	96.34	99.98	13.22
150	106	126	6.83	17.22	16.24	96.03	99.60	12.78
200	75	89	9.07	22.47	21.20	95.96	97.65	2.86
270	53	63	10.58	15.92	15.42	93.52	92.73	0.61
400	38	45	13.28	9.86	10.18	87.68	85.53	4.63
500	25	30	13.80	4.04	4.96	73.71	75.42	2.92
-500 total								
total			100.00	100.00	89.71			51.43

COF solids	35.02 t/h	% Solids, COF	28.24	Pulp	124.01 t/h	Rf = 0.50
CUF solids	335.60 t/h	% Solids, CUF	79.18		423.84 t/h	
CF solids	370.62 t/h					

d50c:	29.05
m:	1.32
Rf:	0.50
SS:	51.43

Answer Report

Solution Cell	Starting	Final
Agnico_Eagle:B92	392.29	51.43

Variable Cells	Starting	Final
Agnico_Eagle:B89..B89	45.00	29.05
Agnico_Eagle:B90..B90	2.50	1.32

Appendix E.2 Classification Performance Calculation of Line 2, AELRD

Classification Curve, Line #2

Screen (MESH)	Size (µm)	Geo Mean (µm)	COF (% ret.)	CUF (% ret.)	CF (%ret.)	Actual Rec. (%	Calc. Rec. (%)	SD
6	3350	3984	0.00	0.12	0.11	99.94	100.00	0.00
8	2360	2807	0.00	0.11	0.10	99.79	100.00	0.05
10	1700	2022	0.00	0.15	0.14	99.87	100.00	0.02
14	1180	1403	0.00	0.41	0.37	99.98	100.00	0.00
20	850	1011	0.01	0.38	0.34	99.63	100.00	0.14
30	600	714	0.02	0.61	0.54	99.58	100.00	0.18
40	425	505	0.10	1.18	1.05	98.93	100.00	1.15
50	300	357	0.48	2.53	2.29	97.60	100.00	5.77
70	212	252	1.34	4.63	4.26	96.42	100.00	12.82
100	150	178	3.63	11.29	10.42	96.03	100.00	15.73
150	106	126	5.85	17.92	16.55	95.98	100.00	16.16
200	75	89	14.52	25.44	24.20	93.17	99.93	45.59
270	53	63	11.35	16.75	16.13	92.00	95.66	13.43
400	38	45	13.65	9.06	9.58	83.79	79.86	15.48
500	25	30	17.78	2.95	4.64	56.38	59.19	7.90
-500 total								
total			100.00	100.00	90.71			134.41

COF solids	34.78 t/h	% Solids, COF	23.14	Pulp	150.31 t/h	Rf = 0.43
CUF solids	270.92 t/h	% Solids, CUF	75.50		358.84 t/h	
CF solids	305.70 t/h					

d50c	39.00
m:	2.73
Rf	0.43
ss:	134.41

Answer Report

Solution Cell	Starting	Final
Agnico_Eagle:B187	238.79	134.41

Variable Cells	Starting	Final
Agnico_Eagle:B184..B184	45.00	39.00
Agnico_Eagle:B185..B185	2.50	2.73

Appendix E.3 Mass Balancing Results of Dome Mine

PDI (September 23, 1997)

Residual sum of squares: 60.7033 Final Results

	Stream	Absolute Solids Flowrate	Pulp Mass Flowrate Meas	Calc	S.D.	Adjust
1	RMFF	100.00	100.0	100.0	0.0	0.0
2	RMD	250.00		250.0		
3	CUF	150.00	150.0	150.0	200.0	0.0
4	COF	100.00		100.0		

	Stream	Relative Solids Flowrate
1	RMFF	100.00
2	RMD	250.00
3	CUF	150.00
4	COF	100.00

Fractional Size Distribution Data

Size	RMD Meas	Calc	SD.	Adj.	CUF Meas	Calc	SD.	Adj.
1/2 inch	1.08	0.87	0.5	-0.2	1.33	1.45	0.5	0.1
3/8 inch	1.10	3.54	0.5	2.4*	7.36	5.90	0.5	-1.5*
3 MESH	2.46	1.46	0.5	-1.0*	1.82	2.42	0.5	0.6*
4 MESH	2.81	1.30	0.5	-1.5*	1.26	2.17	0.5	0.9*
6 MESH	2.01	2.08	0.5	0.1	3.51	3.47	0.5	-0.0
8 MESH	2.43	2.50	0.5	0.1	4.20	4.16	0.5	-0.0
10 MESH	2.63	2.51	0.5	-0.1	4.10	4.17	0.5	0.1
14 MESH	3.89	3.66	0.5	-0.2	5.97	6.11	0.5	0.1
20 MESH	5.19	4.73	0.5	-0.5	7.14	7.42	0.5	0.3
28 MESH	6.16	6.01	0.5	-0.1	8.58	8.67	0.5	0.1
35 MESH	7.41	7.11	0.5	-0.3	8.94	9.12	0.5	0.2
48 MESH	8.03	7.65	0.5	-0.4	8.05	8.28	0.5	0.2
65 MESH	7.86	7.58	0.5	-0.3	6.82	6.99	0.5	0.2
100 MESH	6.03	6.02	0.5	-0.0	4.48	4.48	0.5	0.0
150 MESH	4.50	4.62	0.5	0.1	3.05	2.98	0.5	-0.1
200 MESH	3.63	3.63	0.5	-0.0	2.19	2.19	0.5	0.0
270 MESH	3.76	3.08	0.5	-0.7*	1.58	1.99	0.5	0.4
400 MESH	1.57	2.35	0.5	0.8*	1.60	1.13	0.5	-0.5

Size	COF Meas	Calc	SD.	Adj.
1/2 inch	0.00	0.00	0.1	0.0
3/8 inch	0.00	-0.01	0.1	-0.0
3 MESH	0.00	0.00	0.1	0.0
4 MESH	0.00	0.01	0.1	0.0
6 MESH	0.00	-0.00	0.1	-0.0
8 MESH	0.00	-0.00	0.1	-0.0
10 MESH	0.00	0.00	0.1	0.0
14 MESH	0.00	0.00	0.1	0.0

20 MESH	0.62	0.70	0.3	0.1
28 MESH	1.97	2.03	0.5	0.1
35 MESH	3.97	4.09	0.5	0.1
48 MESH	6.55	6.70	0.5	0.2
65 MESH	8.36	8.47	0.5	0.1
100 MESH	8.33	8.33	0.5	0.0
150 MESH	7.12	7.07	0.5	-0.0
200 MESH	5.78	5.78	0.5	0.0
270 MESH	4.43	4.70	0.5	0.3
400 MESH	4.48	4.17	0.5	-0.3

Appendix E.4 Selection Function Estimation Results of Dome Mine

****
Selection Function Estimation Results
****

Placer Dome Mill, Sept. 23, 1997
Date: 8/21/1998
Tau Plug Flow = 0.10 Tau Small PM = 0.10 Tau Large PM = 0.70

Reference feed flow rate = 100.0 t/h
Current feed flow rate = 100.0 t/h

Breakage Function Matrix

0.00										
0.44	0.00									
0.19	0.44	0.00								
0.09	0.19	0.44	0.00							
0.05	0.09	0.19	0.44	0.00						
0.03	0.05	0.09	0.19	0.44	0.00					
0.03	0.03	0.05	0.09	0.19	0.44	0.00				
0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00			
0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00		
0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00	
0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44	0.00
0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19	0.44
0.00										
0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09	0.19
0.44	0.00									
0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05	0.09
0.19	0.44	0.00								
0.00	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.05
0.09	0.19	0.44	0.00							
0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03
0.05	0.09	0.19	0.44	0.00						
0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.02	0.02	0.03
0.03	0.05	0.09	0.19	0.44	0.00					
0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.02	0.02
0.03	0.03	0.05	0.09	0.19	0.44	0.00				

CLASS	SIEVE SIZE	FEED	MEAS. PROD.	CALC. PROD.	SELEC. FUNC.
1	12700	7.24	0.87	0.87	3.3469
2	8980	6.28	3.54	3.54	1.2857
3	6350	5.35	1.46	1.46	3.4254
4	4490	5.48	1.30	1.30	4.4245
5	3175	5.08	2.08	2.08	3.2968
6	2245	5.48	2.50	2.50	3.0851
7	1587	4.90	2.51	2.51	3.2390
8	1123	5.95	3.66	3.66	2.4374
9	794	6.41	4.73	4.73	1.9630
10	561	6.48	6.01	6.01	1.4880
11	397	6.64	7.11	7.11	1.1389
12	281	5.96	7.65	7.65	0.8613
13	198	5.03	7.58	7.58	0.6386
14	140	3.29	6.02	6.02	0.5720
15	99	2.28	4.62	4.62	0.5726
16	70	1.71	3.63	3.63	0.5933
17	50	1.50	3.08	3.08	0.5904
18	35	1.00	2.35	2.35	0.6631

```
/* ngotc.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <conio.h>
#include <string.h>
#include <alloc.h>
#include <graphics.h>
#include "global.h"

#define EPS  (DBL_EPSILON*100)

extern unsigned _stklen=12226U;

void PrintText(int x,int y,char* text,int forgcolor);
void ShowIntroScreen(void);
void ShowMainMenu(void);
int EstimateSelectionFunction(void);
int SimulateGrindingCircuit(void);
int OptimizeBallSize(void);
int *CreateVectorInt(long nl,long nh);
void FreeVectorInt(int *v,long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
void FreeMatrixD(double **m,long nrl,long nrh,long ncl,long nch);
double **CreateMatrixD(long nrl,long nrh,long ncl,long nch);
double *CreateVectorD(long nl,long nh);
void InitializeGraphics(void);
void TerminateGraphics(void);

double *ptrWrkSpc,*ptrTempDoubleArray;
bool normalisableBreakageFunction=TRUE;
bool normBreakFuncEst=TRUE;
char projectTitle[81]="untitled project";
char projectTitleEst[81]="untitled project";
int sizeClassNumSelecFuncEst,sizeClassNumBallSzOpt,sizeClassNumSimGrCir;
char savescr[4096];
double *screenSize,*ptrScreenSizeEst;
double *dischargeSizeDistribution,*ptrDischargeMeasuredEst,*ptrDischargeCalcEst;
double *feedSizeDistribution,*ptrFeedEst;
double *selectionFunction,*ptrSelectionFunctionEst;
double **breakageFunction,**ptrBreakFuncEst;
double *ptrTauPFSimGrCir,*ptrTauLPMSimGrCir,*ptrTauSPMSimGrCir,
                *ptrRefMillFeedRateSimGrCir,*ptrCurMillFeedRateSimGrCir;
double *ptrTauPFSelecFuncEst,*ptrTauLPMSelecFuncEst,
                *ptrTauSPMSelecFuncEst,*ptrRefMillFeedRateSFEst,
                *ptrCurMillFeedRateSFEst;
double *diag,**t,**tinv,**tdiag,**trans;
double *ptrDiag,**ptrT,**ptrTinv,**ptrTdiag,**ptrTrans;
int *ptr_iwrk;
double *ptrPosKnots,*ptrPosKnots1,*ptrPosKnots2,*ptr_wrk,*ptr_c,*ptr_c1,
                *ptr_c2,*ptr_w,*ptr_w1,*ptr_w2;
```

```
double *ptrParticleSize,*ptrXseries,*ptrSelecFunc,*ptrCurSelecFunc,*ptrNewSelecFunc,
                *ptrFtdSelecFunc,*ptrFtdCurSelecFunc,*ptrFtdNewSelecFunc;
double curBallDia,*ptrCurBallDia;
double newBallDia,*ptrNewBallDia;
char projectTitleBallSzOpt[85];
char logX[4],logY[4];

int main(void){
int i,j,jumper,key='$';
int sizeClassNumbers;
/*
show the current stack size
{
        printf("The stack length is %u\n",_stklen);
        getch();
} */
ptrWrkSpc=CreateVectorD(1,116);
for(i=1;i<=116;i++) ptrWrkSpc[i]=0.0;
ptrTempDoubleArray=CreateVectorD(1,MAXSIZECLASSNO);
screenSize=CreateVectorD(1,MAXSIZECLASSNO);
feedSizeDistribution=CreateVectorD(1,MAXSIZECLASSNO);
dischargeSizeDistribution=CreateVectorD(1,MAXSIZECLASSNO);
selectionFunction=CreateVectorD(1,MAXSIZECLASSNO);
breakageFunction=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
ptrTauPFSimGrCir=(double*) malloc(sizeof(double));
ptrTauSPMSimGrCir=(double*) malloc(sizeof(double));
ptrTauLPMSimGrCir=(double*) malloc(sizeof(double));
ptrRefMillFeedRateSimGrCir=(double*) malloc(sizeof(double));
ptrCurMillFeedRateSimGrCir=(double*) malloc(sizeof(double));
diag=CreateVectorD(1,MAXSIZECLASSNO);
t=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
tinv=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
tdiag=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
trans=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
for(i=1;i<=MAXSIZECLASSNO;i++){
        ptrTempDoubleArray[i]=0.0;
        screenSize[i]=0.0;
        feedSizeDistribution[i]=00.00;
        selectionFunction[i]=EPS;
        dischargeSizeDistribution[i]=0.00;
        diag[i]=0.00;
        for(j=1;j<=MAXSIZECLASSNO;j++){
                breakageFunction[i][j]=0.00;
                t[i][j]=0.0;
                tinv[i][j]=0.0;
                tdiag[i][j]=0.0;
                trans[i][j]=0.0;
        }
}
sizeClassNumSimGrCir=8;
*ptrTauPFSimGrCir=0.1;
```

```
*ptrTauSPMSimGrCir=0.1;
*ptrTauLPMSimGrCir=0.7;
*ptrRefMillFeedRateSimGrCir=100.0;
*ptrCurMillFeedRateSimGrCir=100.0;
ptrScreenSizeEst=CreateVectorD(1,MAXSIZECLASSNO);
ptrFeedEst=CreateVectorD(1,MAXSIZECLASSNO);
ptrDischargeMeasuredEst=CreateVectorD(1,MAXSIZECLASSNO);
ptrDischargeCalcEst=CreateVectorD(1,MAXSIZECLASSNO);
ptrBreakFuncEst=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
ptrSelectionFunctionEst=CreateVectorD(1,MAXSIZECLASSNO);
ptrDiag=CreateVectorD(1,MAXSIZECLASSNO);
ptrT=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
ptrTinv=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
ptrTdiag=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
ptrTrans=CreateMatrixD(1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);

for(i=1;i<=MAXSIZECLASSNO;i++){
        ptrScreenSizeEst[i]=00000.0;
        ptrFeedEst[i]=00.00;
        ptrDischargeMeasuredEst[i]=00.00;
        ptrDischargeCalcEst[i]=0.0;
        ptrSelectionFunctionEst[i]=EPS;
        ptrDiag[i]=0.0;
        for(j=1;j<=MAXSIZECLASSNO;j++){
                ptrBreakFuncEst[i][j]=0.00;
                ptrT[i][j]=0.0;
                ptrTinv[i][j]=0.0;
                ptrTdiag[i][j]=0.0;
                ptrTrans[i][j]=0.0;
        }
}
ptrTauPFSelecFuncEst=(double*) malloc(sizeof(double));
ptrTauSPMSelecFuncEst=(double*) malloc(sizeof(double));
ptrTauLPMSelecFuncEst=(double*) malloc(sizeof(double));
ptrRefMillFeedRateSFEst=(double*) malloc(sizeof(double));
ptrCurMillFeedRateSFEst=(double*) malloc(sizeof(double));
sizeClassNumSelecFuncEst=8;
*ptrTauPFSelecFuncEst=0.1;
*ptrTauSPMSelecFuncEst=0.1;
*ptrTauLPMSelecFuncEst=0.7;
*ptrRefMillFeedRateSFEst=100.0;
*ptrCurMillFeedRateSFEst=100.0;

ptrScreenSizeBallSzOpt=CreateVectorD(0,19);
ptrSelecFuncBallSzOpt=CreateVectorD(0,19);
ptrParticleSize=CreateVectorD(0,19);
ptrXseries=CreateVectorD(0,19);
ptrCurSelecFunc=CreateVectorD(0,19);
ptrNewSelecFunc=CreateVectorD(0,19);
ptrFtdCurSelecFunc=CreateVectorD(0,19);
ptrFtdNewSelecFunc=CreateVectorD(0,19);
```

```
ptrStandardDevBallSzOpt=CreateVectorD(0,19);
ptr_w1=CreateVectorD(0,19);
ptr_w2=CreateVectorD(0,19);
ptrPosKnots1=CreateVectorD(0,19);
ptrPosKnots2=CreateVectorD(0,19);
ptr_c1=CreateVectorD(0,26);
ptr_c2=CreateVectorD(0,26);
ptr_wrk=CreateVectorD(0,714);
ptr_iwrk=CreateVectorInt(0,26);
strcpy(projectTitleBallSzOpt,"Untitled project");
strcpy(logX,"N");
strcpy(logY,"N");
curBallDia=25.4;
newBallDia=25.4;
for(i=0;i<=19;i++){
        ptrScreenSizeBallSzOpt[i]=0.0;
        ptrParticleSize[i]=0.0;
        ptrXseries[i]=0.0;
        ptrSelecFuncBallSzOpt[i]=0.0;
        ptrStandardDevBallSzOpt[i]=1.0;
}
for(i=0;i<=19;i++){
        ptrPosKnots1[i]=0.0;
        ptrPosKnots2[i]=0.0;
        ptr_w1[i]=1.0;
        ptr_w2[i]=1.0;
}
for(i=0;i<=26;i++){
        ptr_c1[i]=0.0;
        ptr_c2[i]=0.0;
        ptr_iwrk[i]=0.0;
}
for(i=0;i<=714;i++) ptr_wrk[i]=0.0;
ShowIntroScreen();
ShowMainMenu();
for(;;){
        key=getch();
        switch(key){
                case 'E':
                case 'e':
                EstimateSelectionFunction();
                ShowMainMenu();
                break;

                case 'B':
                case 'b':
                SimulateGrindingCircuit();
                ShowMainMenu();
                break;

                case 'S':
```

```
            case 's':
            OptimizeBallSize();
            ShowMainMenu();
            break;
        }
        if(key == 'Q' || key == 'q') break;
}
FreeVectorD(ptrTempDoubleArray,1,MAXSIZECLASSNO);
FreeVectorD(ptrWrkSpc,1,116);
FreeVectorD(ptrFeedEst,1,MAXSIZECLASSNO);
FreeVectorD(ptrDischargeMeasuredEst,1,MAXSIZECLASSNO);
FreeVectorD(feedSizeDistribution,1,MAXSIZECLASSNO);
FreeVectorD(dischargeSizeDistribution,1,MAXSIZECLASSNO);
FreeVectorD(ptrDischargeCalcEst,1,MAXSIZECLASSNO);
FreeVectorD(ptrSelectionFunctionEst,1,MAXSIZECLASSNO);
FreeVectorD(selectionFunction,1,MAXSIZECLASSNO);
FreeVectorD(diag,1,MAXSIZECLASSNO);
FreeVectorD(ptrDiag,1,MAXSIZECLASSNO);
FreeMatrixD(t,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(tinv,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(tdiag,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(trans,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(ptrT,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(ptrTinv,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(ptrTdiag,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(ptrTrans,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(breakageFunction,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeMatrixD(ptrBreakFuncEst,1,MAXSIZECLASSNO,1,MAXSIZECLASSNO);
FreeVectorD(ptrScreenSizeBallSzOpt,0,19);
FreeVectorD(ptrSelecFuncBallSzOpt,0,19);
FreeVectorD(ptrParticleSize,0,19);
FreeVectorD(ptrXseries,0,19);
FreeVectorD(ptrCurSelecFunc,0,19);
FreeVectorD(ptrNewSelecFunc,0,19);
FreeVectorD(ptrFtdCurSelecFunc,0,19);
FreeVectorD(ptrFtdNewSelecFunc,0,19);
FreeVectorD(ptr_w1,0,19);
FreeVectorD(ptr_w2,0,19);
FreeVectorD(ptr_c1,0,26);
FreeVectorD(ptr_c2,0,26);
FreeVectorD(ptrPosKnots1,0,19);
FreeVectorD(ptrPosKnots2,0,19);
FreeVectorD(ptrStandardDevBallSzOpt,0,19);
FreeVectorD(ptr_wrk,0,714);
FreeVectorInt(ptr_iwrk,0,26);
clrscr();
return 0;
}
```

```
/* estsfprg.c01 */
int GetSelecFuncEstDataPg1(void);
int GetSelecFuncEstDataPg2(void);
int GetSelecFuncEstDataPg3(void);
int GetSelecFuncEstDataPg4(void);
int GetSelecFuncEstDataPg5(void);
int GetSelecFuncEstDataPg6(void);

int EstimateSelectionFunction(void){
int next=1;

for(;;){
        switch(next){
                case 1:
                next=GetSelecFuncEstDataPg1(); /* getting the general information */
                break;

                case 2:
                next=GetSelecFuncEstDataPg2(); /* getting the feed and discharge size */
                break;

                case 3:
                next=GetSelecFuncEstDataPg3(); /* getting the breakage function */
                break;

                case 4:
                next=GetSelecFuncEstDataPg4(); /* getting the breakage function continued */
                break;

                case 5:
                next=GetSelecFuncEstDataPg5(); /* getting RTD model parameters */
                break;

                case 6:
                next=GetSelecFuncEstDataPg6(); /* getting search settings */
                break;

                case 'Q':
                case 'q':
                break;
        }
        if(next==-1000) break;
}
return 0;
}
```

```
/* ret_sfed.c01 */
#include <dir.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include "global.h"

void PrintText(int x,int y,char* text,int forgcolor);
int HandleEndOfFile(void);
char *GetStringAt(int x,int y,int maxlen);
int GetFileNames(char *ptrExt);

int OpenSelecFuncEstimationFile(void){
FILE    *streamPtr;
char dataFileName[128];
char *ptrString;
char *newPath;
int i,j;
int status=0;
int key='$';
char string[15];
char *FormatErrPrompt=
"Data file format is incorrect! Press any key to continue ...";

gotoxy(1,24);
textcolor(YELLOW);
cprintf("Please enter file name?  ");
gotoxy(42,24);
textcolor(LIGHTGRAY);
cprintf(                    "(type > to change current directory)");
clreol();
textcolor(YELLOW);
ptrString=GetStringAt(25,24,15);
if(ptrString[0]=='>'){
        gotoxy(1,24);
        textcolor(YELLOW);
        cprintf("Please enter new directory:  ");
        clreol();
        textcolor(LIGHTGRAY);
        newPath=GetStringAt(29,24,54);
        if(chdir(newPath)!=0){
                gotoxy(1,24);
                textcolor(LIGHTRED);
                if(strlen(newPath)<=30)
                        cprintf("Cannot change to [%s].  Press any key to continue ...",newPath);
                else
                        cprintf("Cannot change to new directory.  Press any key to continue ...");
                clreol();
                getch();
```

```
                return 2;
        }
        else{
                fflush(stdin);
                GetFileNames("sfd");
                gotoxy(1,24);
                textcolor(YELLOW);
                cprintf("Please enter file name?  ");
                textcolor(YELLOW);
                ptrString=GetStringAt(25,24,15);
        }
}
strcpy(dataFileName,ptrString);
if(strlen(dataFileName)<=8) strcat(dataFileName,".sfd");
if((streamPtr=fopen((const char*) dataFileName,"r"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't open file! press any key to continue ...");
        clreol();
        getch();
        return 2;
}
else{
/* reading one string for the project title */
        while((key=getc(streamPtr))!=EOF){
                if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                else{
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        fgets(projectTitleEst,80,streamPtr);
                        projectTitleEst[strlen(projectTitleEst)-1]='\0';
                        break;
                }
        }
        if(key==EOF){
                HandleEndOfFile();
                return 2;
        }
        /* reading sizeClassNumbers */
        status=0;
        while((key=getc(streamPtr))!=EOF && status<1) {
                if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                else{
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        status=fscanf(streamPtr,"%d",&sizeClassNumSelecFuncEst);
                        if(status<1){
                                PrintText(1,24,FormatErrPrompt,LIGHTRED);
                                clreol();
                                getch();
                                return 2;
                        }
                } /* for else */
```

```
} /* end of while */
if(key==EOF){
        HandleEndOfFile();
        return 2;
}
/* reading the rest of file i.e. sieveSeries, feed size,
selection function */
for(i=1;i<=sizeClassNumSelecFuncEst;i++){
        status=0;
        key= '$';
        while((key=getc(streamPtr))!=EOF&&status<3){
                if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                status=fscanf(streamPtr,"%lf %lf %lf",ptrScreenSizeEst+i,ptrFeedEst+i,
                                                        ptrDischargeMeasuredEst+i);
                if(status<3){
                        PrintText(1,24,FormatErrPrompt,LIGHTRED);
                        clreol();
                        getch();
                        return 2;
                }
        } /* end of while */
        if(key==EOF){
                HandleEndOfFile();
                return 2;
        }
} /* end of for */
/* reading the rest of data file i.e. breakage function values
and test for normalisability */
while((key=getc(streamPtr))!=EOF){
        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
        fscanf(streamPtr,"%s",string);
        if(string[0]!='Y'&&string[0]!='y'&&string[0]!='N'&&string[0]!='n'){
                PrintText(1,24,FormatErrPrompt,LIGHTRED);
                clreol();
                getch();
                return 0;
        }
        else break;
}
if(key==EOF){
        HandleEndOfFile();
        return 2;
}
if(string[0]=='Y'||string[0]=='y'){
        normBreakFuncEst=TRUE;
        for(i=1;i<=sizeClassNumSelecFuncEst;i++){
                status =0;
                key='$';
                while((key=getc(streamPtr))!=EOF&&status<1){
```

```
                        if(key==\'\n\'||key==\' \'||key==\'\t\'||key==\',\') continue;
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        status=fscanf(streamPtr,"%lf",&ptrBreakFuncEst[i][1]);
                        if(status<1){
                                PrintText(1,24,FormatErrPrompt,LIGHTRED);
                                clreol();
                                getch();
                                return 2;
                        }
                } /* end of while */
                if(key==EOF){
                        HandleEndOfFile();
                        return 2;
                }
        } /* end of for */
        for(i=1;i<=sizeClassNumSelecFuncEst;i++)
                for(j=1;j <= sizeClassNumSelecFuncEst;j++)
                        if((i<j)||(i==j))
                                ptrBreakFuncEst[i][j]=0;
                        else if(j>1)
                                ptrBreakFuncEst[i][j]=ptrBreakFuncEst[i-1][j-1];
}
else if(string[0]==\'N\'||string[0]==\'n\'){
        normBreakFuncEst=FALSE;
        for(i=1;i<=sizeClassNumSelecFuncEst;i++)
                for(j=1;j<=sizeClassNumSelecFuncEst;j++){
                        if(i<j){
                                ptrBreakFuncEst[i][j]=0;
                                continue;
                        }
                        status=0;
                        while((key=getc(streamPtr))!=EOF&&status<1){
                                if(key==\'\n\'||key==\' \'||key==\'\t\'||key==\',\')
continue;
                                fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                                status=fscanf(streamPtr,"%lf",&ptrBreakFuncEst[i][j]);
                                if(status<1){
                                        clrscr();
                                        PrintText(1,24,FormatErrPrompt,LIGHTRED);
                                        clreol();
                                        getch();
                                        return 2;
                                }
                        }  /* end of while */
                        if(key==EOF){
                                HandleEndOfFile();
                                return 2;
                        }
                } /* end of for */
        }
        /* reading mean retention times ... */
```

```
                status=0;
                key='$';
                while((key=getc(streamPtr))!=EOF&&status<5){
                        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        status=fscanf(streamPtr,"%lf %lf %lf %lf %lf",ptrTauPFSelecFuncEst,
ptrTauSPMSelecFuncEst,ptrTauLPMSelecFuncEst,
ptrRefMillFeedRateSFEst,ptrCurMillFeedRateSFEst);
                        if(status<5){
                                PrintText(1,24,FormatErrPrompt,LIGHTRED);
                                clreol();
                                getch();
                                return 2;
                        }
                }
        }
        if(fclose(streamPtr)!=0){
                gotoxy(1,24);
                textcolor(LIGHTRED);
                cprintf("Can't close file! Press any key to continue ...\n\r");
                clreol();
                textcolor(LIGHTGRAY);
                getch();
                return 2;
        }
        gotoxy(1,24);
        textcolor(YELLOW);
        cprintf("Data transfered successfully, press any key to continue ...");
        clreol();
        textcolor(LIGHTGRAY);
        getch();
        return 0;
}
```

```
/* sfestdp1.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include "global.h"
#include "gets.h"

#define promptEndPos 47

extern int savescr[4096];

int GetFileNames(char *ptrExt);
void PrintText(int x,int y, char* text,int forgcolor);
void PrintErrMsg(void);
int OpenSelecFuncEstimationFile(void);

int GetSelecFuncEstDataPg1(void){
int key,status,retVal;
int i,j;
bool done=FALSE;
int *address;
char *string;
char *prompt="Open <O>  Change <C>  Next<N>  Quit <Q> ==>";
char isBreakageNormalisable[6];
if(normBreakFuncEst)
        strcpy((char*)isBreakageNormalisable,"yes");
else
        strcpy((char*)isBreakageNormalisable,"no");
clrscr();
PrintText(10,4,"General Information for Selection Function Estimation",
WHITE);
gotoxy(3,6);
cprintf("Project title              %s",projectTitleEst);
gotoxy(3,7);
cprintf("Size class numbers          %d",sizeClassNumSelecFuncEst);
gotoxy(3,8);
cprintf("Normalisable breakage       %s",isBreakageNormalisable);
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
textcolor(LIGHTGRAY);
while(!done){
        key=getch();
        switch(key){
                case 'O':
                case 'o':
                gettext(1,1,80,25,savescr);
                GetFileNames("sfd");
                OpenSelecFuncEstimationFile();
```

```
puttext(1,1,80,25,savescr);
textcolor(LIGHTGRAY);
gotoxy(35,6);
cprintf("%s",projectTitleEst);
clreol();
gotoxy(35,7);
cprintf("%d",sizeClassNumSelecFuncEst);
gotoxy(35,8);
cprintf("%s",isBreakageNormalisable);
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
clreol();
textcolor(LIGHTGRAY);
break;

case 'C':
case 'c':
gotoxy(35,6);
/* cursor at 35,6 */
fflush(stdin);
if((key=getch())!=ESC){
        ungetch(key);
        if(key!='\r') clreol();
        string=GetStringAt(35,6,45);
        if(string[0]=='\0'){
                cprintf("%s",projectTitleEst);
                clreol();
                gotoxy(35,7);
        }
        else{
                strcpy(projectTitleEst,string);
                gotoxy(35,6);
                cprintf("%s",projectTitleEst);
                clreol();
                gotoxy(35,7);
        }
}
else{
        gotoxy(promptEndPos,24);
        break;
}
/* cursor at 35,7 */
fflush(stdin);
if((key=getch())!=ESC){
        ungetch(key);
        address=&sizeClassNumSelecFuncEst;
        status=0;
        while(!done){
                string=GetStringAt(35,7,5);
                if(string[0]=='\0'){
```

```
                        cprintf("%1d",*address);
                        cprintf("          ");
                        gotoxy(35,8);
                        break;
                }
                status=sscanf(string,"%d",address);
                if(status==1){
                        if(*address>MAXSIZECLASSNO||*address<1){
                                PrintErrMsg();
                                PrintText(1,24,prompt,YELLOW);
                                gotoxy(35,7);
                                cprintf("       ");
                                gotoxy(35,7);
                                done=FALSE;
                        }
                        else{
                                gotoxy(35,7);
                                cprintf("%1d",*address);
                                cprintf("          ");
                                gotoxy(35,8);
                                break;
                        }
                }
                else{
                        PrintErrMsg();
                        gotoxy(1,24);
                        textcolor(YELLOW);
                        cprintf(prompt);
                        textcolor(LIGHTGRAY);
                        gotoxy(35,7);
                        cprintf("          ");
                        gotoxy(35,7);
                }
        }
}
else{
        gotoxy(promptEndPos,24);
        break;
}
/* cursor at 35,8 */
fflush(stdin);
if((key=getch())!=ESC){
        ungetch(key);
        if(key!='\r') clreol();
                string[0]='\0';
        while(string[0]!='Y'&&string[0]!='y'&&string[0]!='N'&&string[0]!='n'){
                string=GetStringAt(35,8,6);
                if(string[0]=='\0'){
                        cprintf("%s",isBreakageNormalisable);
                        clreol();
                        gotoxy(promptEndPos,24);
```

```
                                        break;
                                    }
                                    else{

if(string[0]=='Y'||string[0]=='y'||string[0]=='N'||string[0]=='n'){
                                                if(string[0]=='Y'||string[0]=='y')

strcpy((char*)isBreakageNormalisable,"yes");
                                                else

strcpy((char*)isBreakageNormalisable,"no");
                                                gotoxy(35,8);
                                                cprintf("%s",isBreakageNormalisable);
                                                clreol();
                                                if(string[0]=='Y'||string[0]=='y'){
                                                        normBreakFuncEst=TRUE;

for(i=1;i<=sizeClassNumSelecFuncEst;i++){

for(j=1;j<=sizeClassNumSelecFuncEst;j++)

ptrBreakFuncEst[i+1][j+1]=ptrBreakFuncEst[i][j];
                                                                }
                                                }
                                                else  normBreakFuncEst=FALSE;
                                                gotoxy(promptEndPos,24);
                                                break;
                                        } /* end of while */
                                        else{
                                                PrintText(1,24,"Please answer by yes or
no!",LIGHTRED);
                                                clreol();
                                                delay(700);
                                                PrintText(1,24,prompt,YELLOW);
                                        }
                                }
                        }
                }
                else{
                        gotoxy(promptEndPos,24);
                        break;
                }
                break;

                case 'N':
                case 'n':
                retVal=2;
                done=TRUE;
                break;

                case 'Q':
```

```
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
                }  /* end of switch */
        }  /* end of while */
return retVal;
}
```

```
/* sfestdp2.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include "global.h"
#include "gets.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

extern char savescr[4096];
extern  double *ptrTempDoubleArray;

void InitializeGraphics(void);
void TerminateGraphics(void);
void PrintText(int x,int y,char* text,int forgcolor);
void GetColOfData(int row,int column,int noSzClasses,int* xOfColumnPtr,
                                                    int maxlen,char* formatStr,char* formatStrLeftJus,
                                                    double minAllowedRange,double
maxAllowedRange,
                                                    double *tempInputPtr);
int PlotScatterDig(double *xDataVec,double *yDataVec,int m,
                                                     char* title,char* xLab,char* yLab);

enum plottype{SIZEDIST=1,SELECTIONFUNC=2,BREAKAGEFUNC=3};
enum plottype plotType;

int GetSelecFuncEstDataPg2(void){
extern int GraphMode;
extern operationmode mode;
bool done=FALSE;
char *formatStr;
char *formatStrLeftJus;
char *addressChar[20];

int i,noSzClasses,key,retVal;
double minAllowedRange,maxAllowedRange;
double sum;

int row=4;
int column;
int xOfColumn[3]={20,40,60};
int maxlen;
```

```
int* xOfColumnPtr;
double *tempInputPtr;

char *prompt=
"Change <C>  Previous <P>  Next <N>  Graph <G>  Quit <Q> ==>";
mode=SELECTIONFUNCEST;
xOfColumnPtr=&xOfColumn[0]-1;
clrscr();
PrintText(5,1,"Feed Size And Discharge Size Data for Selection Function Estimation",WHITE);
gotoxy(1,3);
cprintf("Size Class No.");
gotoxy(xOfColumnPtr[1],3);
cprintf("Screen Size (μm)");
gotoxy(xOfColumnPtr[2],3);
cprintf("Feed Size (%)");
gotoxy(xOfColumnPtr[3],3);
cprintf("Discharge Size (%)");
gotoxy(5,4);
noSzClasses=sizeClassNumSelecFuncEst;
for(i=1;i<=noSzClasses;i++){
        gotoxy(5,i+3);
        cprintf("%2d",i);
        gotoxy(xOfColumnPtr[1],i+3);
        cprintf("%10.0lf",ptrScreenSizeEst[i]);
        gotoxy(xOfColumnPtr[2],i+3);
        cprintf("%10.2lf",ptrFeedEst[i]);
        gotoxy(xOfColumnPtr[3],i+3);
        cprintf("%10.2lf",ptrDischargeMeasuredEst[i]);
}
PrintText(1,24,prompt,YELLOW);
while(!done){
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                column=1;
                maxlen=15;
                formatStr="%10.0lf";
                formatStrLeftJus="%-10.0lf";
                minAllowedRange=0;
                maxAllowedRange=1e14;
                tempInputPtr=ptrScreenSizeEst;

GetColOfData(row,column,noSzClasses,xOfColumnPtr,maxlen,formatStr,formatStrLeftJus,

minAllowedRange,maxAllowedRange,tempInputPtr);
                column=2;
                maxlen=15;
                formatStr="%10.2lf";
                formatStrLeftJus="%-10.2lf";
                minAllowedRange=0;
```

```
                maxAllowedRange=100;
                tempInputPtr=ptrFeedEst;

GetColOfData(row,column,noSzClasses,xOfColumnPtr,maxlen,formatStr,formatStrLeftJus,

minAllowedRange,maxAllowedRange,tempInputPtr);
                column=3;
                maxlen=15;
                formatStr="%10.2lf";
                formatStrLeftJus="%-10.2lf";
                minAllowedRange=0;
                maxAllowedRange=100;
                tempInputPtr=ptrDischargeMeasuredEst;

GetColOfData(row,column,noSzClasses,xOfColumnPtr,maxlen,formatStr,formatStrLeftJus,

minAllowedRange,maxAllowedRange,tempInputPtr);
                gotoxy(64,24);
                break;

                case 'P':
                case 'p':
                retVal=1;
                done=TRUE;
                break;

                case 'N':
                case 'n':

                sum=0;
                for(i=1;i<=noSzClasses;i++)
                        sum=sum+ptrFeedEst[i];
                if(GT(sum,100)){
                        gettext(1,10,80,14,savescr);

PrintText(2,11,"┌──────────────────────────────────────────────────────────────────────────────┐
",WHITE);
                        PrintText(2,12,"│  The total mass in feed exceeds 100%. Press any key to
continue ...   │ ",WHITE);

PrintText(2,13,"└──────────────────────────────────────────────────────────────────────────────┘
",WHITE);
                        _setcursortype(_NOCURSOR);
                        getch();
                        gotoxy(20,10);
                        puttext(1,10,80,14,savescr);
                        _setcursortype(_NORMALCURSOR);
                        gotoxy(64,24);
                        break;
```

```
                }
                sum=0;
                for(i=1;i<=noSzClasses;i++)
                        sum=sum+ptrDischargeMeasuredEst[i];
                if(GT(sum,100)){
                        gettext(1,10,80,14,savescr);

PrintText(2,11," ┌──────────────────────────────────────────────────────────────────────────────┐
",WHITE);
                        PrintText(2,12," |  The total mass in discharge exceeds 100%.  Press any
key to continue ...   | ",WHITE);

PrintText(2,13," └──────────────────────────────────────────────────────────────────────────────┘
",WHITE);
                        _setcursortype(_NOCURSOR);
                        getch();
                        gotoxy(20,10);
                        puttext(1,10,80,14,savescr);
                        _setcursortype(_NORMALCURSOR);
                        gotoxy(64,24);
                        break;
                }
                retVal=3;
                done=TRUE;
                break;

                case 'G':
                case 'g':
                gettext(1,1,80,25,savescr);
                InitializeGraphics();
                plotType=SIZEDIST;
                sum=0;
                for(i=1;i<=noSzClasses;i++){
                        sum=sum+ptrFeedEst[i];
                        ptrTempDoubleArray[i]=100-sum;
                }
                PlotScatterDig(ptrScreenSizeEst,ptrTempDoubleArray,noSzClasses,
                                                        "Measured Feed Size
Distribution","Particle Size,μm",
                                                        "Mass (%)");
                getch();
                sum=0;
                for(i=1;i<=noSzClasses;i++){
                        sum=sum+ptrDischargeMeasuredEst[i];
                        ptrTempDoubleArray[i]=100-sum;
                }
                PlotScatterDig(ptrScreenSizeEst,ptrTempDoubleArray,noSzClasses,
                                                        "Measured Discharge Size
Distribution","Particle Size, μm",
```

```
                                                    "Mass (%)");
                getch();
                TerminateGraphics();
                puttext(1,1,80,25,savescr);
                gotoxy(64,24);
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
        }
}
return retVal;
}
```

```
/* sfestdp3.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <float.h>
#include "global.h"
#include "gets.h"

#define EPS  (DBL_EPSILON*100)
#define EQ(x,y)  (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y)  ((x)<(y) || EQ(x,y))
#define GE(x,y)  ((y)<(x) || EQ(x,y))
#define LT(x,y)  (!GE(x,y))
#define GT(x,y)  (!LE(x,y))

#define promptEndPos  51

extern char savescr[4096];

void PrintErrMsg(void);
void PrintText(int x,int y,char* text,int forgcolor);
int  CheckSumOfColumns(double **matrixPtr,bool bfNormFlag,int jIterNum,
                                                    int iIterNum,int endPos);

int GetSelecFuncEstDataPg3(void){
bool done=FALSE;
bool canUseArraysValues=TRUE;
int maxIt,retVal;
int row=4;
int column=0;
int xOfColumn;
int maxlen=12;
char *formatStr;
double addressFloat;
double sum;
int status=0;
char *string;
char buffer[10];
int i,j,key;

char *prompt=
"Change <C>   Previous <P>   Next <N>   Quit<Q> ==>";

clrscr();
PrintText(10,1,"Breakage Function Data for Selection Function Estimation",WHITE);
if(normBreakFuncEst)
        PrintText(1,3,"Normalisable breakage function",GREEN);
else
        PrintText(1,3,"Non-normalisable breakage function",GREEN);
```

```
for(i=1;i<=sizeClassNumSelecFuncEst;i++)
        for(j=1;j<=10;j++)
                if(i>=j){
                        xOfColumn=1+(j-1)*8;
                        gotoxy(xOfColumn,i+3);
                        cprintf("%6.4f",ptrBreakFuncEst[i][j]);
                }
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
textcolor(LIGHTGRAY);
while(!done){
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                gotoxy(1,7);
                if(sizeClassNumSelecFuncEst<=10) maxIt=sizeClassNumSelecFuncEst;
                else maxIt=10;
                for(column=1;column<=maxIt;column++){
                        maxlen=9;
                        formatStr="%6.4f";
                        xOfColumn=1+(column-1) * 8;
                        gotoxy(xOfColumn,(column-1)+5);
                        for(row=column+4;row<sizeClassNumSelecFuncEst+4;row++){
                                key=getch();
                                if(key==ESC){
                                        gotoxy(promptEndPos,24);
                                        break;
                                }
                                ungetch(key);
                                putch(key);
                                done=FALSE;
                                canUseArraysValues=TRUE;
                                status=0;
                                while(!done){
                                        string=GetStringAt(xOfColumn,row,maxlen);
                                        if(string[0]=='\0'&&canUseArraysValues){
                                                gotoxy(xOfColumn,row);
                                                cprintf(formatStr,ptrBreakFuncEst[row-3][column]);
                                                if(row<sizeClassNumSelecFuncEst+3)
gotoxy(xOfColumn,row+1);
                                                break;
                                        }

status=sscanf(string,"%lf",&(ptrBreakFuncEst[row-3][column]));
                                        if(status==1&&ptrBreakFuncEst[row-3][column]<=1&&
                                                ptrBreakFuncEst[row-3][column]>=0){
                                                gotoxy(xOfColumn,row);
                                                cprintf(formatStr,ptrBreakFuncEst[row-3][column]);
                                                gotoxy(xOfColumn,row+1);
```

```
                                        if(normBreakFuncEst){

for(i=1;i<=sizeClassNumSelecFuncEst-row+3;i++){

ptrBreakFuncEst[row-3+i][column+i]=ptrBreakFuncEst[row-3][column];
                                                        if(i<=9){
                                                                gotoxy(1+i*8,row+i);

cprintf(formatStr,ptrBreakFuncEst[row-3][column]);
                                                        }
                                                }
                                                gotoxy(xOfColumn,row+1);
                                        }
                                        break;
                                }
                                else{
                                        PrintErrMsg();
                                        gotoxy(1,24);
                                        textcolor(YELLOW);
                                        cprintf(prompt);
                                        textcolor(LIGHTGRAY);
                                        gotoxy(xOfColumn,row);
                                        cprintf("      ");
                                        gotoxy(xOfColumn,row);
                                        done=FALSE;
                                        canUseArraysValues=FALSE;
                                }
                        }
                } /* end of for row */
                if(key==ESC)
                        break;
                if(normBreakFuncEst)
                        break;
        } /* end of for columns */
        gotoxy(promptEndPos,24);
        break;

        case 'P':
        case 'p':
        retVal=2;
        done=TRUE;
        break;

        case 'N':
        case 'n':
        if(sizeClassNumSelecFuncEst>10){
                if(CheckSumOfColumns(ptrBreakFuncEst,normBreakFuncEst,

10,sizeClassNumSelecFuncEst,

promptEndPos)==0){
```

```
                                retVal=4;
                                done=TRUE;
                                break;
                        }
                        else{
                                done=FALSE;
                                break;
                        }
                }
                else{
                        if(CheckSumOfColumns(ptrBreakFuncEst,normBreakFuncEst,
sizeClassNumSelecFuncEst,
sizeClassNumSelecFuncEst,
                                                                promptEndPos)==0){
                                retVal=5;
                                done=TRUE;
                                break;
                        }
                        else{
                                done=FALSE;
                                break;
                        }
                }

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
        }
}
return retVal;
}

int  CheckSumOfColumns(double **matrixPtr,bool bfNormFlag,int jIterNum,
                                                        int iIterNum,int endPos){
int i,j;
double sum;

if(bfNormFlag){
        sum=0;
        for(i=1;i<=iIterNum;i++)
                sum=sum+matrixPtr[i][1];
        if(GT(sum,1)){
                gettext(1,10,80,14,saveacr);

PrintText(2,11,"┌──────────────────────────────────────────────────────────────────────────────┐
",WHITE);
```

```
                PrintText(2,12," |      The sum of bf values in any column must be < = 1. Press any
key ...    | ",WHITE);

PrintText(2,13," └──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────┘
",WHITE);
                _setcursortype(_NOCURSOR);
                getch();
                gotoxy(20,10);
                puttext(1,10,80,14,savescr);
                _setcursortype(_NORMALCURSOR);
                gotoxy(endPos,24);
                return 1;
        }
}
else{
        for(j=1;j<=jIterNum;j++){
                sum=0;
                for(i=1;i<=iIterNum;i++)
                        sum=sum+matrixPtr[i][j];
                if(GT(sum,1)){
                        gettext(1,10,80,15,savescr);

PrintText(2,11," ┌──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────┐
",WHITE);
                        PrintText(2,12," |      The sum of bf values in any column must be < = 1.
Press any key ...    | ",WHITE);

PrintText(2,13," └──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────────┘
",WHITE);
                        _setcursortype(_NOCURSOR);
                        getch();
                        gotoxy(20,10);
                        puttext(1,10,80,15,savescr);
                        _setcursortype(_NORMALCURSOR);
                        gotoxy(endPos,24);
                        return 1;
                }
        }
}
return 0;
}
```

```
/* sfestdp4.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include "global.h"
#include "gets.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

#define promptEndPos  51
#define row14         14
#define row15         row14+1

extern char savescr[4096];

void PrintText(int x,int y,char* text,int forgcolor);
void PrintErrMsg(void);
int CheckSumRestColumns(double **matrixPtr,int jIterNum,
                                                    int iIterNum,int endPos);

int GetSelecFuncEstDataPg4(void){
bool done=FALSE;
bool canUseArraysValues=TRUE;
int i,j,key;
char *formatStr;
int retVal;
int row=row14;
int column=0;
int xOfColumn;
int maxlen=12;
double addressFloat;
int status=0;
char *string;
char prompt[81];
char buffer[10];
double sum;

if(normBreakFuncEst) strcpy(prompt,
"Previous <P>  Next <N>  Quit<Q> ==>");
else  strcpy(prompt,
"Change <C>  Previous <P>  Next <N>  Quit<Q> ==>");
clrscr();
```

```
PrintText(5,1,"Breakage Function Data for Selection Function Estimation (continued)",WHITE);
if(normBreakFuncEst)
PrintText(1,3,"Normalisable breakage function",GREEN);
else
PrintText(1,3,"Non-normalisable breakage function",GREEN);
for(i=11;i<=sizeClassNumSelecFuncEst;i++)
        for(j=11;j<=sizeClassNumSelecFuncEst;j++)
                if(i>=j){
                        xOfColumn=1+(j-11)*8;
                        gotoxy(xOfColumn,(i-11)+row14);
                        cprintf("%6.4f",ptrBreakFuncEst[i][j]);
                }
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
textcolor(LIGHTGRAY);
while(!done){
        key=getch();
        switch(key){
        case 'C':
        case 'c':
        if(normBreakFuncEst) break;
        for(column=11;column<=sizeClassNumSelecFuncEst;column++){
                maxlen=9;
                formatStr="%6.4f";
                xOfColumn=1+(column-11)*8;
                gotoxy(xOfColumn,row15+column-11);
                for(row=row15+column-11;row<=sizeClassNumSelecFuncEst+3;row++){
                        key=getch();
                        if(key==ESC){
                                if(normBreakFuncEst) gotoxy(38,24);
                                else gotoxy(promptEndPos,24);
                                break;
                        }
                        ungetch(key);
                        putch(key);
                        done=FALSE;
                        canUseArraysValues=TRUE;
                        status=0;
                        while(!done){
                                string=GetStringAt(xOfColumn,row,maxlen);
                                if(string[0]=='\0'&&canUseArraysValues){
                                        gotoxy(xOfColumn,row);
                                        cprintf(formatStr,ptrBreakFuncEst[row-3][column]);
                                        if(row<=sizeClassNumSelecFuncEst+3)
gotoxy(xOfColumn,row+1);
                                                break;
                                }
                                status=sscanf(string,"%lf",&(ptrBreakFuncEst[row-3][column]));
                                if(status==1&&ptrBreakFuncEst[row-3][column]<=1&&
                                        ptrBreakFuncEst[row-3][column]>=0){
```

```
                                        gotoxy(xOfColumn,row);
                                        cprintf(formatStr,ptrBreakFuncEst[row-3][column]);
                                        gotoxy(xOfColumn,row+1);
                                        break;
                                }
                                else{
                                        PrintErrMsg();
                                        gotoxy(1,24);
                                        textcolor(YELLOW);
                                        cprintf(prompt);
                                        textcolor(LIGHTGRAY);
                                        gotoxy(xOfColumn,row);
                                        cprintf("       ");
                                        gotoxy(xOfColumn,row);
                                        done=FALSE;
                                        canUseArraysValues=FALSE;
                                }
                        }
                }
                if(key==ESC) break;
        }
        if(normBreakFuncEst)
                        gotoxy(38,24);
        else gotoxy(promptEndPos,24);
                break;

        case 'P':
        case 'p':
        retVal=3;
        done=TRUE;
        break;

        case 'N':
        case 'n':
        if(!normBreakFuncEst){
                if(CheckSumRestColumns(ptrBreakFuncEst,sizeClassNumSelecFuncEst,

sizeClassNumSelecFuncEst,
                                                                promptEndPos)==0){
                        retVal=5;
                        done=TRUE;
                        break;
                }
                else{
                        done=FALSE;
                        break;
                }
        }
        else{
                retVal=5;
                done=TRUE;
```

```
                break;
        }

        case 'Q':
        case 'q':
        retVal=-1000;
        done=TRUE;
        break;
        }  /* end of switch */
} /* end of while */
return retVal;
}

int  CheckSumRestColumns(double **matrixPtr,int jIterNum,
                                                        int iIterNum,int endPos){
int i,j;
double sum;

for(j=11;j<=jIterNum;j++){
        sum=0;
        for(i=1;i<=iIterNum;i++)
                sum=sum+matrixPtr[i][j];
        if(GT(sum,1)){
                gettext(1,10,80,15,savescr);

PrintText(2,11,"┌──────────────────────────────────────────────────────────────
──────────────────────────────────────────────┐
",WHITE);
                PrintText(2,12,"|     The sum of bf values in any column must be <= 1. Press any
key ...    | ",WHITE);

PrintText(2,13,"└──────────────────────────────────────────────────────────────
──────────────────────────────────────────────┘
",WHITE);
                _setcursortype(_NOCURSOR);
                getch();
                gotoxy(20,10);
                puttext(1,10,80,15,savescr);
                _setcursortype(_NORMALCURSOR);
                gotoxy(endPos,24);
                return 1;
        }
}
return 0;
}
```

```
/* sfestdp5.c01 */
#include <conio.h>
#include "global.h"

void PrintText(int x,int y, char* text,int forgcolor);
int GetRtdInfo(double *ptrTauPF,double *ptrTauSPM,double *ptrTauLPM,
                                        double *ptrRefMillFeedRate,double *ptrCurMillFeedRate);

int GetSelecFuncEstDataPg5(){
bool done=FALSE;
int key,retVal;
char scrTitle[81]=
"Residence Time Distribution Data for Selection Function Estimation";
char *prompt =
"Change <C>  Previous <P>  Next <N>  Quit <Q> ==>";

clrscr();
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
textcolor(LIGHTGRAY);
PrintText(5,4,scrTitle,WHITE);
gotoxy(3,6);
cprintf("Reference tau of plug flow:          %.3f",*ptrTauPFSelecFuncEst);
gotoxy(3,7);
cprintf("Reference tau of small perfect mixers:   %.3f",*ptrTauSPMSelecFuncEst);
gotoxy(3,8);
cprintf("Reference tau of large perfect mixer:    %.3f",*ptrTauLPMSelecFuncEst);
gotoxy(3,9);
cprintf("Reference feed rate (t/h):           %.3f",*ptrRefMillFeedRateSFEst);
gotoxy(3,10);
cprintf("Current feed rate (t/h):             %.3f",*ptrCurMillFeedRateSFEst);
gotoxy(52,24);
while(!done){
        key=getch();
        switch(key){
        case 'C':
        case 'c':
        GetRtdInfo(ptrTauPFSelecFuncEst,ptrTauSPMSelecFuncEst,
                                ptrTauLPMSelecFuncEst,ptrRefMillFeedRateSFEst,
                                ptrCurMillFeedRateSFEst);
        gotoxy(52,24);
        break;

        case 'P':
        case 'p':
        if(sizeClassNumSelecFuncEst>10) retVal=4;
        else retVal=3;
        done=TRUE;
        break;
```

```
        case 'N':
        case 'n':
        retVal=6;
        done=TRUE;
        break;

        case 'Q':
        case 'q':
        retVal=-1000;
        done=TRUE;
        break;
 }
}
return retVal;
}
```

```
/* sfestdp6.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include <math.h>
#include <float.h>
#include "global.h"
#include "gets.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y))<=fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

#define promptEndPos 67

extern char savescr[4096];

int WriteToScreenEst(void);
int DoSearch(double* ptrMinPossibleSf,double* ptrMaxPossibleSf,double* ptrPrecision);
void PrintText(int x,int y,char* text,int forgcolor);
void PrintErrMsg(void);
int SaveSelecFuncEstPrgData(void);
int ComputeSelectionFunction(double *ptrLowerSchLim,double *ptrUpperSchLim,
                                                                    double *ptrPrecision);

int GetSelecFuncEstDataPg6(void){
char *string;
char stringf[80];
char prompt[81]=
"Change <C>  Previous <P>  Save data <S>  Run <R>  Quit <Q> ==>";
int i,j,row,key,status,retVal;
int ndig,dec,sign;
float minAllowedValue=0.0;
double *address;
static double lowerSearchLimit=0,upperSearchLimit=100,precision=0.0001;
bool done=FALSE;
bool canUseCurValue=TRUE;

clrscr();
PrintText(5,4,
"Search Specification Data for Selection Function Estimation",WHITE);
gotoxy(3,6);
cprintf("Lower limit of search      %-.0f",lowerSearchLimit);
gotoxy(3,7);
cprintf("Upper limit of search      %-.0f",upperSearchLimit);
gotoxy(3,8);
```

```
cprintf("Desired precision          %-.4f",precision);
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
textcolor(LIGHTGRAY);
while(!done){
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                gotoxy(32,6);
                for(row=6;row<9;row++){
                        switch(row){
                                case 6:
                                address=&lowerSearchLimit;
                                minAllowedValue=0;
                                break;

                                case 7:
                                address=&upperSearchLimit;
                                minAllowedValue=1;
                                break;

                                case 8:
                                address=&precision;
                                minAllowedValue=0;
                                break;
                        }
                        key=getch();
                        if(key==ESC){
                                gotoxy(promptEndPos,24);
                                break;
                        }
                        ungetch(key);
                        putch(key);
                        done=FALSE;
                        canUseCurValue=TRUE;
                        status=0;
                        while(!done){
                                string[0]='\0';
                                string=GetStringAt(32,row,14);
                                if(string[0]=='\0'&& canUseCurValue){
                                                gotoxy(32,row+1);
                                                break;
                                }
                                status =sscanf(string,"%lf",address);
                                if(status==1 && GE(*address,minAllowedValue)){
                                        gotoxy(32,row);
                                        sprintf(stringf,"%.9f",*address);
                                        cprintf("%s",stringf);
                                        clreol();
```

```
                                gotoxy(32,row+1);
                                done = TRUE;
                        }
                        else{
                                PrintErrMsg();
                                gotoxy(1,24);
                                textcolor(YELLOW);
                                cprintf(prompt);
                                clreol();
                                textcolor(LIGHTGRAY);
                                gotoxy(32,row);
                                clreol();
                                gotoxy(32,row);
                                done = FALSE;
                                canUseCurValue = FALSE;
                        }
                }
        }
        gotoxy(promptEndPos,24);
        done=FALSE;
        break;

        case 'P':
        case 'p':
        retVal=5;
        done=TRUE;
        break;

        case 'S':
        case 's':
        SaveSelecFuncEstPrgData();
        gotoxy(1,24);
        textcolor(YELLOW);
        cprintf(prompt);
        textcolor(LIGHTGRAY);
        break;

        case 'R':
        case 'r':
        if(!GT(upperSearchLimit,lowerSearchLimit)){
                gettext(1,10,80,14,savescr);
PrintText(9,11," ┌──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────┐ ",WHITE);
                PrintText(9,12," |  Upper limit of search must be greater than the lower
limit  | " ,WHITE);
PrintText(9,13," └──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────┘ ",WHITE);
                _setcursortype(_NOCURSOR);
                getch();
```

```
                    gotoxy(20,10);
                    puttext(1,10,80,14,savescr);
                    _setcursortype(_NORMALCURSOR);
                    gotoxy(promptEndPos,24);
                    done=FALSE;
                    break;
                }
                retVal=ComputeSelectionFunction(&lowerSearchLimit,&upperSearchLimit,
&precision);
                done=TRUE;
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
                }
        }
return retVal;
}
```

```
/* savesfpd.c01 */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "global.h"

int SaveSelecFuncEstPrgData(void){
int i,j;
FILE *outputStreamPtr;
char ch;
char filename[128];

gotoxy(1,24);
textcolor(YELLOW);
cprintf("Save as: ");
clreol();
textcolor(LIGHTGRAY);
scanf("%128s",filename);
strcat(filename,".sfd");
if((outputStreamPtr=fopen(filename,"w"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't creat output file! Press any key to continue");
        textcolor(LIGHTGRAY);
        getch();
        return 1;
}
fprintf(outputStreamPtr,"\n%s\n",projectTitleEst);
fprintf(outputStreamPtr,"%d\n",sizeClassNumSelecFuncEst);
for(i=1;i<=sizeClassNumSelecFuncEst;i++)
                fprintf(outputStreamPtr,"%10.0lf %6.2f %6.2f\n",ptrScreenSizeEst[i],
                ptrFeedEst[i],ptrDischargeMeasuredEst[i]);
if(normBreakFuncEst){
        fprintf(outputStreamPtr,"yes\n");
        for(i=1;i<=sizeClassNumSelecFuncEst;i++)
        fprintf(outputStreamPtr,"%6.4f\n",ptrBreakFuncEst[i][1]);
}
else{
        fprintf(outputStreamPtr,"no\n");
        for(i=1;i<=sizeClassNumSelecFuncEst;i++)
                for(j=1;j<=sizeClassNumSelecFuncEst;j++){
                        if(i==j) fprintf(outputStreamPtr,"0.0000 \n");
                        else if(i>j)  fprintf(outputStreamPtr,"%6.4f ",
                        ptrBreakFuncEst[i][j]);
                }
}
fprintf(outputStreamPtr,"%3.2f %3.2f %3.2f %3.2f %3.2f\n",*ptrTauPFSelecFuncEst,
                *ptrTauSPMSelecFuncEst,*ptrTauLPMSelecFuncEst,
                *ptrRefMillFeedRateSFEst,*ptrCurMillFeedRateSFEst);
if(fclose(outputStreamPtr)!=0){
        gotoxy(1,24);
```

```
        cprintf("Error in closing output file! Press any key to continue");
        getch();
        return 1;
}
return 0;
}
```

```
/* sfestrun.c01 */
#include <conio.h>
#include <dos.h>
#include "global.h"

void PrintText(int x,int y,char* text,int forgcolor);
int DoSearch(double* const ptrMinPossibleSf,double* const ptrMaxPossibleSf,
                                    double* const ptrPrecision);
int WrtToFileResultEst(void);
int WrtToPrnResultEst(void);

int ComputeSelectionFunction(double *ptrLowerSchLim,double *ptrUpperSchLim,
                                                                    double *ptrPrecision){
bool done=FALSE;
char prompt[]="Print <P> Save result <S> Export <E> Back <B> Quit <Q> ";
int i,key,retVal;

DoSearch(ptrLowerSchLim,ptrUpperSchLim,ptrPrecision);
PrintText(1,24,prompt,YELLOW);
clreol();
while(!done){
        key=getch();
        switch(key){
                case 'S':
                case 's':
                WrtToFileResultEst();
                PrintText(1,24,prompt,YELLOW);
                clreol();
                break;

                case 'E':
                case 'e':
                sizeClassNumSimGrCir=sizeClassNumSelecFuncEst;
                sizeClassNumBallSzOpt=sizeClassNumSelecFuncEst;
                for(i=1;i<=sizeClassNumSelecFuncEst;i++){
                        screenSize[i]=ptrScreenSizeEst[i];
                        ptrScreenSizeBallSzOpt[i-1]=ptrScreenSizeEst[i];
                }
                for(i=1;i<=sizeClassNumSelecFuncEst;i++){
                        selectionFunction[i]=ptrSelectionFunctionEst[i];
                        /* ptrSelecFuncBallSzOpt starts with 0 index */
                        ptrSelecFuncBallSzOpt[i-1]=ptrSelectionFunctionEst[i];
                }
                PrintText(1,24,"Selection function values were exported...",YELLOW);
                clreol();
                delay(700);
                PrintText(1,24,prompt,YELLOW);
                clreol();
                break;

                case 'P':
```

```
                case 'p':
                WrtToPrnResultEst();
                PrintText(1,24,"The output was sent to the printer...",YELLOW);
                clreol();
                delay(700);
                PrintText(1,24,prompt,YELLOW);
                clreol();
                gotoxy(64,24);
                break;

                case 'B':
                case 'b':
                retVal=1;
                done=TRUE;
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
        }
}
return retVal;
}
```

```
/* dosearch.c01 */
#include <conio.h>
#include <math.h>
#include <dos.h>
#include <float.h>
#include "global.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

void PrintText(int x,int y,char* text,int forgcolor);
int TrySf(int start,int end);

int DoSearch(double* ptrMinPossibleSf,double* ptrMaxPossibleSf,
                                    double* ptrPrecision){
int i,j,k,start;
int curline;
struct date today;
bool done=FALSE;
bool useQuestionMark=FALSE;
int curSizeClass=1;
double currentSrchLowLim;
double currentSrchUpLim;
char *msgSmallUpSchLim =
"Upper search limit is low! Press any key to continue ...";
char *msgLargeLowSchLim =
"Lower search limit is high! Press any key to continue ...";

clrscr();
gotoxy(17,3);
textbackground(LIGHTRED);
textcolor(WHITE);
cprintf("                                        ");
gotoxy(17,4);
cprintf("     Selection Function Estimation Data      ");
gotoxy(17,5);
cprintf("                                        ");
cprintf("\r\n");
textbackground(BLACK);
textcolor(LIGHTGRAY);
cprintf(projectTitleEst);
gotoxy(1,7);
getdate((struct date*) &today);
cprintf("Date: %d/%d/%d",today.da_mon,today.da_day,today.da_year);
gotoxy(1,8);
cprintf("Tau Plug Flow = %.2f  Tau Small PM = %.2f Tau Large PM = %.2f",
                *ptrTauPFSelecFuncEst,*ptrTauSPMSelecFuncEst,*ptrTauLPMSelecFuncEst);
```

```
gotoxy(1,9);
cprintf("Reference feed flow rate = %5.1f t/h \r\n",*ptrRefMillFeedRateSFEst);
cprintf("Current feed flow rate = %5.1f t/h \r\n\r\n",*ptrCurMillFeedRateSFEst);
cprintf("Breakage Function Matrix\r\n");
if(sizeClassNumSelecFuncEst<=10)
        for(i=1;i<=sizeClassNumSelecFuncEst;i++)
                for(j=1;j<=sizeClassNumSelecFuncEst;j++){
                        if(i==j) cprintf("0.0000\r\n");
                        else if(i>j) cprintf("%6.4f ",ptrBreakFuncEst[i][j]);
                }
else{
        for(i=1;i<=10;i++)
                for(j=1;j<=10;j++){
                        if(i==j) cprintf("0.0000\r\n");
                        else if(i>j) cprintf("%6.4f ",ptrBreakFuncEst[i][j]);
                }
        gotoxy(1,24);
        textcolor(YELLOW);
        cprintf("Press any key to continue ...");
        clreol();
        textcolor(LIGHTGRAY);
        getch();
        clrscr();
        gotoxy(1,1);
        for(i=11;i<=sizeClassNumSelecFuncEst;i++)
                for(j=11;j<=sizeClassNumSelecFuncEst;j++){
                        if(i==j) cprintf("0.00\r\n");
                        else if(i>j) cprintf("%6.4f ",ptrBreakFuncEst[i][j]);
                }
}
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
gotoxy(17,1);
textbackground(LIGHTRED);
textcolor(WHITE);
cprintf("                                        ");
gotoxy(17,2);
cprintf("  Selection Function Estimation Data (Con'd)  ");
gotoxy(17,3);
cprintf("                                        ");
gotoxy(1,4);
textbackground(BLACK);
textcolor(LIGHTGRAY);
cprintf("\
                                                                              \r\n\
```

```
| CLASS | SIEVE SIZE | FEED | MEAS. PROD. | \r\n\
├──────┼──────────┼──────┼──────────┤\r\n");
if(sizeClassNumSelecFuncEst<=16){
for(i=1;i<=sizeClassNumSelecFuncEst;i++)
cprintf(" | %5d | %10.0lf | %6.2f | %10.2f | \r\n",
i,ptrScreenSizeEst[i],ptrFeedEst[i],ptrDischargeMeasuredEst[i]);
cprintf("\
└──────┴──────────┴──────┴──────────┘");
}
else{
for(i=1;i<=16;i++)
cprintf(" | %5d | %10.0lf | %6.2f | %10.2f | \r\n",
i,ptrScreenSizeEst[i],ptrFeedEst[i],ptrDischargeMeasuredEst[i]);
cprintf("\
└──────┴──────────┴──────┴──────────┘");
PrintText(1,24,"Press any key to continue ...",YELLOW);
clreol();
getch();
clrscr();
gotoxy(17,1);
textbackground(LIGHTRED);
textcolor(WHITE);
cprintf("                                    ");
gotoxy(17,2);
cprintf(" Selection Function Estimation Data (Con'd) ");
gotoxy(17,3);
cprintf("                                    ");
gotoxy(1,4);
textbackground(BLACK);
textcolor(LIGHTGRAY);
cprintf("\
┌──────┬──────────┬──────┬──────────┐\r\n\
| CLASS | SIEVE SIZE | FEED | MEAS. PROD. | \r\n\
├──────┼──────────┼──────┼──────────┤\r\n");
for(i=17;i<=sizeClassNumSelecFuncEst;i++)
cprintf(" | %5d | %10.0lf | %6.2f | %10.2f | \r\n",
i,ptrScreenSizeEst[i],ptrFeedEst[i],ptrDischargeMeasuredEst[i]);
cprintf("\
└──────┴──────────┴──────┴──────────┘");
}
PrintText(1,24,"Press any key to begin the interval-by-interval search ...",YELLOW);
clreol();
getch();
clrscr();
gotoxy(17,11);
```

```
textbackground(LIGHTRED);
textcolor(WHITE);
cprintf("                                        ");
gotoxy(17,12);
cprintf("    Selection Function Estimation Search    ");
gotoxy(17,13);
cprintf("                                        ");
textbackground(BLACK);
PrintText(1,24,"Press any key to continue ...",YELLOW);
clreol();
getch();
clrscr();
textcolor(LIGHTGRAY);
cprintf("\
┌──────┬──────┬──────┬──────┬──────┬──────┐\r\n\
| CLASS | SIEVE SIZE | FEED | MEAS. PROD. | CALC. PROD. | SELEC. FUNC.
| \r\n\
├──────┼──────┼──────┼──────┼──────┼──────┤\r\n");
/* outer while */
while(!done){
        if(curSizeClass > sizeClassNumSelecFuncEst){
                done = TRUE;
                continue;
        }
        start = curSizeClass;
        /* hunting the selection function in an appropriate search interval
        check for the appropriate upper search limit at the begining of search
        calculating product with the max. possible selec. func. (upper limit) */
        ptrSelectionFunctionEst[curSizeClass] = *ptrMaxPossibleSf;
        TrySf(start,curSizeClass+1);
        if(GT(ptrDischargeCalcEst[curSizeClass]-ptrDischargeMeasuredEst[curSizeClass]
                        ,*ptrPrecision)){
                PrintText(1,24,msgSmallUpSchLim,LIGHTRED);
                clreol();
                getch();
                return 1;
        }
        /* check for the appropriate lower search limit at the begining of search */
        if(EQ(*ptrMinPossibleSf,0)) *ptrMinPossibleSf = EPS;
                ptrSelectionFunctionEst[curSizeClass] = *ptrMinPossibleSf;
                TrySf(start,curSizeClass+1);
        /* if with sf set to lower limit, we cannot generate required mass */
                if(LT(ptrDischargeCalcEst[curSizeClass]-
                                ptrDischargeMeasuredEst[curSizeClass],-(*ptrPrecision))){
        /* if we are not at the epsilon limit yet, decrease lower limit */
                if(GT(*ptrMinPossibleSf,EPS)){
                        PrintText(1,24,msgLargeLowSchLim,LIGHTRED);
                        clreol();
                        getch();
```

```
                    return 1;
                }
        /* if we are at the epsilon limit, set SF=epsilon, go to next size class */
                else if(EQ(*ptrMinPossibleSf,EPS)){
                                        if(!useQuestionMark)
                                        cprintf(
                                        " | %5d  | %10.0lf  | %6.2f  | %10.2f  | %10.2f
| %15.6f x  | \r\n",
                                        curSizeClass,ptrScreenSizeEst[curSizeClass],
                                        ptrFeedEst[curSizeClass],
                                        ptrDischargeMeasuredEst[curSizeClass],
                                        ptrDischargeCalcEst[curSizeClass],
                                        ptrSelectionFunctionEst[curSizeClass]);
                                        else
                                        cprintf(
                                        " | %5d  | %10.0lf  | %6.2f  | %10.2f  | %10.2f
| %15.6f ?  | \r\n",
                                        curSizeClass,ptrScreenSizeEst[curSizeClass],
                                        ptrFeedEst[curSizeClass],
                                        ptrDischargeMeasuredEst[curSizeClass],
                                        ptrDischargeCalcEst[curSizeClass],
                                        ptrSelectionFunctionEst[curSizeClass]);
                                        curline=wherey();
                                        if(!useQuestionMark)
                                        PrintText(1,24,
                                        "Unreliable estimate by setting SF to epsilon (symbol x).
Press any key ...",LIGHTRED);
                                        else
                                        PrintText(1,24,
                                        "Unreliable estimate by setting SF to epsilon (symbol ?).
Press any key ...",LIGHTRED);
                                        clreol();
                                        getch();
                                        gotoxy(1,curline);
                                        useQuestionMark=TRUE;
                                        curSizeClass++;
                                        continue;
                                }
                }
                /* we have two boundries and sf is somewhere between the two boundries */
                /* Now, search by bisection, after hunt */
                currentSrchLowLim=*ptrMinPossibleSf;
                currentSrchUpLim=*ptrMaxPossibleSf;
                ptrSelectionFunctionEst[curSizeClass]=

(currentSrchLowLim+currentSrchUpLim)/2.0;
                /* inner while */
                while(TRUE){
                        TrySf(start,curSizeClass+1);
                        if(LT(currentSrchUpLim-currentSrchLowLim,EPS)){
                                if(!useQuestionMark)
```

```
                                        cprintf(
                                        " | %5d  | %10.0lf  | %6.2f  | %10.2f  | %10.2f
| %15.6f✓ | \r\n",
                                        curSizeClass,ptrScreenSizeEst[curSizeClass],
                                        ptrFeedEst[curSizeClass],
                                        ptrDischargeMeasuredEst[curSizeClass],
                                        ptrDischargeCalcEst[curSizeClass],
                                        ptrSelectionFunctionEst[curSizeClass]);
                                else
                                        cprintf(
                                        " | %5d  | %10.0lf  | %6.2f  | %10.2f  | %10.2f
| %15.6f ? | \r\n",
                                        curSizeClass,ptrScreenSizeEst[curSizeClass],
                                        ptrFeedEst[curSizeClass],
                                        ptrDischargeMeasuredEst[curSizeClass],
                                        ptrDischargeCalcEst[curSizeClass],
                                        ptrSelectionFunctionEst[curSizeClass]);
                                        break;
                        }

if(LT(ptrDischargeCalcEst[curSizeClass],ptrDischargeMeasuredEst[curSizeClass]))

currentSrchUpLim=(currentSrchLowLim+currentSrchUpLim)/2;
                        else currentSrchLowLim=(currentSrchLowLim+currentSrchUpLim)/2;

ptrSelectionFunctionEst[curSizeClass]=(currentSrchLowLim+currentSrchUpLim)/2;
                }
                curSizeClass++;
        }
cprintf("\
└──────────┴──────────────────────────┴──────────┴──────────┴──────────┴──────────
──────┴──────────────────────────┴──────────────────────────────────────────┘ ");
return 0;
}
```

```
/* trysf.c01 */
#include <conio.h>
#include <math.h>
#include <assert.h>
#include <float.h>
#include "global.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

int TrySf(int const start,int const end){
int i,j,k;
void PrintText(int x,int y,char* text,int forgcolor);
double tauPF,tauSPM,tauLPM;
double mrt;

/* calculate RTD parameters corresponding to the current feed rate */
assert(*ptrCurMillFeedRateSFEst >= EPS);
mrt = *ptrTauPFSelecFuncEst+2**ptrTauSPMSelecFuncEst+*ptrTauLPMSelecFuncEst;
assert (mrt >= EPS);
tauPF=(*ptrTauPFSelecFuncEst/mrt)*(*ptrRefMillFeedRateSFEst/(*ptrCurMillFeedRateSFEst));
tauSPM=(*ptrTauSPMSelecFuncEst/mrt)*(*ptrRefMillFeedRateSFEst/(*ptrCurMillFeedRateSFEst));
tauLPM=(*ptrTauLPMSelecFuncEst/mrt)*(*ptrRefMillFeedRateSFEst/(*ptrCurMillFeedRateSFEst));
/*      calculation of the "ptrDiag" matrix */
assert(ptrSelectionFunctionEst[start] >= EPS);
for(i=start;i<end;i++){
        double WPF=exp(-ptrSelectionFunctionEst[i]*tauPF);
        double WS=1+ptrSelectionFunctionEst[i]*tauSPM;
        double WL=1+ptrSelectionFunctionEst[i]*tauLPM;
        assert(WL!=0);
        assert(WS!=0);
        ptrDiag[i]=WPF/(WL*pow(WS,2));
}
/*  calculation of the ptrT matrix */
for(i=start;i<end;i++)
        for(j=1;j<end;j++){
                if(i<j) ptrT[i][j]=0.0;
                else    if(i==j)
                                        ptrT[i][j]=ptrSelectionFunctionEst[j];
                                else{
                                        double SUM=0.0;
                                        for(k=1;k<i;k++){
                                        /* SUM should be always positive */
                                                SUM=SUM+ptrBreakFuncEst[i][k]*
ptrSelectionFunctionEst[k]*ptrT[k][j];
                                        }
```

```
                                        if(ptrSelectionFunctionEst[i]==ptrSelectionFunctionEst[j])
ptrSelectionFunctionEst[i]=ptrSelectionFunctionEst[i]*0.999999;
                                        ptrT[i][j]=SUM/(ptrSelectionFunctionEst[i]-
                                ptrSelectionFunctionEst[j]);
                                }
                        }
                        /*      calculation of the ptrTinv matrix */
                        for(i=start;i<end;i++)
                                for(j=1;j<end;j++){
                                        if(i<j) ptrTinv[i][j]=0.0;
                                        if(i==j)
                                                ptrTinv[i][j]=1/ptrSelectionFunctionEst[j];
                                        if(i>j){
                                                double SUM1=0.0;
                                                for(k=1;k<i;k++)
SUM1=SUM1+ptrT[i][k]*ptrTinv[k][j];
                                                        ptrTinv[i][j]=-SUM1/ptrSelectionFunctionEst[i];
                                        }
                                }
                        /*      calculation of the ptrT * ptrDiag */
                        for(i=start;i<end;i++)
                                for(j=1;j<end;j++) ptrTdiag[i][j]=ptrT[i][j]*ptrDiag[j];
                        /*      calculation of the mill transformation matrix */
                        for(i=start;i<end;i++)
                                for(j=1;j<end;j++){
                                        double SUM2=0.0;
                                        for(k=1;k<end;k++) SUM2=SUM2+ptrTdiag[i][k]*ptrTinv[k][j];
                                        ptrTrans[i][j]=SUM2;
                                }
                        /*      calculation of the ball mill discharge */
                        for(i=start;i<end;i++){
                                double SUM3=0.0;
                                for(j=1;j<end;j++) SUM3=SUM3+ptrTrans[i][j]*ptrFeedEst[j];
                                assert(SUM3>=0);
                                ptrDischargeCalcEst[i]=SUM3;
                        }
return 0;
}
```

```
/* wrtfsf.c01 */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include "global.h"

int WrtToFileResultEst(void){
short i,j;
FILE *OutputStream;
char ch;
char *filename;
struct date today;

gotoxy(1,24);
clreol();
textcolor(YELLOW);
cprintf("Save as: ");
textcolor(LIGHTGRAY);
scanf("%128s",filename);
if((OutputStream=fopen(filename,"w"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't creat output file! Press any key to continue");
        textcolor(LIGHTGRAY);
        getch();
        return 1;
}
fprintf(OutputStream,"\n\
                ****************************************\n\
                Selection Function Estimation Results\n\
                ****************************************\n\n");
fprintf(OutputStream,projectTitleEst);
fprintf(OutputStream,"\n");
getdate((struct date*) &today);
fprintf(OutputStream,"Date: %d/%d/%d\n",today.da_mon,today.da_day,today.da_year);
fprintf(OutputStream,"Tau Plug Flow = %5.2f  Tau Small PM = %5.2f Tau Large PM =
%5.2f\n\n",
                *ptrTauPFSelecFuncEst,*ptrTauSPMSelecFuncEst,*ptrTauLPMSelecFuncEst);
fprintf(OutputStream,"Reference feed flow rate = %5.1f t/h \n",*ptrRefMillFeedRateSFEst);
fprintf(OutputStream,"Current feed flow rate = %5.1f t/h \n\n",*ptrCurMillFeedRateSFEst);
fprintf(OutputStream,"Breakage Function Matrix \n");
for(i=1;i<=sizeClassNumSelecFuncEst;i++)
        for(j=1;j<=sizeClassNumSelecFuncEst;j++){
                if(i==j)
                        fprintf(OutputStream,"0.00 \n");
                else if(i>j)
                        fprintf(OutputStream,"%.2f  ",ptrBreakFuncEst[i][j]);
        }
fprintf(OutputStream,"\n\
```

```
┌──────┬──────────────┬──────────┬──────────────┬──────────────┬──────────────┐\n\
| CLASS | SIEVE SIZE | FEED | MEAS. PROD. | CALC. PROD. | SELEC. FUNC.
| \n\
├──────┼──────────────┼──────────┼──────────────┼──────────────┼──────────────┤\n");
for(i=1;i<=sizeClassNumSelecFuncEst;i++)
        fprintf(OutputStream," | %5d | %10.0f | %6.2f | %10.2f | %10.2f | %10.4f
 | \n",i,
ptrScreenSizeEst[i],ptrFeedEst[i],ptrDischargeMeasuredEst[i],
ptrDischargeCalcEst[i],ptrSelectionFunctionEst[i]);
fprintf(OutputStream,
"\
 └──────┴──────────────┴──────────┴──────────────┴──────────────┴──────────────┘ ");
/* close the file */
if(fclose(OutputStream)!=0){
        gotoxy(1,24);
        cprintf("Error in closing output file! Press any key to continue");
        getch();
        return 1;
}
return 0;
}
```

```
/* wrtprbsf.c01 */
#include <stdio.h>
#include <dos.h>
#include "global.h"

int WrtToPrnResultEst(void){
short i,j;
struct date today;
fprintf(stdprn,"\n\n\r\n"
"                ****************************************\r\n"
"                Selection Function Estimation Results\r\n"
"                ****************************************\r\n\n");
fprintf(stdprn,projectTitleEst);
fprintf(stdprn,"\r\n");
getdate((struct date*) &today);
fprintf(stdprn,"Date: %d/%d/%d\r\n",today.da_mon,today.da_day,today.da_year);
fprintf(stdprn,"Tau Plug Flow = %.2f  Tau Small PM = %.2f Tau Large PM = %.2f\r\n\r\n",
            *ptrTauPFSelecFuncEst,*ptrTauSPMSelecFuncEst,*ptrTauLPMSelecFuncEst);
fprintf(stdprn,"Reference feed flow rate = %5.1f t/h \r\n",*ptrRefMillFeedRateSFEst);
fprintf(stdprn,"Current feed flow rate = %5.1f t/h \r\n\r\n",*ptrCurMillFeedRateSFEst);
fprintf(stdprn,"Breakage Function Matrix \r\n");
for(i=1;i<=sizeClassNumSelecFuncEst;i++)
        for(j=1;j<=sizeClassNumSelecFuncEst;j++){
                if(i==j)        fprintf(stdprn,"0.00  \r\n");
                else    if(i>j)
                        fprintf(stdprn,"%.2f ",ptrBreakFuncEst[i][j]);
        }
fprintf(stdprn,"\r\n\
-------------------------------------------------------------------\r\n\
| CLASS | SIEVE SIZE |  FEED  | MEAS. PROD. | CALC. PROD. | SELEC. FUNC. |\r\n\
|-------|------------|--------|-------------|-------------|--------------|\r\n");
for(i=1;i<=sizeClassNumSelecFuncEst;i++)
        fprintf(stdprn,"|%5d  |%10.0f  |%6.2f  |%10.2f  |%10.2f  |%10.4f  |\r\n",i,
        ptrScreenSizeEst[i],ptrFeedEst[i],ptrDischargeMeasuredEst[i],
        ptrDischargeCalcEst[i],ptrSelectionFunctionEst[i]);
fprintf(stdprn,
"\
|-------|------------|--------|-------------|-------------|--------------|");
fprintf(stdprn,"\f");
return 0;
}
```

```
/* simgrcir.c01 */
int GetSimCircuitDataPg1(void);
int GetSimCircuitDataPg2(void);
int GetSimCircuitDataPg3(void);
int GetSimCircuitDataPg4(void);
int GetSimCircuitDataPg5(void);
int SimulateGrindingRun(void);

int SimulateGrindingCircuit(){
int next=1;

for(;;){
        switch(next){
                case 1:
                next=GetSimCircuitDataPg1();
                break;

                case 2:
                next=GetSimCircuitDataPg2();
                break;

                case 3:
                next=GetSimCircuitDataPg3();
                break;

                case 4:
                next=GetSimCircuitDataPg4();
                break;

                case 5:
                next=GetSimCircuitDataPg5();
                break;

                case 6:
                next=SimulateGrindingRun();
                break;

                case 'Q':
                case 'q':
                break;
         }
         if(next==-1000) break;
}
return 0;
}
```

```
/* ret_grsd.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <dir.h>
#include "global.h"

void PrintText(int x,int y,char* text,int forgcolor);
int HandleEndOfFile(void);
char *GetStringAt(int x,int y,int maxlen);
int GetFileNames(char *ptrExt);

int OpenGrindingSimulationFile(void){
FILE    *streamPtr;
char dataFileName[128];
char *newPath;
int i,j;
int status=0;
int key='$';
char string[15];
char *ptrString;
char *FormatErrPrompt=
"Data file format is incorrect! Press any key to continue ...";

gotoxy(1,24);
textcolor(YELLOW);
cprintf(
"Please enter file name?  ");
gotoxy(42,24);
textcolor(LIGHTGRAY);
cprintf(                          "(type > to change current directory)");
clreol();
textcolor(YELLOW);
ptrString=GetStringAt(25,24,15);
if(ptrString[0]=='>'){
        gotoxy(1,24);
        textcolor(YELLOW);
        cprintf("Please enter new directory:  ");
        clreol();
        textcolor(LIGHTGRAY);
        newPath=GetStringAt(29,24,54);
        if(chdir(newPath)!=0){
                gotoxy(1,24);
                textcolor(LIGHTRED);
                if(strlen(newPath)<=30)
                cprintf("Cannot change to [%s].  Press any key to continue ...",newPath);
                else
                cprintf("Cannot change to new directory.  Press any key to continue ...");
                clreol();
```

```
                getch();
                return 2;
        }
        else{
                fflush(stdin);
                GetFileNames("gsd");
                gotoxy(1,24);
                textcolor(YELLOW);
                cprintf(
                "Please enter file name?  ");
                textcolor(YELLOW);
                ptrString=GetStringAt(25,24,15);
        }
}
strcpy(dataFileName,ptrString);
if(strlen(dataFileName)<=8) strcat(dataFileName,".gsd");
if((streamPtr=fopen((const char*) dataFileName,"r"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't open file! press any key to continue ...");
        clreol();
        getch();
        return 2;
}
else{
/* reading one string for project title */
        while((key=getc(streamPtr))!=EOF){
                if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                else{
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        fgets(projectTitle,80,streamPtr);
                        projectTitle[strlen(projectTitle)-1]='\0';
                        break;
                }
        }
        if(key==EOF){
                HandleEndOfFile();
                return 2;
        }
/* reading sizeClassNumbers */
status=0;
        while((key=getc(streamPtr))!=EOF&&status<1) {
                if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                else{
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        status=fscanf(streamPtr,"%d",&sizeClassNumSimGrCir);
                        if(status<1){
                                PrintText(1,24,FormatErrPrompt,LIGHTRED);
                                clreol();
                                getch();
                                return 2;
```

```
                    }
                }
        }
        if(key==EOF){
                HandleEndOfFile();
                return 2;
        }
/* reading the rest of file i.e. sieveSeries, feed size, selection function */
        for(i=1;i<=sizeClassNumSimGrCir;i++){
                status=0;
                key= '$';
                while((key=getc(streamPtr))!=EOF&&status<3){
                        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                                fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                                status=fscanf(streamPtr,"%lf %lf %lf",screenSize+i,
                                                        feedSizeDistribution+i,
                                                        selectionFunction+i);
                                if(status<3){
                                        PrintText(1,24,FormatErrPrompt,LIGHTRED);
                                        clreol();
                                        getch();
                                        return 2;
                                }
                        }
                        if(key==EOF){
                                HandleEndOfFile();
                                return 2;
                        }
                }
/* reading the rest of data file i.e. breakage function values
        and test for normalisability */
        while((key=getc(streamPtr))!=EOF){
                if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                fscanf(streamPtr,"%s",string);
                if(string[0]!='Y'&&string[0]!='y'&&string[0]!='N'&&string[0]!='n'){
                        PrintText(1,24,FormatErrPrompt,LIGHTRED);
                        clreol();
                        getch();
                        return 0;
                }
                else break;
        }
        if(key==EOF){
                HandleEndOfFile();
                return 2;
        }
        if(string[0]=='Y'||string[0]=='y'){
                normalisableBreakageFunction=TRUE;
                for(i=1;i<=sizeClassNumSimGrCir;i++){
                        status=0;
```

```
                    key='$';
                    while((key=getc(streamPtr))!=EOF&&status<1){
                        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        status=fscanf(streamPtr,"%lf",&breakageFunction[i][1]);
                    if(status<1){
                        PrintText(1,24,FormatErrPrompt,LIGHTRED);
                        clreol();
                        getch();
                        return 2;
                    }
                }
                if(key==EOF){
                    HandleEndOfFile();
                    return 2;
                }
            }
            for(i=1;i<=sizeClassNumSimGrCir;i++)
                for(j=1;j<=sizeClassNumSimGrCir;j++)
                if((i<j)||(i==j))
                    breakageFunction[i][j]=0;
                else if(j>1)
                breakageFunction[i][j]=breakageFunction[i-1][j-1];
}
else if(string[0]=='N'||string[0]=='n'){
        normalisableBreakageFunction=FALSE;
        for(i=1;i<=sizeClassNumSimGrCir;i++)
                for(j=1;j<=sizeClassNumSimGrCir;j++){
                    if(i<j){
                        breakageFunction[i][j]=0;
                        continue;
                    }
                status=0;
                while((key=getc(streamPtr))!=EOF&&status<1){
                    if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        status=fscanf(streamPtr,"%lf",&breakageFunction[i][j]);
                            if(status<1){
                                clrscr();
                                PrintText(1,24,FormatErrPrompt,LIGHTRED);
                                clreol();
                                getch();
                                return 2;
                            }
                }
                if(key==EOF){
                    HandleEndOfFile();
                    return 2;
                    }
                }
}
```

```
/* reading mean retention times... */
                status=0;
                key='$';
                while((key=getc(streamPtr))!=EOF&&status<3){
                        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                        status=fscanf(streamPtr,"%lf %lf %lf %lf %lf",ptrTauPFSimGrCir,

ptrTauSPMSimGrCir,ptrTauLPMSimGrCir,

ptrRefMillFeedRateSimGrCir,ptrCurMillFeedRateSimGrCir);
                        if(status<3){
                                PrintText(1,24,FormatErrPrompt,LIGHTRED);
                                clreol();
                                getch();
                                return 2;
                        }
        }
}
if(fclose(streamPtr)!=0){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't close file! Press any key to continue ...\n\r");
        textcolor(LIGHTGRAY);
        getch();
        return 2;
}
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Data transfered successfully, press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
return 0;
}
```

```
/* simgrdp1.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include "global.h"
#include "gets.h"

#define promptEndPos  42

extern int savescr[];

int GetFileNames(char *ptrExt);
void PrintText(int x,int y,char* text,int forgcolor);
void PrintErrMsg(void);
int OpenGrindingSimulationFile(void);

int GetSimCircuitDataPg1(void){
int key,status,retVal;
int i,j;
bool done=FALSE;
int *address;
char *string;
char *prompt="Open <O> Change <C> Next<N> Quit <Q> ==>";
char isBreakageNormalisable[6];

if(normalisableBreakageFunction)
        strcpy((char*)isBreakageNormalisable,"yes");
else
        strcpy((char*)isBreakageNormalisable,"no");
clrscr();
PrintText(10,4,"General Information for Grinding Circuit Simulation",WHITE);
gotoxy(3,6);
cprintf("Project title              %s",projectTitle);
gotoxy(3,7);
cprintf("Size class numbers          %d",sizeClassNumSimGrCir);
gotoxy(3,8);
cprintf("Normalisable breakage       %s",isBreakageNormalisable);
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
clreol();
textcolor(LIGHTGRAY);
while(!done){
        key=getch();
        switch(key){
                case 'O':
                case 'o':
                gettext(1,1,80,25,savescr);
                GetFileNames("gsd");
```

```
OpenGrindingSimulationFile();
puttext(1,1,80,25,savescr);
textcolor(LIGHTGRAY);
gotoxy(35,6);
cprintf("%s",projectTitle);
clreol();
gotoxy(35,7);
cprintf("%d",sizeClassNumSimGrCir);
gotoxy(35,8);
cprintf("%s",isBreakageNormalisable);
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
clreol();
textcolor(LIGHTGRAY);
break;

case 'C':
case 'c':
gotoxy(35,6);
/* cursor at 35,6 */
fflush(stdin);
if((key=getch())!=ESC){
        ungetch(key);
        if(key!='\r') clreol();
        string=GetStringAt(35,6,45);
        if(string[0]=='\0'){
                cprintf("%s",projectTitle);
                clreol();
                gotoxy(35,7);
        }
        else{
                strcpy(projectTitle,string);
                gotoxy(35,6);
                cprintf("%s",projectTitle);
                clreol();
                gotoxy(35,7);
        }
}
else{
        gotoxy(promptEndPos,24);
        break;
}
/* cursor at 35,7 */
fflush(stdin);
if((key=getch())!=ESC){
        ungetch(key);
        address=&sizeClassNumSimGrCir;
        status=0;
        while(!done){
                string=GetStringAt(35,7,5);
```

```
                if(string[0]==\0'){
                        cprintf("%1d",*address);
                        clreol();
                        gotoxy(35,8);
                        break;
                }
                status=sscanf(string,"%d",address);
                if(status==1){
                        if(*address>MAXSIZECLASSNO||*address<1){
                                PrintErrMsg();
                                PrintText(1,24,prompt,YELLOW);
                                gotoxy(35,7);
                                clreol();
                                gotoxy(35,7);
                                done=FALSE;
                        }
                        else{
                                gotoxy(35,7);
                                cprintf("%1d",*address);
                                clreol();
                                gotoxy(35,8);
                                break;
                        }
                }
                else{
                        PrintErrMsg();
                        gotoxy(1,24);
                        textcolor(YELLOW);
                        cprintf(prompt);
                        clreol();
                        textcolor(LIGHTGRAY);
                        gotoxy(35,7);
                        clreol();
                        gotoxy(35,7);
                }
        }
}
else{
        gotoxy(promptEndPos,24);
        break;
}
/* cursor at 35,8 */
fflush(stdin);
if((key=getch())!=ESC){
        ungetch(key);
        if(key!='\r') clreol();
        string[0]='\0';
        while(string[0]!='Y'&&string[0]!='y'&&string[0]!='N'&&
                        string[0]!='n'){
                string=GetStringAt(35,8,6);
                if(string[0]==\0'){
```

```
                                        cprintf("%s",isBreakageNormalisable);
                                        clreol();
                                        gotoxy(promptEndPos,24);
                                        break;
                                }
                                else{
                                        if(string[0]=='Y'||string[0]=='y'||string[0]=='N'||
                                                string[0]=='n'){
                                                if(string[0]=='Y'||string[0]=='y')

strcpy((char*)isBreakageNormalisable,"yes");
                                                else

strcpy((char*)isBreakageNormalisable,"no");
                                                gotoxy(35,8);
                                                cprintf("%s",isBreakageNormalisable);
                                                clreol();
                                                if(string[0]=='Y'||string[0]=='y'){
                                                        normalisableBreakageFunction=TRUE;

for(i=1;i<=sizeClassNumSimGrCir;i++)

for(j=1;j<=sizeClassNumSimGrCir;j++)

breakageFunction[i+1][j+1]=breakageFunction[i][j];
                                                }
                                                else
                                                        normalisableBreakageFunction=FALSE;
                                                gotoxy(promptEndPos,24);
                                                break;
                                        }
                                        else{
                                                PrintText(1,24,"Please answer by yes or
no!",LIGHTRED);
                                                clreol();
                                                delay(700);
                                                PrintText(1,24,prompt,YELLOW);
                                        }
                                }
                        }
                }
                else{
                        gotoxy(promptEndPos,24);
                        break;
                }
                break;

                case 'N':
                case 'n':
                retVal=2;
                done=TRUE;
```

```
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
                }
        }
return retVal;
}
```

```
/* simgrdp2.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include "global.h"
#include "gets.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y))<=fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

extern char savescr[4096];
extern double *ptrTempDoubleArray;
extern enum plottype plotType;

int PlotScatterDig(double *xDataVec,double *yDataVec,int m,
                                                    char* title,char* xLab,char* yLab);
void PrintReminder(void);
void GetColOfData(int row,int column,int noSzClasses,int xOfColumn[],int maxlen,
                                                    char* formatStr,char* formatStrLeftJus,
                                                    double minAllowedRange,double
maxAllowedRange,
                                                    double *tempInputPtr);
void InitializeGraphics(void);
void TerminateGraphics(void);
void PrintText(int x,int y,char* text,int forgcolor);

enum plottype{SIZEDIST=1,SELECTIONFUNC=2,BREAKAGEFUNC=3};
operationmode mode;

int GetSimCircuitDataPg2(void){
char *formatStr;
char *formatStrLeftJus;
char *addressChar[20];
extern int GraphMode;
int i,noSzClasses,key,retVal;
bool done=FALSE;
double minAllowedRange,maxAllowedRange,sum;
int row=4;
int column;
int xOfColumn[3]={20,40,60};
int maxlen;
int* xOfColumnPtr;
double *tempInputPtr;
```

```
char *prompt=
"Change <C> Previous <P> Next<N> Graph<G> Quit<Q> ==>";
mode=SIMULATION;
xOfColumnPtr=&xOfColumn[0]-1;
clrscr();
PrintText(14,1,"Feed Size And Selection Function Data for Simulation",WHITE);
gotoxy(1,3);
cprintf("Size Class No.");
gotoxy(xOfColumnPtr[1],3);
cprintf("Screen Size (μm)");
gotoxy(xOfColumnPtr[2],3);
cprintf("Feed Size (%)");
gotoxy(xOfColumnPtr[3],3);
cprintf("Selection Function");
gotoxy(5,4);
noSzClasses=sizeClassNumSimGrCir;
for(i=1;i<=noSzClasses;i++){
        gotoxy(5,i+3);
        cprintf("%2d",i);
        gotoxy(xOfColumnPtr[1],i+3);
        cprintf("%10.0lf",screenSize[i]);
        gotoxy(xOfColumnPtr[2],i+3);
        cprintf("%10.2lf",feedSizeDistribution[i]);
        gotoxy(xOfColumnPtr[3],i+3);
        cprintf("%15.6lf",selectionFunction[i]);
}
PrintText(1,24,prompt,YELLOW);
clreol();
while(!done){
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                column=1;
                maxlen=15;
                formatStr="%10.0lf";
                formatStrLeftJus="%-10.0lf";
                minAllowedRange=0;
                maxAllowedRange=250000;
                tempInputPtr=screenSize;
                GetColOfData(row,column,noSzClasses,xOfColumnPtr,maxlen,formatStr,

formatStrLeftJus,minAllowedRange,maxAllowedRange,
                                                        tempInputPtr);
                column=2;
                maxlen=15;
                formatStr="%10.2lf";
                formatStrLeftJus="%-10.2lf";
                minAllowedRange=0;
                maxAllowedRange=100;
                tempInputPtr=feedSizeDistribution;
```

```
                GetColOfData(row,column,noSzClasses,xOfColumnPtr,maxlen,formatStr,
formatStrLeftJus,minAllowedRange,maxAllowedRange,
                                                tempInputPtr);

                column=3;
                maxlen=15;
                formatStr="%15.6lf";
                formatStrLeftJus="%-15.6lf";
                minAllowedRange=EPS;
                maxAllowedRange=99999999;
                tempInputPtr=selectionFunction;
                GetColOfData(row,column,noSzClasses,xOfColumnPtr,maxlen,formatStr,
formatStrLeftJus,minAllowedRange,maxAllowedRange,
                                                tempInputPtr);
                gotoxy(53,24);
                break;

                case 'G':
                case 'g':
                gettext(1,1,80,25,savescr);
                InitializeGraphics();
                plotType=SELECTIONFUNC;
                PlotScatterDig(screenSize,selectionFunction,noSzClasses,
                                                "Selection Function Data", "Particle Size,
X (μm)",
                                                "Selection Function");
                getch();
                plotType=SIZEDIST;
                sum=0;
                for(i=1;i<=sizeClassNumSimGrCir;i++){
                        sum=sum+feedSizeDistribution[i];
                        ptrTempDoubleArray[i]=100-sum;
                }
                PlotScatterDig(screenSize,ptrTempDoubleArray,noSzClasses,
                                                "Measured Feed Size Distribution",
"Particle Size, X (μm)",
                                                "Mass (%)");
                getch();
                TerminateGraphics();
                puttext(1,1,80,25,savescr);
                gotoxy(53,24);
                break;

                case 'P':
                case 'p':
                retVal=1;
                done=TRUE;
                break;
```

```
                case 'N':
                case 'n':
                sum=0;
                for(i=1;i<=noSzClasses;i++)
                        sum=sum+feedSizeDistribution[i];
                if(GT(sum,100)){
                        gettext(1,10,80,14,savescr);

PrintText(2,11,"┌──────────────────────────────────────────────────────────────────────────────┐
",WHITE);
                        PrintText(2,12,"|    The total mass in feed exceeds 100%. Press any key to
continue ...    | ",WHITE);

PrintText(2,13,"└──────────────────────────────────────────────────────────────────────────────┘
",WHITE);
                        _setcursortype(_NOCURSOR);
                        getch();
                        gotoxy(20,10);
                        puttext(1,10,80,14,savescr);
                        _setcursortype(_NORMALCURSOR);
                        gotoxy(53,24);
                        break;
                 }
                retVal=3;
                done=TRUE;
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
        }
}
return retVal;
}
```

```
/* simgrdp3.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "global.h"
#include "gets.h"

#define promptEndPos 45

void PrintText(int x,int y,char* text,int forgcolor);
void PrintErrMsg(void);
int CheckSumOfColumns(double **matrixPtr,bool bfNormFlag,
                                                    int jIterNum,int iIterNum,int endPos);

int GetSimCircuitDataPg3(void){
bool done=FALSE;
bool canUseArraysValues=TRUE;
int maxIt,retVal;
int row=4;
int column=0;
int xOfColumn;
int maxlen=12;
char *formatStr;
double addressFloat;
int status=0;
char *string;
char buffer[10];
int i,j,key;
char *prompt="Change <C> Previous <P> Next <N> Quit<Q> ==>";

clrscr();
PrintText(10,1,"Breakage Function Data for Grinding Circuit Simulation",WHITE);
if(normalisableBreakageFunction)
        PrintText(1,3,"Normalisable breakage function",GREEN);
else
        PrintText(1,3,"Non-normalisable breakage function",GREEN);
for(i=1;i<=sizeClassNumSimGrCir;i++)
        for(j=1;j<=10;j++){
                if(i>=j){
                        xOfColumn=1 +(j-1) * 8;
                        gotoxy(xOfColumn ,i+3);
                        cprintf("%6.4f",breakageFunction[i][j]);
                }
        }
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
clreol();
textcolor(LIGHTGRAY);
while(!done){
```

```
        key=getch();
        switch(key){
                case 'C':
                case 'c': gotoxy(1,7);
                if(sizeClassNumSimGrCir<=10) maxIt=sizeClassNumSimGrCir;
                else maxIt=10;
                for(column=1;column<= maxIt;column++){
                        maxlen=9;
                        formatStr="%6.4f";
                        xOfColumn=1 +(column-1) * 8;
                        gotoxy(xOfColumn ,(column-1)+5);
                        for(row=column+4 ;row < sizeClassNumSimGrCir+4 ;row++){
                                key=getch();
                                if(key==ESC){
                                        gotoxy(promptEndPos,24);
                                        break;
                                }
                                ungetch(key);
                                putch(key);
                                done=FALSE;
                                canUseArraysValues=TRUE;
                                status=0;
                                while(!done){
                                        string=GetStringAt(xOfColumn,row,maxlen);
                                        if(string[0]=='\0'&&canUseArraysValues){
                                                gotoxy(xOfColumn,row);

cprintf(formatStr,breakageFunction[row-3][column]);
                                                if(row<sizeClassNumSimGrCir+3)
gotoxy(xOfColumn,row+1);
                                                break;
                                        }

status=sscanf(string,"%lf",&(breakageFunction[row-3][column]));
                                        if(status==1&&breakageFunction[row-3][column]<=1&&
                                                breakageFunction[row-3][column]>=0){
                                                gotoxy(xOfColumn,row);

cprintf(formatStr,breakageFunction[row-3][column]);
                                                gotoxy(xOfColumn,row+1);
                                                if(normalisableBreakageFunction){

for(i=1;i<=sizeClassNumSimGrCir-row+3;i++){

breakageFunction[row-3+i][column+i]=breakageFunction[row-3][column];
                                                                if(i<=9){
                                                                        gotoxy(1+i*8,row+i);

cprintf(formatStr,breakageFunction[row-3][column]);
                                                                }
                                                        }
```

```
                                        gotoxy(xOfColumn,row+1);
                                    }
                                    break;
                                }
                                else{
                                    PrintErrMsg();
                                    gotoxy(1,24);
                                    textcolor(YELLOW);
                                    cprintf(prompt);
                                    textcolor(LIGHTGRAY);
                                    gotoxy(xOfColumn,row);
                                    cprintf("      ");
                                    gotoxy(xOfColumn,row);
                                    done=FALSE;
                                    canUseArraysValues=FALSE;
                                }
                            }
                        }
                        if(key==ESC)
                        break;
                        if(normalisableBreakageFunction)
                        break;
                    }
                    gotoxy(promptEndPos,24);
                    break;

                    case 'P':
                    case 'p':
                    retVal=2;
                    done=TRUE;
                    break;

                    case 'N':
                    case 'n':
                    if(sizeClassNumSimGrCir>10){
                        if(CheckSumOfColumns(breakageFunction,normalisableBreakageFunction,
10,sizeClassNumSimGrCir,
promptEndPos)==0){
                            retVal=4;
                            done=TRUE;
                            break;
                        }
                        else{
                            done=FALSE;
                            break;
                        }
                    }
                    else{
                        if(CheckSumOfColumns(breakageFunction,normalisableBreakageFunction,
```

```
sizeClassNumSimGrCir,

sizeClassNumSimGrCir,

promptEndPos)==0){
                                retVal=5;
                                done=TRUE;
                                break;
                        }
                        else{
                                done=FALSE;
                                break;
                        }
                }

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
                }
        }
return retVal;
}
```

```
/* simgrdp4.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include "global.h"
#include "gets.h"

#define promptEndPos   52
#define row14          14
#define row15          row14+1

void PrintText(int x,int y,char* text,int forgcolor);
void PrintErrMsg(void);
int  CheckSumRestColumns(double **matrixPtr,int jIterNum,
                                                    int iIterNum,int endPos);

int GetSimCircuitDataPg4(void){
bool done=FALSE;
bool canUseArraysValues=TRUE;
int retVal;
int row=row14;
int column=0;
int xOfColumn;
int maxlen=12;
char *formatStr;
double addressFloat;
int status=0;
char *string;
char prompt[81];
char buffer[10];
int i,j,key;

if(normalisableBreakageFunction) strcpy(prompt,
"Previous <P>   Next <N>   Quit <Q> ==>");
else strcpy(prompt,
"Change <C>   Previous <P>   Next <N>   Quit <Q> ==>");
clrscr();
PrintText(5,1,"Breakage Function Data for Grinding Circuit Simulation (continued)",WHITE);
if(normalisableBreakageFunction)
        PrintText(1,3,"Normalisable breakage function",GREEN);
else
        PrintText(1,3,"Non-normalisable breakage function",GREEN);
for(i=11;i<=sizeClassNumSimGrCir;i++)
        for(j=11;j<=sizeClassNumSimGrCir;j++)
                if(i>=j){
                        xOfColumn=1+(j-11)*8;
                        gotoxy(xOfColumn,(i-11)+row14);
                        cprintf("%6.4lf",breakageFunction[i][j]);
                }
```

```
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
textcolor(LIGHTGRAY);
while(!done){
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                if(normalisableBreakageFunction) break;
                for(column=11;column<=sizeClassNumSimGrCir;column++){
                        maxlen=9;
                        formatStr="%6.4f";
                        xOfColumn=1+(column-11)*8;
                        gotoxy(xOfColumn,row15+column-11);
                        for(row=row15+column-11;row<=sizeClassNumSimGrCir+3;row++){
                                key=getch();
                                if(key==ESC){
                                        if(normalisableBreakageFunction) gotoxy(39,24);
                                        else gotoxy(promptEndPos,24);
                                        break;
                                }
                                ungetch(key);
                                putch(key);
                                done=FALSE;
                                canUseArraysValues=TRUE;
                                status=0;
                                while(!done){
                                        string=GetStringAt(xOfColumn,row,maxlen);
                                        if(string[0]=='\0'&&canUseArraysValues){
                                                gotoxy(xOfColumn,row);

cprintf(formatStr,breakageFunction[row-3][column]);
                                                if(row<=sizeClassNumSimGrCir+3)
gotoxy(xOfColumn,row+1);
                                                break;
                                        }

status=sscanf(string,"%lf",&(breakageFunction[row-3][column]));
                                        if(status==1&&breakageFunction[row-3][column]<=1&&
                                                breakageFunction[row-3][column]>=0){
                                                gotoxy(xOfColumn,row);

cprintf(formatStr,breakageFunction[row-3][column]);
                                                gotoxy(xOfColumn,row+1);
                                                break;
                                        }
                                        else{
                                                PrintErrMsg();
                                                gotoxy(1,24);
                                                textcolor(YELLOW);
```

```
                                                cprintf(prompt);
                                                textcolor(LIGHTGRAY);
                                                gotoxy(xOfColumn,row);
                                                cprintf("      ");
                                                gotoxy(xOfColumn,row);
                                                done=FALSE;
                                                canUseArraysValues=FALSE;
                                        }
                                }
                        }
                        if(key==ESC) break;
                }
                if(normalisableBreakageFunction)
                        gotoxy(39,24);
                else gotoxy(promptEndPos,24);
                break;

                case 'P':
                case 'p':
                retVal=3;
                done=TRUE;
                break;

                case 'N':
                case 'n':
                if(!normalisableBreakageFunction){
                        if(CheckSumRestColumns(breakageFunction,sizeClassNumSimGrCir,
sizeClassNumSimGrCir,promptEndPos)==0){
                                retVal=5;
                                done=TRUE;
                                break;
                        }
                        else{
                                done=FALSE;
                                break;
                        }
                }
                else{
                        retVal=5;
                        done=TRUE;
                        break;
                }

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
                }
        }
```

```
return retVal;
}
```

```
/* simgrdp5.c01 */
#include <conio.h>
#include "global.h"

void PrintText(int x,int y,char* text,int forgcolor);
int GetRtdInfo(double *ptrTauPF,double *ptrTauSPM,double *ptrTauLPM,
                                        double *ptrRefMillFeedRate,double *ptrCurMillFeedRate);
int SaveGrCirSimData(void);

int GetSimCircuitDataPg5(void){
bool done=FALSE;
int key,retVal;
char scrTitle[]=
"Residence Time Distribution Data for Grinding Circuit Simulation";
char *prompt=
"Change <C> Previous <P> Save data <S> Run <R> Quit <Q> ==>";
clrscr();
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
clreol();
textcolor(LIGHTGRAY);
PrintText(5,4,scrTitle,WHITE);
gotoxy(3,6);
cprintf("Reference tau of plug flow:              %.3f",*ptrTauPFSimGrCir);
gotoxy(3,7);
cprintf("Reference tau of small perfect mixers:   %.3f",*ptrTauSPMSimGrCir);
gotoxy(3,8);
cprintf("Reference tau of large perfect mixer:    %.3f",*ptrTauLPMSimGrCir);
gotoxy(3,9);
cprintf("Reference feed rate (t/h):               %.3f",*ptrRefMillFeedRateSimGrCir);
gotoxy(3,10);
cprintf("Current feed rate (t/h):                 %.3f",*ptrCurMillFeedRateSimGrCir);
gotoxy(59,24);
while(!done){
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                GetRtdInfo(ptrTauPFSimGrCir,ptrTauSPMSimGrCir,
                                ptrTauLPMSimGrCir,ptrRefMillFeedRateSimGrCir,
                                ptrCurMillFeedRateSimGrCir);
                gotoxy(59,24);
                break;

                case 'P':
                case 'p':
                if(sizeClassNumSimGrCir>10) retVal=4;
                else retVal=3;
                done=TRUE;
                break;
```

```
                case 'S':
                case 's':
                SaveGrCirSimData();
                gotoxy(1,24);
                textcolor(YELLOW);
                cprintf(prompt);
                clreol();
                textcolor(LIGHTGRAY);
                break;

                case 'R':
                case 'r':
                retVal=6;
                done=TRUE;
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
        }
}
return retVal;
}
```

```
/* savegspd.c01 */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "global.h"

int SaveGrCirSimData(void){
int i,j;
FILE *outputStreamPtr;
char ch;
char filename[128];

gotoxy(1,24);
clreol();
textcolor(YELLOW);
cprintf("Save as: ");
textcolor(LIGHTGRAY);
scanf("%128s",filename);
strcat(filename,".gsd");
if((outputStreamPtr=fopen(filename,"w"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't creat output file! Press any key to continue");
        textcolor(LIGHTGRAY);
        getch();
        return 1;
}
/* write data to the file */
fprintf(outputStreamPtr,"%s\n",projectTitle);
fprintf(outputStreamPtr,"%d\n",sizeClassNumSimGrCir);
for(i=1;i<=sizeClassNumSimGrCir ;i++)
        fprintf(outputStreamPtr,"%10.0f  %6.2f  %10.4f\n",screenSize[i],
                        feedSizeDistribution[i],selectionFunction[i]);
if(normalisableBreakageFunction){
        fprintf(outputStreamPtr,"yes\n");
        for(i= 1;i<=sizeClassNumSimGrCir;i++)
                fprintf(outputStreamPtr,"%6.4f  \n",breakageFunction[i][1]);
}
else{
        fprintf(outputStreamPtr,"no\n");
        for(i=1;i<=sizeClassNumSimGrCir;i++)
                for(j=1;j<=sizeClassNumSimGrCir;j++){
                        if(i==j) fprintf(outputStreamPtr,"0.0000 \n");
                        else if(i>j)
                                fprintf(outputStreamPtr,"%6.4f  ",breakageFunction[i][j]);
                }
}
fprintf(outputStreamPtr,"%3.2f %3.2f %3.2f %3.2f %3.2f\n",*ptrTauPFSimGrCir,
                *ptrTauSPMSimGrCir,*ptrTauLPMSimGrCir,*ptrRefMillFeedRateSimGrCir,
                *ptrCurMillFeedRateSimGrCir);
/* close the file */
```

```
if(fclose(outputStreamPtr)!=0){
        gotoxy(1,24);
        cprintf("Error in closing output file! Press any key to continue");
        getch();
        return 1;
}
return 0;
}
```

```
/* simgrrun.c01 */
#include <dos.h>
#include <conio.h>
#include "global.h"

void PrintText(int x,int y,char* text,int forgcolor);
int BallMill(void);
int WriteToScreen(void);
int WriteToPrinter(void);
int WriteToFile(void);

int SimulateGrindingRun(void){
bool done=FALSE;
char key;
char *prompt="Print<P> Save result <S> Back <B> Quit <Q> ";
int retVal;

BallMill();
WriteToScreen();
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
clreol();
textcolor(LIGHTGRAY);
while(!done){
        key=getch();
        switch(key){
                case 'P':
                case 'p':
                WriteToPrinter();
                PrintText(1,24,"printing output ...",        YELLOW);
                clreol();
                delay(700);
                PrintText(1,24,prompt,YELLOW);
                clreol();
                gotoxy(37,24);
                textcolor(LIGHTGRAY);
                break;

                case 'S':
                case 's':
                WriteToFile();
                PrintText(1,24,"saving output ...",          YELLOW);
                gotoxy(1,24);
                textcolor(YELLOW);
                cprintf(prompt);
                clreol();
                textcolor(LIGHTGRAY);
                break;

                case 'B':
```

```
                case 'b':
                retVal=1;
                done=TRUE;
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
        }
}
return retVal;
}
```

```
/* bmsim.c01 */
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <assert.h>
#include <conio.h>
#include "global.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

void ShowMat(double **matPtr,int rows,int cols);
void ShowVec(double *vecPtr,int rows);

int BallMill(void){
int i,j,k;
double tauPF,tauSPM,tauLPM,mrt;
double WPF,WS,WL;
double x;

assert(*ptrCurMillFeedRateSimGrCir>0.0);
for(i=1;i<=sizeClassNumSimGrCir;i++)
        assert(selectionFunction[i]>0.0);
mrt=*ptrTauPFSimGrCir+2**ptrTauSPMSimGrCir+*ptrTauLPMSimGrCir;
assert(mrt>=EPS);
tauPF=(*ptrTauPFSimGrCir/mrt)*(*ptrRefMillFeedRateSimGrCir/(*ptrCurMillFeedRateSimGrCir));
tauSPM=(*ptrTauSPMSimGrCir/mrt)*(*ptrRefMillFeedRateSimGrCir/(*ptrCurMillFeedRateSimGrCir)
);
tauLPM=(*ptrTauLPMSimGrCir/mrt)*(*ptrRefMillFeedRateSimGrCir/(*ptrCurMillFeedRateSimGrCir
));
/*  calculating the t matrix */
for(i=1;i<=sizeClassNumSimGrCir;i++)
        for(j=1;j<=sizeClassNumSimGrCir;j++){
                if(i<j) t[i][j]=0.0;
                if(i==j) t[i][j]=selectionFunction[j];
                if(i>j){
                        double sum=0.0;
                        for(k=1;k<i;k++)
                                sum=sum+breakageFunction[i][k]*selectionFunction[k]*t[k][j];
                        if(EQ(selectionFunction[i],selectionFunction[j]))
                                selectionFunction[i]=selectionFunction[i]*0.99999;
                        t[i][j]=sum/(selectionFunction[i]-selectionFunction[j]);
                }
        }
/*      calculating the tinv matrix */
for(i=1;i<=sizeClassNumSimGrCir;i++)
        for(j=1;j<=sizeClassNumSimGrCir;j++){
```

```
                if(i<j) tinv[i][j]=0.0;
                if(i==j) tinv[i][j]=1/selectionFunction[j];
                if(i>j){
                        double sum=0.0;
                        for(k=1;k<i;k++)
                                sum=sum+t[i][k]*tinv[k][j];
                        tinv[i][j]=-sum/selectionFunction[i];
                }
        }
/*      calculating the diag matrix */
for(i=1;i<=sizeClassNumSimGrCir;i++){
        WPF=exp(-selectionFunction[i]*tauPF);
        WS=1+selectionFunction[i]*tauSPM;
        WL=1+selectionFunction[i]*tauLPM;
        diag[i]=WPF/(WL*pow(WS,2));
}
/*      calculating the t * diag */
for(i=1;i<=sizeClassNumSimGrCir;i++){
        for(j=1;j<=sizeClassNumSimGrCir;j++)
                tdiag[i][j]=t[i][j]*diag[j];
}
/*      calculating the mill transformation matrix */
for(i=1;i<=sizeClassNumSimGrCir;i++)
        for(j=1;j<=sizeClassNumSimGrCir;j++){
                double sum=0.0;
                for(k=1;k<=sizeClassNumSimGrCir;k++)
                        sum=sum+tdiag[i][k]*tinv[k][j];
                trans[i][j]=sum;
        }
/*      calculating the ball mill discharge */
for(i=1;i<=sizeClassNumSimGrCir;i++){
        double sum=0;
        for(j=1;j<=sizeClassNumSimGrCir;j++)
                sum=sum+trans[i][j]*feedSizeDistribution[j];
        dischargeSizeDistribution[i]=sum;
}
return 0;
}

void ShowMat(double **matPtr,int rows,int cols){
int i,j;

clrscr();
for(i=1;i<=rows;i++){
        gotoxy(1,i);
        for(j=1;j<=cols;j++) printf("%6.2f ",matPtr[i][j]);
}
getch();
}

void ShowVec(double *vecPtr,int rows){
```

```
int i;

clrscr();
for(i=1;i<=rows;i++){
        gotoxy(1,i);
        printf("%6.2f ",vecPtr[i]);
}
        getch();
}
```

```
/* wrtprmsg.c01 */
#include <stdio.h>
#include <dos.h>
#include "global.h"

int WriteToPrinter(void){
int i,j;
struct date today;

fprintf(stdprn,"\r\n\r\n\n"
"                  **********************************\r\n"
"                       Simulation Results      \r\n"
"                  **********************************\r\n\n");
fprintf(stdprn,projectTitle);
fprintf(stdprn,"\r\n");
getdate((struct date*) &today);
fprintf(stdprn,
"Date: %d/%d/%d\r\n",today.da_mon,today.da_day,today.da_year);
fprintf(stdprn,
                  "Tau Plug Flow = %.2f  Tau Small PM = %.2f Tau Large PM = %.2f\r\n\r\n",
                  *ptrTauPFSimGrCir,*ptrTauSPMSimGrCir,*ptrTauLPMSimGrCir);
fprintf(stdprn,"Reference feed flow rate = %5.1f t/h \r\n",*ptrRefMillFeedRateSimGrCir);
fprintf(stdprn,"Current feed flow rate = %5.1f t/h \r\n\r\n",*ptrCurMillFeedRateSimGrCir);
fprintf(stdprn,"Breakage Function Matrix \r\n");
for(i=1;i<=sizeClassNumSimGrCir;i++)
        for(j=1;j<=sizeClassNumSimGrCir;j++){
                  if(i==j) fprintf(stdprn,"0.0000  \r\n");
                  else      if(i>j)
                                          fprintf(stdprn,"%6.4f  ",breakageFunction[i][j]);
        }
fprintf(stdprn,"\r\n\
-------------------------------------------------------\r\n\
| CLASS | SIEVE SIZE  |  FEED  |  PRODUCT  | SELEC. FUNC. |\r\n\
|-------|-------------|--------|-----------|--------------|\r\n");
for(i=1;i<=sizeClassNumSimGrCir;i++)
        fprintf(stdprn,
"|  %5d  |  %10.0f  |  %6.2f  |  %10.2f  |  %10.4f  |\r\n",i,
screenSize[i],feedSizeDistribution[i],dischargeSizeDistribution[i],
selectionFunction[i]);
fprintf(stdprn,
"\
|-------|-------------|--------|-----------|--------------|");
fprintf(stdprn,"\f");
return 0;
}
```

```
/* wrtfsimg.c01 */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include "global.h"

int WriteToFile(void){
short i,j;
FILE *OutputStream;
char ch;
char *filename;
struct date today;

gotoxy(1,24);
clreol();
textcolor(YELLOW);
cprintf("Save as: ");
textcolor(LIGHTGRAY);
scanf("%128s",filename);
if((OutputStream=fopen(filename ,"w"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't creat output file! Press any key to continue");
        textcolor(LIGHTGRAY);
        getch();
        return 1;
}
/* write data to the file */
fprintf(OutputStream,"\n\
                ********************\n\
                Simlation Results \n\
                ********************\n\n");
fprintf(OutputStream,projectTitle);
fprintf(OutputStream,"\n");
getdate((struct date*) &today);
fprintf(OutputStream,
"Date: %d/%d/%d\n",today.da_mon,today.da_day,today.da_year);
fprintf(OutputStream,
        "Tau Plug Flow = %5.2f  Tau Small PM = %5.2f Tau Large PM = %5.2f\n\n",
        *ptrTauPFSimGrCir,*ptrTauSPMSimGrCir,*ptrTauLPMSimGrCir);
fprintf(OutputStream,"Reference feed flow rate = %5.1f t/h \n",*ptrRefMillFeedRateSimGrCir);
fprintf(OutputStream,"Current feed flow rate = %5.1f t/h \n\n",*ptrCurMillFeedRateSimGrCir);
fprintf(OutputStream,"Breakage Function Matrix \n");
for(i=1;i<= sizeClassNumSimGrCir;i++)
        for(j=1;j<=sizeClassNumSimGrCir;j++){
                if(i==j) fprintf(OutputStream,"0.0000 \n");
                else if(i>j)
                fprintf(OutputStream,"%6.4f ",breakageFunction[i][j]);
        }
fprintf(OutputStream,"\n\
```

```
┌──────────┬──────────────┬──────────┬──────────────┬──────────────┐\n\
| CLASS | SIEVE SIZE | FEED | PRODUCT | SELEC. FUNC. | \n\
├──────────┼──────────────┼──────────┼──────────────┼──────────────┤\n");
for(i=1;i<=sizeClassNumSimGrCir;i++)
        fprintf(OutputStream,
" | %5d  | %10.0f  | %6.2f  | %10.2f  | %10.4f  | \n",i,
screenSize[i],feedSizeDistribution[i],dischargeSizeDistribution[i],
selectionFunction[i]);
fprintf(OutputStream,
"\ └──────────┴──────────────┴──────────┴──────────────┴──────────────┘ ");
/* close the file */
if(fclose(OutputStream)!=0){
        gotoxy(1,24);
        cprintf("Error in closing output file! Press any key to continue");
        getch();
        return 1;
}
return 0;
}
```

```
/* wrtscrsg.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "global.h"

int WriteToScreen(void){
int i,j;
char ch;
struct date today;

clrscr();
gotoxy(22,3);
textbackground(LIGHTGREEN);
textcolor(WHITE);
cprintf("                              ");
gotoxy(22,4);
cprintf("        Simulation Data       ");
gotoxy(22,5);
cprintf("                              ");
textbackground(BLACK);
textcolor(LIGHTGRAY);
gotoxy(1,7);
cprintf(projectTitle);
cprintf("\r\n");
getdate((struct date*) &today);
cprintf("Date: %d/%d/%d\r\n",today.da_mon,today.da_day,today.da_year);
cprintf("Tau Plug Flow = %.2f  Tau Small PM = %.2f Tau Large PM = %.2f\r\n",
                *ptrTauPFSimGrCir,*ptrTauSPMSimGrCir,*ptrTauLPMSimGrCir);
cprintf("Reference feed flow rate = %5.1f t/h \r\n",*ptrRefMillFeedRateSimGrCir);
cprintf("Current feed flow rate = %5.1f t/h \r\n\r\n",*ptrCurMillFeedRateSimGrCir);
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
gotoxy(1,1);
cprintf("Breakage Function Matrix \r\n");
for(i=1;i<=sizeClassNumSimGrCir;i++){
        if(i==8){
                gotoxy(1,24);
                textcolor(YELLOW);
                cprintf("Press any key to continue ...");
                clreol();
                textcolor(LIGHTGRAY);
                getch();
                clrscr();
        }
```

```
        for(j=1;j<=sizeClassNumSimGrCir;j++){
        if(i==j) cprintf("0.00\r\n");
        else if(i>j) cprintf("%6.4f ",breakageFunction[i][j]);
        }
}
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
gotoxy(22,11);
textbackground(LIGHTGREEN);
textcolor(WHITE);
cprintf("                              ");
gotoxy(22,12);
cprintf("     Simulation Data (Cont'd)     ");
gotoxy(22,13);
cprintf("                              ");
textbackground(BLACK);
textcolor(LIGHTGRAY);
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
cprintf("\r\n\
┌──────────┬──────────────┬──────────┬──────────────────┐\r\n\
│ CLASS │ SIEVE SIZE │ FEED │ SELECTION FUNCTION │\r\n\
├──────────┼──────────────┼──────────┼──────────────────┤\r\n");
for(i=1;i<=sizeClassNumSimGrCir;i++)
        cprintf(
" | %5d  | %10.0f  | %6.2f  | %15.4f       | \r\n",i,
screenSize[i],feedSizeDistribution[i],selectionFunction[i]);
cprintf(
"\
└──────────┴──────────────┴──────────┴──────────────────┘ ");
cprintf("\r\n");
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
```

```
gotoxy(22,11);
textbackground(LIGHTGREEN);
textcolor(WHITE);
cprintf("                              ");
gotoxy(22,12);
cprintf("      Simulation Results      ");
gotoxy(22,13);
cprintf("                              ");
textbackground(BLACK);
textcolor(LIGHTGRAY);
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
cprintf("\r\n\
┌──────────┬──────────────┬──────────┬──────────────┬──────────────────────┐\r\n\
│  CLASS   │  SIEVE SIZE  │   FEED   │   PRODUCT    │  SELECTION FUNCTION  │\r\n\
├──────────┼──────────────┼──────────┼──────────────┼──────────────────────┤\r\n");
for(i=1;i<=sizeClassNumSimGrCir;i++)
        cprintf(
"  %5d    %10.01f     %6.2f    %10.2f     %15.4f          \r\n",i,
screenSize[i],feedSizeDistribution[i],dischargeSizeDistribution[i],
selectionFunction[i]);
cprintf(
"\
└──────────┴──────────────┴──────────┴──────────────┴──────────────────────┘");
cprintf("\r\n");
return 0;
}
```

```
/* ballsize.c01 */
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "global.h"

extern char savescr[4096];
extern enum operationmode mode;

int ShowBallSizeOptMenu(void);
int BallSizeOptDataPg1(void);
int BallSizeOptDataPg2(void);
int BallSizeOptDataPg3(void);
int BallSizeOptRun(void);
int BallSizeOptHelp(void);
double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);

extern char projectTitleBallSzOpt[85];
int *ptrK,k1,k2,*ptr_iopt,*ptr_nest,*ptr_n,*ptr_n1,*ptr_n2,*ptr_lwrk,*ptr_ier;
extern int *ptr_iwrk;
extern double *ptrPosKnots,*ptrPosKnots1,*ptrPosKnots2,*ptr_wrk,*ptr_c,*ptr_c1,
                *ptr_c2,*ptr_w,*ptr_w1,*ptr_w2;
extern double *ptrParticleSize,*ptrXseries,*ptrSelecFunc,*ptrCurSelecFunc,*ptrNewSelecFunc,
                *ptrFtdSelecFunc,*ptrFtdCurSelecFunc,*ptrFtdNewSelecFunc;
double *ptrXb,*ptrXe,*ptrIniS,*ptrIniS1,*ptrIniS2,*ptrS,*ptrDifS,
        *ptrDifS1,*ptrDifS2,*ptr_fp,*ptr_fp1,*ptr_fp2;
extern double curBallDia,*ptrCurBallDia;
extern double newBallDia,*ptrNewBallDia;
double *ptrScreenSizeBallSzOpt,*ptrSelecFuncBallSzOpt,
        *ptrStandardDevBallSzOpt;
double *ptrFactorK;
double *ptr_fin_s,*ptr_fin_s1,*ptr_fin_s2;
int k,iopt,m,nest,n1,n2,lwrk,ier;
double xb,xe,iniS,iniS1,iniS2,s,fin_s1,fin_s2,difS,difS1,difS2,factorK,fp1,
        fp2;

int OptimizeBallSize(void){
int i,next;
bool done;

ptrK=&k;ptr_iopt=&iopt;
ptr_nest=&nest;ptr_n1=&n1;ptr_n2=&n2;
ptr_lwrk=&lwrk;ptr_ier=&ier;ptrXb=&xb;ptrXe=&xe;ptrIniS=&iniS;ptrS=&s;
ptr_fin_s1=&fin_s1;ptr_fin_s2=&fin_s2;ptrDifS=&difS;
ptrFactorK=&factorK;ptr_fp1=&fp1;ptr_fp2=&fp2;

/* default values */
*ptr_iopt=0;
k1=3;
```

```
k2=3;
*ptrK=3;
iniS1=0.0;
iniS2=0.0;
*ptrIniS=0.0;
difS1=0.001;
difS2=0.001;
*ptrDifS=0.001;
*ptrFactorK=4.4e-04;
*ptr_nest=27;
*ptr_lwrk=715;
next=1;
for(;;){
        switch(next){
                case 1:
                next=BallSizeOptDataPg1();
                break;

                case 2:
                next=BallSizeOptDataPg2();
                break;

                case 3:
                next=BallSizeOptDataPg3();
                break;

                case 4:
                next=BallSizeOptRun();
                break;

                case 'Q':
                case 'q':
                break;
        }
        if(next==-1000) break;
}
return 0;
}
```

```
/* ret_bsod.c01 */
#include <dir.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include "global.h"

extern double *ptrParticleSize;
extern double curBallDia,newBallDia;
extern char logX[4],logY[4];

int HandleEndOfFile(void);
void PrintText(int x,int y,char* text,int forgcolor);
char *GetStringAt(int x,int y,int maxlen);
int GetFileNames(char *ptrExt);

int OpenBallSzOptDataFile(void){
FILE *streamPtr;
char dataFileName[128];
char *ptrString,*newPath;
int i,j;
int status=0;
int key='$';
char string[15];
char *FormatErrPrompt=
"Data file format is incorrect! Press any key to continue ...";

gotoxy(1,24);
textcolor(YELLOW);
cprintf("Please enter file name?  ");
gotoxy(42,24);
textcolor(LIGHTGRAY);
cprintf(                          "(type > to change current directory)");
clreol();
textcolor(YELLOW);
ptrString=GetStringAt(25,24,15);
if(ptrString[0]=='>'){
        gotoxy(1,24);
        textcolor(YELLOW);
        cprintf(
        "Please enter new directory:  ");
        clreol();
        textcolor(LIGHTGRAY);
        newPath=GetStringAt(29,24,54);
        if(chdir(newPath)!=0){
                gotoxy(1,24);
                textcolor(LIGHTRED);
                if(strlen(newPath)<=30)
                        cprintf("Cannot change to [%s].  Press any key to continue ...",newPath);
```

```
                else
                        cprintf("Cannot change to new directory.  Press any key to continue ...");
                clreol();
                getch();
                return -1;   // this cause exit to calling module, i.e. data page one
        }
        fflush(stdin);
        GetFileNames("bsd");
        gotoxy(1,24);
        textcolor(YELLOW);
        cprintf("Please enter file name?  ");
        textcolor(YELLOW);
        ptrString=GetStringAt(25,24,15);
}
strcpy(dataFileName,ptrString);
if(strlen(dataFileName)<=8) strcat(dataFileName,".bsd");
if((streamPtr=fopen((const char*) dataFileName,"r"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't open file! press any key to continue ...");
        clreol();
        getch();
        return -1;
}
/* starting to read file after opening input file
        reading one string for project title */
while((key=getc(streamPtr))!=EOF){
        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
        else{
                fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                fgets(projectTitleBallSzOpt,80,streamPtr);
                projectTitleBallSzOpt[strlen(projectTitleBallSzOpt)-1]='\0';
                break;
        }
}
if(key==EOF){
        HandleEndOfFile();
        return -1;
}
/* reading sizeClassNumbers */
status=0;
while((key=getc(streamPtr))!=EOF && status<1){
        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
        else{
                fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                status=fscanf(streamPtr,"%d",&sizeClassNumBallSzOpt);
                if(status<1){
                        PrintText(1,24,FormatErrPrompt,LIGHTRED);
                        clreol();
                        getch();
                        return -1;
```

```
                }
        }
}
if(key==EOF){
        HandleEndOfFile();
        return -1;
}
status=0;
while((key=getc(streamPtr))!=EOF&&status<2){
        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
        status=fscanf(streamPtr,"%lf %lf",&curBallDia,&newBallDia);
        if(status<2){
                PrintText(1,24,FormatErrPrompt,LIGHTRED);
                clreol();
                getch();
                return -1;
        }
}
if(key==EOF){
        HandleEndOfFile();
        return -1;
}
for(i=1;i<=sizeClassNumBallSzOpt;i++){
        status=0;
        while((key=getc(streamPtr))!=EOF&&status<4){
                if(key=='\n'||key==' '||key=='\t'||key==',') continue;
                fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
                status=fscanf(streamPtr,"%lf %lf %lf %lf",
                                                ptrScreenSizeBallSzOpt+i-1,ptrParticleSize+i-1,
                                                ptrSelecFuncBallSzOpt+i-1,
                                                ptrStandardDevBallSzOpt+i-1);
                if(status<4){
                        PrintText(1,24,FormatErrPrompt,LIGHTRED);
                        clreol();
                        getch();
                        return -1;
                }
        }
        if(key==EOF){
                HandleEndOfFile();
                return -1;
        }
}
status=0;
while((key=getc(streamPtr))!=EOF&&status<4){
        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
        status=fscanf(streamPtr,"%lf %d %lf %lf",ptrFactorK,ptrK,ptrIniS,ptrDifS);
        if(status<4){
                PrintText(1,24,FormatErrPrompt,LIGHTRED);
```

```
                clreol();
                getch();
                return -1;
        }
}
if(key==EOF){
        HandleEndOfFile();
        return -1;
}
/* reading logX */
while((key=getc(streamPtr))!=EOF){
        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
        fgets(logX,4,streamPtr);
        logX[3]='\0';
        break;
}
if(key==EOF){
        HandleEndOfFile();
        return -1;
}
/* reading logY */
while((key=getc(streamPtr))!=EOF){
        if(key=='\n'||key==' '||key=='\t'||key==',') continue;
        fseek(streamPtr,ftell(streamPtr)-1,SEEK_SET);
        fgets(logY,4,streamPtr);
        logX[3]='\0';
        break;
}
if(key==EOF){
        HandleEndOfFile();
        return -1;
}
if(fclose(streamPtr)!=0){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't close file! Press any key to continue ...\n\r");
        textcolor(LIGHTGRAY);
        getch();
        return -1;
}
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Data transfered successfully, press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
return 0;
}
```

```
/* bsopdp1.c01 */
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <float.h>
#include <string.h>
#include "global.h"

#define MAXOPTNO 6
#define MINOPTNO 1
#define ESC 27

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

extern int savescr[4096];
extern bool isSecondGraph;
extern int *ptrK,k1,k2;
extern double *ptrFactorK;
extern double *ptrXb,*ptrXe,*ptrIniS,iniS1,iniS2,*ptrS,*ptrDifS,
                                    difS1,difS2,*ptr_fp,*ptr_fp1,*ptr_fp2;
extern char logX[4],logY[4];

void PrintText(int x,int y,char* text,int forgcolor);
char *GetStringAt(int x,int y,int maxlen);
int GetFileNames(char *ptrExt);
int OpenBallSzOptDataFile(void);

int BallSizeOptDataPg1(void){
extern int* ptr_iopt;
bool done;
char *ptrLogX,*ptrLogY,*string;
int retVal,key,option,status,maxlen;

ptrLogX=strupr(logX);
ptrLogY=strupr(logY);
clrscr();
PrintText(20,1,"Ball Size Optimization Program Options",WHITE);
PrintText(3,3,"Morrel's scaling procedure",WHITE);
gotoxy(8,4);
printf("1. K (1/mm)                              %f",*ptrFactorK);
PrintText(3,6,"Spline curve fitting",WHITE);
gotoxy(8,7);
printf("2. Spline degree of the first curve              %d",k1);
gotoxy(8,8);
printf("3. Spline degree of the second curve              %d",k2);
gotoxy(8,9);
```

```
printf("4. Initial smoothing factor                %lf",*ptrIniS);
gotoxy(8,10);
printf("5. Increment/decrement of smoothing factor   %lf",*ptrDifS);
PrintText(3,12,"Graph x and y scales",WHITE);
gotoxy(8,13);
printf("6. Logaritmic particle size values          %c",ptrLogX[0]);
gotoxy(8,14);
printf("7. Logaritmic selection function values     %c",ptrLogY[0]);
gotoxy(1,24);
done=FALSE;
maxlen=20;
while(!done){
        PrintText(3,24,"Open <O>  Change <C>  Next<N>  Quit <Q> ==>
",YELLOW);
        clreol();
        key=getch();
        switch(key){
                case 'O':
                case 'o':
                gettext(1,1,80,25,savescr);
                GetFileNames("bsd");
                OpenBallSzOptDataFile();
                puttext(1,1,80,25,savescr);
                textcolor(LIGHTGRAY);
                gotoxy(55,4);
                cprintf("%f",*ptrFactorK);
                clreol();
                gotoxy(55,7);
                cprintf("%d",k1);
                clreol();
                gotoxy(55,8);
                cprintf("%d",k2);
                clreol();
                gotoxy(55,9);
                cprintf("%f",*ptrIniS);
                clreol();
                gotoxy(55,10);
                cprintf("%f",*ptrDifS);
                clreol();
                gotoxy(55,13);
                ptrLogX=strupr(logX);
                cprintf("%c",ptrLogX[0]);
                clreol();
                gotoxy(55,14);
                ptrLogY=strupr(logY);
                cprintf("%c",ptrLogY[0]);
                clreol();
                gotoxy(3,24);
                textcolor(YELLOW);
                cprintf("Open <O>  Change <C>  Next<N>  Quit <Q> ==>");
                clreol();
```

```
                textcolor(LIGHTGRAY);
                break;

                case 'C':
                case 'c':
                status=0;
                option=0;
                while((status!=1)&&((option>MAXOPTNO)||(option<MINOPTNO))){
                        PrintText(3,24,"Selection: ",LIGHTGRAY);
                        clreol();
                        string=GetStringAt(14,24,maxlen);
                        status=sscanf(string,"%d",&option);
                }
                switch(option){
                        case 1:
                        status=0;
                        while(status!=1||LE(*ptrFactorK,0)){
                                PrintText(3,24,"K factor: ",LIGHTGRAY);
                                clreol();
                                string=GetStringAt(13,24,maxlen);
                                status=sscanf(string,"%lf" ,ptrFactorK);
                        }
                        gotoxy(55,4);
                        clreol();
                        printf("%lf",*ptrFactorK);
                        break;

                        case 2:
                        status=0;
                        while(status!=1||k1<1||k1>5){
                                PrintText(3,24,"Spline degree of the first curve: ",LIGHTGRAY);
                                clreol();
                                string=GetStringAt(37,24,maxlen);
                                status=sscanf(string,"%d",&k1);
                        }
                        gotoxy(55,7);
                        clreol();
                        printf("%d",k1);
                        break;

                        case 3:
                        status=0;
                        while(status!=1||k2<1||k2>5){
                                PrintText(3,24,"Spline degree of the second curve:
",LIGHTGRAY);
                                clreol();
                                string=GetStringAt(38,24,maxlen);
                                status=sscanf(string,"%d",&k2);
                        }
                        gotoxy(55,8);
                        clreol();
```

```
                printf("%d",k2);
                break;

                case 4:
                status=0;
                while(status!=1||LT(*ptrIniS,0)){
                        PrintText(3,24,"Initial smoothing factor: ",LIGHTGRAY);
                        clreol();
                        string=GetStringAt(29,24,maxlen);
                        status=sscanf(string,"%lf",ptrIniS);
                }
                gotoxy(55,9);
                clreol();
                printf("%lf",*ptrIniS);
                break;

                case 5:
                status=0;
                while(status!=1||LT(*ptrDifS,0)){
                        PrintText(3,24,"Increment/decrement of smoothing factor:
",LIGHTGRAY);
                        clreol();
                        string=strupr(GetStringAt(44,24,maxlen));
                        status=sscanf(string,"%lf",ptrDifS);
                }
                gotoxy(55,10);
                clreol();
                printf("%lf",*ptrDifS);
                break;

                case 6:
                PrintText(3,24,"Use logaritms of particle size values (y/n): ",LIGHTGRAY);
                clreol();
                string=strupr(GetStringAt(48,24,maxlen));
                sscanf(string,"%s",logX);
                while(logX[0]!='y'&&logX[0]!='Y'&&logX[0]!='N'&&logX[0]!='n'){
                        PrintText(3,24,"Use logaritms of particle size values (y/n):
",LIGHTGRAY);
                        clreol();
                        string=strupr(GetStringAt(48,24,maxlen));
                        sscanf(string,"%s",logX);
                }
                gotoxy(55,13);
                clreol();
                printf("%c",logX[0]);
                break;

                case 7:
                PrintText(3,24,"Use logaritms of selection function values (y/n):
",LIGHTGRAY);
                clreol();
```

```
                        string=strupr(GetStringAt(53,24,maxlen));
                        sscanf(string,"%s",logY);
                        while(logY[0]!='y'&&logY[0]!='Y'
                                        &&logY[0]!='N'&&logY[0]!='n'){
                                PrintText(3,24,"Use logaritms of selection function values (y/n):
",LIGHTGRAY);
                                clreol();
                                string=strupr(GetStringAt(53,24,maxlen));
                                sscanf(string,"%s",logY);
                        }
                        gotoxy(55,14);
                        clreol();
                        printf("%c",logY[0]);
                        break;
                }
        break;

        case 'N':
        case 'n':
        retVal=2;
        done=TRUE;
        break;

        case 'Q':
        case 'q':
        retVal=-1000;
        done=TRUE;
        break;
        }
}
return retVal;
}

/* Changing settings during curve fitting */
int SetParamOnLine(bool isSecondGraph){
extern int* ptr_iopt;
bool done;
char* string;
int retVal,key,option,status,maxlen;

clrscr();
PrintText(20,1,"Ball Size Optimization Program Options",WHITE);
PrintText(25,3,"Spline Curve Fitting Options",WHITE);
PrintText(3,6,"Spline curve fitting",WHITE);
gotoxy(8,7);
printf("1. Spline degree                    %d",*ptrK);
gotoxy(8,8);
printf("2. Initial smoothing factor          %lf",*ptrIniS);
gotoxy(8,9);
printf("3. Increment/decrement of smoothing factor   %lf",*ptrDifS);
gotoxy(1,24);
```

```
done=FALSE;
maxlen=20;
while(!done){
        PrintText(3,24,"Change <C> Back <B> ==> ",YELLOW);
        clreol();
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                status=0;
                option=0;
                while((status!=1)&&((option>MAXOPTNO)||(option<MINOPTNO))){
                        PrintText(3,24,"Selection: ",LIGHTGRAY);
                        clreol();
                        string=GetStringAt(14,24,maxlen);
                        status=sscanf(string,"%d",&option);
                }
                switch(option){
                        case 1:
                        status=0;
                        while(status!=1||*ptrK<1||*ptrK>5){
                                PrintText(3,24,"Spline degree: ",LIGHTGRAY);
                                clreol();
                                string=GetStringAt(18,24,maxlen);
                                status=sscanf(string,"%d",ptrK);
                        }
                        gotoxy(55,7);
                        clreol();
                        printf("%d",*ptrK);
                        break;

                        case 2:
                        status=0;
                        while(status!=1||*ptrIniS<0){
                                PrintText(3,24,"Initial smoothing factor: ",LIGHTGRAY);
                                clreol();
                                string=GetStringAt(29,24,maxlen);
                                status=sscanf(string,"%lf",ptrIniS);
                        }
                        gotoxy(55,8);
                        clreol();
                        printf("%lf",*ptrIniS);
                        break;
                        case 3:
                        status=0;
                        while(status!=1||*ptrDifS<0){
                                PrintText(3,24,"Increment/decrement of smoothing factor:
",LIGHTGRAY);
                                clreol();
                                string=GetStringAt(44,24,maxlen);
                                status=sscanf(string,"%lf",ptrDifS);
```

```
                    }
                    gotoxy(55,9);
                    clreol();
                    printf("%lf",*ptrDifS);
                    break;
                }
                break;

                case 'B':
                case 'b':
                if(isSecondGraph){
                    k2=*ptrK;
                    iniS2=*ptrIniS;
                    difS2=*ptrDifS;
                }
                else{
                    k1=*ptrK;
                    iniS1=*ptrIniS;
                    difS1=*ptrDifS;
                }
                retVal=-1000;
                done=TRUE;
                break;
        }
}
return retVal;
}
```

```
/* bsopdp2.c01 */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include "global.h"

#define MAXOPTNO 6
#define MINOPTNO 1
#define ESC  27

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y))<=fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

#define promptEndPos 52

extern char savescr[4096];
extern double curBallDia,*ptrCurBallDia;
extern double newBallDia,*ptrNewBallDia;

void PrintText(int x,int y,char* text,int forgcolor);
void PrintErrMsg(void);
char *GetStringAt(int x,int y,int maxlen);

int BallSizeOptDataPg2(void){
extern int* ioptPtr;
bool done;
char* string;
int retVal,key,option,status,row;
double *addressFloat;
int *addressInt;
char *prompt="Change <C>   Previous <P>   Next <N>   Quit <Q> ==>";

ptrCurBallDia=&curBallDia;
ptrNewBallDia=&newBallDia;
if(sizeClassNumBallSzOpt==0)  sizeClassNumBallSzOpt=8;
clrscr();
PrintText(20,1,"Ball Size Optimization Program Data",WHITE);
gotoxy(2,4);
printf("   Project title              %s",projectTitleBallSzOpt);
gotoxy(2,6);
printf("   Size class numbers          %d",sizeClassNumBallSzOpt);
gotoxy(2,8);
printf("   Current diameter of balls   %-3.1f",*ptrCurBallDia);
gotoxy(2,10);
printf("   New diameter of balls       %-3.1f",*ptrNewBallDia);
```

```
gotoxy(1,24);
done=FALSE;
while(!done){
        PrintText(2,24,prompt,YELLOW);
        clreol();
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                gotoxy(34,4);
                /* cursor at 34,4 */
                fflush(stdin);
                if((key=getch())!=ESC){
                        ungetch(key);
                        if(key!='\r') clreol();
                        string=GetStringAt(34,4,45);
                        if(string[0]=='\0'){
                                cprintf("%s",projectTitleBallSzOpt);
                                clreol();
                                gotoxy(34,6);
                        }
                        else{
                                strcpy(projectTitleBallSzOpt,string);
                                gotoxy(34,4);
                                cprintf("%s",projectTitleBallSzOpt);
                                clreol();
                                gotoxy(34,6);
                        }
                }
                else{
                        gotoxy(promptEndPos,24);
                        break;
                }
                /* cursor at 34,6 */
                fflush(stdin);
                if((key=getch())!=ESC){
                        ungetch(key);
                        addressInt=&sizeClassNumBallSzOpt;
                        status=0;
                        while(!done){
                                string=GetStringAt(34,6,5);
                                if(string[0]=='\0'){
                                        cprintf("%1d",*addressInt);
                                        clreol();
                                        gotoxy(34,8);
                                        break;
                                }
                                status =sscanf(string,"%d",addressInt);
                                if(status==1){
                                        if(*addressInt>MAXSIZECLASSNO||*addressInt<1){
                                                PrintErrMsg();
```

```
                                        PrintText(2,24,prompt,YELLOW);
                                        gotoxy(34,6);
                                        clreol();
                                        gotoxy(34,6);
                                        done=FALSE;
                                }
                                else{
                                        gotoxy(34,6);
                                        cprintf("%1d",*addressInt);
                                        clreol();
                                        gotoxy(34,8);
                                        break;
                                }
                        }
                        else{
                                PrintErrMsg();
                                gotoxy(2,24);
                                textcolor(YELLOW);
                                cprintf(prompt);
                                clreol();
                                textcolor(LIGHTGRAY);
                                gotoxy(34,6);
                                clreol();
                                gotoxy(34,6);
                        }
                }
        }
        else{
                gotoxy(promptEndPos,24);
                break;
        }
        /* cursor at 34,8 */
        fflush(stdin);
        if((key=getch())!=ESC){
                ungetch(key);
                addressFloat=ptrCurBallDia;
                status=0;
                while(!done){
                        string=GetStringAt(34,8,10);
                        if(string[0]=='\0'){
                                cprintf("%-3.1f",*addressFloat);
                                clreol();
                                gotoxy(34,10);
                                break;
                        }
                        status=sscanf(string,"%lf",addressFloat);
                        if(status==1){
                                if(LE(*addressFloat,0)){
                                        PrintErrMsg();
                                        PrintText(2,24,prompt,YELLOW);
                                        gotoxy(34,8);
```

```
                                clreol();
                                gotoxy(34,8);
                                done=FALSE;
                        }
                        else{
                                gotoxy(34,8);
                                cprintf("%-3.1f",*addressFloat);
                                clreol();
                                gotoxy(34,10);
                                break;
                        }
                }
                else{
                        PrintErrMsg();
                        gotoxy(2,24);
                        textcolor(YELLOW);
                        cprintf(prompt);
                        clreol();
                        textcolor(LIGHTGRAY);
                        gotoxy(34,8);
                        clreol();
                        gotoxy(34,8);
                }
        }
}
else{
        gotoxy(promptEndPos,24);
        break;
}
/*      cursor at 34,10 */
fflush(stdin);
if((key=getch())!=ESC){
        ungetch(key);
                addressFloat=ptrNewBallDia;
                status=0;
                while(!done){
                        string=GetStringAt(34,10,10);
                        if(string[0]=='\0'){
                                cprintf("%-3.1f",*addressFloat);
                                clreol();
                                gotoxy(promptEndPos,24);
                                break;
                        }
                        status=sscanf(string,"%lf",addressFloat);
                        if(status==1){
                                if(LE(*addressFloat,0)){
                                        PrintErrMsg();
                                        PrintText(2,24,prompt,YELLOW);
                                        gotoxy(34,10);
                                        clreol();
                                        gotoxy(34,10);
```

```
                                        done=FALSE;
                                }
                                else{
                                        gotoxy(34,10);
                                        cprintf("%-3.1f",*addressFloat);
                                        clreol();
                                        gotoxy(promptEndPos,24);
                                        break;
                                }
                        }
                        else{
                                PrintErrMsg();
                                gotoxy(2,24);
                                textcolor(YELLOW);
                                cprintf(prompt);
                                clreol();
                                textcolor(LIGHTGRAY);
                                gotoxy(34,10);
                                clreol();
                                gotoxy(34,10);
                        }
                }
        }
        else{
                gotoxy(promptEndPos,24);
                break;
        }
        break;

        case 'P':
        case 'p':
        retVal=1;
        done=TRUE;
        break;

        case 'N':
        case 'n':
        if(sizeClassNumBallSzOpt==0){
                gettext(1,10,80,15,savescr);

PrintText(2,11,"┌──────────────────────────────────────────────────────────────────────────────┐
",WHITE);
                        PrintText(2,12,"|      Size class numbers must be greater than 0.
Press any key ...       |  ",WHITE);

PrintText(2,13,"└──────────────────────────────────────────────────────────────────────────────┘
",WHITE);
                        _setcursortype(_NOCURSOR);
                        getch();
```

```
                        gotoxy(20,10);
                        puttext(1,10,80,15,savescr);
                        _setcursortype(_NORMALCURSOR);
                        gotoxy(promptEndPos,24);
                        done=FALSE;
                        break;
                }
                if(LE(*ptrCurBallDia,0)){
                        gettext(1,10,80,15,savescr);

PrintText(2,11,"┌──────────────────────────────────────────────────────────────────────────────┐
",WHITE);
                        PrintText(2,12,"|      Grinding ball diameter must be greater than
zero. Press any key ... | ",WHITE);

PrintText(2,13,"└──────────────────────────────────────────────────────────────────────────────┘
",WHITE);
                        _setcursortype(_NOCURSOR);
                        getch();
                        gotoxy(20,10);
                        puttext(1,10,80,15,savescr);
                        _setcursortype(_NORMALCURSOR);
                        gotoxy(promptEndPos,24);
                        done=FALSE;
                        break;
                }
                if(LE(*ptrNewBallDia,0)){
                        gettext(1,10,80,15,savescr);

PrintText(2,11,"┌──────────────────────────────────────────────────────────────────────────────┐
",WHITE);
                        PrintText(2,12,"|      Grinding ball diameter must be greater than
zero. Press any key ... | ",WHITE);

PrintText(2,13,"└──────────────────────────────────────────────────────────────────────────────┘
",WHITE);
                        _setcursortype(_NOCURSOR);
                        getch();
                        gotoxy(20,10);
                        puttext(1,10,80,15,savescr);
                        _setcursortype(_NORMALCURSOR);
                        gotoxy(promptEndPos,24);
                        done=FALSE;
                        break;
                }
                retVal=3;
                done=TRUE;
```

```
                        break;

                        case 'Q':
                        case 'q':
                        retVal=-1000;
                        done=TRUE;
                        break;
                }
        }
return retVal;
}
```

```
/* bsopdp3.c01 */
#include <stdio.h>
#include <conio.h>
#include <float.h>
#include <math.h>
#include "global.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

#define FORTH_ROOT_OF_2   1.1892071
extern char savescr[4096];
extern double *ptrXb,*ptrXe,*ptr_w1;
extern double *ptrParticleSize,*ptrXseries,*ptrCurSelecFunc;
void GetColOfData(int row,int column,int noSzClasses,int* xOfColumnPtr,
                                                    int maxlen,char* formatStr,char* formatStrLeftJus,
                                                    double minAllowedRange,double
maxAllowedRange,
                                                    double *tempInputPtr);
void PrintText(int x,int y,char* text,int forgcolor);
char *GetStringAt(int x,int y,int maxlen);
double FindMax(double* vector,int n);
double FindMin(double* vector,int n);
int SaveBallSzOptData(void);

int BallSizeOptDataPg3(void){
extern bool mode;
bool done;
int status,retVal;
char *string;
char *formatStr;
char *formatStrLeftJus;
char *addressChar[20];
int i,key;
double minAllowedRange,maxAllowedRange;
int row=4;
int column;
int xOfColumn[4]={8,27,44,62};
int maxlen;
int* xOfColumnPtr;
double *tempInputPtr;

char *prompt=
"Change <C>  Previous <P>  Save data <S>  Run <R>  Quit <Q> ==>";
mode=BALLSIZEOPT;
xOfColumnPtr=&xOfColumn[0]-1;
clrscr();
```

```
PrintText(12,1,"Selection Function Data for Ball Size Optimization Program",
WHITE);

gotoxy(1,3);
cprintf("Class");
gotoxy(xOfColumnPtr[1],3);
cprintf("Screen Size (μm)");
gotoxy(xOfColumnPtr[2],3);
cprintf("Particle Size");
gotoxy(xOfColumnPtr[3],3);
cprintf("Selec. Function");
gotoxy(xOfColumnPtr[4],3);
cprintf("Standard Deviation");
gotoxy(5,4);
for(i=1;i<=sizeClassNumBallSzOpt;i++){
        gotoxy(3,i+3);
        cprintf("%2d",i);
        gotoxy(xOfColumnPtr[1],i+3);
        cprintf("%10.0f",ptrScreenSizeBallSzOpt[i-1]);
        gotoxy(xOfColumnPtr[2],i+3);
        cprintf("%10.0f",ptrParticleSize[i-1]);
        gotoxy(xOfColumnPtr[3],i+3);
        cprintf("%15.6f",ptrSelecFuncBallSzOpt[i-1]);
        gotoxy(xOfColumnPtr[4],i+3);
        cprintf("%15.6f",ptrStandardDevBallSzOpt[i-1]);
}
done=FALSE;
while(!done){
        PrintText(2,24,prompt,YELLOW);
        key=getch();
        switch(key){
                case 'C':
                case 'c':
                column=1;
                maxlen=15;
                formatStr="%10.0lf";
                formatStrLeftJus="%-10.0lf";
                minAllowedRange=0;
                maxAllowedRange=1e6;
                tempInputPtr=ptrScreenSizeBallSzOpt-1;
                GetColOfData(row,column,sizeClassNumBallSzOpt,xOfColumnPtr,maxlen,

formatStr,formatStrLeftJus,minAllowedRange,maxAllowedRange,
                                                        tempInputPtr);
                PrintText(2,24,"Set particle sizes to geo-means? [y/n] ",YELLOW);
                clreol();
                while(!done){
                        string=GetStringAt(41,24,4);
                        if(string[0]=='Y'||string[0]=='y'){
                                for(i=1;i<=sizeClassNumBallSzOpt;i++){
```

```
ptrParticleSize[i-1]=ptrScreenSizeBallSzOpt[i-1]*FORTH_ROOT_OF_2;
                                        gotoxy(xOfColumnPtr[2],i+3);
                                        cprintf("%10.0f",ptrParticleSize[i-1]);
                                }
                                done=TRUE;
                        }
                        else if(string[0]=='N'||string[0]=='n')done=TRUE;
                        gotoxy(41,24);
                        clreol();
                }
                done=FALSE;
                PrintText(2,24,prompt,YELLOW);
                clreol();
                column=2;
                maxlen=15;
                formatStr="%10.0lf";
                formatStrLeftJus="%-10.0lf";
                minAllowedRange=0;
                maxAllowedRange=1e6;
                tempInputPtr=ptrParticleSize-1;
                GetColOfData(row,column,sizeClassNumBallSzOpt,xOfColumnPtr,maxlen,

formatStr,formatStrLeftJus,minAllowedRange,maxAllowedRange,
                                                                tempInputPtr);
                column=3;
                maxlen=15;
                formatStr="%10.4lf";
                formatStrLeftJus="%-10.4lf";
                minAllowedRange=EPS;
                maxAllowedRange=1e6;
                tempInputPtr=ptrSelecFuncBallSzOpt-1;
                GetColOfData(row,column,sizeClassNumBallSzOpt,xOfColumnPtr,

maxlen,formatStr,formatStrLeftJus,minAllowedRange,
                                                maxAllowedRange,tempInputPtr);

                column=4;
                maxlen=15;
                formatStr="%15.4lf";
                formatStrLeftJus="%-15.4lf";
                minAllowedRange=EPS;
                maxAllowedRange=1e3;
                tempInputPtr=ptrStandardDevBallSzOpt-1;
                GetColOfData(row,column,sizeClassNumBallSzOpt,xOfColumnPtr,

maxlen,formatStr,formatStrLeftJus,minAllowedRange,
                                        maxAllowedRange,tempInputPtr);
                break;

                case 'P':
                case 'p':
```

```
                retVal=2;
                done=TRUE;
                break;

                case 'S':
                case 's':
                SaveBallSzOptData();
                retVal=3;
                done=TRUE;
                break;

                case 'R':
                case 'r':
                done=TRUE;
                for(i=1;i<=sizeClassNumBallSzOpt;i++){
                        if(LE(ptrScreenSizeBallSzOpt[i-1],0)){
                                gettext(1,10,80,15,savescr);

PrintText(8,11,"┌──────────────────────────────────────────────────────────────────────────────
──────────────────────────────┐ ",WHITE);
                                PrintText(8,12,"|    A screen size cannot be set to zero or
negative    | ",WHITE);

PrintText(8,13,"└──────────────────────────────────────────────────────────────────────────────
──────────────────────────────┘ ",WHITE);
                                _setcursortype(_NOCURSOR);
                                getch();
                                gotoxy(20,10);
                                puttext(1,10,80,15,savescr);
                                _setcursortype(_NORMALCURSOR);
                                gotoxy(41,24);
                                done=FALSE;
                                break;
                        }
                        if(LE(ptrParticleSize[i-1],0)){
                                gettext(1,10,80,15,savescr);

PrintText(8,11,"┌──────────────────────────────────────────────────────────────────────────────
──────────────────────────────┐ ",WHITE);
                                PrintText(8,12,"|    A particle size cannot be set to zero or
negative    | ",WHITE);

PrintText(8,13,"└──────────────────────────────────────────────────────────────────────────────
──────────────────────────────┘ ",WHITE);
                                _setcursortype(_NOCURSOR);
                                getch();
                                gotoxy(20,10);
                                puttext(1,10,80,15,savescr);
                                _setcursortype(_NORMALCURSOR);
                                gotoxy(41,24);
                                done=FALSE;
```

```
                        break;
                    }
                    if(LE(ptrSelecFuncBallSzOpt[i-1],EPS)){
                        gettext(1,10,80,15,savescr);

PrintText(8,11," ┌──────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────┐ ",WHITE);
                        PrintText(8,12," |    A selection function cannot be set to zero or
negative    | ",WHITE);

PrintText(8,13," └──────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────┘ ",WHITE);
                        _setcursortype(_NOCURSOR);
                        getch();
                        gotoxy(20,10);
                        puttext(1,10,80,15,savescr);
                        _setcursortype(_NORMALCURSOR);
                        gotoxy(41,24);
                        done=FALSE;
                        break;
                    }
                }
                if(done) retVal=4;
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
            }
            for(i=1;i<=sizeClassNumBallSzOpt;i++){
                /* index 0 <-> index 10 */
                ptrXseries[i-1]=ptrScreenSizeBallSzOpt[sizeClassNumBallSzOpt-i]*
                                                    FORTH_ROOT_OF_2;
                ptrCurSelecFunc[i-1]=ptrSelecFuncBallSzOpt[sizeClassNumBallSzOpt-i];
            }
*ptrXb=ptrXseries[0];
*ptrXe=ptrXseries[sizeClassNumBallSzOpt-1];
for(i=1;i<=sizeClassNumBallSzOpt;i++) ptr_w1[i-1]=1/ptrStandardDevBallSzOpt[i-1];
}
return retVal;
}
```

```
/* bsoptrun.c01 */
#include <math.h>
#include <float.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <string.h>
#include <graphics.h>
#include <dos.h>
#include "global.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

int ShowBallSizeOptMenu(void);
void CalcNewSelecFunc(void);
extern int GraphMode;              /* The Graphics mode value              */
extern enum operationmode mode;
extern double *ptrPosKnots,*ptrPosKnots1,*ptrPosKnots2,*ptr_wrk,
                                   *ptr_c,*ptr_c1,*ptr_c2,*ptr_w,*ptr_w1,*ptr_w2;
extern double *ptrXseries,*ptrSelecFunc,*ptrCurSelecFunc,*ptrNewSelecFunc,
                                   *ptrFtdSelecFunc,*ptrFtdCurSelecFunc,*ptrFtdNewSelecFunc;
extern int *ptrK,k1,k2,*ptr_iopt,*ptr_nest,*ptr_n,*ptr_n1,*ptr_n2,*ptr_lwrk,
                              *ptr_ier,*ptr_iwrk;

extern double *ptrXb,*ptrXe,*ptrIniS,iniS1,iniS2,*ptrS,*ptrDifS,
                                        difS1,difS2,*ptr_fp,*ptr_fp1,*ptr_fp2;

extern double curBallDia,*ptrCurBallDia;
extern double newBallDia,*ptrNewBallDia;

extern double *ptrScreenSizeBallSzOpt,*ptrSelecFuncBallSzOpt,
                                   *ptrStandardDevBallSzOpt;
extern double *ptrFactorK;
extern char logX[4];
extern char logY[4];
extern double *ptr_fin_s,*ptr_fin_s1,*ptr_fin_s2;

int SetParamOnLine(bool isSecondGraph);
void InitializeGraphics(void);
void TerminateGraphics(void);
void PrintText(int x,int y,char* text,int forgcolor);
int PlotScatterDig(double xDataVec[],double yDataVec[],unsigned long m,
                                                  char* title,char* xLab,char* yLab);
int PlotGraph(double (*ptrFunc) (double *ptr_xCo),double a,double b,
                                          char* title,char* xLab,char* yLab);
int PlotGraphBoth(double (*fPtr) (double* x),double a,double b,char* title,
```

```
                                                    char* xLab,char* yLab);
void splev_(double *t,int *n,double *c__,int *k,double *x,double *y,
                                        int *m,int *ier);
int WrtToScrBallSzOptOutput(void);
int WrtToPrnBallSzOptOutput(void);
int WrtToFileBallSzOptOutput(void);
double Function(double *ptr_xCo);
typedef double (*ptrFunc) (double *ptr_xCo);
ptrFunc ptr_fx;
double xOfMaxCurSelecFunc,xOfMaxNewSelecFunc,*ptr_xOfMaxSelecFunc;
char *ptrsGrTitle,*ptrsXaxis,*ptrsYaxis;
bool isSecondGraph;

int BallSizeOptRun(void){
char *prompt="Print<P> Save result <S> Export<E> Back <B> Quit <Q> ";
int i,key,retVal;
double xDot,dx,y,ySmlFunc,yBigFunc;
bool done;

/* calculating new selection functions using Morrell's approach */
CalcNewSelecFunc();
/* now, setting up graphics to manually fit splines to both current
        and new selection functions */
ptrsGrTitle=(char *) malloc(81);
ptrsXaxis=(char *) malloc(81);
ptrsYaxis=(char *) malloc(81);
/* first fitting a spline curve to the current (known) selection function */
strcpy(ptrsGrTitle,"Spline Curve Fitting To The Known Selection Function Data");
strcpy(ptrsXaxis,"Particle Size, X");
strcpy(ptrsYaxis,"Selection Function, S");
if(logX[0]=='Y'||logX[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                ptrXseries[i-1]=log10(ptrXseries[i-1]);
        xOfMaxCurSelecFunc=log10(xOfMaxCurSelecFunc);
        xOfMaxNewSelecFunc=log10(xOfMaxNewSelecFunc);
        *ptrXb=log10(*ptrXb);
        *ptrXe=log10(*ptrXe);
        strcpy(ptrsXaxis,"Particle Size, log(X)");
}
if(logY[0]=='Y'||logY[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++){
                ptrCurSelecFunc[i-1]=log10(ptrCurSelecFunc[i-1]);
                ptrNewSelecFunc[i-1]=log10(ptrNewSelecFunc[i-1]);
        }
strcpy(ptrsYaxis,"Selection Function, log(S)");
}
ptr_fx=Function;
InitializeGraphics();
ptrK=&k1;
ptrIniS=&iniS1;
ptrDifS=&difS1;
```

```
ptrSelecFunc=ptrCurSelecFunc;
ptrFtdSelecFunc=ptrFtdCurSelecFunc;
ptr_w=ptr_w1;
ptrPosKnots=ptrPosKnots1;
ptr_n=ptr_n1;
ptr_c=ptr_c1;
ptr_fp=ptr_fp1;
ptr_fin_s=ptr_fin_s1;
ptr_xOfMaxSelecFunc=&xOfMaxCurSelecFunc;
isSecondGraph=FALSE;
done=FALSE;
while(!done){
        retVal=PlotGraph(ptr_fx,*ptrXb,*ptrXe,ptrsGrTitle,ptrsXaxis,ptrsYaxis);
        if(retVal==0) done=TRUE;
        else{
                restorecrtmode();
                SetParamOnLine(isSecondGraph);
                setgraphmode(GraphMode);
                ptrK=&k1;
                ptrIniS=&iniS1;
                ptrDifS=&difS1;
        }
}
/* reset spline fitting options to default values */
ptrK=&k2;
ptrIniS=&iniS2;
ptrDifS=&difS2;
ptrSelecFunc=ptrNewSelecFunc;
ptrFtdSelecFunc=ptrFtdNewSelecFunc;
ptr_w=ptr_w2;
ptrPosKnots=ptrPosKnots2;
ptr_n=ptr_n2;
ptr_c=ptr_c2;
ptr_fp=ptr_fp2;
ptr_fin_s=ptr_fin_s2;
ptr_xOfMaxSelecFunc=&xOfMaxNewSelecFunc;
strcpy(ptrsGrTitle,"Spline Curve Fitting To The Scaled Selection Function Data");
isSecondGraph=TRUE;
done=FALSE;
while(!done){
        retVal=PlotGraph(ptr_fx,*ptrXb,*ptrXe,ptrsGrTitle,ptrsXaxis,ptrsYaxis);
        if(retVal==0) done=TRUE;
        else{
                restorecrtmode();
                SetParamOnLine(isSecondGraph);
                setgraphmode(GraphMode);
                ptrK=&k2;
                ptrIniS=&iniS2;
                ptrDifS=&difS2;
        }
}
```

```
strcpy(ptrsGrTitle,"Fitted Spline Curves To Selection Functions");
PlotGraphBoth(ptr_fx,*ptrXb,*ptrXe,ptrsGrTitle,ptrsXaxis,ptrsYaxis);
TerminateGraphics();
WrtToScrBallSzOptOutput();
gotoxy(1,24);
textcolor(YELLOW);
cprintf(prompt);
clreol();
textcolor(LIGHTGRAY);
done=FALSE;
while(!done){
        key=getch();
        switch(key){
                case 'P':
                case 'p':
                WrtToPrnBallSzOptOutput();
                PrintText(1,24,"The output was sent to the printer...",        YELLOW);
                clreol();
                delay(700);
                gotoxy(1,24);
                textcolor(YELLOW);
                cprintf(prompt);
                clreol();
                textcolor(LIGHTGRAY);
                break;

                case 'S':
                case 's':
                WrtToFileBallSzOptOutput();
                gotoxy(1,24);
                textcolor(YELLOW);
                cprintf(prompt);
                clreol();
                textcolor(LIGHTGRAY);
                break;

                case 'E':
                case 'e':
                sizeClassNumSimGrCir=sizeClassNumBallSzOpt;
                for(i=1;i<=sizeClassNumBallSzOpt;i++)
                        screenSize[i]=ptrScreenSizeBallSzOpt[i-1];
                if(logY[0]=='Y'||logY[0]=='y')
                        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                                ptrFtdNewSelecFunc[i-1]=pow(10,ptrFtdNewSelecFunc[i-1]);
                for(i=1;i<=sizeClassNumBallSzOpt;i++)
                        selectionFunction[i]=ptrFtdNewSelecFunc[sizeClassNumBallSzOpt-i];
                PrintText(1,24,
                "Calculated selection functions were exported, press any key to continue ...",
                YELLOW);
                clreol();
                getch();
```

```
                PrintText(1,24,prompt,YELLOW);
                clreol();
                textcolor(LIGHTGRAY);
                break;

                case 'B':
                case 'b':
                retVal=1;
                done=TRUE;
                break;

                case 'Q':
                case 'q':
                retVal=-1000;
                done=TRUE;
                break;
        }
}
free(ptrsGrTitle);
free(ptrsXaxis);
free(ptrsYaxis);
return retVal;
}

double Function(double *ptr_xCo){
int oneOnly;

double yCo=0.0;
oneOnly=1;
splev_(ptrPosKnots,ptr_n,ptr_c,ptrK,ptr_xCo,&yCo,&oneOnly,ptr_ier);
return yCo;
}

void CalcNewSelecFunc(void){
int i;

xOfMaxCurSelecFunc=*ptrFactorK*curBallDia*curBallDia*1000;
xOfMaxNewSelecFunc=*ptrFactorK*newBallDia*newBallDia*1000;
if(GT(curBallDia,newBallDia)){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                if(LE(ptrXseries[i-1],xOfMaxNewSelecFunc)){
                        ptrNewSelecFunc[i-1]=ptrCurSelecFunc[i-1]*(curBallDia/newBallDia);
                        ptr_w2[i-1]=ptr_w1[i-1];
                }
                else    if(GE(ptrXseries[i-1],xOfMaxCurSelecFunc)){
                                ptrNewSelecFunc[i-1]=ptrCurSelecFunc[i-1]*

(newBallDia/curBallDia)*

(newBallDia/curBallDia);
                                ptr_w2[i-1]=ptr_w1[i-1];
```

```
                                }
                                else{
                                        ptrNewSelecFunc[i-1]=2*EPS;
                                        ptr_w2[i-1]=EPS;
                                }
                        }
if(LT(curBallDia,newBallDia)){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                if(GE(ptrXseries[i-1],xOfMaxNewSelecFunc)){
                        ptrNewSelecFunc[i-1]=ptrCurSelecFunc[i-1]*

(newBallDia/curBallDia)*

(newBallDia/curBallDia);
                        ptr_w2[i-1]=ptr_w1[i-1];
                }
                else if(LE(ptrXseries[i-1],xOfMaxCurSelecFunc)){
                                        ptrNewSelecFunc[i-1]=ptrCurSelecFunc[i-1]*

(curBallDia/newBallDia);
                                        ptr_w2[i-1]=ptr_w1[i-1];
                                }
                                else{
                                        ptrNewSelecFunc[i-1]=2*EPS;
                                        ptr_w2[i-1]=EPS;
                                }
                        }
if(EQ(curBallDia,newBallDia)){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                ptrNewSelecFunc[i-1]=ptrCurSelecFunc[i-1];
ptr_w2=ptr_w1;
}
}
```

```
/* savebsod.c01 */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "global.h"

extern double *ptrParticleSize;
extern double curBallDia,newBallDia;
extern char logX[4],logY[4];

int SaveBallSzOptData(void){
short i,j;
FILE *outputStreamPtr;
char ch;
char *filename;

gotoxy(1,24);
clreol();
textcolor(YELLOW);
cprintf("Save as: ");
textcolor(LIGHTGRAY);
scanf("%128s",filename);
strcat(filename,".bsd");
if((outputStreamPtr=fopen(filename,"w"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't creat output file! Press any key to continue");
        textcolor(LIGHTGRAY);
        getch();
        return 0;
}
fprintf(outputStreamPtr,"%s\n",projectTitleBallSzOpt);
fprintf(outputStreamPtr,"%d\n",sizeClassNumBallSzOpt);
fprintf(outputStreamPtr,"%5.1f %5.1f\n",curBallDia,newBallDia);
for(i=1;i<=sizeClassNumBallSzOpt;i++)
        fprintf(outputStreamPtr,"%10.1f %10.1f %10.6f %10.6f\n",
                        ptrScreenSizeBallSzOpt[i-1],ptrParticleSize[i-1],
                        ptrSelecFuncBallSzOpt[i-1],ptrStandardDevBallSzOpt[i-1]);
fprintf(outputStreamPtr,"%10.6f %d %10.6f %10.6f\n",*ptrFactorK,
                *ptrK,*ptrIniS,*ptrDifS);
fprintf(outputStreamPtr,"%c  %c",logX[0],logY[0]);
if(fclose(outputStreamPtr)!=0){
        gotoxy(1,24);
        cprintf("Error in closing output file! Press any key to continue");
        getch();
        return 1;
}
return 0;
}
```

```
/* wrtscrbs.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <float.h>
#include "global.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

extern char logX[4],logY[4];
extern double curBallDia;
extern double newBallDia;
extern double *ptrXseries,*ptrSelecFunc,*ptrCurSelecFunc,*ptrNewSelecFunc,
                                        *ptrFtdCurSelecFunc,*ptrFtdNewSelecFunc;
extern double xOfMaxCurSelecFunc,xOfMaxNewSelecFunc,*ptrXb,*ptrXe;
extern double *ptr_fp1,*ptr_fp2;

int WrtToScrBallSzOptOutput(void){
int i,j;
char ch;
struct date today;

if(logX[0]=='Y'||logX[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                ptrXseries[i-1]=pow(10,ptrXseries[i-1]);
        xOfMaxCurSelecFunc=pow(10,xOfMaxCurSelecFunc);
        *ptrXb=pow(10,*ptrXb);
        *ptrXe=pow(10,*ptrXe);
}
if(logY[0]=='Y'||logY[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++){
                ptrCurSelecFunc[i-1]=pow(10,ptrCurSelecFunc[i-1]);
                ptrNewSelecFunc[i-1]=pow(10,ptrNewSelecFunc[i-1]);
                ptrFtdCurSelecFunc[i-1]=pow(10,ptrFtdCurSelecFunc[i-1]);
                ptrFtdNewSelecFunc[i-1]=pow(10,ptrFtdNewSelecFunc[i-1]);
        }
        xOfMaxNewSelecFunc=pow(10,xOfMaxNewSelecFunc);
}
clrscr();
gotoxy(15,1);
textbackground(LIGHTGREEN);
textcolor(WHITE);
cprintf("                                        ");
gotoxy(15,2);
```

```
cprintf("          Selection Function Scaling Results          ");
gotoxy(15,3);
cprintf("                                                      ");
textbackground(BLACK);
textcolor(LIGHTGRAY);
gotoxy(1,4);
cprintf(projectTitleBallSzOpt);
gotoxy(1,5);
getdate((struct date*) &today);
cprintf("Date: %d/%d/%d\r\n",today.da_mon,today.da_day,today.da_year);
cprintf("Current ball size = %.2f mm.\r\n",curBallDia);
cprintf("New ball size = %.2f mm.\r\n",newBallDia);
cprintf("K = %f (1/mm.)\r\n",*ptrFactorK);
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
gotoxy(15,1);
textbackground(LIGHTGREEN);
textcolor(WHITE);
cprintf("                                                      ");
gotoxy(15,2);
textcolor(WHITE);
cprintf("      Selection Function Scaling Results, Cont'd      ");
gotoxy(15,3);
cprintf("                                                      ");
textbackground(BLACK);
textcolor(LIGHTGRAY);
gotoxy(1,4);
cprintf("\
┌──────┬──────────────┬──────────────┬──────────────┬──────────────┬──────────────┐\r\n\
| CLASS | SCREEN SIZE | PARTICLE SIZE | EST.SEL.FUNC. | STD.DEV. | SCALED SEL.FUNC. | \r\n\
├──────┼──────────────┼──────────────┼──────────────┼──────────────┼──────────────┤\r\n");
for(i=1;i<=sizeClassNumBallSzOpt;i++)
        cprintf(" | %4d | %10.0f | %10.0f | %12.4f | %9.4f | %15.4f | \r\n",i,
                        ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i],
                        ptrCurSelecFunc[sizeClassNumBallSzOpt-i],
                        ptrStandardDevBallSzOpt[i-1],
                        ptrNewSelecFunc[sizeClassNumBallSzOpt-i]);
cprintf(
"\
└──────┴──────────────┴──────────────┴──────────────┴──────────────┴──────────────┘");
cprintf("\r\n");
gotoxy(1,24);
```

```
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
gotoxy(15,1);
textbackground(LIGHTGREEN);
textcolor(WHITE);
cprintf("                                                    ");
gotoxy(15,2);
cprintf("          Spline Curve Fitting Results              ");
gotoxy(15,3);
cprintf("                                                    ");
gotoxy(1,4);
textbackground(BLACK);
textcolor(LIGHTGRAY);
cprintf("Weighted SSR for the first curve: %f  \r\n",*ptr_fp1);
cprintf("Weighted SSR for the second curve: %f \r\n",*ptr_fp2);
gotoxy(1,24);
textcolor(YELLOW);
cprintf("Press any key to continue ...");
clreol();
textcolor(LIGHTGRAY);
getch();
clrscr();
gotoxy(15,1);
textbackground(LIGHTGREEN);
textcolor(WHITE);
cprintf("                                                    ");
gotoxy(15,2);
textcolor(WHITE);
cprintf("       Spline Curve Fitting Results, Cont'd         ");
gotoxy(15,3);
cprintf("                                                    ");
textbackground(BLACK);
textcolor(LIGHTGRAY);
gotoxy(1,4);
cprintf("\
┌──────┬──────────┬──────────────┬──────────────┬──────────────┐\r\n\
| CLASS | SCREEN SIZE | PARTICLE SIZE | CALC.EST.SEL.FUNC. | CALC.SCALED
SEL.FUNC. | \r\n\
├──────┼──────────┼──────────────┼──────────────┼──────────────┤\r\n");
for(i=1;i<=sizeClassNumBallSzOpt;i++)
        if(GT(ptrFtdCurSelecFunc[sizeClassNumBallSzOpt-i],1e9)&&
                LT(ptrFtdNewSelecFunc[sizeClassNumBallSzOpt-i],1e9))
        cprintf("\
| %4d | %8.0f   | %10.0f   | ***************  | %15.4f      | \r\n",i,
                ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i],
```

```
                ptrFtdNewSelecFunc[sizeClassNumBallSzOpt-i]);
        else            if(LT(ptrFtdCurSelecFunc[sizeClassNumBallSzOpt-i],1e9)&&
                GT(ptrFtdNewSelecFunc[sizeClassNumBallSzOpt-i],1e9))
        cprintf("\
| %4d | %8.0f   | %10.0f   | %15.4f   |  ****************   | \r\n",i,
                ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i],
                ptrFtdCurSelecFunc[sizeClassNumBallSzOpt-i]);

        else    if(GT(ptrFtdCurSelecFunc[sizeClassNumBallSzOpt-i],1e9)&&
                GT(ptrFtdNewSelecFunc[sizeClassNumBallSzOpt-i],1e9))
        cprintf("\
| %4d | %8.0f   | %10.0f   | **************   |  ****************   | \r\n",i,
                ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i]);

        else
        cprintf("\
| %4d | %8.0f   | %10.0f   | %15.4f   | %15.4f      | \r\n",i,
                        ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i],
                        ptrFtdCurSelecFunc[sizeClassNumBallSzOpt-i],
                        ptrFtdNewSelecFunc[sizeClassNumBallSzOpt-i]);
cprintf(
"\
└──────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘ ");
cprintf("\r\n");
if(logX[0]=='Y'||logX[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                ptrXseries[i-1]=log10(ptrXseries[i-1]);
        xOfMaxCurSelecFunc=log10(xOfMaxCurSelecFunc);
        *ptrXb=log10(*ptrXb);
        *ptrXe=log10(*ptrXe);
}
if(logY[0]=='Y'||logY[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++){
                ptrCurSelecFunc[i-1]=log10(ptrCurSelecFunc[i-1]);
                ptrNewSelecFunc[i-1]=log10(ptrNewSelecFunc[i-1]);
                ptrFtdCurSelecFunc[i-1]=log10(ptrFtdCurSelecFunc[i-1]);
                ptrFtdNewSelecFunc[i-1]=log10(ptrFtdNewSelecFunc[i-1]);
        }
        xOfMaxNewSelecFunc=log10(xOfMaxNewSelecFunc);
}
return 0;
}
```

```
/* wrtprnbs.c01 */
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include "global.h"

extern char logX[4],logY[4];
extern double curBallDia;
extern double newBallDia;
extern double *ptrXseries,*ptrSelecFunc,*ptrCurSelecFunc,*ptrNewSelecFunc,
                                        *ptrFtdCurSelecFunc,*ptrFtdNewSelecFunc;
extern double xOfMaxCurSelecFunc,xOfMaxNewSelecFunc,*ptrXb,*ptrXe;
extern double *ptr_fp1,*ptr_fp2;

int WrtToPrnBallSzOptOutput(void);

int WrtToPrnBallSzOptOutput(void){
int i,j;
char ch;
struct date today;

if(logX[0]=='Y'||logX[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                ptrXseries[i-1]=pow(10,ptrXseries[i-1]);
        xOfMaxCurSelecFunc=pow(10,xOfMaxCurSelecFunc);
        *ptrXb=pow(10,*ptrXb);
        *ptrXe=pow(10,*ptrXe);
}
if(logY[0]=='Y'||logY[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++){
                ptrCurSelecFunc[i-1]=pow(10,ptrCurSelecFunc[i-1]);
                ptrNewSelecFunc[i-1]=pow(10,ptrNewSelecFunc[i-1]);
                ptrFtdCurSelecFunc[i-1]=pow(10,ptrFtdCurSelecFunc[i-1]);
                ptrFtdNewSelecFunc[i-1]=pow(10,ptrFtdNewSelecFunc[i-1]);
        }
        xOfMaxNewSelecFunc=pow(10,xOfMaxNewSelecFunc);
}
fprintf(stdprn,"\r\n\r\n"
"                ***********************************************\r\n");
fprintf(stdprn,
"                    Selection Function Scaling Results     \r\n");
fprintf(stdprn,
"                ***********************************************\r\n");
fprintf(stdprn,projectTitleBallSzOpt);
fprintf(stdprn,"\r\n");
getdate((struct date*) &today);
fprintf(stdprn,"Date: %d/%d/%d\r\n",today.da_mon,today.da_day,today.da_year);
fprintf(stdprn,"Current ball size = %.2f mm.\r\n",curBallDia);
fprintf(stdprn,"New ball size = %.2f mm.\r\n",newBallDia);
fprintf(stdprn,"K = %f (1/mm.)\r\n",*ptrFactorK);
fprintf(stdprn,"\
```

```
----------\r\n\
|CLASS|SCREEN SIZE|PARTICLE SIZE|EST.SEL.FUNC.| STD.DEV. |SCALED
SEL.FUNC.|\r\n\
----------\r\n");
for(i=1;i<=sizeClassNumBallSzOpt;i++)
        fprintf(stdprn,
"|%4d |%10.0f |%10.0f  |%12.4f |%9.4f |%15.4f |\r\n",i,
ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i],
ptrCurSelecFunc[sizeClassNumBallSzOpt-i],
ptrStandardDevBallSzOpt[i-1],
ptrNewSelecFunc[sizeClassNumBallSzOpt-i]);
fprintf(stdprn,
"\
----------\r\n");
fprintf(stdprn,
"               **********************************************\r\n");
fprintf(stdprn,
"                       Spline Curve Fitting Results      \r\n");
fprintf(stdprn,
"               **********************************************\r\n");
fprintf(stdprn,"Weighted SSR for the first curve: %f \r\n",*ptr_fp1);
fprintf(stdprn,"Weighted SSR for the second curve: %f \r\n",*ptr_fp2);
fprintf(stdprn,"\
----------\r\n\
|CLASS|SCREEN SIZE|PARTICLE SIZE|CALC.EST.SEL.FUNC.|CALC.SCALED
SEL.FUNC.|\r\n\
----------\r\n");
for(i=1;i<=sizeClassNumBallSzOpt;i++)
        fprintf(stdprn,"|%4d |%8.0f  |%10.0f  |%15.4f  |%15.4f      |\r\n",i,
                        ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i],
                        ptrFtdCurSelecFunc[sizeClassNumBallSzOpt-i],
                        ptrFtdNewSelecFunc[sizeClassNumBallSzOpt-i]);
fprintf(stdprn,
"\
----------\r\n");
fprintf(stdprn,"\r\n\r\n");
fprintf(stdprn,"\f");
if(logX[0]=='Y'||logX[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                ptrXseries[i-1]=log10(ptrXseries[i-1]);
        xOfMaxCurSelecFunc=log10(xOfMaxCurSelecFunc);
        *ptrXb=log10(*ptrXb);
        *ptrXe=log10(*ptrXe);
}
if(logY[0]=='Y'||logY[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++){
                ptrCurSelecFunc[i-1]=log10(ptrCurSelecFunc[i-1]);
                ptrNewSelecFunc[i-1]=log10(ptrNewSelecFunc[i-1]);
                ptrFtdCurSelecFunc[i-1]=log10(ptrFtdCurSelecFunc[i-1]);
                ptrFtdNewSelecFunc[i-1]=log10(ptrFtdNewSelecFunc[i-1]);
        }
```

```
        xOfMaxNewSelecFunc=log10(xOfMaxNewSelecFunc);
}
return 0;
}
```

```
/* wrtfbso.c01 */
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <math.h>
#include "global.h"

extern char logX[4],logY[4];
extern double curBallDia;
extern double newBallDia;
extern double *ptrXseries,*ptrSelecFunc,*ptrCurSelecFunc,*ptrNewSelecFunc,
                                        *ptrFtdCurSelecFunc,*ptrFtdNewSelecFunc;
extern double xOfMaxCurSelecFunc,xOfMaxNewSelecFunc,*ptrXb,*ptrXe;
extern double *ptr_fp1,*ptr_fp2;

int WrtToFileBallSzOptOutput(void);

int WrtToFileBallSzOptOutput(void){
short i,j;
FILE *OutputStream;
char ch;
struct date today;
char *filename;

gotoxy(1,24);
clreol();
textcolor(YELLOW);
cprintf("Save as: ");
textcolor(LIGHTGRAY);
scanf("%128s",filename);
if((OutputStream=fopen(filename,"w"))==NULL){
        gotoxy(1,24);
        textcolor(LIGHTRED);
        cprintf("Can't creat output file! Press any key to continue ...");
        textcolor(LIGHTGRAY);
        getch();
        return 1;
}
/* write some data to the file */
if(logX[0]=='Y'||logX[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                ptrXseries[i-1]=pow(10,ptrXseries[i-1]);
        xOfMaxCurSelecFunc=pow(10,xOfMaxCurSelecFunc);
        *ptrXb=pow(10,*ptrXb);
        *ptrXe=pow(10,*ptrXe);
}
if(logY[0]=='Y'||logY[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++){
                ptrCurSelecFunc[i-1]=pow(10,ptrCurSelecFunc[i-1]);
                ptrNewSelecFunc[i-1]=pow(10,ptrNewSelecFunc[i-1]);
```

```
                ptrFtdCurSelecFunc[i-1]=pow(10,ptrFtdCurSelecFunc[i-1]);
                ptrFtdNewSelecFunc[i-1]=pow(10,ptrFtdNewSelecFunc[i-1]);
        }
        xOfMaxNewSelecFunc=pow(10,xOfMaxNewSelecFunc);
}
fprintf(OutputStream,"\n\n"
"          ************************************************************\n");
fprintf(OutputStream,
"                    Selection Function Scaling Results          \n");
fprintf(OutputStream,
"          ************************************************************\n");
fprintf(OutputStream,projectTitleBallSzOpt);
fprintf(OutputStream,"\n");
getdate((struct date*) &today);
fprintf(OutputStream,"Date: %d/%d/%d\n",today.da_mon,today.da_day,today.da_year);
fprintf(OutputStream,"Current ball size = %.2f mm.\n",curBallDia);
fprintf(OutputStream,"New ball size = %.2f mm.\n",newBallDia);
fprintf(OutputStream,"K = %f (1/mm.)\n",*ptrFactorK);
fprintf(OutputStream,"\
-------------------------------------------------------------------\n\
|CLASS|SCREEN SIZE|PARTICLE SIZE|EST.SEL.FUNC.| STD.DEV. |SCALED
SEL.FUNC.|\n\
-------------------------------------------------------------------\n");
for(i=1;i<=sizeClassNumBallSzOpt;i++)
        fprintf(OutputStream,"|%4d |%10.0f |%10.0f  |%12.4f |%9.4f |%15.4f |\n",i,
                        ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i],
                        ptrCurSelecFunc[sizeClassNumBallSzOpt-i],
                        ptrStandardDevBallSzOpt[i-1],
                        ptrNewSelecFunc[sizeClassNumBallSzOpt-i]);
fprintf(OutputStream,
"\
-------------------------------------------------------------------\n");
fprintf(OutputStream,"\n\n"
"          ************************************************************\n");
fprintf(OutputStream,
"                    Spline Curve Fitting Results          \n");
fprintf(OutputStream,
"          ************************************************************\n");
fprintf(OutputStream,"Weighted SSR for the first curve: %f \n",*ptr_fp1);
fprintf(OutputStream,"Weighted SSR for the second curve: %f \n",*ptr_fp2);
fprintf(OutputStream,"\
-------------------------------------------------------------------\n\
|CLASS|SCREEN SIZE|PARTICLE SIZE|CALC.EST.SEL.FUNC.|CALC.SCALED
SEL.FUNC.|\n\
-------------------------------------------------------------------\n");
for(i=1;i<=sizeClassNumBallSzOpt;i++)
        fprintf(OutputStream,"|%4d |%8.0f  |%10.0f  |%15.4f  |%15.4f     |\n",i,
                        ptrScreenSizeBallSzOpt[i-1],ptrXseries[sizeClassNumBallSzOpt-i],
                        ptrFtdCurSelecFunc[sizeClassNumBallSzOpt-i],
                        ptrFtdNewSelecFunc[sizeClassNumBallSzOpt-i]);
fprintf(OutputStream,
```

```
"\
-------------------------------------------------------------------------\n");
if(logX[0]=='Y'||logX[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++)
                ptrXseries[i-1]=log10(ptrXseries[i-1]);
        xOfMaxCurSelecFunc=log10(xOfMaxCurSelecFunc);
        *ptrXb=log10(*ptrXb);
        *ptrXe=log10(*ptrXe);
}
if(logY[0]=='Y'||logY[0]=='y'){
        for(i=1;i<=sizeClassNumBallSzOpt;i++){
                ptrCurSelecFunc[i-1]=log10(ptrCurSelecFunc[i-1]);
                ptrNewSelecFunc[i-1]=log10(ptrNewSelecFunc[i-1]);
                ptrFtdCurSelecFunc[i-1]=log10(ptrFtdCurSelecFunc[i-1]);
                ptrFtdNewSelecFunc[i-1]=log10(ptrFtdNewSelecFunc[i-1]);
        }
        xOfMaxNewSelecFunc=log10(xOfMaxNewSelecFunc);
}
/* close the file */
if(fclose(OutputStream)!=0){
        gotoxy(1,24);
        cprintf("Error in closing output file! Press any key to continue");
        getch();
        return 1;
}
return 0;
}
```

```
/* getrtd.c01 */-
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include "global.h"
#include "gets.h"

void PrintErrMsg(void);

int GetRtdInfo(double *ptrTauPF,double *ptrTauSPM,double *ptrTauLPM,
                                        double *ptrRefMillFeedRate,double *ptrCurMillFeedRate)
{
int i,row,key,status,retVal;
bool done=FALSE;
bool canUseCurValue=TRUE;
const float minAllowedValue=0.0;
double *address;
char *string;

gotoxy(45,6);
row=6;
for(row=6;row<11;row++){
        switch(row){
                case 6:
                address=ptrTauPF;
                break;

                case 7:
                address=ptrTauSPM;
                break;

                case 8:
                address=ptrTauLPM;
                break;

                case 9:
                address=ptrRefMillFeedRate;
                break;

                case 10:
                address=ptrCurMillFeedRate;
                break;
        }
        key=getch();
        if(key==ESC){
                gotoxy(50,24);
                break;
        }
        ungetch(key);
```

```
putch(key);
done=FALSE;
canUseCurValue=TRUE;
status=0;
while(!done){
        string=GetStringAt(45,row,18);
        if(string[0]=='\0'&&canUseCurValue){
                gotoxy(45,row);
                cprintf("%.3f",*address);
                cprintf("          ");
                gotoxy(45,row+1);
                break;
        }
        switch(row){
                case 6:
                status=sscanf(string,"%lf",ptrTauPF);
                address= ptrTauPF;
                break;

                case 7:
                status=sscanf(string,"%lf",ptrTauSPM);
                address=ptrTauSPM;
                break;

                case 8:
                status=sscanf(string,"%lf",ptrTauLPM);
                address=ptrTauLPM;
                break;

                case 9:
                status=sscanf(string,"%lf",ptrRefMillFeedRate);
                address=ptrRefMillFeedRate;
                break;

                case 10:
                status=sscanf(string,"%lf",ptrCurMillFeedRate);
                address=ptrCurMillFeedRate;
                break;
        }
        if(status==1&&*address>=minAllowedValue){
                gotoxy(45,row);
                cprintf("%.3f",*address);
                cprintf("          ");
                gotoxy(45,row+1);
                done=TRUE;
        }
        else{
                PrintErrMsg();
                gotoxy(45,row);
                cprintf("          ");
                gotoxy(45,row);
```

```
                    done=FALSE;
                    canUseCurValue=FALSE;
                }
            }
        }
return 0;
}
```

```
/* graphics.c01 */
#include <graphics.h>
#include <dos.h>
#include <math.h>
#include <float.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include "global.h"

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

extern double *ptrWrkSpc;
extern char *logX;
extern char *logY;
extern int sizeClassNumBallSzOpt;
extern int *ptrK,k1,k2,*ptr_iopt,*ptr_nest,*ptr_n,*ptr_n1,*ptr_n2,*ptr_lwrk,
                        *ptr_ier,*ptr_iwrk;
extern double *ptrPosKnots,*ptrPosKnots1,*ptrPosKnots2,*ptr_wrk,*ptr_c,
                        *ptr_c1,*ptr_c2,*ptr_w,*ptr_w1,*ptr_w2;
extern double *ptrXseries,*ptrSelecFunc,*ptrCurSelecFunc,*ptrNewSelecFunc,
                        *ptrFtdSelecFunc,*ptrFtdCurSelecFunc,*ptrFtdNewSelecFunc;
extern double *ptrXb,*ptrXe,*ptrIniS,*ptrS,*ptrDifS,*ptr_fp1,*ptr_fp2;
extern double curBallDia,*ptrCurBallDia;
extern double newBallDia,*ptrNewBallDia;
extern double *ptrScreenSizeBallSzOpt,*ptrSelecFuncBallSzOpt,
                        *ptrStandardDevBallSzOpt;
extern double *ptrFactorK;
extern double *ptr_fin_s,*ptr_fin_s1,*ptr_fin_s2;
extern int iopt,m,nest,n1,n2,lwrk,ier;
extern double xb,xe,iniS,s,fin_s1,fin_s2,difS,factorK,fp1,fp2;
extern double xOfMaxCurSelecFunc,xOfMaxNewSelecFunc,*ptr_xOfMaxSelecFunc;
extern bool isSecondGraph;
int GraphDriver;            /* The Graphics device driver          */
int GraphMode;              /* The Graphics mode value             */
int ErrorCode;              /* Reports any graphics errors         */
double AspectRatio;         /* Aspect ratio of a pixel on the screen*/
struct palettetype palette; /* Used to read palette info           */

int xMaxScr,yMaxScr;
int xGrad,yGrad,xDivNo,yDivNo;
int leftX,leftY,rightX,rightY;
int xC[639],yC[479];
double xLeftMrk,xRightMrk,yDownMrk,yUpMrk,xDiv,yDiv,xScale,yScale,xDot;
```

```
enum plottype{SIZEDIST=1,SELECTIONFUNC=2,BREAKAGEFUNC=3};
extern enum plottype plotType;
extern int *ptrK,*ptr_iopt,*ptr_nest,*ptr_n,*ptr_lwrk,*ptr_ier,*ptr_iwrk;
extern double *ptrXb,*ptrXe,*ptrS,*ptr_fp;
extern double *ptr_w1,*ptr_w2,*ptr_wrk;

/* uses the following interfaces */
void curfit_(int* iopt,int* m,double* x,double* y,double* w,double* xb,
double* xe,int* k,double* s,int* nest,int* n,double* t,double* c__,
double* fp,double* wrk,int* lwrk,int* iwrk,int* ier);
double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
void Locate(double *xx,unsigned long n,double x,unsigned long *j);
double FindMin(const double *vector,int n);
double FindMax(const double *vector,int n);

/* provides interfaces */
int PlotScatterDig(double *xDataVec,double *yDataVec,int m,
                                                    char* title,char* xLab,char* yLab);
int PlotGraph(double (*ptrFunc) (double *ptr_xCo),double a,double b,
                                        char* title,char* xLab,char* yLab);
int PlotGraphBoth(double (*fPtr) (double* x),double a,double b,char* title,
                                                    char* xLab,char* yLab);
void SetPlotPane(char* xLab,char* yLab);
void InitializeGraphics(void);
void TerminateGraphics(void);
double FindDiv(double xMin,double xMax);
void SetGraphWindow(char *title);
void changetextstyle(int font,int direction,int charsize);
int gprintf(int *xloc,int *yloc,char *fmt,...);
void GradX(void);
void AnotYaxGrad(void);
void GradY(void);
void AnotXaxGrad(void);
void DrawSymbols(const double *xDataVec,const double *yDataVec,int  m);
void AnotateScatterDiagram(void);
void AnotateGraph(void);
void AnotateGraphBoth(void);
void UpdateS(double  *ptr_S,double *ptr_fp);

/* function to plot a scatter diagram*/
int PlotScatterDig(double *xDataVec,double *yDataVec,int m,
                                                    char *title,char *xLab,char *yLab){
int i;
double xMin,xMax,yMin,yMax;
struct viewporttype vp;

xMaxScr=getmaxx();
yMaxScr=getmaxy();
/* setting viewport after getting xMaxScr and yMaxScr */
SetGraphWindow(title);
```

```
getviewsettings(&vp);
leftX=vp.left+(int) (0.2*xMaxScr);
leftY=vp.top+(int) (0.08*yMaxScr);
rightX=vp.right-(int) (0.15*xMaxScr);
rightY=vp.bottom-(int) (0.20*yMaxScr);
SetPlotPane(xLab,yLab);
/* xDataVec starts from 1,i.e with a unit offset */
xMin=FindMin(xDataVec,m);
xMax=FindMax(xDataVec,m);
yMin=FindMin(yDataVec,m);
yMax=FindMax(yDataVec,m);
xDiv=FindDiv(xMin,xMax);
if(GE(xMin,0)) xLeftMrk=((long) (xMin/xDiv))* xDiv;
else xLeftMrk=((long) (xMin/xDiv)-1)*xDiv;
if(GE(xMax,0)) xRightMrk=((long) (xMax/xDiv)+1)*xDiv;
else xRightMrk=((long) (xMax/xDiv))*xDiv;
if(plotType!=SIZEDIST){
        yDiv=FindDiv(yMin,yMax);
        if(GE(yMin,0)) yDownMrk=((long) (yMin/yDiv))*yDiv;
        else    yDownMrk=((long) (yMin/yDiv)-1)*yDiv;
        if(GE(yMax,0))  yUpMrk=((long) (yMax/yDiv)+1)*yDiv;
        else    yUpMrk=((long) (yMax/yDiv))*yDiv;
}
else{
        yUpMrk=100.00;
        yDownMrk=0.00;
        yDiv=10.0;
}
xDivNo=(int) ((xRightMrk-xLeftMrk)/xDiv);
yDivNo=(int) ((yUpMrk-yDownMrk)/yDiv);
xScale=((rightX-leftX)/(xRightMrk-xLeftMrk));
yScale=((rightY-leftY)/(yUpMrk-yDownMrk));
setlinestyle(SOLID_LINE,1,NORM_WIDTH);
GradX();
AnotXaxGrad();
setlinestyle(DASHED_LINE,1,NORM_WIDTH);
GradY();
AnotYaxGrad();
AnotateScatterDiagram();
setfillstyle(1,WHITE);
DrawSymbols(xDataVec,yDataVec,m);
return 0;
}

/* function to plot spline curves fitted to selection functions */
int PlotGraph(double (*ptrFunc) (double *ptr_xCo),double a,double b,
                                        char* title,char* xLab,char* yLab){
int i,key,maxIt;
double y,ySml,yBig,ySmlData,yBigData,ySmlFunc,yBigFunc,ySmlFuncZeroS,
                yBigFuncZeroS,ySmlFuncHugeS,yBigFuncHugeS;
double xMin,xMax,dx;
```

```
struct viewporttype vp;

xMaxScr=getmaxx();
yMaxScr=getmaxy();
/* setting viewport after getting xMaxScr and yMaxScr */
SetGraphWindow(title);  /* the big red rectangle */
getviewsettings(&vp);
leftX=vp.left+(int) (0.15*xMaxScr);
leftY=vp.top+(int) (0.08*yMaxScr);
rightX=vp.right-(int) (0.1*xMaxScr);
rightY=vp.bottom-(int) (0.25*yMaxScr);
SetPlotPane(xLab,yLab);
AnotateGraph();
xMin=a;
xMax=b;
xDiv=FindDiv(xMin,xMax);
if(GE(xMin,0)) xLeftMrk=(long) (xMin/xDiv)* xDiv;
else xLeftMrk=((long) (xMin/xDiv)-1)*xDiv;
if(GE(xMax,0)) xRightMrk=((long) (xMax/xDiv)+1)*xDiv;
else xRightMrk=(long) (xMax/xDiv)*xDiv;
xScale=(rightX-leftX)/(double) (xRightMrk-xLeftMrk);
maxIt=(int) (((b*xScale)-(a*xScale))/2);
xDivNo=(int) ((xRightMrk-xLeftMrk)/xDiv);
dx=2*(1/xScale);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);
GradX();
AnotXaxGrad();
/* xDataVec starts from 1, i.e. with a unit offset */
ySmlData=FindMin(ptrSelecFunc-1,sizeClassNumBallSzOpt);
yBigData=FindMax(ptrSelecFunc-1,sizeClassNumBallSzOpt);
ySmlData=ySmlData-(0.05*ySmlData);
yBigData=yBigData+(0.05*yBigData);
/* calculating Min and Max of curve with zero S */
*ptrS=0.0;
curfit_(ptr_iopt,&sizeClassNumBallSzOpt,ptrXseries,
                    ptrSelecFunc,ptr_w,ptrXb,ptrXe,ptrK,ptrS,ptr_nest,
                    ptr_n,ptrPosKnots,ptr_c,ptr_fp,ptr_wrk,ptr_lwrk,ptr_iwrk,
                    ptr_ier);
xDot=a;
ySmlFuncZeroS=yBigFuncZeroS=0.0;
for(i=1;i<=maxIt;i++){
        y=(*ptrFunc)(&xDot);
        if(LT(y,ySmlFuncZeroS)) ySmlFuncZeroS=y;
        if(GT(y,yBigFuncZeroS)) yBigFuncZeroS=y;
        xDot+=dx;
}
/* calculating Min and Max of curve with huge S */
*ptrS=10000000000;
curfit_(ptr_iopt,&sizeClassNumBallSzOpt,ptrXseries,
                    ptrSelecFunc,ptr_w,ptrXb,ptrXe,ptrK,ptrS,ptr_nest,
                    ptr_n,ptrPosKnots,ptr_c,ptr_fp,ptr_wrk,ptr_lwrk,ptr_iwrk,
```

```
                    ptr_ier);
xDot=a;
ySmlFuncHugeS=yBigFuncHugeS=0.0;
for(i=1;i<=maxIt;i++){
        y=(*ptrFunc)(&xDot);
        if(LT(y,ySmlFuncHugeS)) ySmlFuncHugeS=y;
        if(GT(y,yBigFuncHugeS)) yBigFuncHugeS=y;
        xDot+=dx;
}
if(LT(ySmlFuncZeroS-ySmlFuncHugeS,-EPS)) ySmlFunc=ySmlFuncZeroS;
else ySmlFunc=ySmlFuncHugeS;
if(GT(yBigFuncZeroS-yBigFuncHugeS,EPS)) yBigFunc=yBigFuncZeroS;
else  yBigFunc=yBigFuncHugeS;
if(LT(ySmlFunc-ySmlData,-EPS)) ySml=ySmlFunc;
else ySml=ySmlData;
if(GT(yBigFunc-yBigData,EPS)) yBig=yBigFunc;
else yBig=yBigData;
yDiv=FindDiv(ySml,yBig);
if(GE(ySml,0)) yDownMrk=(long) (ySml/yDiv)*yDiv;
else yDownMrk=((long) (ySml/yDiv)-1)*yDiv;
if(GE(yBig,0)) yUpMrk=((long) (yBig/yDiv)+1)*yDiv;
else yUpMrk=(long) (yBig/yDiv)*yDiv;
yDivNo=(yUpMrk-yDownMrk)/yDiv;
yScale=(rightY-leftY)/(yUpMrk-yDownMrk);
setlinestyle(DASHED_LINE,1,NORM_WIDTH);
GradY();
AnotYaxGrad();
setlinestyle(SOLID_LINE,1,NORM_WIDTH);
if(isSecondGraph) setcolor(LIGHTRED);
else setcolor(YELLOW);
if(LT(((*ptr_xOfMaxSelecFunc-xLeftMrk)*xScale),(rightX-leftX))&&
        GT(((*ptr_xOfMaxSelecFunc-xLeftMrk)*xScale),0))
        line(leftX+(*ptr_xOfMaxSelecFunc-xLeftMrk)*xScale,rightY,
                leftX+(*ptr_xOfMaxSelecFunc-xLeftMrk)*xScale,leftY);
if(isSecondGraph) setfillstyle(1,LIGHTRED);
else setfillstyle(1,YELLOW);
/* drawing selection function data point... assumes a unit offset */
DrawSymbols(ptrXseries-1,ptrSelecFunc-1,sizeClassNumBallSzOpt);
/* a curve when s=0 is plotted at the begining of manual curve fitting */
*ptrS=*ptrIniS;
curfit_(ptr_iopt,&sizeClassNumBallSzOpt,ptrXseries,
                ptrSelecFunc,ptr_w,ptrXb,ptrXe,ptrK,ptrS,ptr_nest,
                ptr_n,ptrPosKnots,ptr_c,ptr_fp,ptr_wrk,ptr_lwrk,ptr_iwrk,
                ptr_ier);
setlinestyle(SOLID_LINE,1,1);
if(isSecondGraph) setcolor(LIGHTRED);
else setcolor(YELLOW);
xDot=a;
y=(*ptrFunc) (&xDot);
xC[0]=leftX+(int) ((a-xLeftMrk)*xScale);
yC[0]=rightY-(int)((y-yDownMrk)*yScale);
```

```
moveto(xC[0],yC[0]);
for(i=1;i<=maxIt;i++){
        y=(*ptrFunc) (&xDot);
        xC[i]=leftX+ (int) ((xDot-xLeftMrk)*xScale);
        yC[i]=rightY-(int) ((y-yDownMrk)*yScale);
        if(yC[i]>leftY&&yC[i]<rightY) lineto(xC[i],yC[i]);
        else moveto(xC[i],yC[i]);
        xDot+=dx;
}
GradX();
/* updating smooting factor on the screen */
UpdateS(ptrS,ptr_fp);
/* loop for fitting a spline curve to data by changing smoothing factor, s */
while((key=getch())!=0){
        if(key=='\r'){
                *ptr_fin_s=*ptrS;
                break;
        }
        else    if(key=='C'||key=='c') return 1;
                        /* general pointers are used as alias names for different sets of data like
                        ptr_w for ptr_w1 and ptr_w2...*/
                        else    if(key=='i'||key=='I'){
                                        *ptrS+=(double) *ptrDifS;

curfit_(ptr_iopt,&sizeClassNumBallSzOpt,ptrXseries,

ptrSelecFunc,ptr_w,ptrXb,ptrXe,ptrK,ptrS,ptr_nest,

ptr_n,ptrPosKnots,ptr_c,ptr_fp,ptr_wrk,ptr_lwrk,
                                                        ptr_iwrk,ptr_ier);
                                }
                                else    if(key=='d'||key=='D'){
                                                        if(LT(*ptrS,EPS)) *ptrS=0;
                                                        else *ptrS-=(double) *ptrDifS;

curfit_(ptr_iopt,&sizeClassNumBallSzOpt,ptrXseries,

ptrSelecFunc,ptr_w,ptrXb,ptrXe,ptrK,ptrS,

ptr_nest,ptr_n,ptrPosKnots,ptr_c,ptr_fp,

ptr_wrk,ptr_lwrk,ptr_iwrk,ptr_ier);
                                                }
                                                else continue;
        /* drawing fitted spline */
        moveto(xC[0],yC[0]);
        setcolor(BLACK);
        setlinestyle(SOLID_LINE,1,1);
        for(i=1;i<=maxIt;i++)
                if(yC[i]>leftY&&yC[i]<rightY&&yC[i-1]>leftY&&yC[i-1]<rightY)
                        lineto(xC[i],yC[i]);
```

```
                else
                        moveto(xC[i],yC[i]);
        /* after erasing the previous curve, destroyed parts
        must be restored */
        SetPlotPane(xLab,yLab);
        setlinestyle(DASHED_LINE,1,NORM_WIDTH);
        GradY();
        AnotYaxGrad();
        setlinestyle(SOLID_LINE,1,NORM_WIDTH);
        if(isSecondGraph) setcolor(LIGHTRED);
        else setcolor(YELLOW);
        if(LT(((*ptr_xOfMaxSelecFunc-xLeftMrk)*xScale),(rightX-leftX))&&
                GT(((*ptr_xOfMaxSelecFunc-xLeftMrk)*xScale),0))
                line(leftX+(*ptr_xOfMaxSelecFunc-xLeftMrk)*xScale,rightY,leftX+
                        (*ptr_xOfMaxSelecFunc-xLeftMrk)*xScale,leftY);
        if(isSecondGraph) setfillstyle(1,LIGHTRED);
        else setfillstyle(1,YELLOW);
        /* drawing data points... assumes a unit offset */
        DrawSymbols(ptrXseries-1,ptrSelecFunc-1,sizeClassNumBallSzOpt);
        setlinestyle(SOLID_LINE,1,1);
        if(isSecondGraph) setcolor(LIGHTRED);
        else setcolor(YELLOW);
        /* now, the new curve is plotted using the spline
        function returned by curfit based on new smoothing factor */
        xDot=a;
        y=(*ptrFunc) (&xDot);
        xC[0]=leftX+(int) ((a-xLeftMrk)*xScale);
        yC[0]=rightY-(int)((y-yDownMrk)*yScale);
        moveto(xC[0],yC[0]);
        for(i=1;i<=maxIt;i++){
                y=(*ptrFunc) (&xDot);
                xC[i]=leftX+ (int) ((xDot-xLeftMrk)*xScale);
                yC[i]=rightY-(int) ((y-yDownMrk)*yScale);
                if(yC[i]>leftY&&yC[i]<rightY&&yC[i-1]>leftY&&yC[i-1]<rightY)
                        lineto(xC[i],yC[i]);
                else
                        moveto(xC[i],yC[i]);
                        xDot+=dx;
        }
        GradX();
        /* updating smooting factor on the screen */
        UpdateS(ptrS,ptr_fp);
}
/* evaluating function (fitted or calculated selection
functions) for each size class */
for(i=1;i<=sizeClassNumBallSzOpt;i++)
        ptrFtdSelecFunc[i-1]=y=(*ptrFunc) (ptrXseries+i-1);
return 0;
}

int PlotGraphBoth(double (*ptrFunc) (double* x),double a,double b,char* title,
```

```
                                                    char* xLab,char* yLab){
int i,key,maxIt;
double y,ySml,yBig,ySmlData,yBigData,ySmlFunc,yBigFunc,ySmlFunc1,
                    yBigFunc1,ySmlFunc2,yBigFunc2;
double xMin,xMax,dx;
struct viewporttype vp;

xMaxScr=getmaxx();
yMaxScr=getmaxy();
/* sets viewport */
SetGraphWindow(title);
getviewsettings(&vp);
leftX=vp.left+(int) (0.15*xMaxScr);
leftY=vp.top+(int) (0.08*yMaxScr);
rightX=vp.right-(int) (0.1*xMaxScr);
rightY=vp.bottom-(int) (0.2*yMaxScr);
SetPlotPane(xLab,yLab);
AnotateGraphBoth();
xMin=a;
xMax=b;
xDiv=FindDiv(xMin,xMax);
if(GE(xMin,0)) xLeftMrk=(long) (xMin/xDiv)*xDiv;
else xLeftMrk=((long) (xMin/xDiv)-1)*xDiv;
if(GE(xMax,0)) xRightMrk=((long) (xMax/xDiv)+1)*xDiv;
else xRightMrk=(long) (xMax/xDiv)*xDiv;
xScale=(rightX-leftX)/(double)(xRightMrk-xLeftMrk);
maxIt=(int) (b*xScale)-(int) (a*xScale);
xDivNo=(int) ((xRightMrk-xLeftMrk)/xDiv);
dx=1/xScale;
setlinestyle(SOLID_LINE,1,NORM_WIDTH);
GradX();
AnotXaxGrad();
/* xDataVec starts from 1, i.e. with a unit offset */
ySmlData=FindMin(ptrSelecFuncBallSzOpt-1,sizeClassNumBallSzOpt);
yBigData=FindMax(ptrSelecFuncBallSzOpt-1,sizeClassNumBallSzOpt);
ySmlData=ySmlData-(0.05*ySmlData);
yBigData=yBigData+(0.05*yBigData);
ptrPosKnots=ptrPosKnots1;
ptrK=&k1;
ptr_n=ptr_n1;
ptr_c=ptr_c1;
xDot=a;
ySmlFunc1=yBigFunc1=0.0;
for(i=1;i<=maxIt;i++){
        y=(*ptrFunc)(&xDot);
        if(LT(y,ySmlFunc1)) ySmlFunc1=y;
        if(GT(y,yBigFunc1)) yBigFunc1=y;
        xDot+=dx;
}
ptrPosKnots=ptrPosKnots2;
ptrK=&k2;
```

```
ptr_n=ptr_n2;
ptr_c=ptr_c2;
xDot=a;
ySmlFunc2=yBigFunc2=0.0;
for(i=1;i<=maxIt;i++){
        y=(*ptrFunc)(&xDot);
        if(LT(y,ySmlFunc2)) ySmlFunc2=y;
        if(GT(y,yBigFunc2)) yBigFunc2=y;
        xDot += dx;
}
if(LT(ySmlFunc1-ySmlFunc2,-EPS)) ySmlFunc=ySmlFunc1;
else ySmlFunc=ySmlFunc2;
if(GT(yBigFunc1-yBigFunc2,EPS)) yBigFunc=yBigFunc1;
else yBigFunc=yBigFunc2;
if(LT(ySmlData-ySmlFunc,-EPS)) ySml=ySmlData;
else if(GT(ySmlData-ySmlFunc,EPS)) ySml=ySmlFunc;
else ySml=ySmlData=ySmlFunc;
if(GT(yBigData-yBigFunc,EPS)) yBig=yBigData;
else if(LT(yBigData-yBigFunc,-EPS)) yBig=yBigFunc;
else yBig=yBigData=yBigFunc;

yDiv=FindDiv(ySml,yBig);
if(GE(ySml,0)) yDownMrk=(long) (ySml/yDiv)*yDiv;
else yDownMrk=((long) (ySml/yDiv)-1)*yDiv;
if(GE(yBig,0)) yUpMrk=((long) (yBig/yDiv)+1)*yDiv;
else yUpMrk=(long) (yBig/yDiv)*yDiv;
yDivNo=(yUpMrk-yDownMrk)/yDiv;
yScale=(rightY-leftY)/(yUpMrk-yDownMrk);
setlinestyle(DASHED_LINE,1,NORM_WIDTH);
GradY();
AnotYaxGrad();
setlinestyle(SOLID_LINE,1,NORM_WIDTH);
setcolor(YELLOW);
if(LT(((xOfMaxCurSelecFunc-xLeftMrk)*xScale),(rightX-leftX))&&
        GT(((xOfMaxCurSelecFunc-xLeftMrk)*xScale),0))
        line(leftX+(xOfMaxCurSelecFunc-xLeftMrk)*xScale,rightY,
          leftX+(xOfMaxCurSelecFunc-xLeftMrk)*xScale,leftY);
/* plotting the first selected curve */
ptrPosKnots=ptrPosKnots1;
ptrK=&k1;
ptr_n=ptr_n1;
ptr_c=ptr_c1;
setfillstyle(1,YELLOW);
/* drawing data points... assumes a unit offset */
DrawSymbols(ptrXseries-1,ptrCurSelecFunc-1,sizeClassNumBallSzOpt);
setcolor(YELLOW);
setlinestyle(SOLID_LINE,1,1);
xDot=a;
y=(*ptrFunc) (&xDot);
xC[0]=leftX+(int) ((a-xLeftMrk)*xScale);
yC[0]=rightY-(int)((y-yDownMrk)*yScale);
```

```
moveto(xC[0],yC[0]);
for(i=1;i<=maxIt;i++){
        y=(*ptrFunc) (&xDot);
        xC[i]=leftX+ (int) ((xDot-xLeftMrk)*xScale);
        yC[i]=rightY-(int) ((y-yDownMrk)*yScale);
        lineto(xC[i],yC[i]);
        xDot+=dx;
}
/* plotting the second selected curve */
setlinestyle(SOLID_LINE,1,NORM_WIDTH);
setcolor(LIGHTRED);
if(LT(((xOfMaxNewSelecFunc-xLeftMrk)*xScale),(rightX-leftX))&&
        GT(((xOfMaxNewSelecFunc-xLeftMrk)*xScale),0))
        line(leftX+(xOfMaxNewSelecFunc-xLeftMrk)*xScale,rightY,
          leftX+(xOfMaxNewSelecFunc-xLeftMrk)*xScale,leftY);
ptrPosKnots=ptrPosKnots2;
ptrK=&k2;
ptr_n=ptr_n2;
ptr_c=ptr_c2;
setfillstyle(1,LIGHTRED);
/* drawing data points... assumes a unit offset */
DrawSymbols(ptrXseries-1,ptrNewSelecFunc-1,sizeClassNumBallSzOpt);
setcolor(LIGHTRED);
setlinestyle(SOLID_LINE,1,1);
xDot=a;
y=(*ptrFunc) (&xDot);
xC[0]=leftX+(int) ((a-xLeftMrk)*xScale);
yC[0]=rightY-(int)((y-yDownMrk)*yScale);
moveto(xC[0],yC[0]);
for(i=1;i<=maxIt;i++){
        y=(*ptrFunc) (&xDot);
        xC[i]=leftX+ (int) ((xDot-xLeftMrk)*xScale);
        yC[i]=rightY-(int) ((y-yDownMrk)*yScale);
        lineto(xC[i],yC[i]);
        xDot+=dx;
}
/* letting the user to see plotted curves */
getch();
return 0;
}

/* initializes the graphics system and reports any errors which occured */
void InitializeGraphics(void){
int xasp,yasp;    /* Used to read the aspect ratio*/

registerbgidriver(EGAVGA_driver);
registerbgifont(sansserif_font);
registerbgifont(small_font);
GraphDriver=DETECT; /* Request auto-detection */
initgraph(&GraphDriver,&GraphMode,"");
ErrorCode=graphresult(); /* Read result of initialization*/
```

```
if(ErrorCode!=grOk){    /* Error occured during init        */
        printf(" Graphics System Error: %s\n",grapherrormsg(ErrorCode));
        delay(100);
        exit(1);
}
getpalette(&palette); /* Read the palette from board */
getaspectratio(&xasp,&yasp); /* read the hardware aspect    */
AspectRatio=(double)xasp/(double)yasp; /* Get correction factor */
}

void TerminateGraphics(void){
cleardevice();
closegraph();
}

void SetGraphWindow(char *title){
struct viewporttype vp;

cleardevice();      /* Clear graphics screen */
setcolor(LIGHTCYAN);  /* Set current color to white        */
setviewport(0,0,xMaxScr,yMaxScr,1);      /* Open port to full screen */
changetextstyle(SMALL_FONT,HORIZ_DIR,0);
setusercharsize(4,3,4,3);
settextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(xMaxScr/2,15,title);
/* draws a solid single line around the current viewport*/
setcolor(RED);
setlinestyle(SOLID_LINE,0,3);
getviewsettings(&vp);
rectangle(0,0,vp.right-vp.left,vp.bottom-vp.top);
}

void changetextstyle(int font,int direction,int charsize){
int ErrorCode;

settextstyle(font,direction,charsize);
ErrorCode=graphresult(); /* check result */
if(ErrorCode!=grOk){ /* if error occured */
        closegraph();
        printf(" Graphics System Error: %s\n",grapherrormsg(ErrorCode));
        delay(100);
        exit(1);
        }
}
/* used like PRINTF except the output is sent to the screen in graphics
        mode at the specified co-ordinate */
int gprintf(int *xloc,int *yloc,char *fmt,...){
va_list  argptr;                    /* Argument list pointer  */
char str[140];                      /* Buffer to build sting into        */
int cnt;                            /* Result of SPRINTF for return */
```

```
va_start(argptr,fmt);              /* Initialize va_ functions          */
cnt = vsprintf(str,fmt,argptr);    /* prints string to buffer  */
outtextxy(*xloc,*yloc,str);        /* Send string in graphics mode */
va_end(argptr);                    /* Close va_ functions */
return(cnt);                       /* Return the conversion count */
}

/* function to set plot pane */
void SetPlotPane(char* xLab,char* yLab){
struct viewporttype vp;

getviewsettings(&vp);
setlinestyle(SOLID_LINE,1,2);
setcolor(WHITE);
rectangle(leftX,leftY,rightX,rightY);

changetextstyle(SMALL_FONT,HORIZ_DIR,0);
setusercharsize(4,2,4,2);
settextjustify(CENTER_TEXT,TOP_TEXT);
setcolor(YELLOW);
outtextxy((rightX-leftX)/2+leftX,rightY+25,xLab);

changetextstyle(SMALL_FONT,VERT_DIR,0);
setusercharsize(4,2,4,2);
settextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(vp.left+20,(rightY-leftY)/2+leftY,yLab);
}

/* function to find x and y divisions */
double FindDiv(double MinNum,double MaxNum){
double div10;
double div;
int i,j;
unsigned long* index;

j=1;
for(i=-14;i<15;i++){
        ptrWrkSpc[j]=pow10(i)*1;
        j++;
        ptrWrkSpc[j]=pow10(i)*2;
        j++;
        ptrWrkSpc[j]=pow10(i)*5;
        j++;
        ptrWrkSpc[j]=pow10(i)*8;
        j++;
}
if(EQ(MaxNum,MinNum)){
         if(MinNum==0.0) MinNum -= 0.05;
         else MinNum=MinNum-MinNum*0.05;
         if(MaxNum==0.0) MaxNum += 0.05;
         else MaxNum=MaxNum+MaxNum*0.05;
```

```
}
div10=(MaxNum-MinNum)/10;
Locate(ptrWrkSpc,116,div10,index);
div=ptrWrkSpc[*index+1];
return div;
}

/* function to graduate x axis */
void GradX(void){
int i;

setcolor(WHITE);
for(i=1;i<=xDivNo;i++){
        if(rightX > leftX+(int) (i*(xDiv*xScale)))  /* these are two integers */
                line(leftX+(int) (i*(xDiv*xScale)),rightY,
                        leftX+(int) (i*(xDiv*xScale)),rightY-5);
}
}

/* function to anotate x graduate */
void AnotXaxGrad(void){
char buffer[]="format string";
int i;
double gradsIndicator;

changetextstyle(SMALL_FONT,HORIZ_DIR,0);
setusercharsize(5,4,5,4);
settextjustify(CENTER_TEXT,TOP_TEXT);
yGrad=rightY+5;
for(i=1;i<=xDivNo+2;i=i+2){
        xGrad=leftX+(int) ((i-1)*xDiv*xScale);
        gradsIndicator=xLeftMrk+(i-1)*xDiv;
        if((xDiv<=1)&&(xDiv>0.1))
        strcpy(buffer,"%.1lf");
        else if((xDiv<=0.1)&&(xDiv>0.01))
        strcpy(buffer,"%.2lf");
        else if((xDiv<=0.01)&&(xDiv>0.001))
        strcpy(buffer,"%.3lf");
        else if((xDiv<=0.001)&&(xDiv>0.0001))
        strcpy(buffer,"%.4lf");
        else if((xDiv<=0.0001)&&(xDiv>0.00001))
        strcpy(buffer,"%.5lf");
        else if((xDiv<=0.00001))
        strcpy(buffer,"%.1e");
        else if ((xLeftMrk<= -10000) || (xLeftMrk >= 10000))
        strcpy(buffer,"%.1e");
        else
        strcpy(buffer,"%.2lf");
        if(xRightMrk-gradsIndicator >= 0)
        gprintf(&xGrad,&yGrad,buffer,gradsIndicator);
}
```

```
}

/* function to graduate y axis */
void GradY(void){
int i;
setcolor(WHITE);
for(i=1;i<=yDivNo;i++){
        if(rightY>leftY+ (int) (i*(yDiv*yScale)))    /* these are two integers */
        line(leftX,leftY+(int) (i*(yDiv*yScale)),
                rightX,leftY+(int) i*(yDiv*yScale));
}
}

/* function to anotate y graduates */
void AnotYaxGrad(void){
int i;
char buffer[]="format string";
double gradsIndicator;

changetextstyle(SMALL_FONT,HORIZ_DIR,0);
setusercharsize(5,4,5,4);
settextjustify(RIGHT_TEXT,CENTER_TEXT);
xGrad=leftX-2;
for(i=1;i<=yDivNo+2;i++){
        yGrad=rightY-(int) ((i-1)*yDiv*yScale);
        gradsIndicator=yDownMrk+(i-1)*yDiv;

        if((yDiv<=1)&&(yDiv>0.1))
        strcpy(buffer,"%.1f");
        else if((yDiv<=0.1) &&(yDiv>0.01))
        strcpy(buffer,"%.2f");
        else if((yDiv<=0.01)&&(yDiv>0.001))
        strcpy(buffer,"%.3f");
        else if((yDiv<=0.001)&&(yDiv>0.0001))
        strcpy(buffer,"%.4f");
        else if((yDiv<=0.0001)&&(yDiv>0.00001))
        strcpy(buffer,"%.5f");
        else if((yDiv<=0.00001))
        strcpy(buffer,"%.1e");
        else if((yDownMrk >= 1000) || (yDownMrk<=-1000))
        strcpy(buffer,"%.0e");
        else
        strcpy(buffer,"%.2f");
        if(yUpMrk-gradsIndicator >= 0)
        gprintf(&xGrad,&yGrad,buffer,gradsIndicator);
}
}

/* function to draw symbols at XYs */
void    DrawSymbols(const double *xDataVec,const double* yDataVec,int m){
int i;
```

```
int xP,yP;

for(i=1;i<=m;i++){
        xP=((xDataVec[i]-xLeftMrk)*xScale)+leftX;
        yP=((yUpMrk-yDataVec[i])*yScale)+leftY;
        if(LT(((yUpMrk-yDataVec[i])*yScale),(rightY-leftY)) &&
        GT(((yUpMrk-yDataVec[i])*yScale),0)){
                moveto((int) xP,(int) yP);
                bar(getx()-2,gety()-2,getx()+2,gety()+2);
        }
}
}

void AnotateScatterDiagram(void){
int charW,charH;
struct viewporttype vp;

getviewsettings(&vp);
changetextstyle(SMALL_FONT,HORIZ_DIR,0);
setusercharsize(7,5,7,5);
settextjustify(RIGHT_TEXT,BOTTOM_TEXT);
charW=textwidth("M");
charH=textheight("H");
setcolor(YELLOW);
outtextxy(xMaxScr-charW-2,yMaxScr-charH,"Press any key to continue...");
}

void AnotateGraph(void){
int charW,charH;
struct viewporttype vp;

getviewsettings(&vp);
setcolor(WHITE);
changetextstyle(SMALL_FONT,HORIZ_DIR,0);
setusercharsize(7,5,7,5);
settextjustify(LEFT_TEXT,BOTTOM_TEXT);
charW=textwidth("M");
charH=textheight("H");
outtextxy(charW,yMaxScr-5*charH,"Smoothing factor    ==> ");
outtextxy(charW,yMaxScr-4*charH,"Sum of Sq. Residual ==> ");
setcolor(LIGHTGRAY);
outtextxy(charW,yMaxScr-3*charH,"Press <I> or <D> to increase or decrease smoothing factor");
outtextxy(charW,yMaxScr-2*charH,"Press <C> to change spline curve fitting options");
setcolor(YELLOW);
charW=textwidth("Press <Enter> to select the curve");
outtextxy(xMaxScr-charW-2,yMaxScr-charH,"Press <Enter> to select the curve");
}

void AnotateGraphBoth(void){
int charW,charH;
struct viewporttype vp;
```

```
getviewsettings(&vp);
setcolor(YELLOW);
changetextstyle(SMALL_FONT,HORIZ_DIR,0);
setusercharsize(7,5,7,5);
settextjustify(RIGHT_TEXT,BOTTOM_TEXT);
charW=textwidth("M");
charH=textheight("H");
outtextxy(xMaxScr-charW-2,yMaxScr-charH,"Press any key to continue...");
}

void UpdateS(double  *ptr_S,double *ptr_fp){
char *str;
int charW,charH;
int x,y;

setcolor(WHITE);
changetextstyle(SMALL_FONT,HORIZ_DIR,0);
setusercharsize(7,5,7,5);
settextjustify(LEFT_TEXT,BOTTOM_TEXT);
charW=textwidth("M");
charH=textheight("H");
setfillstyle(1,BLACK);
x=charW+textwidth("Sum of Sq. Residual ==> ");
y=yMaxScr-5*charH;
bar(x,y-charH,x+x,y+charH);
gprintf(&x,&y,"%f",*ptr_S);
y=yMaxScr-4*charH;
gprintf(&x,&y,"%f",*ptr_fp);
}
```

```
// splfit.c01
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

void curfit_(int* iopt,int* m,double* x,double* y,double* w,double* xb,
double* xe,int* k,double* s,int* nest,int* n,double* t,double* c__,
double* fp,double* wrk,int* lwrk,int* iwrk,int* ier);

double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
void curfit_(int *iopt,int *m,double *x,double *y,double *w,
double *xb,double *xe,int *k,double *s,int *nest,int *n,double *t,
double *c__,double *fp,double *wrk,int *lwrk,int *iwrk,int *ier);
void fpback_(double *a,double *z__,int *n,int *k,double *c__,int *nest);
void fpbspl_(double *t,int *n,int *k,double *x,int *l,double *h__);
void fpchec_(double *x,int *m,double *t,int *n,int *k,int *ier);
void fpcurf_(int *iopt,double *x,double *y,double *w,int *m,double *xb,double *xe,
int *k,double *s,int *nest,double *tol,int *maxit,int *k1,int *k2,int *n,
double *t,double *c__,double *fp,double *fpint,double *z__,double *a,double *b,
double *g,double * q,int *nrdata,int *ier);
void fpdisc_(double *t,int *n,int *k2,double *b,int *nest);
void fpgivs_(double *piv,double *ww,double *cos__,double *sin__);
void fpknot_(double *x,int *m,double *t,int *n,double *fpint,
int *nrdata,int *nrint,int *nest,int *istart);
double fprati_(double *p1,double  *f1,double *p2,double *f2,double *p3,double *f3);
void fprota_(double *cos__,double *sin__,double *a,double *b);
void splev_(double *t,int *n,double *c__,int *k,double *x,double *y,
int *m,int *ier);
static int c__1=1;
//.....................................................................
void curfit_(int* iopt,int* m,double* x,double* y,double* w,double* xb,
double* xe,int* k,double* s,int* nest,int* n,double* t,double* c__,
double* fp,double* wrk,int* lwrk,int* iwrk,int* ier){
                    /* System generated locals */
                    int i__1;
                    int i;
                    /* Local variables */
                    static int nmin,i__,j,maxit,k1,k2,lwest,ia,ib,ig;
                    static int iq,iz;
                    static int ifp;
                    static double tol;

                    /* Parameter adjustments */
                    --w;
                    --y;
                    --x;
                    --iwrk;
                    --c__;
                    --t;
                    --wrk;
```

```
                /* Function Body */
                maxit=20;
                tol  =0.001;
/*  before starting computations a data check is made. if the input data
*/
/*  are invalid,control is immediately repassed to the calling program.
*/
                *ier=10;
                if(*k<=0 || *k>5){
        goto L50;
                }
                k1=*k+1;
                k2=k1+1;
                if(*iopt<-1 || *iopt>1){
        goto L50;
                }
                nmin=2*k1;
                if(*m<k1 || *nest<nmin){
        goto L50;
                }
                lwest=*m*k1+*nest *(*k*3+7);
                if(*lwrk<lwest){
        goto L50;
                }
                if(*xb>x[1] || *xe<x[*m] || w[1]<=0.0){
        goto L50;
                }
                i__1=*m;
                for(i__=2;i__<=i__1;++i__){
        if(x[i__-1]>=x[i__] || w[i__]<=0.0){
                        goto L50;
        }
/* L10: */
                }
                if(*iopt>=0){
        goto L30;
                }
                if(*n<nmin || *n>*nest){
        goto L50;
                }
                j=*n;
                i__1=k1;
                for(i__=1;i__<=i__1;++i__){
        t[i__]=*xb;
        t[j]=*xe;
        --j;
/* L20: */
                }
                fpchec_(&x[1],m,&t[1],n,k,ier);
                if(*ier != 0){
        goto L50;
```

```
                } else {
        goto L40;
                }
L30:
                if(*s<0.0){
        goto L50;
                }
                if(*s==0.0 && *nest<*m+k1){
        goto L50;
                }
                *ier=0;
/* we partition the working space and determine the spline approximation.
*/
L40:
                ifp=1;
                iz=ifp+*nest;
                ia=iz+*nest;
                ib=ia+*nest*k1;
                ig=ib+*nest*k2;
                iq=ig+*nest*k2;
                fpcurf_(iopt,&x[1],&y[1],&w[1],m,xb,xe,k,s,nest,&tol,&maxit,&
                        k1,&k2,n,&t[1],&c__[1],fp,&wrk[ifp],&wrk[iz],&wrk[ia],&
                        wrk[ib],&wrk[ig],&wrk[iq],&iwrk[1],ier);
L50: ;

} /* curfit_ */
//.....................................................................
void fpback_(double* a,double* z__,int* n,int* k,double* c__,int* nest){
                /* System generated locals */
                int a_dim1,a_offset,i__1,i__2;

                /* Local variables */
                static int i__,j,l,m,i1;
                static double store;
                static int k1;

                /* Parameter adjustments */
                --c__;
                --z__;
                a_dim1=*nest;
                a_offset=a_dim1+1;
                a -= a_offset;

                /* Function Body */
                k1=*k-1;
                c__[*n]=z__[*n] / a[*n+a_dim1];
                i__=*n-1;
                if(i__==0){
        goto L30;
                }
                i__1=*n;
```

```
                for(j=2;j<=i__1;++j){
        store=z__[i__];
        il=kl;
        if(j<=kl){
                        il=j-1;
        }
        m=i__;
        i__2=il;
        for(l=1;l<=i__2;++l){
                        ++m;
                        store -= c__[m]*a[i__ +(l+1)*a_dim1];
/* L10: */
        }
        c__[i__]=store / a[i__+a_dim1];
        --i__;
/* L20: */
                }
L30: ;

} /* fpback_ */

void fpbspl_(double* t,int* n,int* k,double* x,int* l,double* h__){
                /* System generated locals */
                int i__1,i__2;

                /* Local variables */
                static double f;
                static int i__,j;
                static double hh[5];
                static int li,lj;
                static double one;

                /* Parameter adjustments */
                --t;
                --h__;

                /* Function Body */
                one =1.0;
                h__[1]=one;
                i__1=*k;
                for(j=1;j<=i__1;++j){
        i__2=j;
        for(i__=1;i__<=i__2;++i__){
                        hh[i__-1]=h__[i__];
/* L10: */
        }
        h__[1]=0.0;
        i__2=j;
        for(i__=1;i__<=i__2;++i__){
                        li=*l+i__;
                        lj=li-j;
```

```
                    f=hh[i__-1] /(t[li]-t[lj]);
                    h__[i__] += f *(t[li]-*x);
                    h__[i__+1]=f *(*x-t[lj]);
/* L20: */
        }
            }

} /* fpbspl_ */

void fpchec_(double* x,int* m,double* t,int* n,int* k,int* ier){
            /* System generated locals */
            int i__1;

            /* Local variables */
            static int i__,j,l,k1,k2;
            static double tj,tl;
            static int nk1,nk2,nk3;

            /* Parameter adjustments */
            --x;
            --t;

            /* Function Body */
            k1=*k+1;
            k2=k1+1;
            nk1=*n-k1;
            nk2=nk1+1;
            *ier=10;
/*  check condition no 1 */
            if(nk1<k1 || nk1>*m){
        goto L80;
            }
/*  check condition no 2 */
            j=*n;
            i__1=*k;
            for(i__=1;i__<=i__1;++i__){
        if(t[i__]>t[i__+1]){
                    goto L80;
        }
        if(t[j]<t[j-1]){
                    goto L80;
        }
        --j;
/* L20: */
            }
/*  check condition no 3 */
            i__1=nk2;
            for(i__=k2;i__<=i__1;++i__){
        if(t[i__]<=t[i__-1]){
                    goto L80;
        }
```

```
/* L30: */
                }
/*  check condition no 4 */
                if(x[1]<t[k1] || x[*m]>t[nk2]){
        goto L80;
                }
/*  check condition no 5 */
                if(x[1]>=t[k2] || x[*m]<=t[nk1]){
        goto L80;
                }
                i__=1;
                l=k2;
                nk3=nk1-1;
                if(nk3<2){
        goto L70;
                }
                i__1=nk3;
                for(j=2;j<=i__1;++j){
        tj=t[j];
        ++l;
        tl=t[l];
L40:
        ++i__;
        if(i__>=*m){
                        goto L80;
        }
        if(x[i__]<=tj){
                        goto L40;
        }
        if(x[i__]>=tl){
                        goto L80;
        }
/* L60: */
                }
L70:
                *ier=0;
L80: ;

} /* fpchec_ */

void fpcurf_(int* iopt,double* x,double* y,double* w,int* m,double* xb,
double* xe,int* k,double* s,int* nest,double* tol,int* maxit,
int*       k1,int* k2,int* n,double* t,double*  c__,double* fp,double* fpint,
double* z__,double* a,double* b,double* g,double* q,int* nrdata,int* ier){
                /* System generated locals */
                int a_dim1,a_offset,b_dim1,b_offset,g_dim1,g_offset,q_dim1,
                        q_offset,i__1,i__2,i__3,i__4,i__5;
                double r__1;

                /* Local variables */
                static double half;
```

```
static int nmin,iter,nmax;
static double fpms,term,pinv,h__[7];
static int i__,j,l;
static double p,fpold,fpart,f1,f2,f3;
static int i1,i2;
static double store;
static int i3,k3;
static double p1,p2,p3;
static int l0,nplus,nrint,n8;
static int it;
static double rn,wi,xi,yi;
static double fp0;
static int mk1,nk1;
static double acc,one,cos__,sin__;
static int new__;
static double piv;
static int ich1,ich3;
static double con1,con4,con9;
static int npl1;

/* Parameter adjustments */
--w;
--y;
--x;
--nrdata;
--z__;
--fpint;
--c__;
--t;
q_dim1 = *m;
q_offset = q_dim1 + 1;
q -= q_offset;
a_dim1 = *nest;
a_offset = a_dim1 + 1;
a -= a_offset;
g_dim1 = *nest;
g_offset = g_dim1 + 1;
g -= g_offset;
b_dim1 = *nest;
b_offset = b_dim1 + 1;
b -= b_offset;

/* Function Body */
one = 1.0;
con1 = 0.1;
con9 = 0.9;
con4 = 0.04;
half = 0.5;

/* part 1: determination of the number of knots and their position     c
```

```
/* determine nmin,the number of knots for polynomial approximation. */
                nmin=*k1*2 ;
                if(*iopt<0){
        goto L60;
                }
/* calculation of acc,the absolute tolerance for the root of f(p)=s. */
                acc=*tol**s;
/* determine nmax,the number of knots for spline interpolation. */
                nmax=*m+*k1;
                if(*s>0.0){
        goto L45;
                }
/* if s=0,s(x) is an interpolating spline. */
/* test whether the required storage space exceeds the available one. */
                *n=nmax;
                if(nmax>*nest){
        goto L420;
                }
/* find the position of the interior knots in case of interpolation. */
L10:
                mk1=*m-*k1;
                if(mk1==0){
        goto L60;
                }
                k3=*k / 2;
                i__=*k2;
                j=k3+2;
                if(k3*2==*k){
        goto L30;
                }
                i__1=mk1;
                for(l=1;l<=i__1;++l){
        t[i__]=x[j];
        ++i__;
        ++j;
/* L20: */
                }
                goto L60;
L30:
                i__1=mk1;
                for(l=1;l<=i__1;++l){
        t[i__]=(x[j]+x[j-1])*half;
        ++i__;
        ++j;
/* L40: */
                }
                goto L60;.

L45:
                if(*iopt==0){
        goto L50;
```

```
                }
                if(*n==nmin){
        goto L50;
                }
                fp0=fpint[*n];
                fpold=fpint[*n-1];
                nplus=nrdata[*n];
                if(fp0>*s){
        goto L60;
                }
L50:
                *n=nmin;
                fpold=0.0;
                nplus=0;
                nrdata[1]=*m-2;
/*  main loop for the different sets of knots. m is a save upper bound */
/*  for the number of trials. */
L60:
                i__1=*m;
                for(iter=1;iter<=i__1;++iter){
        if(*n==nmin){
                        *ier=-2;
        }
/*  find nrint,tne number of knot intervals. */
        nrint=*n-nmin+1;
/*  find the position of the additional knots which are needed for */
/*  the b-spline representation of s(x). */
        nk1=*n-*k1;
        i__=*n;
        i__2=*k1;
        for(j=1;j<=i__2;++j){
                        t[j]=*xb;
                        t[i__]=*xe;
                        --i__;
/* L70: */
        }

        *fp=0.0;
/*  initialize the observation matrix a. */
        i__2=nk1;
        for(i__=1;i__<=i__2;++i__){
                        z__[i__]=0.0;
                        i__3=*k1;
                        for(j=1;j<=i__3;++j){
                a[i__+j*a_dim1]=0.0;
/* L80: */
                        }
        }
        l=*k1;
        i__3=*m;
        for(it=1;it<=i__3;++it){
```

```
/* fetch the current data point x(it),y(it). */
                        xi=x[it];
                        wi=w[it];
                        yi=y[it]*wi;
/* search for knot interval t(l)<=xi<t(l+1). */
L85:
                        if(xi<t[l+1] || l==nk1){
                goto L90;
                        }
                        ++l;
                        goto L85;
/* evaluate the(k+1) non-zero b-splines at xi and store them in
q. */
L90:
                        fpbspl_(&t[1],n,k,&xi,&l,h__);
                        i__2=*k1;
                        for(i__=1;i__<=i__2;++i__){
                q[it+i__*q_dim1]=h__[i__-1];
                h__[i__-1] *= wi;
/* L95: */
                        }
/* rotate the new row of the observation matrix into triangle. */
                        j=l-*k1;
                        i__2=*k1;
                        for(i__=1;i__<=i__2;++i__){
                ++j;
                piv=h__[i__-1];
                if(piv==0.0){
                                goto L110;
                }
/* calculate the parameters of the givens transformation. */
                fpgivs_(&piv,&a[j+a_dim1],&cos__,&sin__);
/* transformations to right hand side. */
                fprota_(&cos__,&sin__,&yi,&z__[j]);
                if(i__==*k1){
                                goto L120;
                }
                i2=1;
                i3=i__+1;
                i__4=*k1;
                for(i1=i3;i1<=i__4;++i1){
                                ++i2;
/* transformations to left hand side. */
                                fprota_(&cos__,&sin__,&h__[i1-1],&a[j+i2*a_dim1])
                                        ;
/* L100: */
                }
L110:
                ;
                        }
/* add contribution of this row to the sum of squares of residual
```

```
 */
/* right hand sides. */
L120:
/* Computing 2nd power */
                        r__1=yi;
                        *fp += r__1*r__1;
/* L130: */
        }
        if(*ier==-2){
                        fp0=*fp;
        }
        fpint[*n]=fp0;
        fpint[*n-1]=fpold;
        nrdata[*n]=nplus;
/* backward substitution to obtain the b-spline coefficients. */
        fpback_(&a[a_offset],&z__[1],&nk1,k1,&c__[1],nest);
/* test whether the approximation sinf(x) is an acceptable solution.
*/
        if(*iopt<0){
                        goto L440;
        }
        fpms=*fp-*s;
        if(fabs(fpms)<acc){
                        goto L440;
        }
/* if f(p=inf)<s accept the choice of knots. */
        if(fpms<0.0){
                        goto L250;
        }
/* if n=nmax,sinf(x) is an interpolating spline. */
        if(*n==nmax){
                        goto L430;
        }
/* increase the number of knots. */
/* if n=nest we cannot increase the number of knots because of */
/* the storage capacity limitation. */
        if(*n==*nest){
                        goto L420;
        }
/* determine the number of knots nplus we are going to add. */
        if(*ier==0){
                        goto L140;
        }
        nplus=1;
        *ier=0;
        goto L150;
L140:
        npl1=nplus*2;
        rn= nplus;
        if(fpold-*fp>acc){
                        npl1=rn*fpms /(fpold-*fp);
```

```
        }
/* Computing MIN */
/* Computing MAX */
        i__4=npl1,i__5=nplus / 2,i__4=max(i__4,i__5);
        i__3=nplus*2,i__2=max(i__4,1);
        nplus=min(i__3,i__2);
L150:
        fpold=*fp;
/*  compute the sum((w(i)*(y(i)-s(x(i))))**2) for each knot interval
*/
/*  t(j+k)<=x(i)<=t(j+k+1) and store it in fpint(j),j=1,2,...nrint
. */
        fpart =0.0;
        i__=1;
        l=*k2;
        new__=0;
        i__3=*m;
        for(it=1;it<=i__3;++it){
                        if(x[it]<t[l] || l>nk1){
                goto L160;
                        }
                        new__=1;
                        ++l;
L160:
                        term=0.0;
                        l0=l-*k2;
                        i__2=*k1;
                        for(j=1;j<=i__2;++j){
                ++l0;
                term += c__[l0]*q[it+j*q_dim1];
/* L170: */
                        }
/* Computing 2nd power */
                        r__1=w[it] *(term-y[it]);
                        term=r__1*r__1;
                        fpart += term;
                        if(new__==0){
                goto L180;
                        }
                        store=term*half;
                        fpint[i__]=fpart-store;
                        ++i__;
                        fpart=store;
                        new__=0;
L180:
                        ;
        }
        fpint[nrint]=fpart;
        i__3=nplus;
        for(l=1;l<=i__3;++l){
/*  add a new knot. */
```

```
                        fpknot_(&x[1],m,&t[1],n,&fpint[1],&nrdata[1],&nrint,nest,&
                                c__1);
/*  if n=nmax we locate the knots as for interpolation. */
                        if(*n==nmax){
                goto L10;
                        }
/*  test whether we cannot further increase the number of knots.
*/
                        if(*n==*nest){
                goto L200;
                        }
/* L190: */
        }
/*  restart the computations with the new set of knots. */
L200:
        ;
                }
/*  test whether the least-squares kth degree polynomial is a solution */
/*  of our approximation problem. */
L250:
                if(*ier==-2){
        goto L440;
                }

/*  part 2: determination of the smoothing spline sp(x).                c

/*  evaluate the discontinuity jump of the kth derivative of the */
/*  b-splines at the knots t(l),l=k+2,...n-k-1 and store in b. */
                fpdisc_(&t[1],n,k2,&b[b_offset],nest);
/*  initial value for p. */
                p1=0.0;
                f1=fp0-*s;
                p3=- one;
                f3=fpms;
                p=0.0;
                i__1=nk1;
                for(i__=1;i__<=i__1;++i__){
        p += a[i__+a_dim1];
/* L255: */
                }
                rn =(double) nk1;
                p=rn / p;
                ich1=0;
                ich3=0;
                n8=*n-nmin;
/*  iteration process to find the root of f(p)=s. */
                i__1=*maxit;
                for(iter=1;iter<=i__1;++iter){
/*  the rows of matrix b with weight 1/p are rotated into the */
/*  triangularised observation matrix a which is stored in g. */
        pinv=one / p;
```

```
        i__3=nk1;
        for(i__=1;i__<=i__3;++i__){
                        c__[i__]=z__[i__];
                        g[i__+*k2*g_dim1]=0.0;
                        i__2=*k1;
                        for(j=1;j<=i__2;++j){
                g[i__+j*g_dim1]=a[i__+j*a_dim1];
/* L260: */
                        }
        }
        i__2=n8;
        for(it=1;it<=i__2;++it){
/*  the row of matrix b is rotated into triangle by givens transfo
rmation */
                        i__3=*k2;
                        for(i__=1;i__<=i__3;++i__){
                h__[i__-1]=b[it+i__*b_dim1]*pinv;
/* L270: */
                        }
                        yi=0.0;
                        i__3=nk1;
                        for(j=it;j<=i__3;++j){
                piv=h__[0];
/*  calculate the parameters of the givens transformation. */
                fpgivs_(&piv,&g[j+g_dim1],&cos__,&sin__);
/*  transformations to right hand side. */
                fprota_(&cos__,&sin__,&yi,&c__[j]);
                if(j==nk1){
                                goto L300;
                }
                i2=*k1;
                if(j>n8){
                                i2=nk1-j;
                }
                i__4=i2;
                for(i__=1;i__<=i__4;++i__){
/*  transformations to left hand side. */
                                i1=i__+1;
                                fprota_(&cos__,&sin__,&h__[i1-1],&g[j+i1*g_dim1])
                                        ;
                                h__[i__-1]=h__[i1-1];
/* L280: */
                }
                h__[i2]=0.0;
/* L290: */
                        }
L300:
                        ;
        }
/*  backward substitution to obtain the b-spline coefficients. */
        fpback_(&g[g_offset],&c__[1],&nk1,k2,&c__[1],nest);
```

```
/* computation of f(p). */
        *fp=0.0;
        l=*k2;
        i__2=*m;
        for(it=1;it<=i__2;++it){
                        if(x[it]<t[l] || l>nk1){
                goto L310;
                        }
                        ++l;
L310:
                        l0=l-*k2;
                        term=0.0;
                        i__3=*k1;
                        for(j=1;j<=i__3;++j){
                ++l0;
                term += c__[l0]*q[it+j*q_dim1];
/* L320: */
                        }
/* Computing 2nd power */
                        r__1=w[it] *(term-y[it]);
                        *fp += r__1*r__1;
/* L330: */
        }
/* test whether the approximation sp(x) is an acceptable solution. */
        fpms=*fp-*s;
        if(fabs(fpms)<acc){
                        goto L440;
        }
/* test whether the maximal number of iterations is reached. */
        if(iter==*maxit){
                        goto L400;
        }
/* carry out one more step of the iteration process. */
        p2=p;
        f2=fpms;
        if(ich3 != 0){
                        goto L340;
        }
        if(f2-f3>acc){
                        goto L335;
        }
/* our initial choice of p is too large. */
        p3=p2;
        f3=f2;
        p *= con4;
        if(p<=p1){
                        p=p1*con9+p2*con1;
        }
        goto L360;
L335:
        if(f2<0.0){
```

```
                        ich3=1;
        }
L340:
        if(ich1 != 0){
                        goto L350;
        }
        if(f1-f2>acc){
                        goto L345;
        }
/*  our initial choice of p is too small */
        p1=p2;
        f1=f2;
        p /= con4;
        if(p3<0.0){
                        goto L360;
        }
        if(p>=p3){
                        p=p2*con1+p3*con9;
        }
        goto L360;
L345:
        if(f2>0.0){
                        ich1=1;
        }
/*  test whether the iteration process proceeds as theoretically */
/*  expected. */
L350:
        if(f2>=f1 || f2<=f3){
                        goto L410;
        }
/*  find the new value for p. */
        p=fprati_(&p1,&f1,&p2,&f2,&p3,&f3);
L360:
        ;
                }
/*  error codes and messages. */
L400:
                *ier=3;
                goto L440;
L410:
                *ier=2;
                goto L440;
L420:
                *ier=1;
                goto L440;
L430:
                *ier=-1;
L440: ;
} /* fpcurf_ */

void fpdisc_(double* t,int* n,int* k2,double* b,int* nest){
```

```
        /* System generated locals */
        int b_dim1,b_offset,i__1,i__2,i__3;

        /* Local variables */
        static double prod,h__[12];
        static int i__,j,k,l,nrint,k1;
        static double an;
        static int ik,jk,lj,lk,lp,nk1;
        static double fac;
        static int lmk;

        /* Parameter adjustments */
        --t;
        b_dim1 = *nest;
        b_offset=b_dim1+1;
        b -= b_offset;

        /* Function Body */
        k1 = *k2-1;
        k=k1-1;
        nk1 = *n-k1;
        nrint=nk1-k;
        an =(double) nrint;
        fac=an /(t[nk1+1]-t[k1]);
        i__1 =nk1;
        for(l = *k2;l <=i__1;++l){
    lmk=l-k1;
    i__2=k1;
    for(j=1;j <=i__2;++j){
                ik=j+k1;
                lj=l+j;
                lk=lj-*k2;
                h__[j-1]=t[l]-t[lk];
                h__[ik-1]=t[l]-t[lj];
/* L10: */
    }
    lp=lmk;
    i__2=*k2;
    for(j=1;j <=i__2;++j){
                jk=j;
                prod=h__[j-1];
                i__3=k;
                for(i__=1;i__ <=i__3;++i__){
            ++jk;
            prod=prod*h__[jk-1]*fac;
/* L20: */
                }
                lk=lp+k1;
                b[lmk+j*b_dim1] =(t[lk]-t[lp]) / prod;
                ++lp;
/* L30: */
```

```
        }
/* L40: */
                }

} /* fpdisc_ */

void fpgivs_(double* piv,double* ww,double* cos__,double* sin__){
                /* System generated locals */
                double r__1;
                /* Local variables */
                static double store,dd,one;

                one=1.0;
                store=fabs(*piv);
                if(store>=*ww){
/* Computing 2nd power */
        r__1=*ww / *piv;
        dd=store*sqrt(one+r__1*r__1);
                }
                if(store<*ww){
/* Computing 2nd power */
        r__1=*piv / *ww;
        dd=*ww*sqrt(one+r__1*r__1);
                }
                *cos__=*ww / dd;
                *sin__=*piv / dd;
                *ww=dd;

} /* fpgivs_ */

void fpknot_(double* x,int* m,double* t,int* n,double* fpint,int* nrdata,
int* nrint,int* nest,int* istart){
                /* System generated locals */
                int i__1;

                /* Local variables */
                static int next,j,k,ihalf;
                static double fpmax;
                static int maxpt;
                static double am,an;
                static int jj,jk,jbegin,maxbeg,number,jpoint,nrx;

                /* Parameter adjustments */
                --x;
                --nrdata;
                --fpint;
                --t;

                /* Function Body */
                k =(*n-*nrint-1) / 2;
/*  search for knot interval t(number+k)<=x<=t(number+k+1) where */
```

```
/*  fpint(number) is maximal on the condition that nrdata(number) */
/*  not equals zero. */
                fpmax=0.0;
                jbegin=*istart;
                i__1=*nrint;
                for(j=1;j<=i__1;++j){
        jpoint=nrdata[j];
        if(fpmax>=fpint[j] || jpoint==0){
                        goto L10;
        }
        fpmax=fpint[j];
        number=j;
        maxpt=jpoint;
        maxbeg=jbegin;
L10:
        jbegin=jbegin+jpoint+1;
/* L20: */
                }
/*  let coincide the new knot t(number+k+1) with a data point x(nrx) */
/*  inside the old knot interval t(number+k)<=x<=t(number+k+1). */
                ihalf=maxpt / 2+1;
                nrx=maxbeg+ihalf;
                next=number+1;
                if(next>*nrint){
        goto L40;
                }
/*  adjust the different parameters. */
                i__1=*nrint;
                for(j=next;j<=i__1;++j){
        jj=next+*nrint-j;
        fpint[jj+1]=fpint[jj];
        nrdata[jj+1]=nrdata[jj];
        jk=jj+k;
        t[jk+1]=t[jk];
/* L30: */
                }
L40:
                nrdata[number]=ihalf-1;
                nrdata[next]=maxpt-ihalf;
                am=maxpt;
                an=nrdata[number];
                fpint[number]=fpmax*an / am;
                an=nrdata[next];
                fpint[next]=fpmax*an / am;
                jk=next+k;
                t[jk]=x[nrx];
                ++(*n);
                ++(*nrint);

} /* fpknot_ */
```

```
double fprati_(double* p1,double* f1,double* p2,double* f2,double* p3,double* f3){
                /* System generated locals */
                double ret_val;

                /* Local variables */
                static double p,h1,h2,h3;

                if(*p3 >0.0){
        goto L10;
                }
/*  value of p in case p3=infinity. */
                p =(*p1 *(*f1-*f3)**f2-*p2 *(*f2-*f3)**f1) /((*f1-*f2)**
                        f3);
                goto L20;
/*  value of p in case p3 ^= infinity. */
L10:
                h1=*f1 *(*f2-*f3);
                h2=*f2 *(*f3-*f1);
                h3=*f3 *(*f1-*f2);
                p=-(double)(*p1**p2*h3+*p2**p3*h1+*p3**p1*h2)/(*
                        p1*h1+*p2*h2+*p3*h3);
/*  adjust the value of p1,f1,p3 and f3 such that f1>0 and f3<0. */
L20:
                if(*f2<0.0){
        goto L30;
                }
                *p1=*p2;
                *f1=*f2;
                goto L40;
L30:
                *p3=*p2;
                *f3=*f2;
L40:
                ret_val=p;
                return ret_val;
} /* fprati_ */

void fprota_(double* cos__,double* sin__,double* a,double* b){
                static double stor1,stor2;

                stor1=*a;
                stor2=*b;
                *b=*cos__*stor2+*sin__*stor1;
                *a=*cos__*stor1-*sin__*stor2;

} /* fprota_ */

void splev_(double* t,int* n,double*  c__,int* k,double* x,double* y,
int* m,int* ier){
                /* System generated locals */
                int i__1,i__2;
```

```
                /* Local variables */
                static double h__[6];
                static int i__,j,l,k1,l1;
                static double tb;
                static int ll;
                static double te,sp;
                extern /* Subroutine */ void fpbspl_();
                static int nk1;
                static double arg;

                /* Parameter adjustments */
                --c__;
                --t;
                --y;
                --x;

                /* Function Body */
                *ier=10;
                if((i__1=*m-1)<0){
        goto L100;
                } else if(i__1==0){
        goto L30;
                } else {
        goto L10;
                }
L10:
                i__1=*m;
                for(i__=2;i__<=i__1;++i__){
        if(x[i__]<x[i__-1]){
                        goto L100;
        }
/* L20: */
                }
L30:
                *ier=0;
/*  fetch tb and te,the boundaries of the approximation interval. */
                k1=*k+1;
                nk1=*n-k1;
                tb=t[k1];
                te=t[nk1+1];
                l=k1;
                l1=l+1;
/*  main loop for the different points. */
                i__1=*m;
                for(i__=1;i__<=i__1;++i__){
/*  fetch a new x-value arg. */
        arg=x[i__];
        if(arg<tb){             //tb i.e. t begin
                        arg=tb;             //te i.e. t end
        }
        if(arg>te){
```

```
                        arg=te;
        }
/* search for knot interval t(l)<=arg<t(l+1) */
L40:
        if(arg<t[l1] || l==nk1){
                        goto L50;
        }
        l=l1;
        l1=l+1;
        goto L40;
/* evaluate the non-zero b-splines at arg. */
L50:
        fpbspl_(&t[1],n,k,&arg,&l,h__);
/* find the value of s(x) at x=arg. */
        sp=0.0;
        ll=l-k1;
        i__2=k1;
        for(j=1;j<=i__2;++j){
                        ++ll;
                        sp += c__[ll]*h__[j-1];
/* L60: */
        }
        y[i__]=sp;
/* L80: */
                }
L100: ;

} /* splev_ */
```

```
/* menus.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int ShowMainMenu(void){
clrscr();
gotoxy(1,5);
textcolor(WHITE);
cprintf("    MAIN MENU\r\n\r\n");
textcolor(LIGHTGRAY);
textbackground(BLACK);
cprintf("    E  Estimate a mill selection function\r\n");
cprintf("    S  Scale an estimated mill selection function\r\n");
cprintf("    B  Simulate a ball mill in an open circuit\r\n");
cprintf("    Q  Quit\r\n\r\n");
gotoxy(1,24);
textcolor(YELLOW);
cprintf("    Please enter your choice: ");
textcolor(LIGHTGRAY);
return 0;
}
```

```
/* introscr.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <dos.h>

int ShowIntroScreen(void){
int i;

clrscr();
gotoxy(1,5);
printf("   Numerical Grinding Optimization Tools in C (NGOTC), MAY 1998\n");
printf("   McGill University\n");
printf("   Department of Mining and Metallurgical Engineering\n");
printf("   Mineral Processing Group\n");
printf("   Developer:  Akbar Farzanegan\n");
printf("   Supervisor:  Prof. André R. Laplante\n\n\n");
printf("                                                    \n\n");
printf("   \n");
printf("   \n");
printf("   \n");
printf("   \n");
printf("   \n\n");
printf("   Please press any key to continue ...");
getch();
return 0;
}
```

```
/* utility.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#include <dir.h>
#include "gets.h"
#include "global.h"

extern char savescr[4096];

int GetFileNames(char *ptrExt);
void PrintText(int x,int y,char* text,int forgcolor);
int HandleEndOfFile(void);
char *GetStringAt(int x,int y,int maxlen);
void GetColOfData(int row,int column,int noSzClasses,int* xOfColumnPtr,
                                                    int maxlen,char* formatStr,char* formatStrLeftJus,
                                                    double minAllowedRange,double
maxAllowedRange,
                                                    double *tempInputPtr);
void PrintErrMsg(void);
double FindMax(const double* vector,int n);
double FindMin(const double* vector,int n);
void Locate(double *xx,unsigned long n,double x,unsigned long *j);
int CheckInput(int column,double input,double min,double max);

void PrintText(int x,int y,char* text,int forgcolor){
gotoxy(x,y);
textcolor(forgcolor);
cprintf(text);
textcolor(LIGHTGRAY);
}

char *GetStringAt(int x,int y,int maxlen){
static char buffer[MAXLEN];

buffer[0]=maxlen-2;
gotoxy(x,y);
return strdup(cgets(buffer));
}

void GetColOfData(int row,int column,int noSzClasses,int* xOfColumnPtr,
                                                    int maxlen,char* formatStr,char* formatStrLeftJus,
                                                    double minAllowedRange,double
maxAllowedRange,
                                                    double *tempInputPtr){
int CheckInput(int column,double input,double min,double max);
enum bool done=FALSE;
enum bool canUseArraysValues=TRUE;
int i,key;
```

```
int status=0;
char *string;

for(row=4;row<noSzClasses+4;row++){
        _setcursortype(_NOCURSOR);
        gotoxy(xOfColumnPtr[column],row);
        if(column>3) clreol();
        else printf("                ");
        gotoxy(xOfColumnPtr[column],row);
        printf(formatStrLeftJus,tempInputPtr[row-3]);
        gotoxy(xOfColumnPtr[column],row);
        _setcursortype(_NORMALCURSOR);
        key=getch();
        if(key==ESC){
                _setcursortype(_NOCURSOR);
                gotoxy(xOfColumnPtr[column],row);
                printf("                ");
                gotoxy(xOfColumnPtr[column],row);
                cprintf(formatStr,tempInputPtr[row-3]);
                break;
        }
        ungetch(key);
        putch(key);
        done=FALSE;
        status=0;
        while(!done){
                _setcursortype(_NORMALCURSOR);
                string=GetStringAt(xOfColumnPtr[column],row,maxlen);
                if(string[0]=='\0'&&canUseArraysValues){
                        gotoxy(xOfColumnPtr[column],row);
                        printf("                ");
                        gotoxy(xOfColumnPtr[column],row);
                        cprintf(formatStr,tempInputPtr[row-3]);
                        _setcursortype(_NOCURSOR);
                        done=TRUE;
                        continue;
                }
                status=sscanf(string,"%lf",&tempInputPtr[row-3]);
                if(status==1){
                        if(CheckInput(column,tempInputPtr[row-3],minAllowedRange,
                                maxAllowedRange)) done=TRUE;
                        else{
                                PrintErrMsg();
                                gotoxy(xOfColumnPtr[column],row);
                                printf("                ");
                                done=FALSE;
                                canUseArraysValues=FALSE;
                        }
                        if(!done) continue;
                        _setcursortype(_NOCURSOR);
                        gotoxy(xOfColumnPtr[column],row);
```

```
                        printf("                ");
                        gotoxy(xOfColumnPtr[column],row);
                        cprintf(formatStr,tempInputPtr[row-3]);
                        if(column==1&&row==4)
                                for(i=5;i<=noSzClasses+3;i++){
                                        tempInputPtr[i-3]=tempInputPtr[i-4]/(double)sqrt(2);
                                        gotoxy(xOfColumnPtr[column],i);
                                        printf("                ");
                                        gotoxy(xOfColumnPtr[column],i);
                                        cprintf(formatStr,tempInputPtr[i-3]);
                                }
                        canUseArraysValues=TRUE;
                        continue;
                }
                else{
                        fflush(stdin);
                        string[0]='\0';
                        PrintErrMsg();
                        _setcursortype(_NOCURSOR);
                        gotoxy(xOfColumnPtr[column],row);
                        printf("                ");
                        done=FALSE;
                        canUseArraysValues=FALSE;
                }
        }
}
_setcursortype(_NORMALCURSOR);
}

void PrintErrMsg(void){
char savescr[810];

gettext(1,10,80,14,savescr);
PrintText(16,11," ┌──────────────────────────────────────────────────────────────────────────────┐ ",WHITE);
PrintText(16,12," | Incorrect input, press any key to continue ... | ",WHITE);
PrintText(16,13," └──────────────────────────────────────────────────────────────────────────────┘ ",WHITE);
_setcursortype(_NOCURSOR);
getch();
gotoxy(20,10);
puttext(1,10,80,14,savescr);
_setcursortype(_NORMALCURSOR);
}

double FindMax(const double* vector,int n){
/* assuming a unit offset for v */
double max=vector[1];
int iter;

for(iter=1;iter<=n;iter++)
```

```
max=max(max,vector[iter]);
return max;
}

double FindMin(const double* vector,int n){
// assuming a unit offset
double min=vector[1];
int iter;

for(iter=1;iter<=n;iter++)
min=min(min,vector[iter]);
return min;
}

void Locate(double *xx,unsigned long n,double x,unsigned long *j){
unsigned long ju,jm,j1;
int ascnd;

j1=0;
ju=n+1;
ascnd=(xx[n]>xx[1]);
while(ju-j1>1){
        jm=(ju+j1) >> 1;
        if(x>xx[jm]==ascnd) j1=jm;
        else      ju=jm;
}
*j=j1;
}

int CheckInput(int column,double input,double min,double max){
int retValue=1;
        if(input<min||input>max) retValue=0;
return retValue;
}

int HandleEndOfFile(void){
char key;

clrscr();
gotoxy(1,24);
textcolor(LIGHTRED);
cprintf("Unexpected EOF before reading all data! Press any key to continue ...");
textcolor(LIGHTGRAY);
getch();
return 0;
}

int GetFileNames(char *ptrExt){
struct ffblk ffblk;
char filename[128]="*.";
char *ptrCurDir;
```

```
int done,counter;

ptrCurDir=getcwd(NULL,MAXPATH);
if(ptrCurDir==NULL){
  clrscr();
  PrintText(5,3,"Error reading current working directory",LIGHTGRAY);
}
strcat(filename,ptrExt);
clrscr();
gotoxy(10,2);
textcolor(WHITE);
cprintf("Data files (%s) in current directory %s\n",filename,ptrCurDir);
textcolor(LIGHTGRAY);
done=findfirst(filename,&ffblk,0);
gotoxy(1,4);
counter=1;
while(!done){
        if(counter<=20)
        gotoxy(1,counter+3);
        else if(counter<=40)  gotoxy(21,counter-20+3);
        else if(counter<=60)  gotoxy(41,counter-40+3);
        else if(counter<=80)  gotoxy(61,counter-60+3);
        else break;
        cprintf("%s",ffblk.ff_name);
        done=findnext(&ffblk);
        counter++;
}
return 0;
}
```

```
/* memmange.c */
#include <stdio.h>
#include <stdlib.h>
#define NR_END 1
#define FREE_ARG char*

typedef struct{
        double l;
} ballmill;

typedef struct {
        double  Dc;  /* inside diameter of the cyclone */
        double  Di;  /* inside diameter of the cyclone inlet */
        double  Do;  /* inside diameter of the cyclone overflow or vortex finder */
        double  Du;  /* inside diameter of the cyclone underflow or apex orifice */
        double  h;   /* free vortex height of the cyclone */
} cyclone;

ballmill *CreateVectorBM(long nl,long nh);
void FreeVectorBM(ballmill *v,long nl,long nh);
cyclone *CreateVectorCYC(long nl,long nh);
void FreeVectorCYC(cyclone *v,long nl,long nh);
void FatalError(char errorText[]);
int *CreateVectorInt(long nl,long nh);
void FreeVectorInt(int *v,long nl,long nh);
float *CreateVector(long nl,long nh);
void FreeVector(float *v,long nl,long nh);
double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
int **CreateMatrixInt(long nrl,long nrh,long ncl,long nch);
void FreeMatrixInt(int **m,long nrl,long nrh,long ncl,long nch);
float **CreateMatrix(long nrl,long nrh,long ncl,long nch);
void FreeMatrix(float **m,long nrl,long nrh,long ncl,long nch);
double **CreateMatrixD(long nrl,long nrh,long ncl,long nch);
void FreeMatrixD(double **m,long nrl,long nrh,long ncl,long nch);

ballmill *CreateVectorBM(long nl,long nh){
ballmill *v;

v=(ballmill *) malloc((size_t) ((nh-nl+1+NR_END)*sizeof(ballmill)));
if(!v)
        FatalError("Memory allocation failure in CreateVectorBM()");
return v-nl+NR_END;
}

void FreeVectorBM(ballmill *v,long nl,long nh){
free((FREE_ARG) (v+nl-NR_END));
}

cyclone *CreateVectorCYC(long nl,long nh){
cyclone *v;
```

```
v=(cyclone *) malloc((size_t) ((nh-nl+1+NR_END)*sizeof(cyclone)));
if(!v)
        FatalError("Memory allocation failure in CreateVectorCYC()");
return v-nl+NR_END;
}

void FreeVectorCYC(cyclone *v,long nl,long nh){
free((FREE_ARG) (v+nl-NR_END));
}


int *CreateVectorInt(long nl,long nh){
int *v;

v=(int *) malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
if(!v)
        FatalError("Memory allocation failure in CreateVectorInt()");
return v-nl+NR_END;
}

void FreeVectorInt(int *v,long nl,long nh){
free((FREE_ARG) (v+nl-NR_END));
}

float *CreateVector(long nl,long nh){
float *v;

v=(float *) malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
if(!v)
        FatalError("Memory allocation failure in CreateVector()");
return v-nl+NR_END;
}

void FreeVector(float *v,long nl,long nh){
free((FREE_ARG) (v+nl-NR_END));
}

double *CreateVectorD(long nl,long nh){
double *v;

v=(double *) malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
if(!v)
        FatalError("Memory allocation failure in CreateVectorD()");
return v-nl+NR_END;
}

void FreeVectorD(double *v,long nl,long nh){
free ((FREE_ARG) (v+nl-NR_END));
}

int **CreateMatrixInt(long nri,long nrh,long ncl,long nch){
```

```
long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
int **m;

/* allocate pointers to rows */
m=(int **) malloc((size_t) ((nrow+NR_END) *sizeof(int*)));
if (!m)
        FatalError("Memory allocation failure 1 in CreateMatrix()");
m+=NR_END;
m-=nrl;
/* allocate rows and set pointers to them */
m[nrl]=(int *) malloc((size_t) ((nrow*ncol+NR_END)*sizeof(int)));
if(!m[nrl])
        FatalError("Memory allocation failure 2 in CreateMatrix()");
m[nrl]+=NR_END;
m[nrl]-=ncl;
for(i=nrl+1;i<=nrh;i++)
        m[i]=m[i-1]+ncol;
/* return pointer to array of pointers to rows */
return m;
}

void FreeMatrixInt(int **m,long nrl,long nrh,long ncl,long nch){
free((FREE_ARG) (m[nrl]+ncl-NR_END));
free((FREE_ARG) (m+nrl-NR_END));
}

float **CreateMatrix(long nrl,long nrh,long ncl,long nch){
long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
float **m;

/* allocate pointers to rows */
m=(float **) malloc((size_t)((nrow+NR_END) *sizeof(float*)));
if(!m)
        FatalError("Memory allocation failure 1 in CreateMatrix()");
m+=NR_END;
m-=nrl;
/* allocate rows and set pointers to them */
m[nrl]=(float *) malloc((size_t) ((nrow*ncol+NR_END)*sizeof(float)));
if(!m[nrl])
        FatalError("Memory allocation failure 2 in CreateMatrix()");
m[nrl]+=NR_END;
m[nrl]-=ncl;
for(i=nrl+1;i<=nrh;i++)
        m[i]=m[i-1]+ncol;
/* return pointer to array of pointers to rows */
return m;
}

void FreeMatrix(float **m,long nrl,long nrh,long ncl,long nch){
free((FREE_ARG) (m[nrl]+ncl-NR_END));
free((FREE_ARG) (m+nrl-NR_END));
```

```
}

double **CreateMatrixD(long nrl,long nrh,long ncl,long nch){
long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
double **m;

/* allocate pointers to rows */
m=(double **) malloc((size_t) ((nrow+NR_END) *sizeof(double*)));
if(!m)
        FatalError("Memory allocation failure 1 in CreateMatrixD()");
m+=NR_END;
m-=nrl;
/* allocate rows and set pointers to them */
m[nrl]=(double *) malloc((size_t) ((nrow*ncol+NR_END)*sizeof(double)));
if(!m[nrl])
        FatalError("Memory allocation failure 2 in CreateMatrixD()");
m[nrl]+=NR_END;
m[nrl]-=ncl;
for(i=nrl+1;i<=nrh;i++)
m[i]=m[i-1]+ncol;
/* return pointer to array of pointers to rows */
return m;
}

void FreeMatrixD(double **m,long nrl,long nrh,long ncl,long nch){
free((FREE_ARG) (m[nrl]+ncl-NR_END));
free((FREE_ARG) (m+nrl-NR_END));
}

void FatalError(char errorText[]){
fprintf(stderr,"%s\n",errorText);
fprintf(stderr,"exit ...\n");
exit(EXIT_FAILURE);
}
```

```
/* ngotc.h */
extern unsigned _stklen = 8192;

int AcquireData (void);
int SaveData(void);
int BallMill(void);
int ShowMainMenu(void);
int SimulateCircuit(void);
int HelpMainMenu(void);
int WriteToScreen(void);
int WriteToFile(void);
int WriteToPrinter(void);
int GetDataFromUser(void);
int ExitProgram(void);
```

```
/* gets.h */

#define MAXLEN 7  /* Maximum string size (+3) : 4 characters can be entered
                     at keyboard */

char *GetStringAt(int x, int y, int maxlen);
```

```
/* global.h */

#define ESC 27
#define MAXSIZECLASSNO 20

typedef enum bool {FALSE=0,TRUE=1} bool;
typedef enum operationmode{SELECTIONFUNCEST=1,SIMULATION=2,BALLSIZEOPT=3}
                                                    operationmode;

extern char projectTitle[81];
extern double *screenSize;
extern double *feedSizeDistribution;
extern double *selectionFunction;
extern double *dischargeSizeDistribution;
extern bool normalisableBreakageFunction;
extern double **breakageFunction;
extern double *diag;
extern double **t;
extern double **tinv;
extern double **tdiag;
extern double **trans;

extern int sizeClassNumSimGrCir;
extern double *ptrTauPFSimGrCir;
extern double *ptrTauLPMSimGrCir;
extern double *ptrTauSPMSimGrCir;
extern double *ptrRefMillFeedRateSimGrCir;
extern double *ptrCurMillFeedRateSimGrCir;
extern char *projectTitlePtr;
extern bool *normalisableBreakageFunctionPtr;
extern bool savedData;

extern char projectTitleEst[];
extern bool normBreakFuncEst;
extern int sizeClassNumSelecFuncEst;
extern double *ptrScreenSizeEst;
extern double *ptrFeedEst;
extern double *ptrDischargeMeasuredEst;
extern double *ptrDischargeCalcEst;
extern double **ptrBreakFuncEst;
extern double *ptrSelectionFunctionEst;
extern double *ptrTauPFSelecFuncEst;
extern double *ptrTauLPMSelecFuncEst;
extern double *ptrTauSPMSelecFuncEst;
extern double *ptrRefMillFeedRateSFEst;
extern double *ptrCurMillFeedRateSFEst;
extern double *ptrDiag;
extern double **ptrT;
extern double **ptrTinv;
extern double **ptrTdiag;
extern double **ptrTrans;
```

```
extern double *ptrFactorK,*ptrIniS, *ptrDifS;
extern int *ptrK;
extern char projectTitleBallSzOpt[];
extern int sizeClassNumBallSzOpt;
extern double *ptrScreenSizeBallSzOpt;
extern double *ptrSelecFuncBallSzOpt;
extern double *ptrStandardDevBallSzOpt;
```

```
/* bmcs.c */
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include <math.h>
#include <process.h>

/* code for various physical and conceptual units in the circuits */
#define BM_UNIT_CODE          1
#define HCYC_UNIT_CODE        2
#define JUNC_NODE_CODE        3
#define SPLIT_NODE_CODE       4
#define FIXCLASS_NODE_CODE    5
#define CNVRG_BLOCK_CODE      100

/* constants */
#define EPS  0.000001
#define rho 1 /* water density as default for suspending fluid density */
#define mu  1 /* water viscosity as default for suspending liquid viscosity */

typedef enum bool {FALSE=0,TRUE=1} bool;

int *CreateVectorInt(long nl,long nh);
void FreeVectorInt(int *v,long nl,long nh);
int **CreateMatrixInt(long nrl,long nrh,long ncl,long nch);
void FreeMatrixInt(int **m,long nrl,long nrh,long ncl,long nch);
int ReadCircuitSpec(int *cirNo);
int ReadNumsOfNodesStrms(int cirNo,int *nodeNums,int *streamNums);
int ReadSimulationStatus(int *curNodeNumPtr,int cnvrgNodeNums,bool *convergence,int
*compIterPtr);
int UpdateSimulationStatus(int curNodeNum,int cnvrgNodeNums,bool *convergence,int compIter);
int ReadConnectivityMatrix(int **conMatPtr,int cirNo);
int ReadStreamMatrix(int streamNums,double **strmMatPtr);
int ReadStreamsSizeDist(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int UpdateStrmSizeMat(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int ReadCycloneNums(int **conMatPtr,int nodeNums,int *cycPackNums);
int GetFileNames(char *ptrExt);
int SelectInputFile(char *dFileName,char *extPtr);
int ReadInputFile(char *dFileName);
int CreateInputFile(char *dFileName);
int CreateSimuStatusFile(int cnvrgNodeNums);
int CreateStrmsMatFile(void);
int CreateStrmsSizeMatFile(void);
int CreateCycParamFile(int cycPackNums);
int WriteReport(void);
int CheckIfExeCanProceed(void);
char *GetStringAt(int x,int y,int maxlen);

int main(){
```

```
char str[32],dFileName[15],extPtr[4],*strPtr;
int **conMatPtr;
int i,j,iter,curNodeNum;
int cirNo,nodeNums,streamNums;
int cycPackNums,cnvrgNodeNums;
int retVal;
long int t;
bool done;
bool *cnvrgVecPtr;

clrscr();
gotoxy(1,5);
printf("   BALL MILLING CIRCUITS SIMULATOR (BMCS), MAY 1998\n");
printf("   McGill University\n");
printf("   Dept. of Mining and Metallurgical Engineering\n");
printf("   Mineral Processing Group\n");
printf("   Developer:  Akbar Farzanegan\n");
printf("   Supervisor:  Prof. André R. Laplante\n\n\n");
printf("                              STEP 1                    \n\n");
printf("   At this step a circuit flowsheet must be selected by the user. \n");
printf("   The number of the selected circuit is passed to the simulator \n");
printf("   which identifies the circuit in a text file named 'conmat.lib'. \n");
printf("   This file must be in the same directory that the 'bmcs.exe' \n");
printf("   file exists.\n\n");
printf("   Please press any key to continue ... ");
getch();
system("circuits.exe");
ReadCircuitSpec(&cirNo);
if(cirNo==14||cirNo==15||cirNo==19||cirNo==20){
        clrscr();
        gotoxy(1,5);
        printf("    This program cannot be used to simulate the selected\n");
        printf("    circuit.  Only circuits that include ball mills or/and\n");
        printf("    hydrocyclones can be simulated.\n\n");
        printf("    Exit BMCS to DOS ...");
        gotoxy(1,24);
        exit(EXIT_FAILURE);
}
else if(cirNo>=21&&cirNo<=48){
        clrscr();
        gotoxy(1,5);
        printf("    This program cannot be used to simulate the selected\n");
        printf("    circuit.  Only circuits that include ball mills or/and\n");
        printf("    hydrocyclones can be simulated.\n\n");
        printf("    Exit BMCS to DOS ...");
        gotoxy(1,24);
        exit(EXIT_FAILURE);
}
else if(cirNo==50||cirNo==51||cirNo==55){
        clrscr();
        gotoxy(1,5);
```

```
        printf("    This program cannot be used to simulate the selected\n");
        printf("    circuit.  Only circuits that include ball mills or/and\n");
        printf("    hydrocyclones can be simulated.\n\n");
        printf("    Exit BMCS to DOS ...");
        gotoxy(1,24);
        exit(EXIT_FAILURE);
}
/* read number of nodes and streams from the selected circuit connectivity
        matrix */
ReadNumsOfNodesStrms(cirNo,&nodeNums,&streamNums);
clrscr();
gotoxy(1,5);
printf("                                STEP 2                  \n\n");
printf("    To simulate the selected circuit, a data file must be created\n");
printf("    with the extension c## and saved under the same directory that\n");
printf("    'bmcs.exe' file exists.  The data file format must be consistent \n");
printf("    with the circuit discription given in the 'conmat.lib' file.\n\n");

printf("    Do you want the edit.com program to be run to create\n");
printf("    a new data file or modify an existing one (y/n)? ");
strPtr=GetStringAt(54,13,4);
while(strPtr[0]!='y'&&strPtr[0]!='Y'&&strPtr[0]!='N'&&strPtr[0]!='n'){
        gotoxy(54,13);
        clreol();
        strPtr=GetStringAt(54,13,4);
}
if(strPtr[0]=='Y'||strPtr[0]=='y')
        system("edit.com /b untitled.c##");
/* make extension string */
strcpy(extPtr,"c");
itoa(cirNo,str,10);
strcat(extPtr,str);
/* read data file prepared by the user ********.c## */
/* create initial files */
retVal=1;
while(retVal!=0){
        GetFileNames(extPtr);
        SelectInputFile(dFileName,extPtr);
        retVal=CreateInputFile(dFileName);
}
CreateStrmsMatFile();
CreateStrmsSizeMatFile();
/* allocate exact memory needed for reading connectivity matrix */
conMatPtr=CreateMatrixInt(1,nodeNums,1,streamNums+3);
for(i=1;i<=nodeNums;i++)
        for(j=1;j<=streamNums+3;j++)
                conMatPtr[i][j]=0.00;
ReadConnectivityMatrix(conMatPtr,cirNo);
cnvrgNodeNums=0;
for(i=1;i<=nodeNums;i++)
        if(conMatPtr[i][2]==CNVRG_BLOCK_CODE) cnvrgNodeNums++;
```

```
CreateSimuStatusFile(cnvrgNodeNums);
cnvrgVecPtr=(bool *) CreateVectorInt(1,cnvrgNodeNums);
for(i=1;i<=cnvrgNodeNums;i++)
        cnvrgVecPtr[i]=0;
ReadSimulationStatus(&curNodeNum,cnvrgNodeNums,cnvrgVecPtr,&iter);
ReadCycloneNums(conMatPtr,nodeNums,&cycPackNums);
CreateCycParamFile(cycPackNums);
clrscr();
gotoxy(1,5);
printf("    PLEASE WAIT...");
gotoxy(1,7);
printf("    ITERATION ");
gotoxy(58,24);
printf("(Press Ctrl+C to quit)");
done=FALSE;
while(!done){
        gotoxy(15,7);
        printf("%3d",iter);
                for(i=1;i<=nodeNums;i++){
                        switch(conMatPtr[i][2]){
                                case BM_UNIT_CODE:
                                gotoxy(1,9);
                                clreol();
                                printf("    running ballmill.exe .....");
                                system("ballmill.exe");
                                CheckIfExeCanProceed();
                                break;

                                case HCYC_UNIT_CODE:
                                gotoxy(1,9);
                                clreol();
                                printf("    running cyclone.exe .....");
                                system("cyclone.exe");
                                CheckIfExeCanProceed();
                                break;

                                case JUNC_NODE_CODE:
                                gotoxy(1,9);
                                clreol();
                                printf("    running junction.exe .....");
                                system("junction.exe");
                                CheckIfExeCanProceed();
                                break;

                                case SPLIT_NODE_CODE:
                                gotoxy(1,9);
                                clreol();
                                printf("    running split.exe .....");
                                system("    split.exe");
                                CheckIfExeCanProceed();
                                break;
```

```
                                case FIXCLASS_NODE_CODE:
                                gotoxy(1,9);
                                clreol();
                                printf("    running fixclass.exe .....");
                                system("    fixclass.exe");
                                CheckIfExeCanProceed();
                                break;

                                case CNVRG_BLOCK_CODE:
                                gotoxy(1,9);
                                clreol();
                                printf("    running converge.exe ...");
                                system("converge.exe");
                                CheckIfExeCanProceed();
                                break;
                        }
                }
if(cnvrgNodeNums==0) break; /* if there is no convergence node, go out of while loop */
else{
        ReadSimulationStatus(&curNodeNum,cnvrgNodeNums,cnvrgVecPtr,&iter);
        for(i=1;i<=cnvrgNodeNums;i++)
                if(cnvrgVecPtr[i]) done=TRUE;
                else{
                        done=FALSE;
                        break; /* if encounter the first non-converged block do not check others,
                                                go out of for loop */
                }
}
iter++;
UpdateSimulationStatus(curNodeNum,cnvrgNodeNums,cnvrgVecPtr,iter);
}
WriteReport();
/* clear Ctrl+C reminder  */
gotoxy(1,24);
clreol();
/* inform the user that simulation has been completed */
gotoxy(1,12);
printf("                                STEP 3                          \n\n");
printf("    The simulation was completed. The output file has been saved\n");
printf("    under the name 'bmcs.out'.\n\n");
printf("    Do you want to see the file (y/n)? ");

strPtr=GetStringAt(40,17,4);
while(strPtr[0]!='y'&&strPtr[0]!='Y'&&strPtr[0]!='N'&&strPtr[0]!='n'){
        gotoxy(40,17);
        clreol();
        strPtr=GetStringAt(40,17,4);
}
if(strPtr[0]=='Y'||strPtr[0]=='y') system("edit.com /b bmcs.out");
clrscr();
gotoxy(1,5);
```

```
printf("    Exit to DOS ...");
for(t=1;t<=4000000;t++) ;
clrscr();
return 0;
}
```

```
/* circuits.c01 */
typedef enum bool {FALSE=0,TRUE=1} bool;
#ifdef __TINY__
#error BGIDEMO will not run in the tiny model.
#endif

#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

#include <graphics.h>

#define MAXLEN  20
#define TRUE    1               /* Define some handy constants   */
#define FALSE   0               /* Define some handy constants   */
#define PI      3.14159         /* Define a value for PI         */
#define ON      1                       /* Define some handy constants   */
#define OFF     0                       /* Define some handy constants   */

#define NFONTS 11

#define ESC  27

#define MILL_LENGTH  50
#define MILL_DIAMETER 35
#define CYC_TOP_H     12
#define CYC_DIAMETER  24
#define CYC_H  40

#define H_SPC  55
#define SMALL_H_SPC 15
#define V_SPC   90
#define SMALL_V_SPC 20

#define ArrowW  6
#define ArrowL  8


char *Fonts[NFONTS]={
  "DefaultFont",  "TriplexFont",  "SmallFont",
  "SansSerifFont","GothicFont","ScriptFont","SimplexFont","TriplexScriptFont",
  "ComplexFont","EuropeanFont","BoldFont"
};

char *LineStyles[]={
  "SolidLn", "DottedLn", "CenterLn", "DashedLn", "UserBitLn"
};
```

```
char *FillStyles[]={
  "EmptyFill", "SolidFill",    "LineFill",    "LtSlashFill",
  "SlashFill", "BkSlashFill",  "LtBkSlashFill","HatchFill",
  "XHatchFill","InterleaveFill","WideDotFill", "CloseDotFill"
};

int    GraphDriver;              /* The Graphics device driver              */
int    GraphMode;                /* The Graphics mode value                 */
double AspectRatio;              /* Aspect ratio of a pixel on the screen*/
int    MaxX,MaxY;                /* The maximum resolution of the screen */
int    MaxColors;                /* The maximum # of colors available    */
int    ErrorCode;                /* Reports any graphics errors                */
struct palettetype palette;      /* Used to read palette info          */

/*       Function prototypes */
void Initialize(void);
void DisplayCircuits(void);
void MainWindow(char *header);
void StatusLine(char *msg);
void DrawBorder(void);
void changetextstyle(int font,int direction,int charsize);
int  gprintf(int *xloc,int *yloc,char *fmt,... );

void DrawPrimaryMill(int x,int y,int d,int l);
void DrawBallMill(int x,int y,int d,int l);
void DrawCyclone(int x,int y,int d,int l,int h);
void DrawScreen(int x1,int y1,int x2,int y2,int x3,int y3);
void DrawCrusher(int x,int y);
void RightArrow(int x,int y);
void RightArrowW(int x,int y);
void LeftArrow(int x,int y);
void UpArrowW(int x,int y);
void UpArrow(int x,int y);
void DownArrow(int x,int y);
void DownArrowW(int x,int y);

int LoadCircuits(int *cirNo);
void DrawFlowsheet1(void);
void DrawFlowsheet2(void);
void DrawFlowsheet3(void);
void DrawFlowsheet4(void);
void DrawFlowsheet5(int x,int y);
void DrawFlowsheet6(int xPnt,int yPnt);
void DrawFlowsheet7(int xPnt,int yPnt);
void DrawFlowsheet8(int xPnt,int yPnt);
void DrawFlowsheet9(int xPnt,int yPnt);
void DrawFlowsheet10(int xPnt,int yPnt);
void DrawFlowsheet11(int xPnt,int yPnt);
void DrawFlowsheet12(int xPnt,int yPnt);
void DrawFlowsheet13(int xPnt,int yPnt);
void DrawFlowsheet14(int xPnt,int yPnt);
```

```
void DrawFlowsheet15(int xPnt,int yPnt);
void DrawFlowsheet16(int xPnt,int yPnt);
void DrawFlowsheet17(int xPnt,int yPnt);
void DrawFlowsheet18(int xPnt,int yPnt);
void DrawFlowsheet19(int xPnt,int yPnt);
void DrawFlowsheet20(int xPnt,int yPnt);
void DrawFlowsheet21(int xPnt,int yPnt);
void DrawFlowsheet22(int xPnt,int yPnt);
void DrawFlowsheet23(int xPnt,int yPnt);
void DrawFlowsheet24(int xPnt,int yPnt);
void DrawFlowsheet25(int xPnt,int yPnt);
void DrawFlowsheet26(int xPnt,int yPnt);
void DrawFlowsheet27(int xPnt,int yPnt);
void DrawFlowsheet28(int xPnt,int yPnt);
void DrawFlowsheet29(int xPnt,int yPnt);
void DrawFlowsheet30(int xPnt,int yPnt);
void DrawFlowsheet31(int xPnt,int yPnt);
void DrawFlowsheet32(int xPnt,int yPnt);
void DrawFlowsheet33(int xPnt,int yPnt);
void DrawFlowsheet34(int xPnt,int yPnt);
void DrawFlowsheet35(int xPnt,int yPnt);
void DrawFlowsheet36(int xPnt,int yPnt);
void DrawFlowsheet37(int xPnt,int yPnt);
void DrawFlowsheet38(int xPnt,int yPnt);
void DrawFlowsheet39(int xPnt,int yPnt);
void DrawFlowsheet40(int xPnt,int yPnt);
void DrawFlowsheet41(int xPnt,int yPnt);
void DrawFlowsheet42(int xPnt,int yPnt);
void DrawFlowsheet43(int xPnt,int yPnt);
void DrawFlowsheet44(int xPnt,int yPnt);
void DrawFlowsheet45(int xPnt,int yPnt);
void DrawFlowsheet46(int xPnt,int yPnt);
void DrawFlowsheet47(int xPnt,int yPnt);
void DrawFlowsheet48(int xPnt,int yPnt);
void DrawFlowsheet49(int xPnt,int yPnt);
void DrawFlowsheet50(int xPnt,int yPnt);
void DrawFlowsheet51(int xPnt,int yPnt);
void DrawFlowsheet52(int xPnt,int yPnt);
void DrawFlowsheet53(int xPnt,int yPnt);
void DrawFlowsheet54(int xPnt,int yPnt);
void DrawFlowsheet55(int xPnt,int yPnt);
void DrawFlowsheet56(int xPnt,int yPnt);
int WrtOutputFile(int cirNo);
char *GetStringAt(int x,int y,int maxlen);

int main(){
int cirNo;

Initialize();                    /* Set system into Graphics mode      */
LoadCircuits(&cirNo);
WrtOutputFile(cirNo);
```

```
closegraph();                   /* Return the system to text mode       */
return 0;
}

/* INITIALIZE: Initializes the graphics system and reports
        any errors which occured. */

void Initialize(void){
int xasp,yasp;                  /* Used to read the aspect ratio*/

registerbgidriver(EGAVGA_driver);
registerbgifont(sansserif_font);
registerbgifont(small_font);
GraphDriver=DETECT;                     /* Request auto-detection */
initgraph(&GraphDriver,&GraphMode," ");
ErrorCode=graphresult();                /* Read result of initialization*/
if(ErrorCode!= grOk){           /* Error occured during init       */
        printf(" Graphics System Error: %s\n",grapherrormsg(ErrorCode));
        exit(1);
}
getpalette(&palette);           /* Read the palette from board     */
MaxColors=getmaxcolor() + 1;    /* Read maximum number of colors*/
MaxX=getmaxx();
MaxY=getmaxy();                         /* Read size of screen             */
getaspectratio(&xasp,&yasp);    /* read the hardware aspect        */
AspectRatio=(double)xasp /(double)yasp; /* Get correction factor   */
}

/*      CIRCLEDEMO: Display a random pattern of circles on the screen   */
/*      until the user says enough. */
int LoadCircuits(int *cirNo){
char str[5],*strPtr;
int ch,key;
long i;

bool done=FALSE;
MainWindow("GRINDING CIRCUIT FLOWSHEETS");
StatusLine("Previous <P>  Next <N>  GoTo <G>  Escape <Esc>");

*cirNo=1;
if(*cirNo==1)
  DrawFlowsheet1();

while(!done){
        key=getch();
        if(key==ESC){
                closegraph();
                gotoxy(1,5);
                printf("    Please enter the circuit number.  If the circuit flowsheet cannot\n");
                printf("    be found among the pre-defined ones, you have to assign a number\n");
                printf("    to your circuit greater than the number of the last predefined \n");
```

```
                printf("    flowsheet and must also add its connectivity matrix with the correct \n");
                printf("    format to the 'commat.lib' file.\n\n");
                printf("    Circuit number? ");

                strPtr=GetStringAt(21,11,5);
                *cirNo=0;
                while(((sscanf(strPtr,"%i",cirNo))!=1)||(*cirNo<1)){
                        gotoxy(21,11);
                        clreol();
                        strPtr=GetStringAt(21,11,5);
                }
                clrscr();
                gotoxy(1,5);
                printf("    Do you want to return to circuits screens to change the circuit\n");
                printf("    number selected (y/n)? ");
                strPtr=GetStringAt(28,6,4);
                while(strPtr[0]!='y'&&strPtr[0]!='Y'&&strPtr[0]!='N'&&strPtr[0]!='n'){
                        gotoxy(28,6);
                        clreol();
                        strPtr=GetStringAt(28,6,4);
                }
                if(strPtr[0]=='N'||strPtr[0]=='n'){
                        done=TRUE;
                        clrscr();
                        break;
                }
                Initialize();
                MainWindow("GRINDING CIRCUIT FLOWSHEETS");
                StatusLine("Previous <P>  Next <N>  GoTo <G>  Escape <Esc>");
                *cirNo=1;
                DrawFlowsheet1();
        }
        else    if((*cirNo>1)&&(key=='P'||key=='p'))
                        (*cirNo)--;
                else    if((*cirNo<56)&&(key=='N'||key=='n'))
                                (*cirNo)++;
                        else    if(key=='G'||key=='g'){
                                        closegraph();
                                        gotoxy(1,5);
                                        printf("    Please enter the circuit
number? ");
                                        strPtr=GetStringAt(38,5,5);
                                        *cirNo=0;

while(((sscanf(strPtr,"%i",cirNo))!=1)||(*cirNo<1)){
                                                gotoxy(38,5);
                                                clreol();

strPtr=GetStringAt(38,5,5);
                                        }
                                        if(*cirNo>56){
```

```
                                                    gotoxy(1,5);
                                                    clreol();
                                                    printf("    The graphical
representation of the entered circuit number does\n");
                                                    printf("    not
exit!",*cirNo);
                                                    gotoxy(1,9);
                                                    printf("    Press any key
to continue ...");
                                                    getch();
                                                    Initialize();

MainWindow("GRINDING CIRCUIT FLOWSHEETS");
                                                    StatusLine("Previous
<P>  Next <N>  GoTo  <G>  Escape <Esc>");
                                                    *cirNo=1;
                                                }
                                                else{
                                                    Initialize();

MainWindow("GRINDING CIRCUIT FLOWSHEETS");
                                                    StatusLine("Previous
<P>  Next <N>  GoTo  <G>  Escape <Esc>");
                                                }
                                            }
                                            else continue;

        switch(*cirNo){
                case 1:
                DrawFlowsheet1();
                break;

                case 2:
                DrawFlowsheet2();
                break;

                case 3:
                DrawFlowsheet3();
                break;

                case 4:
                DrawFlowsheet4();
                break;

                case 5:
                DrawFlowsheet5(MaxX/2-50,MaxY/2-50);
                break;

                case 6:
                DrawFlowsheet6(MaxX/2,MaxY/2);
                break;
```

```
case 7:
DrawFlowsheet7(MaxX/2-50,MaxY/2-50);
break;

case 8:
DrawFlowsheet8(MaxX/2-50,MaxY/2-50);
break;

case 9:
DrawFlowsheet9(MaxX/2-50,MaxY/2-50);
break;

case 10:
DrawFlowsheet10(MaxX/2-50,MaxY/2-50);
break;

case 11:
DrawFlowsheet11(MaxX/2-50,MaxY/2-40);
break;

case 12:
DrawFlowsheet12(MaxX/2-50,MaxY/2-40);
break;

case 13:
DrawFlowsheet13(MaxX/2+100,MaxY/2-50);
break;

case 14:
DrawFlowsheet14(MaxX/2+150,MaxY/2-50);
break;

case 15:
DrawFlowsheet15(MaxX/2+150,MaxY/2-50);
break;

case 16:
DrawFlowsheet16(MaxX/2+25,MaxY/2-25);
break;

case 17:
DrawFlowsheet17(MaxX/2+25,MaxY/2-25);
break;

case 18:
DrawFlowsheet18(MaxX/2+25,MaxY/2-25);
break;

case 19:
DrawFlowsheet19(MaxX/2+25,MaxY/2-25);
break;
```

```
case 20:
DrawFlowsheet20(MaxX/2+25,MaxY/2-25);
break;

case 21:
DrawFlowsheet21(MaxX/2+25,MaxY/2-25);
break;

case 22:
DrawFlowsheet22(MaxX/2+25,MaxY/2-25);
break;

case 23:
DrawFlowsheet23(MaxX/2+25,MaxY/2-25);
break;

case 24:
DrawFlowsheet24(MaxX/2+25,MaxY/2-25);
break;

case 25:
DrawFlowsheet25(MaxX/2+125,MaxY/2-25);
break;

case 26:
DrawFlowsheet26(MaxX/2+150,MaxY/2-50);
break;

case 27:
DrawFlowsheet27(MaxX/2+150,MaxY/2-50);
break;

case 28:
DrawFlowsheet28(MaxX/2+25,MaxY/2-25);
break;

case 29:
DrawFlowsheet29(MaxX/2+25,MaxY/2-25);
break;

case 30:
DrawFlowsheet30(MaxX/2+25,MaxY/2-25);
break;

case 31:
DrawFlowsheet31(MaxX/2+25,MaxY/2-25);
break;

case 32:
DrawFlowsheet32(MaxX/2+25,MaxY/2-25);
break;
```

```
case 33:
DrawFlowsheet33(MaxX/2+25,MaxY/2-25);
break;

case 34:
DrawFlowsheet34(MaxX/2+25,MaxY/2-25);
break;

case 35:
DrawFlowsheet35(MaxX/2+25,MaxY/2-25);
break;

case 36:
DrawFlowsheet36(MaxX/2+25,MaxY/2-25);
break;

case 37:
DrawFlowsheet37(MaxX/2+125,MaxY/2-25);
break;

case 38:
DrawFlowsheet38(MaxX/2+150,MaxY/2-50);
break;

case 39:
DrawFlowsheet39(MaxX/2+150,MaxY/2-50);
break;

case 40:
DrawFlowsheet40(MaxX/2+25,MaxY/2-25);
break;

case 41:
DrawFlowsheet41(MaxX/2+25,MaxY/2-25);
break;

case 42:
DrawFlowsheet42(MaxX/2+25,MaxY/2-25);
break;

case 43:
DrawFlowsheet43(MaxX/2+25,MaxY/2-25);
break;

case 44:
DrawFlowsheet44(MaxX/2+25,MaxY/2-25);
break;

case 45:
DrawFlowsheet45(MaxX/2+25,MaxY/2-25);
break;
```

```
                case 46:
                DrawFlowsheet46(MaxX/2+25,MaxY/2-25);
                break;

                case 47:
                DrawFlowsheet47(MaxX/2+25,MaxY/2-25);
                break;

                case 48:
                DrawFlowsheet48(MaxX/2+25,MaxY/2-25);
                break;

                case 49:
                DrawFlowsheet49(MaxX/2+150,MaxY/2-50);
                break;

                case 50:
                DrawFlowsheet50(MaxX/2+175,MaxY/2-50);
                break;

                case 51:
                DrawFlowsheet51(MaxX/2+175,MaxY/2-50);
                break;

                case 52:
                DrawFlowsheet52(MaxX/2+75,MaxY/2-50);
                break;

                case 53:
                DrawFlowsheet53(MaxX/2+75,MaxY/2-50);
                break;

                case 54:
                DrawFlowsheet54(MaxX/2+100,MaxY/2-50);
                break;

                case 55:
                DrawFlowsheet55(MaxX/2+100,MaxY/2-50);
                break;

                case 56:
                DrawFlowsheet56(MaxX/2+100,MaxY/2-50);
                break;
                }
        }
return 0;
}

void DrawBallMill(int x,int y,int d,int l){
int i;
```

```
rectangle(x-l/2-(l/5)-(l/7),y-d/2,x-l/2-(l/5),y+d/2);
rectangle(x-l/2-(l/5),y-d/2-(d/5),x-l/2,y+d/2+(d/5));
rectangle(x-l/2,y-d/2,x+l/2,y+d/2);
rectangle(x+l/2,y-d/4,x+l/2+l/10,y+d/4);
for(i=1; i<=11; i++)
        circle(x-l/2+(i*4),y+d/2-3,2);
changetextstyle(DEFAULT_FONT,HORIZ_DIR,0);
settextjustify(CENTER_TEXT,TOP_TEXT);
setusercharsize(6,1,6,1);
outtextxy(x-10,y-2,"BM");
}

void DrawPrimaryMill(int x,int y,int d,int l){
int i;

rectangle(x-l/2-(l/5)-(l/7),y-d/2,x-l/2-(l/5),y+d/2);
rectangle(x-l/2-(l/5),y-d/2-(d/5),x-l/2,y+d/2+(d/5));
rectangle(x-l/2,y-d/2,x+l/2,y+d/2);
rectangle(x+l/2,y-d/4,x+l/2+l/10,y+d/4);
moveto(x-l/2+2,y+d/2-4);
for(i=1;i<=12;i++){
  lineto(getx()+2,gety()+2);
  lineto(getx()+2,gety()-2);
}
moveto(x-l/2+2,y+d/2-7);
for(i=1;i<=12;i++){
  lineto(getx()+2,gety()+2);
  lineto(getx()+2,gety()-2);
}
changetextstyle(DEFAULT_FONT,HORIZ_DIR,0);
settextjustify(CENTER_TEXT,TOP_TEXT);
setusercharsize(6,1,6,1);
outtextxy(x-6,y-2,"PM");
}

void DrawCyclone(int x,int y,int d,int l,int h){
rectangle(x-l/2,y-d/2,x+l/2,y+d/2);
line(x-l/2,y+d/2,x,y+h);
line(x,y+h,x+l/2,y+d/2);
changetextstyle(DEFAULT_FONT,HORIZ_DIR,0);
settextjustify(CENTER_TEXT,TOP_TEXT);
setusercharsize(6,1,6,1);
outtextxy(x-8,y-3,"C");
}

void DrawScreen(int x1,int y1,int x2,int y2,int x3,int y3){
moveto(x1,y1);
setlinestyle(DASHED_LINE,0,1);
lineto(x2,y2);
setlinestyle(SOLID_LINE,0,1);
lineto(x3,y3);
```

```
lineto(x1,y1);
}

void DrawCrusher(int x,int y){
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;
int x6,y6,x7,y7,x8,y8,x9,y9,x10,y10,x11,y11;

x1=x-0.6*SMALL_H_SPC;
y1=y;
x2=x1+0.15*SMALL_H_SPC;
y2=y1+0.6*SMALL_V_SPC;
x3=x2-0.85*SMALL_H_SPC;
y3=y2+0.9*SMALL_V_SPC;
x4=x3+2*(x1+((x-x1)/2)-x3);
y4=y3;
x5=x2+2*(x1+((x-x1)/2)-x2);
y5=y2;

x6=x3-0.15*SMALL_H_SPC;
y6=y3;
x7=x6;
y7=y1+0.2*SMALL_V_SPC;
x8=x2-0.4*SMALL_H_SPC;
y8=y2;

x9=x4+0.15*SMALL_H_SPC;
y9=y4;
x10=x9;
y10=y+0.2*SMALL_V_SPC;
x11=x5+0.4*SMALL_H_SPC;
y11=y5;

moveto(x,y);
lineto(x1,y1);
lineto(x2,y2);
lineto(x3,y3);
lineto(x4,y4);
lineto(x5,y5);
lineto(x,y);

moveto(x6,y6);
lineto(x7,y7);
lineto(x8,y8);
lineto(x6,y6);

moveto(x9,y9);
lineto(x10,y10);
lineto(x11,y11);
lineto(x9,y9);
}
```

```
void RightArrow(int x,int y){
/* our polygon array */
int poly[6];

lineto(x,y);
poly[0]=x;        /* 1st vertex */
poly[1]=y;
poly[2]=x-ArrowL; /* 2nd */
poly[3]=y-ArrowW/2;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly[4]=x-ArrowL; /* 3rd */
poly[5]=y+ArrowW/2;
/* draw a filled polygon */
fillpoly(3,poly);
}

void RightArrowW(int x,int y){
/* our polygon array */
int poly[6];

setlinestyle(DASHED_LINE,0,1);
lineto(x,y);
poly[0]=x;        /* 1st vertex */
poly[1]=y;
poly[2]=x-ArrowL; /* 2nd */
poly[3]=y-ArrowW/2;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly[4]=x-ArrowL; /* 3rd */
poly[5]=y+ArrowW/2;
/* draw a filled polygon */
fillpoly(3,poly);
setlinestyle(SOLID_LINE,0,1);
}

void LeftArrow(int x,int y){
/* our polygon array */
int poly[6];

lineto(x,y);
poly[0]=x;        /* 1st vertex */
poly[1]=y;
poly[2]=x+ArrowL; /* 2nd */
poly[3]=y+ArrowW/2;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly[4]=x+ArrowL; /* 3rd */
poly[5]=y-ArrowW/2;
/* draw a filled polygon */
fillpoly(3,poly);
}

void UpArrow(int x,int y){
```

```
/* our polygon array */
int poly[6];

lineto(x,y);
poly[0]=x;        /* 1st vertex */
poly[1]=y;
poly[2]=x-ArrowW/2; /* 2nd */
poly[3]=y+ArrowL;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly[4]=x+ArrowW/2; /* 3rd */
poly[5]=y+ArrowL;
/* draw a filled polygon */
fillpoly(3,poly);
}

void UpArrowW(int x,int y){
/* our polygon array */
int poly[6];

setlinestyle(DASHED_LINE,0,1);
lineto(x,y);
poly[0]=x;        /* 1st vertex */
poly[1]=y;
poly[2]=x-ArrowW/2; /* 2nd */
poly[3]=y+ArrowL;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly[4]=x+ArrowW/2; /* 3rd */
poly[5]=y+ArrowL;
/* draw a filled polygon */
fillpoly(3,poly);
setlinestyle(SOLID_LINE,0,1);
}

void DownArrow(int x,int y){
/* our polygon array */
int poly[6];
lineto(x,y);
poly[0]=x;        /* 1st vertex */
poly[1]=y;
poly[2]=x+ArrowW/2; /* 2nd */
poly[3]=y-ArrowL;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly[4]=x-ArrowW/2; /* 3rd */
poly[5]=y-ArrowL;
/* draw a filled polygon */
fillpoly(3,poly);
}

void DownArrowW(int x,int y){
/* our polygon array */
int poly[6];
```

```
setlinestyle(DASHED_LINE,0,1);
lineto(x,y);
poly[0]=x;        /* 1st vertex */
poly[1]=y;
poly[2]=x+ArrowW/2; /* 2nd */
poly[3]=y-ArrowL;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly[4]=x-ArrowW/2; /* 3rd */
poly[5]=y-ArrowL;
/* draw a filled polygon */
fillpoly(3,poly);
setlinestyle(SOLID_LINE,0,1);
}

/*      MAINWINDOW: Establish the main window for the demo and set   */
/*      a viewport for the demo code.                                */
/*                                                                   */
void MainWindow(char *header){
int height;

cleardevice();                  /* Clear graphics screen  */
setcolor(LIGHTCYAN);            /* Set current color to white      */
setviewport(0,0,MaxX,MaxY,1);   /* Open port to full screen        */

height=5*textheight("H");       /* Get basic text height      */

changetextstyle(DEFAULT_FONT,HORIZ_DIR,0);
settextjustify(CENTER_TEXT,TOP_TEXT);
setusercharsize(6,1,6,1);
outtextxy(MaxX/2-100,25,header);
setcolor(WHITE);
setviewport(0,height+4,MaxX,MaxY-(height+4),1);
DrawBorder();
setviewport(1,height+5,MaxX-1,MaxY-(height+5),1);
}

/*      STATUSLINE: Display a status line at the bottom of the screen.   */
void StatusLine(char *msg){
int height;
setviewport(0,0,MaxX,MaxY,1);  /* Open port to full screen      */
changetextstyle(DEFAULT_FONT,HORIZ_DIR,0);
settextjustify(LEFT_TEXT,TOP_TEXT);
setusercharsize(6,4,6,4);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
setfillstyle(EMPTY_FILL,0);

height=5*textheight("H");       /* Detemine current height      */
bar(0,MaxY-(height+4),MaxX,MaxY);
rectangle(0,MaxY-(height+4),MaxX,MaxY);
setcolor(YELLOW);               /* Set current color to white      */
outtextxy(5,MaxY-(height-10),msg);
```

```
setcolor(WHITE);                    /* Set current color to white      */
setviewport(1,height+5,MaxX-1,MaxY-(height+5),1);
}
/*      DRAWBORDER: Draw a solid single line around the current
        viewport. */

void DrawBorder(void){
struct viewporttype vp;

setcolor(MaxColors-1);              /* Set current color to white      */
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
getviewsettings(&vp);
rectangle(0,0,vp.right-vp.left,vp.bottom-vp.top);
}

/*      CHANGETEXTSTYLE: similar to settextstyle,but checks for        */
/*      errors that might occur whil loading the font file.            */
void changetextstyle(int font,int direction,int charsize){
int ErrorCode;

graphresult();                      /* clear error code                */
settextstyle(font,direction,charsize);
ErrorCode=graphresult();                    /* check result            */
  if(ErrorCode!=grOk){              /* if error occured                */
        closegraph();
        printf(" Graphics System Error: %s\n",grapherrormsg(ErrorCode));
        exit(1);
  }
}

/*      GPRINTF: Used like PRINTF except the output is sent to the     */
/*      screen in graphics mode at the specified co-ordinate.          */
int gprintf(int *xloc,int *yloc,char *fmt,...){
va_list  argptr;                    /* Argument list pointer  */
char str[140];                      /* Buffer to build sting into      */
int cnt;                            /* Result of SPRINTF for return */

va_start(argptr,fmt);               /* Initialize va_ functions        */

cnt=vsprintf(str,fmt,argptr);       /* prints string to buffer  */
outtextxy(*xloc,*yloc,str);         /* Send string in graphics mode */
*yloc += textheight("H") + 2;       /* Advance to next line            */

va_end(argptr);                     /* Close va_ functions             */
return(cnt);                        /* Return the conversion count     */
}

int WrtOutputFile(int cirNo){
FILE *outputStreamPtr;

/* open output file */
```

```
if(!(outputStreamPtr=fopen("circuit.spc","w")))
        printf("cannot creat output file!  Press any key to exit program");
else
        fprintf(outputStreamPtr,"%d",cirNo);
/* close file */
fclose(outputStreamPtr);
return (-1);
}

char *GetStringAt(int x,int y,int maxlen){
static char buffer[MAXLEN];

buffer[0]=maxlen-2;
gotoxy(x,y);
return strdup(cgets(buffer));
}
```

```
/* funcs1.c01 */
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

#define MILL_LENGTH 50
#define MILL_DIAMETER 35
#define CYC_TOP_H 12
#define CYC_DIAMETER 24
#define CYC_H 40

#define H_SPC 55
#define SMALL_H_SPC 15
#define V_SPC 90
#define SMALL_V_SPC 20

extern int MaxX,MaxY;          /* The maximum resolution of the screen */

void changetextstyle(int font,int direction,int charsize);

void DrawPrimaryMill(int x,int y,int d,int l);
void DrawBallMill(int x,int y,int d,int l);
void DrawCyclone(int x,int y,int d,int l,int h);
void DrawScreen(int x1,int y1,int x2,int y2,int x3,int y3);
void DrawCrusher(int x,int y);
void RightArrow(int x,int y);
void RightArrowW(int x,int y);
void LeftArrow(int x,int y);
void UpArrowW(int x,int y);
void UpArrow(int x,int y);
void DownArrow(int x,int y);
void DownArrowW(int x,int y);

void DrawFlowsheet1(void){
/* BM, a single ball mill in open circuit */
int millX,millY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=MaxX/2;
millY=MaxY/2-100;

mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
/* Draw fresh feed */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
/* mill discharge */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
```

```
RightArrow(getx()+H_SPC,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 1");
}

void DrawFlowsheet2(void){
/* C, a single classification unit in open circuit */
int  cycX,cycY;
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
setfillstyle(1,WHITE);
cycX=MaxX/2;
cycY=MaxY/2-100;
DrawCyclone(cycX,cycY,2*CYC_TOP_H,2*CYC_DIAMETER,2*CYC_H);
/* CF */
moveto(cycX-CYC_DIAMETER-H_SPC,cycY);
RightArrow(cycX-CYC_DIAMETER,gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H);
lineto(getx(),cycY-CYC_TOP_H-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF */
moveto(cycX,cycY+2*CYC_H);
DownArrow(getx(),gety()+2.5*SMALL_V_SPC);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 2");
}

void DrawFlowsheet3(void){
/* a JUNCTION node */
int x,y;
int poly1[6],poly2[6];

setlinestyle(SOLID_LINE,0,1);
clearviewport();
setfillstyle(1,WHITE);
x=MaxX/2;
y=MaxY/2-100;
moveto(x-2*H_SPC,y);
RightArrow(x,y);
moveto(x-H_SPC,y-V_SPC);
lineto(x,y);
poly1[0] = x;        /* 1st vertex */
poly1[1] = y;
poly1[2] = poly1[0]-2; /* 2nd */
poly1[3] = poly1[1]-10;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly1[4] = poly1[0]-7; /* 3rd */
poly1[5] = poly1[1]-8;
```

```
/* draw a filled polygon */
fillpoly(3,poly1);
moveto(x-H_SPC,y+V_SPC);
lineto(x,y);
poly2[0] = x;        /* 1st vertex */
poly2[1] = y;
poly2[2] = poly2[0]-7; /* 2nd */
poly2[3] = poly2[1]+8;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly2[4] = poly2[0]-2; /* 3rd */
poly2[5] = poly2[1]+10;
/* draw a filled polygon */
fillpoly(3,poly2);
RightArrow(x+2.5*H_SPC,y);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 3");
}

void DrawFlowsheet4(void){
/* a SPLIT node */
int x,y;
int poly1[6],poly2[6];

setlinestyle(SOLID_LINE,0,1);
clearviewport();
setfillstyle(1,WHITE);
x=MaxX/2;
y=MaxY/2-100;
moveto(x-2.5*H_SPC,y);
RightArrow(x,y);
lineto(x+H_SPC,y-V_SPC);
poly1[0] = getx();       /* 1st vertex */
poly1[1] = gety();
poly1[2] = poly1[0]-8; /* 2nd */
poly1[3] = poly1[1]+7;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly1[4] = poly1[0]-2; /* 3rd */
poly1[5] = poly1[1]+10;
/* draw a filled polygon */
fillpoly(3,poly1);
moveto(x,y);
lineto(x+H_SPC,y+V_SPC);
poly2[0] = getx();       /* 1st vertex */
poly2[1] = gety();
poly2[2] = poly2[0]-8; /* 2nd */
poly2[3] = poly2[1]-7;
/* 3th vertex. fillpoly automatically closes the polygon. */
poly2[4] = poly2[0]-2; /* 3rd */
poly2[5] = poly2[1]-10;
/* draw a filled polygon */
fillpoly(3,poly2);
```

```
moveto(x,y);
RightArrow(x+2*H_SPC,y);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 4");
}

void DrawFlowsheet5(int xPnt,int yPnt){
/* BM-C, a ball mill with classification */
int  cycX,cycY,millX,millY;
int mLoffset;

clearviewport();
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
moveto(millX-mLoffset-H_SPC-SMALL_H_SPC,millY);
RightArrow(millX-mLoffset-SMALL_H_SPC,millY);
RightArrow(millX-mLoffset,millY);
/* water addition */
moveto(millX-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 5");
}

void DrawFlowsheet6(int xPnt,int yPnt){
/* BM-RC, a ball mill with reversed classification */
int cycX,cycY,millX,millY;

clearviewport();
setfillstyle(1,WHITE);
```

```
millX=xPnt+40;
millY=yPnt-40;
cycX=millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC/2);
lineto(cycX-CYC_DIAMETER/2-2*SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
moveto(getx()-H_SPC,cycY);
RightArrow(getx()+H_SPC,cycY);
moveto(getx(),cycY-V_SPC);
DownArrowW(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 6");
}

void DrawFlowsheet7(int xPnt,int yPnt){
/* BM-TSFC, a ball mill with two stage fine classification */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY;
int mLoffset,mRoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
/* FF, fresh feed */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset-SMALL_H_SPC,millY);
RightArrow(millX-mLoffset,millY);
/* water addition */
moveto(millX-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
```

```
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 7");
}

void DrawFlowsheet8(int xPnt,int yPnt){
/* BM-TSMC, a ball mill with two stage medium classification */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
/* FF */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset-SMALL_H_SPC,millY);
RightArrow(millX-mLoffset,millY);
/* water addition */
moveto(millX-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
```

```
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 8");
}

void DrawFlowsheet9(int xPnt,int yPnt){
/* BM-TSCC, a ball mill with two stage coarse classification */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
/* FF */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset-SMALL_H_SPC,millY);
RightArrow(millX-mLoffset,millY);
/* water addition */
moveto(millX-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
```

```
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 9");
}

void DrawFlowsheet10(int xPnt,int yPnt){
/* BM-RTSFC, a ball mill with reversed two stage fine classification */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY;
int mRoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
```

```
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
moveto(getx(),gety()-V_SPC);
DownArrowW(getx(),gety()+V_SPC);
moveto(getx()-H_SPC,gety());
RightArrow(getx()+H_SPC,gety());
RightArrow(cycX-(CYC_DIAMETER/2),cycY);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 10");
}

void DrawFlowsheet11(int xPnt,int yPnt){
/* BM-RTSMC, a ball mill with two stage medium classification */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
```

```
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(cycX-(CYC_DIAMETER/2)-SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* water addition */
moveto(cycX-(CYC_DIAMETER/2)-SMALL_H_SPC,gety()-V_SPC);
DownArrowW(getx(),cycY);
/* FF */
moveto(getx()-H_SPC,cycY);
RightArrow(getx()+H_SPC,cycY);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(millX-mLoffset,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 11");
}

void DrawFlowsheet12(int xPnt,int yPnt){
/* BM-RTSCC, a ball mill with two stage coarse classification */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
```

```
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(cycX-(CYC_DIAMETER/2)-SMALL_H_SPC,millY);
UpArrow(getx(),cycY);
RightArrow(cycX-(CYC_DIAMETER/2),cycY);
/* water addition */
moveto(cycX-(CYC_DIAMETER/2)-SMALL_H_SPC,cycY-V_SPC);
DownArrowW(getx(),cycY);
/* FF */
moveto(getx()-H_SPC,cycY);
RightArrow(getx()+H_SPC,cycY);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(millX-mLoffset,millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 12");
}

void DrawFlowsheet13(int xPnt,int yPnt){
/* BM-BM-C: ball mill in reversed closed circuit */
int  cycX,cycY,millX,millY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
```

```
mill_1X=cycX-CYC_DIAMETER/2-2*H_SPC;
/* FF to BM1 */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
/* BM1 */
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* BMD1 to CF */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,millY);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
/* BM2 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 13");
}

void DrawFlowsheet14(int xPnt,int yPnt){
/* BM-Scr-C: a ball mill with screen and classification */
int  cycX,cycY,millX,millY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
```

```
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
mill_1X=cycX-CYC_DIAMETER/2-2*H_SPC;
/* fresh feed to the first BM */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
/* BM1 */
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x3,y3);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
RightArrow(mill_1X-mLoffset,millY);
/* BM2 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
/* connect cyclone underflow to mill feed */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
```

```
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 14");
}

void DrawFlowsheet15(int xPnt,int yPnt){
/* BM-Scr-Cr-C: a ball mill, screen, crusher and classification */
int  cycX,cycY,millX,millY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
mill_1X=cycX-CYC_DIAMETER/2-2*H_SPC;
/* FF to BM1 */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
/* BM1 */
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* BMD1 -> CF */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x3,y3);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
```

```
lineto(mill_1X-mLoffset-SMALL_H_SPC,getY());
DownArrow(getx(),millY);
RightArrow(mill_1X-mLoffset,millY);
/* BM2 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD -> BMF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 15");
}

void DrawFlowsheet16(int xPnt,int yPnt){
/* BM-BM-TSFC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
```

```
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 16");
}

void DrawFlowsheet17(int xPnt,int yPnt){
/* BM-BM-TSMC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
```

```
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 17");
}

void DrawFlowsheet18(int xPnt,int yPnt){
/* BM-BM-TSCC */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
```

```
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 18");
}
```

```
void DrawFlowsheet19(int xPnt,int yPnt){
/* BM-Scr-BM-TSFC */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
```

```
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 19");
}

void DrawFlowsheet20(int xPnt,int yPnt){
/* BM-Scr-BM-STMC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
```

```
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(cycX-(CYC_DIAMETER/2)-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),millY);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 20");
}

void DrawFlowsheet21(int xPnt,int yPnt){
/* BM-Scr-BM-TSCC */
```

```
int cycX,cycY,mill1X,mill1Y,cyc2InX,cyc2InY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
mill1X=xPnt;
mill1Y=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=mill1X-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,mill1Y);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,mill1Y);
RightArrow(mill_1X-mLoffset,mill1Y);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,mill1Y+V_SPC);
UpArrowW(getx(),mill1Y);
DrawBallMill(mill_1X,mill1Y,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,mill1Y-2,"1");
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),mill1Y);
RightArrow(getx()+SMALL_H_SPC,mill1Y);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),mill1Y);
DrawBallMill(mill1X,mill1Y,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill1X+7,mill1Y-2,"2");
cycX=mill1X+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=mill1Y-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* BMD -> CF */
moveto(mill1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),mill1Y);
```

```
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,millY);
UpArrow(getx(),cyc2InY-CYC_TOP_H/2-SMALL_V_SPC);
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(millX-mLoffset,millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 21");
}

void DrawFlowsheet22(int xPnt,int yPnt){
/* BM-Scr-Cr-BM-TSFC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
```

```
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 22");
```

```
}

void DrawFlowsheet23(int xPnt,int yPnt){
/* BM-Scr-Cr-BM-TSMC */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* wat addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
```

```
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),millY);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 23");
}

void DrawFlowsheet24(int xPnt,int yPnt){
/* BM-Scr-Cr-BM-TSCC */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* wat addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
```

```
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cyc2InY-CYC_TOP_H/2-SMALL_V_SPC);
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
```

```
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(millX-mLoffset,millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 24");
}
```

```
/* funcs2.c01 */
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

#define MILL_LENGTH  50
#define MILL_DIAMETER 35
#define CYC_TOP_H     12
#define CYC_DIAMETER  24
#define CYC_H  40

#define H_SPC  55
#define SMALL_H_SPC 15
#define V_SPC   90
#define SMALL_V_SPC 20

extern int    MaxX,MaxY;                    /* The maximum resolution of the screen */
void changetextstyle(int font,int direction,int charsize);
void DrawPrimaryMill(int x,int y,int d,int l);
void DrawBallMill(int x,int y,int d,int l);
void DrawCyclone(int x,int y,int d,int l,int h);
void DrawScreen(int x1,int y1,int x2,int y2,int x3,int y3);
void DrawCrusher(int x,int y);
void RightArrow(int x,int y);
void RightArrowW(int x,int y);
void LeftArrow(int x,int y);
void UpArrowW(int x,int y);
void UpArrow(int x,int y);
void DownArrow(int x,int y);
void DownArrowW(int x,int y);

void DrawFlowsheet25(int xPnt,int yPnt)
{
/* PM-BM-RC */
int  cycX,cycY,millX,millY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
cycX=millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-H_SPC;
cycY=millY-V_SPC;
mill_1X=cycX-CYC_DIAMETER/2-1.5*H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* PMD */
```

```
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,millY);
lineto(getx(),cycY);
lineto(getx()+SMALL_H_SPC,cycY);
moveto(cycX-CYC_DIAMETER/2-H_SPC,cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
RightArrow(cycX-CYC_DIAMETER/2,gety());
moveto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety()-V_SPC);
DownArrowW(getx(),cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC/2);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 25");
}

void DrawFlowsheet26(int xPnt,int yPnt){
/* PM-Scr-BM-RC */
int  cycX,cycY,millX,millY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
mill_1X=cycX-CYC_DIAMETER/2-2*H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
```

```
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
/* BM1 */
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BM1 to CF */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x3,y3);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
RightArrow(mill_1X-mLoffset,millY);
/* BM2 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
/* CUF to BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 26");
}

void DrawFlowsheet27(int xPnt,int yPnt){
/* PM-Scr-Cr-BM-RC */
int  cycX,cycY,millX,millY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
setfillstyle(1,WHITE);
```

```
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
mill_1X=cycX-CYC_DIAMETER/2-2*H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
/* BM1 */
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BM1 to CF */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x3,y3);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
RightArrow(mill_1X-mLoffset,millY);
/* BM2 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
/* CUF to BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
```

```
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 27");
}

void DrawFlowsheet28(int xPnt,int yPnt){
/* PM-BM-TSFC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
```

```
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 28");
}

void DrawFlowsheet29(int xPnt,int yPnt){
/* PM-BM-TSMC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
```

```
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 29");
}

void DrawFlowsheet30(int xPnt,int yPnt){
/* PM-BM-TSCC */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
```

```
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 30");
}

void DrawFlowsheet31(int xPnt,int yPnt){
/* PM-Scr-BM-RTSFC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;
clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
```

```
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
```

```
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 31");
}

void DrawFlowsheet32(int xPnt,int yPnt){
/* PM-Scr-BM-RTSMC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(cycX-(CYC_DIAMETER/2)-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
```

```
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),millY);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 32");
}

void DrawFlowsheet33(int xPnt,int yPnt){
/* PM-Scr-BM-STSC: two stage coarse */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
```

```
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* BMD to CD */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,millY);
UpArrow(getx(),cyc2InY-CYC_TOP_H/2-SMALL_V_SPC);
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(millX-mLoffset,millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 33");
}
```

```
void DrawFlowsheet34(int xPnt,int yPnt){
/* PM-Scr-Cr-BM-RTSFC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
```

```
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 34");
}

void DrawFlowsheet35(int xPnt,int yPnt){
/* PM-Scr-Cr-BM-RTSMC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
```

```
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),millY);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
```

```
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 35");
}

void DrawFlowsheet36(int xPnt,int yPnt){
/* PM-Scr-Cr-BM-RTSCC */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
```

```
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cyc2InY-CYC_TOP_H/2-SMALL_V_SPC);
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* c2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
LeftArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(millX-mLoffset,millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 36");
}
```

```
/* funcs3.c01 */
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

#define MILL_LENGTH  50
#define MILL_DIAMETER 35
#define CYC_TOP_H     12
#define CYC_DIAMETER  24
#define CYC_H  40

#define H_SPC  55
#define SMALL_H_SPC 15
#define V_SPC  90
#define SMALL_V_SPC 20

extern int    MaxX,MaxY;                    /* The maximum resolution of the screen */
void changetextstyle(int font,int direction,int charsize);
void DrawPrimaryMill(int x,int y,int d,int l);
void DrawBallMill(int x,int y,int d,int l);
void DrawCyclone(int x,int y,int d,int l,int h);
void DrawScreen(int x1,int y1,int x2,int y2,int x3,int y3);
void DrawCrusher(int x,int y);
void RightArrow(int x,int y);
void RightArrowW(int x,int y);
void LeftArrow(int x,int y);
void UpArrowW(int x,int y);
void UpArrow(int x,int y);
void DownArrow(int x,int y);
void DownArrowW(int x,int y);

void DrawFlowsheet37(int xPnt,int yPnt){
/* PM-BM-RC-RePM, a portion of CUF returns to the primary mill */
int  cycX,cycY,millX,millY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
cycX=millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-H_SPC;
cycY=millY-V_SPC;
mill_1X=cycX-CYC_DIAMETER/2-1.5*H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* PMD */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
```

```
lineto(getx()+SMALL_H_SPC,millY);
lineto(getx(),cycY);
lineto(getx()+SMALL_H_SPC,cycY);
moveto(cycX-CYC_DIAMETER/2-H_SPC,cycY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
RightArrow(cycX-CYC_DIAMETER/2,gety());
moveto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety()-V_SPC);
DownArrowW(getx(),cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC/2);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 37");
}

void DrawFlowsheet38(int xPnt,int yPnt){
/* PM-Scr-BM-RC-RePM */
int  cycX,cycY,millX,millY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
mill_1X=cycX-CYC_DIAMETER/2-2*H_SPC;
```

```
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
/* water addition. */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
/* BM1 */
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BMD to C */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x3,y3);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,y3);
moveto(x2,y2);
UpArrow(x2,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
RightArrow(mill_1X-mLoffset,millY);
/* BM2 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
/* CUF to BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
LeftArrow(x2,gety());
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 38");
}

void DrawFlowsheet39(int xPnt,int yPnt){
/* PM-Scr-Cr-BM-RC-RePM */
int cycX,cycY,millX,millY,mill_1X;
```

```
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
mill_1X=cycX-CYC_DIAMETER/2-2*H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
/* BM1 */
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x3,y3);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,y2-1.5*V_SPC);
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
DownArrow(getx(),cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
RightArrow(mill_1X-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(cycX,millY);
```

```
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
LeftArrow(mill_1X-0.3*SMALL_H_SPC,gety());
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 39");
}

void DrawFlowsheet40(int xPnt,int yPnt){
/* PM-BM-TSFC-RePM */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
```

```
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 40");
}

void DrawFlowsheet41(int xPnt,int yPnt){
/* PM-BM-TSMC-RePM */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
```

```
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+0.75*V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 41");
}

void DrawFlowsheet42(int xPnt,int yPnt){
/* PM-BM-TSCC-RePM */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
/* FF */
```

```
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H+0.75*SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 42");
}
```

```
void DrawFlowsheet43(int xPnt,int yPnt){
/* PM-Scr-BM-TSFC-RePM */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
UpArrow(x2,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
```

```
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
LeftArrow(x2,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 43");
}

void DrawFlowsheet44(int xPnt,int yPnt){
/* PM-Scr0BM-TSMC-RePM */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
```

```
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
UpArrow(x2,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
LeftArrow(x2,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 44");
```

```
}

void DrawFlowsheet45(int xPnt,int yPnt){
/* PM-Scr-BM-TSCC-RePM */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
```

```
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(millX-mLoffset,millY);
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H+0.75*SMALL_V_SPC);
LeftArrow(mill_1X-mLoffset-SMALL_H_SPC,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 45");
}

void DrawFlowsheet46(int xPnt,int yPnt){
/* PM-Scr-Cr-BM-TSFC-RePM */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
```

```
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
LeftArrow(mill_1X-mLoffset-SMALL_H_SPC,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
```

```
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 46");
}

void DrawFlowsheet47(int xPnt,int yPnt){
/* PM-Scr-Cr-BM-TSMC-RePM */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"1");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
```

```
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
moveto(cycX,cycY+CYC_H+0.75*SMALL_V_SPC);
LeftArrow(mill_1X-mLoffset-SMALL_H_SPC,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 47");
}

void DrawFlowsheet48(int xPnt,int yPnt){
/* PM-Scr-Cr-BM-RePM */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X;
int mLoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_1X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
```

```
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawPrimaryMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_1X,gety());
DrawCrusher(mill_1X,gety());
moveto(mill_1X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"2");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD to CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
```

```
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H+0.75*SMALL_V_SPC);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(millX-mLoffset,millY);
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H+0.75*SMALL_V_SPC);
LeftArrow(mill_1X-mLoffset-SMALL_H_SPC,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 48");
}
```

```
/* funcs4.c01 */
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

#define MILL_LENGTH 50
#define MILL_DIAMETER 35
#define CYC_TOP_H    12
#define CYC_DIAMETER 24
#define CYC_H 40

#define H_SPC 55
#define SMALL_H_SPC 15
#define V_SPC 90
#define SMALL_V_SPC 20

extern int   MaxX,MaxY;                    /* The maximum resolution of the screen */

void changetextstyle(int font,int direction,int charsize);

void DrawPrimaryMill(int x,int y,int d,int l);
void DrawBallMill(int x,int y,int d,int l);
void DrawCyclone(int x,int y,int d,int l,int h);
void DrawScreen(int x1,int y1,int x2,int y2,int x3,int y3);
void DrawCrusher(int x,int y);
void RightArrow(int x,int y);
void RightArrowW(int x,int y);
void LeftArrow(int x,int y);
void UpArrowW(int x,int y);
void UpArrow(int x,int y);
void DownArrow(int x,int y);
void DownArrowW(int x,int y);

void DrawFlowsheet49(int xPnt,int yPnt){
/* BM-BM-BM-RC */
int cycX,cycY,millX,millY,mill_1X,mill_2X;
int mLoffset;
clearviewport();

setfillstyle(1,WHITE);
millX=xPnt+40;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
```

```
mill_2X=cycX-CYC_DIAMETER/2-2*H_SPC;
mill_1X=mill_2X-mLoffset-(MILL_LENGTH/2)-(MILL_LENGTH/10)-H_SPC;
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
/* BM1 */
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* BM2 */
DrawBallMill(mill_2X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_2X+7,millY-2,"2");
/* BM1->BM2 */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(mill_2X-mLoffset,millY);
/* BM1D -> CF */
moveto(mill_2X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,millY);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
/* BM3 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"3");
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 49");
}

void DrawFlowsheet50(int xPnt,int yPnt){
/* BM-BM-Scr-BM-RC */
int cycX,cycY,millX,millY,mill_1X,mill_2X;
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
```

```
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
mill_2X=cycX-CYC_DIAMETER/2-2*H_SPC;
mill_1X=mill_2X-mLoffset-(MILL_LENGTH/2)-(MILL_LENGTH/10)-H_SPC;
/* BM1 */
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
RightArrow(mill_1X-mLoffset,millY);
/* BM2 */
DrawBallMill(mill_2X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_2X+7,millY-2,"2");
/* BM1->BM2 */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(mill_2X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_2X-mLoffset,millY);
/* BMD2 to CF */
moveto(mill_2X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x3,y3);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_2X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
RightArrow(mill_2X-mLoffset,millY);
/* BM2 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
```

```
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"3");
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 50");
}

void DrawFlowsheet51(int xPnt,int yPnt){
/* BM-BM-Scr-Cr-BM-RC */
int  cycX,cycY,millX,millY,mill_1X,mill_2X;
int mLoffset;
int x1,y1,x2,y2,x3,y3;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
cycX=millX-mLoffset-H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),cycY-(CYC_TOP_H/2)-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
mill_2X=cycX-CYC_DIAMETER/2-2*H_SPC;
mill_1X=mill_2X-mLoffset-(MILL_LENGTH/2)-(MILL_LENGTH/10)-H_SPC;
/* BM1 */
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(getx()+H_SPC-SMALL_H_SPC,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
RightArrow(mill_1X-mLoffset,millY);
/* BM1 */
```

```
DrawBallMill(mill_2X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_2X+7,millY-2,"2");
/* BM1->BM2 */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(mill_2X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_2X-mLoffset,millY);
/* BMD1 -> CF */
moveto(mill_2X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x3,y3);
RightArrow(cycX-CYC_DIAMETER/2-SMALL_H_SPC,y3);
moveto(x2,y2);
lineto(x2,y2-1.5*V_SPC);
LeftArrow(mill_2X,gety());
DrawCrusher(mill_2X,gety());
moveto(mill_2X-0.3*SMALL_H_SPC,y2-1.5*V_SPC+(1.5*SMALL_V_SPC));
lineto(getx(),gety()+SMALL_V_SPC);
lineto(mill_2X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
RightArrow(mill_2X-mLoffset,millY);
/* BM2 */
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"3");
/* CUF -> BMF */
moveto(cycX,cycY+CYC_H);
lineto(cycX,millY);
RightArrow(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7),millY);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,millY);
lineto(getx(),gety()+V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),y3);
lineto(getx(),cycY);
RightArrow(cycX-CYC_DIAMETER/2,cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
settextjustify(CENTER_TEXT,TOP_TEXT);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 51");
}

void DrawFlowsheet52(int xPnt,int yPnt){
/* BM-BM-BM-TSFC */
```

```
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X,mill_2X;
int mLoffset,mRoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_2X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
mill_1X=mill_2X-mLoffset-(MILL_LENGTH/2)-(MILL_LENGTH/10)-H_SPC;
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(mill_2X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_2X+7,millY-2,"2");
/* BM1->BM2 */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(mill_2X-mLoffset,millY);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"3");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
```

```
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 52");
}

void DrawFlowsheet53(int xPnt,int yPnt){
/* BM-BM-BM-TSMC */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X,mill_2X;
int mLoffset,mRoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
millY=yPnt;
mill_2X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
mill_1X=mill_2X-mLoffset-(MILL_LENGTH/2)-(MILL_LENGTH/10)-H_SPC;
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(mill_2X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_2X+7,millY-2,"2");
/* BM1->BM2 */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(mill_2X-mLoffset,millY);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"3");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
```

```
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+0.75*V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 53");
}

void DrawFlowsheet54(int xPnt,int yPnt){
/* BM-BM-BM-TSCC */
int cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X,mill_2X;
int mLoffset;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_2X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10);
mill_1X=mill_2X-mLoffset-(MILL_LENGTH/2)-(MILL_LENGTH/10)-H_SPC;
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
```

```
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY-V_SPC);
DownArrowW(getx(),millY);
DrawBallMill(mill_2X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_2X+7,millY-2,"2");
/* BM1->BM2 */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(mill_2X-mLoffset,millY);
/* BM1D to BM2F */
moveto(millX-mLoffset-H_SPC,millY);
RightArrow(millX-mLoffset,millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"3");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
lineto(getx()+SMALL_H_SPC,gety());
lineto(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=cycY+CYC_H+2.5*SMALL_V_SPC;
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
lineto(cycX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
UpArrow(getx(),cycY);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC/2);
lineto(millX-mLoffset-SMALL_H_SPC,gety());
UpArrow(getx(),millY);
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 54");
}

void DrawFlowsheet55(int xPnt,int yPnt){
/* BM-BM-Scr-BM-TSFC */
int  cycX,cycY,millX,millY,cyc2InX,cyc2InY,mill_1X,mill_2X;
```

```
int mLoffset,mRoffset;
int x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;

clearviewport();
setfillstyle(1,WHITE);
millX=xPnt;
millY=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_2X=millX-mLoffset-H_SPC-(MILL_LENGTH/2)-(MILL_LENGTH/10)-2*SMALL_H_SPC;
mill_1X=mill_2X-mLoffset-(MILL_LENGTH/2)-(MILL_LENGTH/10)-H_SPC;
DrawBallMill(mill_1X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,millY-2,"1");
/* FF */
moveto(mill_1X-mLoffset-H_SPC,millY);
RightArrow(mill_1X-mLoffset-SMALL_H_SPC,millY);
RightArrow(mill_1X-mLoffset,millY);
/* water addition */
moveto(mill_1X-mLoffset-SMALL_H_SPC,millY+V_SPC);
UpArrowW(getx(),millY);
DrawBallMill(mill_2X,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_2X+7,millY-2,"2");
/* BM1->BM2 */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(mill_2X-mLoffset,millY);
moveto(mill_2X+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
RightArrow(getx()+SMALL_H_SPC,millY);
x1=getx();
y1=gety();
x2=x1+2*SMALL_H_SPC;
y2=y1+1.5*SMALL_V_SPC;
x3=x2-SMALL_H_SPC;
y3=y2+0.5*SMALL_V_SPC;
DrawScreen(x1,y1,x2,y2,x3,y3);
moveto(x2,y2);
lineto(x2,y2-V_SPC);
lineto(mill_1X-mLoffset-SMALL_H_SPC,gety());
DownArrow(getx(),millY);
DrawBallMill(millX,millY,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(millX+7,millY-2,"3");
cycX=millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+H_SPC;
cycY=millY-V_SPC;
moveto(x3,y3);
lineto(x3+2*SMALL_H_SPC,y3);
lineto(getx(),cycY);
RightArrow(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10)+SMALL_H_SPC,cycY);
DrawCyclone(cycX,cycY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* BMD -> CF */
moveto(millX+(MILL_LENGTH/2)+(MILL_LENGTH/10),millY);
```

```
lineto(getx()+SMALL_H_SPC,gety());
UpArrow(getx(),gety()-V_SPC);
RightArrow(cycX-(CYC_DIAMETER/2),gety());
/* COF */
moveto(cycX,cycY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
cyc2InX=getx();
cyc2InY=gety();
/* CUF */
moveto(cycX,cycY+CYC_H);
lineto(getx(),gety()+V_SPC);
lineto(millX-(MILL_LENGTH/2)-(MILL_LENGTH/5)-(MILL_LENGTH/7)-SMALL_H_SPC,gety());
lineto(getx(),millY);
RightArrow(getx()+SMALL_H_SPC,gety());
/* C2 */
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-SMALL_V_SPC);
RightArrow(getx()+H_SPC,gety());
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+V_SPC);
LeftArrow(cycX,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 55");
}

void DrawFlowsheet56(int xPnt,int yPnt){
int  cyc1InX,cyc1InY,cyc2InX,cyc2InY,mill_1X,mill_1Y,mill_2X,mill_2Y;
int mLoffset,mRoffset;

clearviewport();
setfillstyle(1,WHITE);
mill_1X=mill_2X=xPnt;
mill_1Y=yPnt;
mLoffset=(MILL_LENGTH/2)+(MILL_LENGTH/5)+(MILL_LENGTH/7);
mill_2Y=mill_1Y+0.75*V_SPC;
cyc1InX=cyc2InX=mill_1X-4*H_SPC;
cyc1InY=mill_1Y-0.75*V_SPC;
cyc2InY=cyc1InY-V_SPC;
DrawBallMill(mill_1X,mill_1Y,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_1X+7,mill_1Y-2,"1");
DrawBallMill(mill_2X,mill_2Y,MILL_DIAMETER,MILL_LENGTH);
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(mill_2X+7,mill_2Y-2,"2");
DrawCyclone(cyc1InX+CYC_DIAMETER/2,cyc1InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
DrawCyclone(cyc2InX+CYC_DIAMETER/2,cyc2InY,CYC_TOP_H,CYC_DIAMETER,CYC_H);
moveto(40,mill_2Y+0.5*V_SPC);
```

```
RightArrow(cyc1InX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
moveto(getx(),gety()+0.5*V_SPC);
UpArrowW(getx(),gety()-0.5*V_SPC);
lineto(getx(),cyc2InY);
RightArrow(cyc2InX,cyc2InY);
moveto(cyc1InX-CYC_DIAMETER/2-SMALL_H_SPC,cyc1InY);
RightArrow(cyc1InX,cyc1InY);
/* COF1 */
moveto(cyc1InX+CYC_DIAMETER/2,cyc1InY-CYC_TOP_H/2);
lineto(getx(),gety()-0.75*SMALL_V_SPC);
lineto(getx()+4*H_SPC,gety());
UpArrow(getx(),cyc2InY-CYC_TOP_H/2-0.5*SMALL_V_SPC);
/* COF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY-CYC_TOP_H/2);
lineto(getx(),gety()-0.5*SMALL_V_SPC);
RightArrow(getx()+4*H_SPC,gety());
RightArrow(getx()+H_SPC,gety());
/* CUF1 */
moveto(cyc1InX+CYC_DIAMETER/2,cyc1InY+CYC_H);
lineto(getx(),mill_2Y+5);
RightArrow(mill_2X-mLoffset,mill_2Y+5);
moveto(cyc1InX+CYC_DIAMETER/2,mill_1Y+5);
RightArrow(mill_1X-mLoffset,mill_1Y+5);
/* CUF2 */
moveto(cyc2InX+CYC_DIAMETER/2,cyc2InY+CYC_H);
lineto(getx(),gety()+0.75*SMALL_V_SPC);
lineto(getx()+2*H_SPC,gety());
lineto(getx(),mill_2Y-5);
RightArrow(mill_2X-mLoffset,gety());
moveto(cyc2InX+CYC_DIAMETER/2+2*H_SPC,mill_1Y-5);
RightArrow(mill_1X-mLoffset,gety());
/* BMD1 */
moveto(mill_1X+(MILL_LENGTH/2)+(MILL_LENGTH/10),mill_1Y);
lineto(getx()+2*SMALL_H_SPC,gety());
DownArrow(getx(),mill_2Y);
/* BMD2 */
moveto(mill_2X+(MILL_LENGTH/2)+(MILL_LENGTH/10),mill_2Y);
RightArrow(getx()+2*SMALL_H_SPC,gety());
lineto(getx(),gety()+0.50*V_SPC);
LeftArrow(cyc1InX-CYC_DIAMETER/2-SMALL_H_SPC,gety());
changetextstyle(SMALL_FONT,HORIZ_DIR,8);
outtextxy(MaxX/2-20,MaxY/2+100,"Circuit 56");
}
```

```
/* ballmill.c01 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include <conio.h>
#include <assert.h>

/* constants */
#define STRMMATCOLS  4
#define rho 1 /* water density as default for suspending fluid density */
#define mu  1 /* water viscosity as default for suspending liquid viscosity */

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

/* code for various physical and conceptual units in the circuits */
#define BM_UNIT_CODE          1
#define HCYC_UNIT_CODE        2
#define JUNC_NODE_CODE        3
#define SPLIT_NODE_CODE       4
#define FIXCLASS_NODE_CODE    5
#define CNVRG_BLOCK_CODE      100

typedef enum bool {FALSE=0,TRUE=1} bool;

typedef struct{
        double *sf;
        char bFnormalizabilityFlag[5];
        double **bf;
        double referenceFeedRate;
        double referenceTauPF;
        double referenceTauSPM;
        double referenceTauLPM;
} ballmill;

int *CreateVectorInt(long nl, long nh);
void FreeVectorInt(int *v, long nl, long nh);
double *CreateVectorD(long nl, long nh);
void FreeVectorD(double *v, long nl, long nh);
int **CreateMatrixInt(long nrl, long nrh, long ncl, long nch);
void FreeMatrixInt(int **m, long nrl, long nrh, long ncl, long nch);
double **CreateMatrixD(long nrl, long nrh, long ncl, long nch);
void FreeMatrixD(double **m, long nrl, long nrh, long ncl, long nch);

int ReadCircuitSpec(int *cirNo);
```

```
int ReadNumsOfNodesStrms(int cirNo,int *nodeNums,int *streamNums);
int ReadSizeClassNums(int *sizeClassNums);
int ReadSimulationStatus(int *curNodeNumPtr,int cnvrgNodeNums,bool *convergence,int
*compIterPtr);
int UpdateSimulationStatus(int curNodeNum,int cnvrgNodeNums,bool *convergence,int compIter);
int ReadConnectivityMatrix(int **conMatPtr,int cirNo);
int ReadStreamMatrix(int streamNums,double **strmMatPtr);
int UpdateStreamMatrix(int streamNums,double **strmMatPtr);
int ReadStreamsSizeDist(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int UpdateStrmSizeMat(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int ReadBallMillInputData(int **conMatPtr,int curNodeNum,int sizeClassNums,ballmill *b);
int FindBallMillStreamsIndex(int streamNums,int curNodeNum,int **conMatPtr,int *bmF,int *bmD);
int HandleIncorrectData(char *str);

int main(void){
char str[160];
int **conMatPtr;
int i,j,k,iter;
int sizeClassNums,curNodeNum;
int cirNo,nodeNums,streamNums;
int BMFstrmIndex,BMDstrmIndex;
int cnvrgNodeNums;
double **strmSizeMatPtr,**strmMatPtr;
double *diag,**t,**tinv,**tdiag,**trans;
double tauPF,tauSPM,tauLPM,mrt;
double WPF,WS,WL;
double sum,sum1,sum2,sum3,sum4;
ballmill b;
ballmill *bmPtr;
bool *cnvrgVecPtr;

bmPtr=&b;
/* read general data */
ReadCircuitSpec(&cirNo);
ReadNumsOfNodesStrms(cirNo,&nodeNums,&streamNums);
ReadSizeClassNums(&sizeClassNums);
/* check data */
if(iter<2){
        if(sizeClassNums<2){
                str[0]='\0';
                strcat(str,"Cannot accept size class numbers less than two."
                                        "  This might happen because of incorrect data or file\n"
                                        "  format.  Please check data file for possible errors.");
                HandleIncorrectData(str);
        }
}
/* allocate exact memory needed for reading connectivity matrix */
conMatPtr=CreateMatrixInt(1,nodeNums,1,streamNums+3);
for(i=1;i<=nodeNums;i++)
        for(j=1;j<=streamNums+3;j++)
                conMatPtr[i][j]=0.00;
```

```
ReadConnectivityMatrix(conMatPtr,cirNo);
cnvrgNodeNums=0;
for(i=1;i<=nodeNums;i++)
        if(conMatPtr[i][2]==CNVRG_BLOCK_CODE) cnvrgNodeNums++;
cnvrgVecPtr=(bool *) CreateVectorInt(1,cnvrgNodeNums);
        for(i=1;i<=cnvrgNodeNums;i++)
                cnvrgVecPtr[i]=0;
ReadSimulationStatus(&curNodeNum,cnvrgNodeNums,cnvrgVecPtr,&iter);
/* allocate exact memory needed for reading streams information */
strmMatPtr=CreateMatrixD(1,streamNums,1,STRMMATCOLS);
for(i=1;i<=streamNums;i++)
        for(j=1;j<=STRMMATCOLS;j++)
                strmMatPtr[i][j]=0.00;
ReadStreamMatrix(streamNums,strmMatPtr);
/* allocate exact memory needed for reading size distributions;
        including pan fraction */
strmSizeMatPtr=CreateMatrixD(1,sizeClassNums,1,streamNums+2);
for(i=1;i<=sizeClassNums;i++)
        for(j=1;j<=streamNums+2;j++)
                strmSizeMatPtr[i][j]=0.00;
ReadStreamsSizeDist(strmSizeMatPtr,sizeClassNums,streamNums);
/* check data */
if(iter<2){
for(i=1;i<=sizeClassNums;i++)
        for(j=3;j<=streamNums+2;j++)
                if(LT(strmSizeMatPtr[i][j],0) || GT(strmSizeMatPtr[i][j],100)){
                        str[0]='\0';
                        strcat(str,"Cannot accept a percent mass less than zero or greater\n"
                                                * than 100. This might happen because of
incorrect data or file\n"
                                                * format. Please check data file for possible
errors.");
                        HandleIncorrectData(str);
                }
}
bmPtr->sf=CreateVectorD(1,sizeClassNums-1);
/* initialize sf vector */
for(i=1;i<=sizeClassNums-1;i++)
        bmPtr->sf[i]=0.0;
bmPtr->bf=CreateMatrixD(1,sizeClassNums-1,1,sizeClassNums-1);
/* initialize bf matrix */
for(i=1;i<=sizeClassNums-1;i++)
        for(j=1;j<=sizeClassNums-1;j++)
                bmPtr->bf[i][j]=0.0;
FindBallMillStreamsIndex(streamNums,curNodeNum,conMatPtr,&BMFstrmIndex,&BMDstrmIndex);
ReadBallMillInputData(conMatPtr,curNodeNum,sizeClassNums,bmPtr);
/* check data */
if(iter<2){
        str[0]='\0';
        /* check sf values */
        for(i=1;i<=sizeClassNums-1;i++)
```

```
            if(LE(bmPtr->sf[i],0)){
                strcat(str,"Cannot accept a selection function value less than or \n"
                        "    equal to zero. This might happen because of incorrect data\n"
                        "    or file format. Please check data file for possible errors.");
                HandleIncorrectData(str);
            }
        /* check for equal sf values */
        for(i=2;i<=sizeClassNums-1;i++)
            if(EQ(bmPtr->sf[i],bmPtr->sf[i-1])){
                strcat(str,"Cannot accept two equal consecutive selection function values.\n"
                        "    Please correct data file by introducing a small change in
one\n"
                        "    of the selection function values.");
                HandleIncorrectData(str);
            }
        /* check bf normalizability flag */
        if(strcmp(bmPtr->bFnormalizabilityFlag,"ON")!=0 &&
            strcmp(bmPtr->bFnormalizabilityFlag,"OFF")!=0){
            strcat(str,"Breakage function normalizability flag must be either\n"
                        "    ON or OFF. This might happen because of incorrect\n"
                        "    data or file format. Please check data file for \n"
                        "    possible errors.");
            HandleIncorrectData(str);
        }
        /* check individual bf values */
        for(i=1;i<=sizeClassNums-1;i++){
            for(j=1;j<=sizeClassNums-1;j++)
                if(LT(bmPtr->bf[i][j],0) || GT(bmPtr->bf[i][j],1)){
                    strcat(str,"Cannot accept a breakage function value less than zero
or\n"
                        "    greater than 1. This might happen because of incorrect data
\n"
                        "    or file format. Please check data file for possible errors.");
                    HandleIncorrectData(str);
                }
        }
        /* check sum of bf values */
        for(j=1;j<=sizeClassNums-1;j++){
            sum=0;
            for(i=1;i<=sizeClassNums-1;i++)
                sum=sum+bmPtr->bf[i][j];
            if(GT(sum,1)){
                strcat(str,"The sum of breakage function values for each column cannot be
\n"
                        "    greater than 1. This might happen because of incorrect \n"
                        "    data or file format. Please check data file for possible\n"
                        "    errors.");
                HandleIncorrectData(str);
            }
        }
        /* check RTD related values */
```

```
        if(LE(bmPtr->referenceFeedRate,0)){
                strcat(str,"Cannot accept a reference feed rate less than or equal to zero.\n"
                                "  This might happen because of incorrect data or file\n"
                                "  format.  Please check data file for possible errors.");
                HandleIncorrectData(str);
        }
        if(LT(bmPtr->referenceTauPF,0) || LT(bmPtr->referenceTauSPM,0) ||
LT(bmPtr->referenceTauLPM,0)){
                strcat(str,"Cannot accept negative reference mean retention times.\n"
                                "  This might happen because of incorrect data or file\n"
                                "  format.  Please check data file for possible errors.");
                HandleIncorrectData(str);
        }
        if(LE(strmMatPtr[BMFstrmIndex][2],0)){
                strcat(str,"Cannot accept a ball mill feed rate less than zero.\n"
                                "  This might happen because of incorrect data or file\n"
                                "  format.  Please check data file for possible errors.");
                HandleIncorrectData(str);
        }
}
/*  allocating exact memory for all matrices */
diag=CreateVectorD(1,sizeClassNums-1);
t=CreateMatrixD(1,sizeClassNums-1,1,sizeClassNums-1);
tinv=CreateMatrixD(1,sizeClassNums-1,1,sizeClassNums-1);
tdiag=CreateMatrixD(1,sizeClassNums-1,1,sizeClassNums-1);
trans=CreateMatrixD(1,sizeClassNums-1,1,sizeClassNums-1);
for(i=1;i<=sizeClassNums-1;i++){
        diag[i]=0.00;
        for(j=1;j<=sizeClassNums-1;j++){
                t[i][j]=0.0;
                tinv[i][j]=0.0;
                tdiag[i][j]=0.0;
                trans[i][j]=0.0;
        }
}
/* calculate  taus */
mrt=bmPtr->referenceTauPF+(2*bmPtr->referenceTauSPM)+bmPtr->referenceTauLPM;
assert(GT(mrt,0));
tauPF=(bmPtr->referenceFeedRate/strmMatPtr[BMFstrmIndex][2])*(bmPtr->referenceTauPF/mrt);
tauSPM=(bmPtr->referenceFeedRate/strmMatPtr[BMFstrmIndex][2])*(bmPtr->referenceTauSPM/mrt
);
tauLPM=(bmPtr->referenceFeedRate/strmMatPtr[BMFstrmIndex][2])*(bmPtr->referenceTauLPM/mr
t);
/*      calculate the "diag" matrix */
for(i=1;i<=sizeClassNums-1;i++){
        WPF=exp(-bmPtr->sf[i]*tauPF);
        WS=1+bmPtr->sf[i]*tauSPM;
        WL=1+bmPtr->sf[i]*tauLPM;
        diag[i]=WPF/(WL*pow(WS,2));
}
/*  calculation of the t matrix */
```

```
for(i=1;i<=sizeClassNums-1;i++)
        for(j=1;j<=sizeClassNums-1;j++)
                if(i<j) t[i][j]=0.0;
                else    if(i==j) t[i][j]=bmPtr->sf[j];
                                else{
                                        sum=0.0;
                                        for(k=1;k<i;k++)
sum=sum+bmPtr->bf[i][k]*bmPtr->sf[k]*t[k][j];
                                        if(EQ(bmPtr->sf[i],bmPtr->sf[j]))
bmPtr->sf[i]=bmPtr->sf[i]*0.999999;
                                        t[i][j]=sum/(bmPtr->sf[i]-bmPtr->sf[j]);
                                }
/*      calculation of the tinv matrix */
for(i=1;i<=sizeClassNums-1;i++)
        for(j=1;j<=sizeClassNums-1;j++){
                if(i<j) tinv[i][j]=0;
                if(i==j)
                        tinv[i][j]=1/bmPtr->sf[j];
                if(i>j){
                        sum1=0;
                        for(k=1;k<i;k++) sum1=sum1+t[i][k]*tinv[k][j];
                                tinv[i][j]=-sum1/bmPtr->sf[i];
                }
        }
/*      calculation of the t * diag */
for(i=1;i<=sizeClassNums-1;i++)
        for(j=1;j<=sizeClassNums-1;j++)         tdiag[i][j]=t[i][j]*diag[j];
/*      calculation of the mill transformation matrix */
for(i=1;i<=sizeClassNums-1;i++)
        for(j=1;j<=sizeClassNums-1;j++){
                sum2=0;
                for(k=1;k<=sizeClassNums-1;k++) sum2=sum2+tdiag[i][k]*tinv[k][j];
                trans[i][j]=sum2;
        }
/*      calculation of the ball mill discharge */
for(i=1;i<=sizeClassNums-1;i++){
        sum3=0;
        for(j=1;j<=sizeClassNums-1;j++)
sum3=sum3+trans[i][j]*strmSizeMatPtr[j][BMFstrmIndex+2];
        strmSizeMatPtr[i][BMDstrmIndex+2] = sum3;
}
sum4=0;
for(i=1;i<=sizeClassNums-1;i++) sum4=sum4+strmSizeMatPtr[i][BMDstrmIndex+2];
strmSizeMatPtr[sizeClassNums][BMDstrmIndex+2]=100-sum4;
strmMatPtr[BMDstrmIndex][2]=strmMatPtr[BMFstrmIndex][2];
strmMatPtr[BMDstrmIndex][3]=strmMatPtr[BMFstrmIndex][3];
strmMatPtr[BMDstrmIndex][4]=strmMatPtr[BMFstrmIndex][4];
if(curNodeNum<nodeNums)  curNodeNum++;
else curNodeNum=1;
UpdateStreamMatrix(streamNums,strmMatPtr);
UpdateStrmSizeMat(strmSizeMatPtr,sizeClassNums,streamNums);
```

```
UpdateSimulationStatus(curNodeNum,cnvrgNodeNums,cnvrgVecPtr,iter);
FreeMatrixInt(conMatPtr,1,nodeNums,1,streamNums+3);
FreeMatrixD(strmMatPtr,1,streamNums,1,STRMMATCOLS);
FreeMatrixD(strmSizeMatPtr,1,sizeClassNums,1,streamNums+2);
return 0;
}

int FindBallMillStreamsIndex(int streamNums,int curNodeNum,int **conMatPtr,int *bmF,int *bmD){
int j;
for(j=4;j<=streamNums+3;j++)
        if(conMatPtr[curNodeNum][j]==+1) *bmF=j-3;
        else if(conMatPtr[curNodeNum][j]==-1) *bmD=j-3; /* -1 is BMD stream code */
return 0;
}
```

```
/* cyclone.c01 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include <conio.h>
#include <assert.h>

/* code for various physical and conceptual units in the circuits */
#define BM_UNIT_CODE          1
#define HCYC_UNIT_CODE        2
#define JUNC_NODE_CODE        3
#define SPLIT_NODE_CODE       4
#define FIXCLASS_NODE_CODE    5
#define CNVRG_BLOCK_CODE      100


/* constants */
#define STRMMATCOLS 4
#define rho 1 /* water density as default for suspending fluid density */
#define mu  1 /* water viscosity as default for suspending liquid viscosity */

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y))<=fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

typedef enum bool {FALSE=0,TRUE=1} bool;

typedef struct{
        double  Dc;     /* inside diameter of the cyclone */
        double  Di;  /* inside diameter of the cyclone inlet */
        double  Do;  /* inside diameter of the cyclone overflow or vortex finder */
        double  Du;  /* inside diameter of the cyclone underflow or apex orifice */
        double  h;   /* free vortex height of the cyclone */
} cyclone;

typedef struct{
        double d50;     /* cut size */
        double Rf;    /* recovery of water to cyclone underflow */
        double m;     /* sharpness of separation */
} cycparam;

int *CreateVectorInt(long nl,long nh);
void FreeVectorInt(int *v,long nl,long nh);
double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
int **CreateMatrixInt(long nrl,long nrh,long ncl,long nch);
```

```
void FreeMatrixInt(int **m,long nrl,long nrh,long ncl,long nch);
double **CreateMatrixD(long nrl,long nrh,long ncl,long nch);
void FreeMatrixD(double **m,long nrl,long nrh,long ncl,long nch);

int ReadCircuitSpec(int *cirNo);
int ReadNumsOfNodesStrms(int cirNo,int *nodeNums,int *streamNums);
int ReadSizeClassNums(int *sizeClassNums);
int ReadSimulationStatus(int *curNodeNumPtr,int cnvrgNodeNums,bool *convergence,int
*compIterPtr);
int UpdateSimulationStatus(int curNodeNum,int cnvrgNodeNums,bool *convergence,int compIter);
int ReadConnectivityMatrix(int **conMatPtr,int cirNo);
int ReadStreamMatrix(int streamNums,double **strmMatPtr);
int UpdateStreamMatrix(int streamNums,double **strmMatPtr);
int ReadStreamsSizeDist(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int UpdateStrmSizeMat(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int FindCycStreamsIndex(int streamNums,int curNodeNum,int **conMatPtr,int *cF,int *cOF,int
*cUF);
int ReadCycInputData(int **conMatPtr,int curNodeNum,cyclone *c,double *rhosPtr,
                                                    int *cycNums,double
*d50CorrFactorPtr,double *pCorrFactorPtr,
                                                    double *sCorrFactorPtr,double
*mCorrFactorPtr,
                                                    double *RfCorrFactorPtr,int *maxIter);
int ReadCycloneNums(int **conMatPtr,int nodeNums,int *cycPakNums);
int CreateCycParamFile(int cycPakNums);
int ReadCycParams(double **cycParamMatPtr,int cycPakNums);
int WriteCycParams(double **cycParamMatPtr,int cycPakNums);
int HandleIncorrectData(char *str);

int main(){
FILE *simStatusFilePtr;
char str[160];
int iter;
int **conMatPtr;
int cirNo,nodeNums,streamNums;
int sizeClassNums,curNodeNum;
int i,j,r,iterCount,maxIter;
int cycPakNums,cycNums;
int cFstrmIndex,cOFstrmIndex,cUFstrmIndex;
int cnvrgNodeNums;
double sum;
double **strmSizeMatPtr,**strmMatPtr,**cycParamMatPtr;
double *SbySsolidsCUF,*SbySsolidsCOF;
double sumSolidsCOF;
double cFliquidTonnage,cFsolidsTonnage;
double Q;       /* volumetric flow rate of cyclone feed slurry */
double phi; /* volumetric fraction of solids in the feed slurry */
double phis;/* mass fraction solids in the feed slurry */
double S;  /* volumetric slurry split between COF and CUF */
double P;  /* pressure drop across a hydrocyclone */
double H;       /* preasure drop expressed in head of feed slurry */
```

```
double Rv; /* rec. of feed volume to the underflow product */
double Rs; /* rec. of feed solides to the underflow product */
double rhos; /* solid density and liquid density, respectively */
double rhop; /* solid density of pulp */
double RfOld,RfNew;
double Lu,Lu20,Lum;
double cUFsolids,cUFwater,cUFperSolids,cOFsolids,cOFwater,cOFperSolids;
double d50CorrFactor,pCorrFactor,sCorrFactor,mCorrFactor,RfCorrFactor;
cyclone c;
cycparam cp;
bool done,*cnvrgVecPtr;

c.Dc=0;
c.Di=0;
c.Do=0;
c.Du=0;
c.h=0;
rhos=0;
cycNums=0;
maxIter=0;
ReadCircuitSpec(&cirNo);
ReadNumsOfNodesStrms(cirNo,&nodeNums,&streamNums);
ReadSizeClassNums(&sizeClassNums);
/* allocate memory for work space */
SbySsolidsCUF=CreateVectorD(1,sizeClassNums);
SbySsolidsCOF=CreateVectorD(1,sizeClassNums);
for(i=1;i<=sizeClassNums;i++){
        SbySsolidsCUF[i]=0.00;
        SbySsolidsCOF[i]=0.00;
}
/* allocate exact memory needed for reading connectivity matrix */
conMatPtr=CreateMatrixInt(1,nodeNums,1,streamNums+3);
for(i=1;i<=nodeNums;i++)
        for(j=1;j<=streamNums+3;j++)
                conMatPtr[i][j]=0.00;
ReadConnectivityMatrix(conMatPtr,cirNo);
cnvrgNodeNums=0;
for(i=1;i<=nodeNums;i++)
  if(conMatPtr[i][2]==CNVRG_BLOCK_CODE)
          cnvrgNodeNums++;
cnvrgVecPtr=(bool *) CreateVectorInt(1,cnvrgNodeNums);
for(i=1;i<=cnvrgNodeNums;i++)
        cnvrgVecPtr[i]=0;
ReadSimulationStatus(&curNodeNum,cnvrgNodeNums,cnvrgVecPtr,&iter);
/* allocate exact memory needed for reading streams information */
strmMatPtr=CreateMatrixD(1,streamNums,1,STRMMATCOLS);
for(i=1;i<=streamNums;i++)
        for(j=1;j<=STRMMATCOLS;j++)
                strmMatPtr[i][j]=0.00;
ReadStreamMatrix(streamNums,strmMatPtr);
ReadSizeClassNums(&sizeClassNums);
```

```
/* allocate exact memory needed for reading size distributions */
strmSizeMatPtr=CreateMatrixD(1,sizeClassNums,1,streamNums+2);
for(i=1;i<=sizeClassNums;i++)
        for(j=1;j<=streamNums+2;j++)
                strmSizeMatPtr[i][j]=0.00;
ReadStreamsSizeDist(strmSizeMatPtr,sizeClassNums,streamNums);
ReadCycloneNums(conMatPtr,nodeNums,&cycPakNums);
/* allocate exact memory for reading cyclone parameters */
cycParamMatPtr=CreateMatrixD(1,cycPakNums,1,6);
for(i=1;i<=cycPakNums;i++)
        for(j=1;j<=6;j++)
                cycParamMatPtr[i][j]=0.0;
ReadCycParams(cycParamMatPtr,cycPakNums);
r=1;
for(i=1;i<=nodeNums;i++)
        if(conMatPtr[i][2]==HCYC_UNIT_CODE){
                cycParamMatPtr[r][1]=conMatPtr[i][3];
                r++;
        }
FindCycStreamsIndex(streamNums,curNodeNum,conMatPtr,&ccFstrmIndex,&ccOFstrmIndex,&ccUFstrmI
ndex);
ReadCycInputData(conMatPtr,curNodeNum,&c,&rhos,&cycNums,&d50CorrFactor,

&pCorrFactor,&sCorrFactor,&mCorrFactor,&RfCorrFactor,&maxIter);
/*  check data */
if(iter<2){
str[0]='\0';
if(LE(c.Dc,0)){
        strcat(str,"Cannot accept cyclone diameter less than or equal to zero.\n"
                                "  This might happen because of incorrect data or file\n"
                                "  format.  Please check data file for possible errors.");
        HandleIncorrectData(str);
}
if(LE(c.Di,0)){
        strcat(str,"Cannot accept cyclone inlet diameter less than or equal to \n"
                                "  zero.  This might happen because of incorrect data or \n"
                                "  file format.  Please check data file for possible errors.");
        HandleIncorrectData(str);
}
if(LE(c.Do,0)){
        strcat(str,"Cannot accept cyclone vortex finder diameter less than or \n"
                                "  equal to zero.  This might happen because of incorrect \n"
                                "  data or file format.  Please check data file for possible\n"
                                "  errors.");
        HandleIncorrectData(str);
}
if(LE(c.Du,0)){
        strcat(str,"Cannot accept apex diameter less than or equal to zero.\n"
                                "  This might happen because of incorrect data or file\n"
                                "  format.  Please check data file for possible errors.");
        HandleIncorrectData(str);
```

```
}
if(LE(c.h,0)){
        strcat(str,"Cannot accept cyclone free vortex height less than or \n"
                                "  equal to zero. This might happen because of incorrect\n"
                                "  data or file format. Please check data file for\n"
                                "  possible errors.");
        HandleIncorrectData(str);
}
if(LE(rhos,0)){
        strcat(str,"Cannot accept solid density less than or equal to zero.\n"
                                "  This might happen because of incorrect data or file\n"
                                "  format. Please check data file for possible errors.");
        HandleIncorrectData(str);
}
if(cycNums<=0){
        strcat(str,"Cannot accept cyclone numbers of a cyclopak less than 1.\n"
                                "  This might happen because of incorrect data or file\n"
                                "  format. Please check data file for possible errors.");
        HandleIncorrectData(str);
}
if(LE(d50CorrFactor,0) || LE(pCorrFactor,0) || LE(sCorrFactor,0) || LE(mCorrFactor,0)
        || LE(RfCorrFactor,0)){
        strcat(str,"Cannot accept any adjusting factor less than or equal to\n"
                                "  zero. This might happen because of incorrect data or\n"
                                "  file format. Please check data file for possible errors.");
        HandleIncorrectData(str);
}
if(maxIter<=0){
        strcat(str,"Cannot accept maximum number of iterations less than or \n"
                                "  equal to zero. This might happen because of incorrect \n"
                                "  data or file format. Please check data file for possible\n"
                                "  errors.");
        HandleIncorrectData(str);
}
}
/* Begin calculations */
cFliquidTonnage=strmMatPtr[cFstrmIndex][4]/cycNums;
cFsolidsTonnage=strmMatPtr[cFstrmIndex][2]/cycNums;
phi=100*(cFsolidsTonnage/rhos)/((cFliquidTonnage/rho)+(cFsolidsTonnage/rhos));
rhop=(phi/100*rhos)+(1-phi/100)*rho;
Q=cFsolidsTonnage*1000/(rhop*60);
/* Calculate d50 */
cp.d50=(50.5*pow(c.Dc,0.46)*pow(c.Di,0.6)*pow(c.Do,1.21)*
                exp(0.063*phi))/(pow(c.Du,0.71)*pow(c.h,0.38)*pow(Q,0.45)*pow((rhos-rho),0.5));
cp.d50=d50CorrFactor*cp.d50;
/* Calculate P in kilo pascals */
P=(1.88*pow(Q,1.78)*exp(0.0055*phi))/(pow(c.Dc,0.37)*pow(c.Di,0.94)*
        pow(c.h,0.28)*pow((c.Du*c.Du+c.Do*c.Do),0.87));
P=pCorrFactor*P;
/* converting P from kilo pascal to meter of pulp */
H=P/(rhop*9.8066);
```

```
/* calculate S  dimensionless */
S=(1.9*pow(c.Du/c.Do,3.31)*pow(c.h,0.54)*pow((c.Du*c.Du+c.Do*c.Do),0.36)*
        exp(0.0054*phi))/(pow(H,0.24)*pow(c.Dc,1.11));
S=sCorrFactor*S;
/* calculate Rs and Rf using iterative method */
Rv=S/(1+S);
/* calculate m */
cp.m=1.94*exp(-1.58*Rv)*pow((pow(c.Dc,2)*c.h/Q),0.15);
cp.m=mCorrFactor*cp.m;
/* begin iterations to calculate Rf */
RfOld=1;
RfNew=0.00001;
for(iterCount=1;iterCount<=maxIter;iterCount++){
        /* step 1: calculate sum of the rec. of solids to CUF for each size class */
        for(i=1;i<=sizeClassNums;i++)
                SbySsolidsCUF[i]=cFsolidsTonnage*(strmSizeMatPtr[i][cFstrmIndex+2]/100)*

(RfNew+(1-RfNew)*(1-exp(-0.693*pow((strmSizeMatPtr[i][2]/cp.d50),cp.m))));
        sum=0;
        for(i=1;i<=sizeClassNums;i++) sum=sum+SbySsolidsCUF[i];
        for(i=1;i<=sizeClassNums;i++)
                strmSizeMatPtr[i][cUFstrmIndex+2]=100*SbySsolidsCUF[i]/sum;
        /* step 2: calculate Rs, total rec. of feed solids to CUF */
        Rs=sum/cFsolidsTonnage;
        /* calculate Rf  dimensionless */
        RfNew=(Rv-(Rs*phi/100))/(1-phi/100);
        if(EQ(RfNew,RfOld)){
                cp.Rf=RfNew;
                sumSolidsCOF=cFsolidsTonnage-sum;
                for(i=1;i<=sizeClassNums;i++){

SbySsolidsCOF[i]=(cFsolidsTonnage*(strmSizeMatPtr[i][cFstrmIndex+2]/100)-
                                                        SbySsolidsCUF[i]);
                        if(fabs(SbySsolidsCOF[i])<=EPS)  SbySsolidsCOF[i]=0.00;
                        strmSizeMatPtr[i][cOFstrmIndex+2]=100*SbySsolidsCOF[i]/sumSolidsCOF;
                }
                break;
        }
        if(iterCount==maxIter){
                clrscr();
                gotoxy(1,5);
                printf("    Rope discharge condition might exist or the maximum number of\n");
                printf("    iterations to calculate Rf is too low.  The simulator currently\n");
                printf("    cannot predict cyclone performance under rope discharge operating\n");
                printf("    conditions.  Datafile must be modified to prevent this situation.\n");
                printf("    First try to use a very large maximum iteration number.  Then, if\n");
                printf("    the situation persists, you have to check other simulation data\n");
                printf("    such as percent solids and cyclone geometry.\n\n\n");
                if(!(simStatusFilePtr=fopen("simstat.lst","w"))){
                        printf("Cannot open the file simstat.lst");
                        exit(EXIT_FAILURE);
```

```
                }
                fprintf(simStatusFilePtr,"exit");
                fclose(simStatusFilePtr);
                printf("     Please press any key to exit");
                getch();
                exit(EXIT_FAILURE);
        }
RfOld=RfNew;
}
cp.Rf=RfCorrFactor*cp.Rf;
/* calculate CUF stream solids, %solids and water for each cyclone */
cUFsolids=Rs*cFsolidsTonnage;
cUFwater=cp.Rf*cFliquidTonnage;
cUFperSolids=100*cUFsolids/(cUFsolids+cUFwater);
/* calculate COF stream solids, %solids and water */
cOFsolids=cFsolidsTonnage-cUFsolids;
cOFwater=(1-cp.Rf)*cFliquidTonnage;
cOFperSolids=100*cOFsolids/(cOFsolids+cOFwater);
/* calculate CUF stream solids, %solids and water */
strmMatPtr[cUFstrmIndex][2]=cUFsolids*cycNums;
strmMatPtr[cUFstrmIndex][4]=cUFwater*cycNums;
strmMatPtr[cUFstrmIndex][3]=cUFperSolids;
/* calculate COF stream solids, %solids and water */
strmMatPtr[cOFstrmIndex][2]=cOFsolids*cycNums;
strmMatPtr[cOFstrmIndex][4]=cOFwater*cycNums;
strmMatPtr[cOFstrmIndex][3]=cOFperSolids;
for(i=1;i<=cycPakNums;i++)
        if(cycParamMatPtr[i][1]==conMatPtr[curNodeNum][3]){
                cycParamMatPtr[i][2]=cp.d50;
                cycParamMatPtr[i][3]=P;
                cycParamMatPtr[i][4]=S;
                cycParamMatPtr[i][5]=cp.m;
                cycParamMatPtr[i][6]=cp.Rf;
        }
if(curNodeNum<nodeNums)  curNodeNum++;
else curNodeNum=1;
/* update files */
UpdateStreamMatrix(streamNums,strmMatPtr);
UpdateStrmSizeMat(strmSizeMatPtr,sizeClassNums,streamNums);
UpdateSimulationStatus(curNodeNum,cnvrgNodeNums,cnvrgVecPtr,iter);
WriteCycParams(cycParamMatPtr,cycPakNums);
/* free allocated memory blocks */
FreeVectorD(SbySsolidsCUF,1,sizeClassNums);
FreeVectorD(SbySsolidsCOF,1,sizeClassNums);
FreeMatrixInt(conMatPtr,1,nodeNums,1,streamNums+3);
FreeMatrixD(strmMatPtr,1,streamNums,1,STRMMATCOLS);
FreeMatrixD(strmSizeMatPtr,1,sizeClassNums,1,streamNums+2);
return 0;
}

int FindCycStreamsIndex(int streamNums,int curNodeNum,int **conMatPtr,
```

```
                                                    int *cF,int *cOF,int *cUF){
int j;

for(j=4;j<=streamNums+3;j++)
        if(conMatPtr[curNodeNum][j]==+1) *cF=j-3;
        else if(conMatPtr[curNodeNum][j]==-2) *cOF=j-3; /* -2 is COF stream code */
        else if(conMatPtr[curNodeNum][j]==-3) *cUF=j-3; /* -3 is CUF stream code */
return 0;
}
```

```
/* junction.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <float.h>
#define STRMMATCOLS      4
#define PULP_STRM_CODE   1
#define CYCLONE_STRM_OF  -2
#define CYCLONE_STRM_UF  -3
#define WATER_STRM_CODE  1000

/* code for various physical and conceptual units in the circuits */
#define BM_UNIT_CODE          1
#define HCYC_UNIT_CODE        2
#define JUNC_NODE_CODE        3
#define SPLIT_NODE_CODE       4
#define FIXCLASS_NODE_CODE    5
#define CNVRG_BLOCK_CODE      100

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y))<=fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

typedef enum bool {FALSE=0,TRUE=1} bool;

int *CreateVectorInt(long nl,long nh);
double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
int **CreateMatrixInt(long nrl,long nrh,long ncl,long nch);
void FreeMatrixInt(int **m,long nrl,long nrh,long ncl,long nch);
double **CreateMatrixD(long nrl,long nrh,long ncl,long nch);
void FreeMatrixD(double **m,long nrl,long nrh,long ncl,long nch);

int ReadCircuitSpec(int *cirNo);
int ReadNumsOfNodesStrms(int cirNo,int *nodeNums,int *streamNums);
int ReadSizeClassNums(int *sizeClassNums);
int ReadSimulationStatus(int *curNodeNumPtr,int cnvrgNodeNums,
                                                    bool *convergence,int
*compIterPtr);
int UpdateSimulationStatus(int curNodeNum,int cnvrgNodeNums,
                                                        bool *convergence,int
iter);
int ReadConnectivityMatrix(int **conMatPtr,int cirNo);
int ReadStreamMatrix(int streamNums,double **strmMatPtr);
int UpdateStreamMatrix(int streamNums,double **strmMatPtr);
int ReadStreamsSizeDist(double **strmSizeMatPtr,int sizeClassNums,
                                                    int streamNums);
```

```
int UpdateStrmSizeMat(double **strmSizeMatPtr,int sizeClassNums,
                                                    int streamNums);
int FindNumsInputStrmJunction(int streamNums,int curNodeNum,
                                                                    int
**conMatPtr,int *nodeInputNums);
int CheckWatAddStrmExist(int streamNums,int curNodeNum,int **conMatPtr,
                                                        bool *existWatAdd);
int FindJunctionPntStreamsIndex(int streamNums,int curNodeNum,int **conMatPtr,
                                                                    int
*nodeInputIndexes,int *nodeWatInputIndex,
                                                                    int
*nodeOutputIndex,int *nodeInputNums);
int HandleIncorrectData(char *str);

int main(){
char str[160];
int **conMatPtr;
int i,j,k,iter;
int sizeClassNums,curNodeNum;
int cirNo,nodeNums,streamNums;
int *nodeInputIndexes, nodeInputNums;
int  nodeWatInputIndex,nodeOutputIndex;
int cnvrgNodeNums;
double **strmSizeMatPtr, **strmMatPtr;
double sum,massOfSolidsInPulps,massOfWaterInPulps,totalMassOfPulp;
bool existWatAdd;
bool *cnvrgVecPtr;

ReadCircuitSpec(&cirNo);
ReadNumsOfNodesStrms(cirNo,&nodeNums,&streamNums);
ReadSizeClassNums(&sizeClassNums);
/* allocate exact memory needed for reading connectivity matrix */
conMatPtr=CreateMatrixInt(1,nodeNums,1,streamNums+3);
for(i=1;i<=nodeNums;i++)
        for(j=1;j<=streamNums+3;j++)
                conMatPtr[i][j]=0.00;
ReadConnectivityMatrix(conMatPtr,cirNo);
cnvrgNodeNums=0;
for(i=1;i<=nodeNums;i++)
        if(conMatPtr[i][2]==CNVRG_BLOCK_CODE) cnvrgNodeNums++;
cnvrgVecPtr=(bool *) CreateVectorInt(1,cnvrgNodeNums);
        for(i=1;i<=cnvrgNodeNums;i++)
                cnvrgVecPtr[i]=0;
ReadSimulationStatus(&curNodeNum,cnvrgNodeNums,cnvrgVecPtr,&iter);
/* allocate exact memory needed for reading streams information */
strmMatPtr=CreateMatrixD(1,streamNums,1,STRMMATCOLS);
for(i=1;i<=streamNums;i++)
        for(j=1;j<=STRMMATCOLS;j++)
                strmMatPtr[i][j]=0.00;
ReadStreamMatrix(streamNums,strmMatPtr);
/* check data */
```

```
if(iter<2){
for(i=1;i<=streamNums;i++)
        for(j=2;j<=STRMMATCOLS;j++)
                if(LT(strmMatPtr[i][j],0)){
                        str[0]='\0';
                        strcat(str,"Cannot accept negative flow rates or percent solids."
                                                "  This might happen because of incorrect data
or file\n"
                                                "  format.  Please check data file for possible
errors.");
                        HandleIncorrectData(str);
                }
}
/* allocate exact memory needed for reading size distributions */
strmSizeMatPtr=CreateMatrixD(1,sizeClassNums,1,streamNums+2);
for(i=1;i<=sizeClassNums;i++)
        for(j=1;j<=streamNums+2;j++)
                strmSizeMatPtr[i][j]=0.00;
ReadStreamsSizeDist(strmSizeMatPtr,sizeClassNums,streamNums);
/* check data */
if(iter<2)
        for(i=1;i<=sizeClassNums;i++)
                for(j=3;j<=streamNums+2;j++)
                        if(LT(strmSizeMatPtr[i][j],0) || GT(strmSizeMatPtr[i][j],100)){
                                str[0]='\0';
                                strcat(str,"Cannot accept a percent mass less than zero or greater
than 100");
                                HandleIncorrectData(str);
                        }
FindNumsInputStrmJunction(streamNums,curNodeNum,conMatPtr,&nodeInputNums);
nodeInputIndexes=CreateVectorInt(1,nodeInputNums);
FindJunctionPntStreamsIndex(streamNums,curNodeNum,conMatPtr,nodeInputIndexes,

&nodeWatInputIndex,&nodeOutputIndex,
                                                                        &nodeInputNums);
massOfSolidsInPulps=0.0;
for(i=1;i<=nodeInputNums;i++)
        massOfSolidsInPulps=massOfSolidsInPulps+strmMatPtr[nodeInputIndexes[i]][2];
for(i=1;i<=sizeClassNums;i++){
        sum=0;
        for(j=1;j<=nodeInputNums;j++)

sum=sum+strmSizeMatPtr[i][nodeInputIndexes[j]+2]*strmMatPtr[nodeInputIndexes[j]][2];
        strmSizeMatPtr[i][nodeOutputIndex+2]=sum/massOfSolidsInPulps;
}
/* calculate output solids, %solids, water */
strmMatPtr[nodeOutputIndex][2]=massOfSolidsInPulps;
massOfWaterInPulps=0;
for(i=1;i<=nodeInputNums;i++)
        massOfWaterInPulps=massOfWaterInPulps+strmMatPtr[nodeInputIndexes[i]][4];
existWatAdd=FALSE;
```

```
CheckWatAddStrmExist(streamNums,curNodeNum,conMatPtr,&existWatAdd);
if(existWatAdd)
        totalMassOfPulp=massOfSolidsInPulps+massOfWaterInPulps+
                                                strmMatPtr[nodeWatInputIndex][4];
else  totalMassOfPulp=massOfSolidsInPulps+massOfWaterInPulps;
strmMatPtr[nodeOutputIndex][3]=100*massOfSolidsInPulps/totalMassOfPulp;
if(existWatAdd)
        strmMatPtr[nodeOutputIndex][4]=massOfWaterInPulps+

strmMatPtr[nodeWatInputIndex][4];
else strmMatPtr[nodeOutputIndex][4]=massOfWaterInPulps;
if(curNodeNum<nodeNums)  curNodeNum++;
else curNodeNum=1;
/* update files */
UpdateStreamMatrix(streamNums,strmMatPtr);
UpdateStrmSizeMat(strmSizeMatPtr,sizeClassNums,streamNums);
UpdateSimulationStatus(curNodeNum,cnvrgNodeNums,cnvrgVecPtr,iter);
/* free allocated memory blocks */
FreeMatrixInt(conMatPtr,1,nodeNums,1,streamNums+3);
FreeMatrixD(strmMatPtr,1,streamNums,1,STRMMATCOLS);
FreeMatrixD(strmSizeMatPtr,1,sizeClassNums,1,streamNums+2);
return 0;
}

FindNumsInputStrmJunction(int streamNums,int curNodeNum,
                                                        int **conMatPtr,int
*nodeInputNums){
int i,j;

i=1;
for(j=4;j<=streamNums+3;j++)
        if(conMatPtr[curNodeNum][j]==+1) i++;
*nodeInputNums=i-1;
return 0;
}

CheckWatAddStrmExist(int streamNums,int curNodeNum,int **conMatPtr,
                                                bool *existWatAdd){
int j;

for(j=4;j<=streamNums+3;j++)
        if(conMatPtr[curNodeNum][j]==WATER_STRM_CODE) *existWatAdd=TRUE;
return 0;
}

int FindJunctionPntStreamsIndex(int streamNums,int curNodeNum,int **conMatPtr,
                                                                        int
*nodeInputIndexes,int *nodeWatInputIndex,
                                                                        int
*nodeOutputIndex,int *nodeInputNums){
int i,j;
```

```
i=1;
for(j=4; j<=streamNums+3; j++){
        if(conMatPtr[curNodeNum][j]==+1){
                nodeInputIndexes[i]=j-3;
                i++;
        }
        else if(conMatPtr[curNodeNum][j]==WATER_STRM_CODE) *nodeWatInputIndex=j-3;
        else if(conMatPtr[curNodeNum][j]==-1) *nodeOutputIndex=j-3;
}
*nodeInputNums=i-1;
return 0;
}
```

```
/* split.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <float.h>

#define STRMMATCOLS      4
#define PULP_STRM_CODE   1
#define CYCLONE_STRM_OF  -2
#define CYCLONE_STRM_UF  -3
#define WATER_STRM_CODE  1000

/* code for various physical and conceptual units in the circuits */
#define BM_UNIT_CODE          1
#define HCYC_UNIT_CODE        2
#define JUNC_NODE_CODE        3
#define SPLIT_NODE_CODE       4
#define FIXCLASS_NODE_CODE    5
#define CNVRG_BLOCK_CODE      100

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

typedef enum bool {FALSE=0,TRUE=1} bool;

int *CreateVectorInt(long nl,long nh);
double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
int **CreateMatrixInt(long nrl,long nrh,long ncl,long nch);
void FreeMatrixInt(int **m,long nrl,long nrh,long ncl,long nch);
double **CreateMatrixD(long nrl,long nrh,long ncl,long nch);
void FreeMatrixD(double **m,long nrl,long nrh,long ncl,long nch);
int ReadCircuitSpec(int *cirNo);
int ReadNumsOfNodesStrms(int cirNo,int *nodeNums,int *streamNums);
int ReadSizeClassNums(int *sizeClassNums);
int ReadSimulationStatus(int *curNodeNumPtr,int cnvrgNodeNums,
                                                    bool *convergence,int
*compIterPtr);
int UpdateSimulationStatus(int curNodeNum,int cnvrgNodeNums,
                                                    bool *convergence,int
compIter);
int ReadConnectivityMatrix(int **conMatPtr,int cirNo);
int ReadStreamMatrix(int streamNums,double **strmMatPtr);
int UpdateStreamMatrix(int streamNums, double **strmMatPtr);
int ReadStreamsSizeDist(double **strmSizeMatPtr,int sizeClassNums,
                                                    int streamNums);
```

```
int UpdateStrmSizeMat(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int ReadSplitOutputNums(int **conMatPtr,int curNodeNum,int *splitOutputNums);
int ReadSplitInputData(int **conMatPtr,int curNodeNum,double **splitMatPtr);
int FindNumsOutputsSplitter(int streamNums,int curNodeNum,int **conMatPtr,int *splitOutputNums);
int FindSplitStrmsIndex(int streamNums,int curNodeNum,int **conMatPtr,
                                                    int *splitOutputIndexes,int
*splitInputIndex);
int HandleIncorrectData(char *str);

int main(){
char str[160];
int **conMatPtr;
int i,j,r,iter;
int sizeClassNums,curNodeNum;
int cirNo,nodeNums,streamNums;
int splitInputIndex,splitOutputNums;
int *splitOutputIndexes;
int cnvrgNodeNums;
double **strmSizeMatPtr, **strmMatPtr,**splitMatPtr;
double sum,massOfSolidsInPulps,massOfWaterInPulps,totalMassOfPulp;
bool done,*cnvrgVecPtr;

ReadCircuitSpec(&cirNo);
ReadNumsOfNodesStrms(cirNo,&nodeNums,&streamNums);
/* allocate exact memory needed for reading connectivity matrix */
conMatPtr=CreateMatrixInt(1,nodeNums,1,streamNums+3);
for(i=1;i<=nodeNums;i++)
        for(j=1;j<=streamNums+3;j++)
                conMatPtr[i][j]=0.00;
ReadConnectivityMatrix(conMatPtr,cirNo);
cnvrgNodeNums=0;
for(i=1;i<=nodeNums;i++)
  if(conMatPtr[i][2]==CNVRG_BLOCK_CODE)
          cnvrgNodeNums++;
cnvrgVecPtr=(bool *) CreateVectorInt(1,cnvrgNodeNums);
        for(i=1;i<=cnvrgNodeNums;i++)
                cnvrgVecPtr[i]=0;
ReadSimulationStatus(&curNodeNum,cnvrgNodeNums,cnvrgVecPtr,&iter);
/* allocate exact memory needed for reading streams information */
strmMatPtr=CreateMatrixD(1,streamNums,1,STRMMATCOLS);
for(i=1;i<=streamNums;i++)
        for(j=1;j<=STRMMATCOLS;j++)
                strmMatPtr[i][j]=0.00;
ReadStreamMatrix(streamNums,strmMatPtr);
/* check data */
if(iter<2){
        for(i=1;i<=streamNums;i++)
                for(j=2;j<=STRMMATCOLS;j++)
                        if(LT(strmMatPtr[i][j],0)){
                                str[0]='\0';
                                strcat(str,"Cannot accept negative flow rates or percent solids."
```

```
                                                    " This might happen because of
incorrect data or file\n"
                                                    " format. Please check data file for
possible errors.");
                                HandleIncorrectData(str);
                        }
}
ReadSizeClassNums(&sizeClassNums);
/* allocate exact memory needed for reading size distributions */
strmSizeMatPtr=CreateMatrixD(1,sizeClassNums,1,streamNums+2);
for(i=1;i<=sizeClassNums;i++)
        for(j=1;j<=streamNums+2;j++)
                strmSizeMatPtr[i][j]=0.00;
ReadStreamsSizeDist(strmSizeMatPtr,sizeClassNums,streamNums);
ReadSplitOutputNums(conMatPtr,curNodeNum,&splitOutputNums);
splitMatPtr=CreateMatrixD(1,splitOutputNums,1,2);
for(i=1;i<=splitOutputNums;i++)
        for(j=1;j<=2;j++)
                splitMatPtr[i][j]=0.00;
ReadSplitInputData(conMatPtr,curNodeNum,splitMatPtr);
if(iter<2){
        sum=0;
        for(i=1;i<=splitOutputNums;i++)
                sum=sum+splitMatPtr[i][2];
        if(!EQ(sum,100)){
                str[0]='\0';
                        strcat(str,"The sum of splitting factors for a split node must be\n"
                                                    " equal to 100. This might happen because of
incorrect\n"
                                                    " data or file format. Please check data file
for\n"
                                                    " possible errors.");
                        HandleIncorrectData(str);
        }
}
FindNumsOutputsSplitter(streamNums,curNodeNum,conMatPtr,&splitOutputNums);
splitOutputIndexes=CreateVectorInt(1,splitOutputNums);
        for(i=1;i<=splitOutputNums;i++)
                splitOutputIndexes[i]=0;
/* find convergence point streams indexes */
FindSplitStrmsIndex(streamNums,curNodeNum,conMatPtr,splitOutputIndexes,&splitInputIndex);
/* calculate splitter output streams variables */
for(i=1;i<=splitOutputNums;i++){
        for(r=1;r<=splitOutputNums;r++)
                if(splitOutputIndexes[i]==splitMatPtr[r][1])break;
        strmMatPtr[splitOutputIndexes[i]][2]=strmMatPtr[splitInputIndex][2]*

         splitMatPtr[r][2]/100;
        strmMatPtr[splitOutputIndexes[i]][4]=strmMatPtr[splitInputIndex][4]*

         splitMatPtr[r][2]/100;
```

```
        if(GT(strmMatPtr[splitOutputIndexes[i]][2],0))
                strmMatPtr[splitOutputIndexes[i]][3]=100*strmMatPtr[splitOutputIndexes[i]][2]/

        (strmMatPtr[splitOutputIndexes[i]][2]+

          strmMatPtr[splitOutputIndexes[i]][4]);
        else strmMatPtr[splitOutputIndexes[i]][3]=0.0;
        for(r=1;r<=sizeClassNums;r++)
          strmSizeMatPtr[r][splitOutputIndexes[i]+2]=strmSizeMatPtr[r][splitInputIndex+2];
}
if(curNodeNum<nodeNums) curNodeNum++;
else curNodeNum=1;
/* update files */
UpdateStreamMatrix(streamNums,strmMatPtr);
UpdateStrmSizeMat(strmSizeMatPtr,sizeClassNums,streamNums);
UpdateSimulationStatus(curNodeNum,cnvrgNodeNums,cnvrgVecPtr,iter);
/* free pointers */
FreeMatrixInt(conMatPtr,1,nodeNums,1,streamNums+3);
FreeMatrixD(strmMatPtr,1,streamNums,1,STRMMATCOLS);
FreeMatrixD(strmSizeMatPtr,1,sizeClassNums,1,streamNums+2);
return 0;
}

FindNumsOutputsSplitter(int streamNums,int curNodeNum,int **conMatPtr,
                                                        int *splitOutputNums){
int i,j;

i=1;
for(j=4;j<=streamNums+3;j++)
        if(conMatPtr[curNodeNum][j]==-1) i++;
*splitOutputNums=i-1;
return 0;
}

int FindSplitStrmsIndex(int streamNums,int curNodeNum,int **conMatPtr,
                                                        int *splitOutputIndexes,int
*splitInputIndex){
int i,j;
i=1;
for(j=4;j<=streamNums+3;j++){
        if(conMatPtr[curNodeNum][j]==-1){
                splitOutputIndexes[i]=j-3;
                i++;
        }
        else if(conMatPtr[curNodeNum][j]==+1) *splitInputIndex=j-3;
}
return 0;
}
```

```
/* fixclass.c01 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include <conio.h>
#include <assert.h>

/* code for various physical and conceptual units in the circuits */
#define BM_UNIT_CODE          1
#define HCYC_UNIT_CODE        2
#define JUNC_NODE_CODE        3
#define SPLIT_NODE_CODE       4
#define FIXCLASS_NODE_CODE    5
#define CNVRG_BLOCK_CODE      100

/* constants */
#define STRMMATCOLS 4

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y))<=fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

typedef enum bool {FALSE=0,TRUE=1} bool;

int *CreateVectorInt(long nl,long nh);
void FreeVectorInt(int *v,long nl,long nh);
float *CreateVector(long nl,long nh);
void FreeVector(float *v,long nl,long nh);
double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
int **CreateMatrixInt(long nrl,long nrh,long ncl,long nch);
void FreeMatrixInt(int **m,long nrl,long nrh,long ncl,long nch);
double **CreateMatrixD(long nrl,long nrh,long ncl,long nch);
void FreeMatrixD(double **m,long nrl,long nrh,long ncl,long nch);

int ReadCircuitSpec(int *cirNo);
int ReadNumsOfNodesStrms(int cirNo,int *nodeNums,int *streamNums);
int ReadSizeClassNums(int *sizeClassNums);
int ReadSimulationStatus(int *curNodeNumPtr,int cnvrgNodeNums,
                                                    bool *convergence,int
*compIterPtr);
int UpdateSimulationStatus(int curNodeNum,int cnvrgNodeNums,bool *convergence,int compIter);
int ReadConnectivityMatrix(int **conMatPtr,int cirNo);
int ReadStreamMatrix(int streamNums,double **strmMatPtr);
int UpdateStreamMatrix(int streamNums,double **strmMatPtr);
int ReadStreamsSizeDist(double **strmSizeMatPtr,
```

```
                                                    int sizeClassNums,int
streamNums);
int UpdateStrmSizeMat(double **strmSizeMatPtr,int sizeClassNums,
                                                    int streamNums);
int FindClassifierStreamsIndex(int streamNums,int curNodeNum,int **conMatPtr,
                                                    int *sF,int *sOF,int
*sUF);
int ReadClassifierInputData(int **conMatPtr,int curNodeNum,int sizeClassNums,
                                                    float *cPtr,float *watSplit);
int HandleIncorrectData(char *str);

int main(){
FILE *simStatusFilePtr;
char str[160];
int **conMatPtr;
int cirNo,nodeNums,streamNums;
int sizeClassNums,curNodeNum;
int i,j,iter;
int sFstrmIndex,sOFstrmIndex,sUFstrmIndex;
int cnvrgNodeNums;
double sum;
double **strmSizeMatPtr,**strmMatPtr;
double *SbySsolidsScrOF,*SbySsolidsScrUF;
double sumSolidsScrUF;
double sFliquidTonnage,sFsolidsTonnage;
double Rsol; /* rec. of feed solides to the underflow product */
double sUFsolids,sUFwater,sUFperSolids,sOFsolids,sOFwater,sOFperSolids;
float *opc;
float watSplit;
bool done,*cnvrgVecPtr;

ReadCircuitSpec(&cirNo);
ReadNumsOfNodesStrms(cirNo,&nodeNums,&streamNums);
ReadSizeClassNums(&sizeClassNums);
/* allocate memory */
SbySsolidsScrOF=CreateVectorD(1,sizeClassNums);
SbySsolidsScrUF=CreateVectorD(1,sizeClassNums);
for(i=1;i<=sizeClassNums;i++){
        SbySsolidsScrOF[i]=0.00;
        SbySsolidsScrUF[i]=0.00;
}
/* allocate exact memory needed for reading connectivity matrix */
conMatPtr=CreateMatrixInt(1,nodeNums,1,streamNums+3);
for(i=1;i<=nodeNums;i++)
        for(j=1;j<=streamNums+3;j++)
                conMatPtr[i][j]=0.00;
ReadConnectivityMatrix(conMatPtr,cirNo);
cnvrgNodeNums=0;
for(i=1;i<=nodeNums;i++)
  if(conMatPtr[i][2]==CNVRG_BLOCK_CODE)
          cnvrgNodeNums++;
```

```
cnvrgVecPtr=(bool *) CreateVectorInt(1,cnvrgNodeNums);
        for(i=1;i<=cnvrgNodeNums;i++)
                cnvrgVecPtr[i]=0;
ReadSimulationStatus(&curNodeNum,cnvrgNodeNums,cnvrgVecPtr,&iter);
/* allocate exact memory needed for reading streams information */
strmMatPtr=CreateMatrixD(1,streamNums,1,STRMMATCOLS);
for(i=1;i<=streamNums;i++)
        for(j=1;j<=STRMMATCOLS;j++)
                strmMatPtr[i][j]=0.00;
ReadStreamMatrix(streamNums,strmMatPtr);
/* allocate exact memory needed for reading size distributions */
strmSizeMatPtr=CreateMatrixD(1,sizeClassNums,1,streamNums+2);
for(i=1;i<=sizeClassNums;i++)
        for(j=1;j<=streamNums+2;j++)
                strmSizeMatPtr[i][j]=0.00;
ReadStreamsSizeDist(strmSizeMatPtr,sizeClassNums,streamNums);
FindClassifierStreamsIndex(streamNums,curNodeNum,conMatPtr,

&sFstrmIndex,&sOFstrmIndex,&sUFstrmIndex);
/* allocate memory to read oversize partition coefficients */
opc=CreateVector(1,sizeClassNums);
/* initialize c vector */
for(i=1;i<=sizeClassNums;i++) opc[i]=0.0;
ReadClassifierInputData(conMatPtr,curNodeNum,sizeClassNums,opc,&watSplit);
/* check data */
if(iter<2){
        str[0]='\0';
        for(i=1;i<=sizeClassNums;i++)
                if(LT(opc[i],0)||GT(opc[i],100)){
                        strcat(str,"Cannot accept an oversize partition coefficient less than zero or\n"
                                "  greater than 100.  This might happen because of incorrect
data\n"
                                "  or file format.  Please check data file for possible errors.");
                        HandleIncorrectData(str);
                }
        if(LT(watSplit,0)||GT(watSplit,100)){
                strcat(str,"Cannot accept a water split factor less than zero or greater\n"
                                "  than 100.  This might happen because of incorrect data or\n"
                                "  file format.  Please check data file for possible errors.");
                HandleIncorrectData(str);
        }
}
/* begin calculations */
sFliquidTonnage=strmMatPtr[sFstrmIndex][4];
sFsolidsTonnage=strmMatPtr[sFstrmIndex][2];
for(i=1;i<=sizeClassNums;i++){
        SbySsolidsScrOF[i]=sFsolidsTonnage*

(strmSizeMatPtr[i][sFstrmIndex+2]/100)*opc[i]/100;
        SbySsolidsScrUF[i]=sFsolidsTonnage*
                                                (strmSizeMatPtr[i][sFstrmIndex+2]/100)-
```

```
                                                    SbySsolidsScrOF[i];
}
sum=0;
for(i=1;i<=sizeClassNums;i++) sum=sum+SbySsolidsScrOF[i];
sumSolidsScrUF=sFsolidsTonnage-sum;
for(i=1;i<=sizeClassNums;i++){
        strmSizeMatPtr[i][sOFstrmIndex+2]=100*SbySsolidsScrOF[i]/sum;
        strmSizeMatPtr[i][sUFstrmIndex+2]=100*SbySsolidsScrUF[i]/sumSolidsScrUF;
}
/* step 2: calculate Rsol, total rec. of feed solids to SOF */
Rsol=sum/sFsolidsTonnage;
/* calculate SOF stream solids, %solids and water for each cyclone */
sOFsolids=Rsol*sFsolidsTonnage;
sOFwater=watSplit*sFliquidTonnage/100;
sOFperSolids=100*sOFsolids/(sOFsolids+sOFwater);
/* calculate SUF stream solids, %solids and water */
sUFsolids=sFsolidsTonnage-sOFsolids;
sUFwater=(100-watSplit)*sFliquidTonnage/100;
sUFperSolids=100*sUFsolids/(sUFsolids+sUFwater);
strmMatPtr[sOFstrmIndex][2]=sOFsolids;
strmMatPtr[sOFstrmIndex][4]=sOFwater;
strmMatPtr[sOFstrmIndex][3]=sOFperSolids;
strmMatPtr[sUFstrmIndex][2]=sUFsolids;
strmMatPtr[sUFstrmIndex][4]=sUFwater;
strmMatPtr[sUFstrmIndex][3]=sUFperSolids;
if(curNodeNum<nodeNums)  curNodeNum++;
else curNodeNum=1;
/* update files */
UpdateStreamMatrix(streamNums,strmMatPtr);
UpdateStrmSizeMat(strmSizeMatPtr,sizeClassNums,streamNums);
UpdateSimulationStatus(curNodeNum,cnvrgNodeNums,cnvrgVecPtr,iter);
/* free allocated memory blocks */
FreeVectorD(SbySsolidsScrOF,1,sizeClassNums);
FreeVectorD(SbySsolidsScrUF,1,sizeClassNums);
FreeMatrixInt(conMatPtr,1,nodeNums,1,streamNums+3);
FreeMatrixD(strmMatPtr,1,streamNums,1,STRMMATCOLS);
FreeMatrixD(strmSizeMatPtr,1,sizeClassNums,1,streamNums+2);
FreeVector(opc,1,sizeClassNums);
return 0;
}

int FindClassifierStreamsIndex(int streamNums,int curNodeNum,int **conMatPtr,
                                                                    int *sF,int *sOF,int
*sUF){
int j;
for(j=4;j<=streamNums+3;j++)
        if(conMatPtr[curNodeNum][j]==+1) *sF=j-3;
        else if(conMatPtr[curNodeNum][j]==-2) *sOF=j-3; /* -2 is SOF stream code */
        else if(conMatPtr[curNodeNum][j]==-3) *sUF=j-3; /* -3 is SUF stream code */
return 0;
}
```

```
/* converge.c01 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <float.h>

#define STRMMATCOLS      4
#define PULP_STRM_CODE   1
#define CYCLONE_STRM_OF  -2
#define CYCLONE_STRM_UF  -3
#define WATER_STRM_CODE  1000

/* code for various physical and conceptual units in the circuits */
#define BM_UNIT_CODE          1
#define HCYC_UNIT_CODE        2
#define JUNC_NODE_CODE        3
#define SPLIT_NODE_CODE       4
#define FIXCLASS_NODE_CODE    5
#define CNVRG_BLOCK_CODE      100

#define EPS (DBL_EPSILON*100)
#define EQ(x,y) (fabs((x)-(y)) <= fabs((x)+(y))*EPS)
#define LE(x,y) ((x)<(y) || EQ(x,y))
#define GE(x,y) ((y)<(x) || EQ(x,y))
#define LT(x,y) (!GE(x,y))
#define GT(x,y) (!LE(x,y))

typedef enum bool {FALSE=0,TRUE=1} bool;

int *CreateVectorInt(long nl,long nh);
double *CreateVectorD(long nl,long nh);
void FreeVectorD(double *v,long nl,long nh);
int **CreateMatrixInt(long nrl,long nrh,long ncl,long nch);
void FreeMatrixInt(int **m,long nrl,long nrh,long ncl,long nch);
double **CreateMatrixD(long nrl,long nrh,long ncl,long nch);
void FreeMatrixD(double **m,long nrl,long nrh,long ncl,long nch);

int ReadCircuitSpec(int *cirNo);
int ReadNumsOfNodesStrms(int cirNo,int *nodeNums,int *streamNums);
int ReadSizeClassNums(int *sizeClassNums);
int ReadSimulationStatus(int* curNodeNumPtr,int cnvrgNodeNums,bool *convergence,int
*compIterPtr);
int UpdateSimulationStatus(int curNodeNum,int cnvrgNodeNums,bool *convergence,int compIter);
int ReadConnectivityMatrix(int **conMatPtr,int cirNo);
int ReadStreamMatrix(int streamNums,double **strmMatPtr);
int UpdateStreamMatrix(int streamNums,double **strmMatPtr);
int ReadStreamsSizeDist(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int UpdateStrmSizeMat(double **strmSizeMatPtr,int sizeClassNums,int streamNums);
int ReadConvergeInputData(int **conMatPtr,int curNodeNum,double *tolerance);
int FindConvergePntStrmsIndex(int streamNums,int curNodeNum,int **conMatPtr,int
```

```
*nodeInputIndex,int *nodeOutputIndex);
int HandleIncorrectData(char *str);

int main(){
char str[160];
int **conMatPtr;
int i,j,k,iter;
int sizeClassNums,curNodeNum;
int cirNo,nodeNums,streamNums;
int nodeInputIndex,nodeOutputIndex;
int cnvrgNodeNums;
double **strmSizeMatPtr,**strmMatPtr;
double sum,massOfSolidsInPulps,massOfWaterInPulps,totalMassOfPulp;
double tolerance,maxDif;
bool *cnvrgVecPtr;

ReadCircuitSpec(&cirNo);
ReadNumsOfNodesStrms(cirNo,&nodeNums,&streamNums);
ReadSizeClassNums(&sizeClassNums);
/* allocate exact memory needed for reading connectivity matrix */
conMatPtr=CreateMatrixInt(1,nodeNums,1,streamNums+3);
for(i=1;i<=nodeNums;i++)
        for(j=1;j<=streamNums+3;j++)
                conMatPtr[i][j]=0.00;
ReadConnectivityMatrix(conMatPtr,cirNo);
cnvrgNodeNums=0;
for(i=1;i<=nodeNums;i++)
          if(conMatPtr[i][2]==CNVRG_BLOCK_CODE)
                  cnvrgNodeNums++;
cnvrgVecPtr=(bool *) CreateVectorInt(1,cnvrgNodeNums);
        for(i=1;i<=cnvrgNodeNums;i++)
                cnvrgVecPtr[i]=0;
ReadSimulationStatus(&curNodeNum,cnvrgNodeNums,cnvrgVecPtr,&iter);
/* allocate exact memory needed for reading streams information */
strmMatPtr=CreateMatrixD(1,streamNums,1,STRMMATCOLS);
for(i=1;i<=streamNums;i++)
        for(j=1;j<=STRMMATCOLS;j++)
                strmMatPtr[i][j]=0.00;
ReadStreamMatrix(streamNums,strmMatPtr);
/* allocate exact memory needed for reading size distributions */
strmSizeMatPtr=CreateMatrixD(1,sizeClassNums,1,streamNums+2);
for(i=1;i<=sizeClassNums;i++)
        for(j=1;j<=streamNums+2;j++)
                strmSizeMatPtr[i][j]=0.00;
ReadStreamsSizeDist(strmSizeMatPtr,sizeClassNums,streamNums);
ReadConvergeInputData(conMatPtr,curNodeNum,&tolerance);
if(iter<2){
        if(LT(tolerance,EPS)){
                str[0]='\0';
                strcat(str,"To converge at any convergence node, a tolerance value \n"
                                        "    greater than epsilon is required.  This might
```

```
happen\n"
                                        "   because of incorrect data or file format.  Please \n"
                                        "   check data file for possible errors.");
                HandleIncorrectData(str);
        }
}
/* find convergence point streams indexes */
FindConvergePntStrmsIndex(streamNums,curNodeNum,conMatPtr,&nodeInputIndex,&nodeOutputInde
x);
/* begin to test if convergence has been reached or not */
maxDif=strmMatPtr[nodeOutputIndex][2]*tolerance/100;
if(LE(strmMatPtr[nodeInputIndex][2]-strmMatPtr[nodeOutputIndex][2],maxDif))
        cnvrgVecPtr[conMatPtr[curNodeNum][3]]=TRUE;
strmMatPtr[nodeOutputIndex][2]=strmMatPtr[nodeInputIndex][2];
strmMatPtr[nodeOutputIndex][3]=strmMatPtr[nodeInputIndex][3];
strmMatPtr[nodeOutputIndex][4]=strmMatPtr[nodeInputIndex][4];
for(i=1;i<=sizeClassNums;i++)
        strmSizeMatPtr[i][nodeOutputIndex+2]=strmSizeMatPtr[i][nodeInputIndex+2];
if(curNodeNum<nodeNums)  curNodeNum++;
else curNodeNum=1;
/* update files */
UpdateStreamMatrix(streamNums,strmMatPtr);
UpdateStrmSizeMat(strmSizeMatPtr,sizeClassNums,streamNums);
UpdateSimulationStatus(curNodeNum,cnvrgNodeNums,cnvrgVecPtr,iter);
/* free pointers */
FreeMatrixInt(conMatPtr,1,nodeNums,1,streamNums+3);
FreeMatrixD(strmMatPtr,1,streamNums,1,STRMMATCOLS);
FreeMatrixD(strmSizeMatPtr,1,sizeClassNums,1,streamNums+2);
return 0;
}

int FindConvergePntStrmsIndex(int streamNums,int curNodeNum,
                                                                        int
**conMatPtr,int *nodeInputIndex,
                                                                        int
*nodeOutputIndex){
int j;

for(j=4;j<=streamNums+3;j++)
        if(conMatPtr[curNodeNum][j]==+1) *nodeInputIndex=j-3;
        else if(conMatPtr[curNodeNum][j]==-1) *nodeOutputIndex=j-3;
return 0;
}
```

```
;;;    ***********************************************************
;;;    *    Grinding Circuits Optimization Supervisor (GCOS)    *
;;;    *                        By                              *
;;;    *                  Akbar Farzanegan                      *
;;;    *                   August 1998                          *
;;;    ***********************************************************

;=============================================================
;                      TEMPLATES MODULE
;=============================================================
(defmodule TEMPLATES (export ?ALL))

(deftemplate numparam
(slot name)
(slot min (default -1.1e4932) (type NUMBER))
(slot max (default +1.1e4932) (type NUMBER)))

(deftemplate nonnumparam
(slot name)
(multislot values)
(multislot menu)
(slot convert))

(deftemplate parameter
(slot name)
(multislot value)
(slot certainty (default 100.0)))

(deftemplate conclusion-text
(slot text))

(deftemplate conclusion-string
(slot string)
(slot arg1)
(slot arg2))

;=============================================================
;                      FUNCTIONS MODULE
;=============================================================
(defmodule FUNCTIONS (export ?ALL))

(deffunction FUNCTIONS::yes-or-no-p(?question)
(bind ?x bogus)
(while (and (neq ?x y)(neq ?x n))
        (system cls)
        (printout t crlf crlf crlf ?question)
        (bind ?x (lowcase (sym-cat (read)))))
```

```
(if (eq ?x y) then TRUE else FALSE))

(deffunction processhelp(?help)
(system cls)
(printout t crlf crlf crlf
"                              GCOS HELP
    ---------------------------------------------------------------")

(printout t crlf ?help)
(printout t crlf crlf "
    You can quit any time by typing quit, bye or system.  If
    you do not know the answer just type <unknown> or <u>.


    Please press any key to continue ... ")
(system presskey))

(deffunction post(?text)
(system cls)
(printout t crlf crlf crlf ?text crlf)
(printout t crlf "     Please press any key to continue ... ")
(system presskey))

(deffunction ask-numeric(?question ?help ?min ?max)
(system cls)
(printout t crlf crlf crlf crlf crlf crlf crlf crlf
"          GCOS> " ?question)
(bind ?answer (read))
(if (lexemep ?answer) then (bind ?answer (lowcase ?answer)))
(if (or (eq ?answer h)(eq ?answer help)(eq ?answer "?"))
then    (processhelp ?help))
(if (or (eq ?answer u)(eq ?answer unknown)) then (return unknown))
(while (or (not (numberp ?answer)) (< ?answer ?min) (> ?answer ?max) ) do
(if (or (eq ?answer q)(eq ?answer quit)(eq ?answer bye)(eq ?answer system))
        then    (set-current-module RESET)(focus RESET)(break))
(system cls)
(printout t crlf crlf crlf crlf crlf crlf crlf crlf
"          GCOS> " ?question)
(bind ?answer (read))
(if (lexemep ?answer) then (bind ?answer (lowcase ?answer)))
(if (or (eq ?answer h)(eq ?answer help)(eq ?answer "?"))
        then (processhelp ?help))
(if (or (eq ?answer q)(eq ?answer quit)(eq ?answer bye)(eq ?answer system))
        then (set-current-module RESET)(focus RESET)(break))
(if (or (eq ?answer u)(eq ?answer unknown)) then (return unknown)))
(return ?answer))

(deffunction ask-non-numeric(?question ?help $?menu)
(system cls)
(printout t crlf crlf crlf crlf crlf crlf crlf crlf
"          GCOS> " ?question)
```

```
(bind ?answer (read))
(if (lexemep ?answer) then (bind ?answer (lowcase ?answer)))
(if (or (eq ?answer h)(eq ?answer help)(eq ?answer "?"))
        then (processhelp ?help))
(if (or (eq ?answer u)(eq ?answer unknown)) then (return unknown))
(while (not(member$ ?answer ?menu)) do
(if (or (eq ?answer q)(eq ?answer quit)(eq ?answer bye)(eq ?answer system))
        then (set-current-module RESET)(focus RESET)(break))
(system cls)
(printout t crlf crlf crlf crlf crlf crlf crlf crlf
"       GCOS> " ?question)
(bind ?answer (read))
(if (or (eq ?answer h)(eq ?answer help)(eq ?answer "?"))
        then (processhelp ?help))
(if (or (eq ?answer q)(eq ?answer quit)(eq ?answer bye)(eq ?answer system))
      then (set-current-module RESET)(focus RESET)(break))
(if (or (eq ?answer u)(eq ?answer unknown)) then (return unknown)))
(return ?answer))

(deffunction check-simulation-status(?input-file)
(open ?input-file  src "r")
(bind ?simulation-status (read src))
(close src)
(return  ?simulation-status))

;=========================================================
;                    QUERY MODULE
;=========================================================
(defmodule QUERY (import TEMPLATES ?ALL)(import FUNCTIONS ?ALL)(export ?ALL))

(deftemplate QUERY::question
        (slot parameter (default ?NONE))
        (slot module (default ?NONE))
        (slot the-question (default ?NONE))
        (multislot precursors (default ?DERIVE))
        (slot already-asked (default FALSE))
        (slot help (default ?NONE)))

(defrule QUERY::ask-numeric-question
?f <- (question (parameter ?the-parameter)
                (module ?the-module)
                (the-question ?the-question)
                (precursors)
                (already-asked FALSE)
                (help ?the-help))
    (parameter (name current-module)(value ?the-module))
    (numparam (name ?the-parameter) (min ?min) (max ?max))
 =>
(modify ?f (already-asked TRUE))
```

```
(assert (parameter (name ?the-parameter)
(value  (ask-numeric ?the-question ?the-help ?min ?max)))))

(defrule QUERY::ask-non-numeric-question
?f <- (question (parameter ?the-parameter)
                (module ?the-module)
                (the-question ?the-question)
                (precursors)
                (already-asked FALSE)
                (help ?the-help))
    (parameter (name current-module)(value ?the-module))
    (nonnumparam (name ?the-parameter)(menu $?the-menu))
=>
(modify ?f (already-asked TRUE))
(assert (user-response ?the-parameter
        (ask-non-numeric ?the-question ?the-help $?the-menu))))

(defrule QUERY::precursor-is-satisfied
?f <- (question (already-asked FALSE)
    (precursors ?name is ?value $?rest))
    (parameter (name ?name) (value ?value))
=>
(if (eq (nth 1 ?rest) and)
then (modify ?f (precursors (rest$ ?rest)))
else (modify ?f (precursors ?rest))))

(defrule QUERY::precursor-is-not-satisfied
?f <- (question (already-asked FALSE)
                (precursors ?name is-not ?value $?rest))
    (parameter (name ?name)(value ~?value))
=>
(if (eq (nth 1 ?rest) and)
then (modify ?f (precursors (rest$ ?rest)))
else (modify ?f (precursors ?rest))))

(defrule QUERY::Set-NonNumParam-to-Unknown
?f <- (user-response ?name unknown)
(nonnumparam (name ?name))
=>
(retract ?f)
(assert (parameter (name ?name) (value unknown))))

(defrule QUERY::convert
?f <- (user-response ?name ?x&~unknown)
(nonnumparam (name ?name)(values $?values)(menu $?the-menu)(convert yes))
=>
(retract ?f)
(assert (parameter (name ?name) (value (nth$ (member$ ?x $?the-menu) ?values)))))

(defrule QUERY::no-convert
?f <- (user-response ?name ?x)
```

```
(nonnumparam (name ?name)(values $?values)(menu $?the-menu)(convert no))
=>
(retract ?f)
(assert (parameter (name ?name) (value ?x))))

;=========================================================
=========================
;                    MAIN MODULE
;=========================================================
============================
(defmodule MAIN (export ?ALL))

(defrule MAIN::welcome
=>
(bind ?text
"          ***************************************************
           * Grinding Circuits Optimization Supervisor (GCOS) *
          * August 1998                                      *
           * McGill University                                *
           * Mining and Metallurgical Engineering Department  *
           * Mineral Processing Group                         *
           * ***************************************************

           GCOS is a knowledge-based expert system to assist
           a mineral processing engineer to optimize a ball
           milling circuit.  The system will ask a series of
           questions to reach a conclusion or a number of
           conclusions.")
(system cls)
(printout t crlf crlf crlf ?text)
(printout t crlf crlf
           "          Please press any key to continue ... ")
(system presskey))

(defrule MAIN::start
=>
(set-fact-duplication TRUE)
(focus INITIALIZATION))

;=========================================================
=========================
;                  INITIALIZATION MODULE
;=========================================================
=========================
(defmodule INITIALIZATION (import MAIN deftemplate initial-fact)
                          (import TEMPLATES ?ALL)
                          (import QUERY ?ALL))

(deffacts INITIALIZATION::initial-state
(nonnumparam (name current-module)
        (values QUERY INITIALIZATION BALLMILL HYDROCYCLONE MODSIM
```

```
CONCLUSION RESET))

(nonnumparam (name consultation-topic)
          (values BALLMILL HYDROCYCLONE CIRCUIT MODSIM)
          (menu 1 2 3 4)
            (convert yes))

(question (parameter consultation-topic)
          (module INITIALIZATION)
          (the-question
"Please choose one of the following topics?

               1 ball mill
               2 hydrocyclone
               3 circuit
               4 modelling and simulation

                  ==> ")
        (help
"   GCOS can assist you in off-line optimization of ball mills,
    classifications, and full grinding circuits. Also, it guides
    a novice engineer in modelling and simulation of a ball mill/
    hydrocyclone circuit. Please choose one of the options by
    typing corresponding menu item."))

(parameter (name current-module)(value INITIALIZATION)))

(defrule focus-on-selected-module
(parameter (name consultation-topic) (value ?ct&~unknown))
=>
(assert (parameter (name current-module)(value ?ct)))
(focus ?ct))

(defrule focus-on-query-module
=>
(focus QUERY))

(defrule go-conclusion
(parameter (name consultation-topic) (value unknown))
=>
(assert (conclusion-text (text
"   - The system cannot proceed without setting a consultation
    topic.")))
(focus CONCLUSION))

;=============================================================
;                    BALL MILL  MODULE
;=============================================================
(defmodule BALLMILL (import MAIN deftemplate initial-fact)
```

```
                (import TEMPLATES ?ALL)
                (import QUERY ?ALL))

(deffacts BALLMILL::Initial-facts

(numparam (name mill-filling)(min 0)(max 100))
(numparam (name charge-to-roof-distance)(min 0)(max 10))
(numparam (name liner-wear)(min 0))
(numparam (name Wi-l)(min 0))
(numparam (name Wi-o)(min 0))
(numparam (name mill-speed)(min 0))
(numparam (name percent-of-discharge-coarser-than-hump-particle-size)(min 0))
(numparam (name hump-selection-function)(min 0))
(numparam (name top-size-selection-function)(min 0))
(numparam (name mill-diameter)(min 0)(max 7))
(numparam (name F80)(min 0))
(numparam (name ore-specific-gravity)(min 0))
(numparam (name make-up-ball-size)(min 0))

(nonnumparam
        (name optimization-objective)
      (values increase-throughput-or-grind-fineness decrease-operating-costs unknown)
      (menu i d u)
        (convert yes))

(nonnumparam
        (name operation-mode)
      (values  wet dry)
      (menu w d)
        (convert yes))

(nonnumparam
      (name discharge-mechanism)
      (values  overflow diaphragm)
      (menu o d)
      (convert yes))

(nonnumparam
        (name balls-material)
      (values  steel silica)
      (menu s i)
        (convert yes))

(nonnumparam
        (name mill-liner-condition-checked)
        (values  yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name ball-mill-feed-can-get-coarser)
```

```
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name circuit-type)
      (values  open closed)
      (menu o c)
      (convert yes))

(nonnumparam
        (name selection-function-curve-shape)
      (values is-a-straight-line has-a-small-hump has-a-large-hump)
      (menu l sh lh)
        (convert yes))

(nonnumparam
        (name ball-size-increased-in-past)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name ball-size-decreased-in-past)
        (values yes no)
        (menu y n)
      (convert yes))


(question (parameter optimization-objective)
        (module BALLMILL)
        (the-question

"What is your optimization objective?

            I  increase throughput or grind fineness
            D  decrease operating costs
            U  unknown

                ==> ")
        (help
"   It is important to establish a clear objective for the
    optimization exercise.  A defined objective directs decision
    making regarding changes which must be made in circuit
    operation to achieve that objective.

    Please note that If there are more than one ball mill in your
    grinding circuit, then you have to consider one ball mill at a
    time.")))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;; START > optimization objective: increase-throughput-or-grind-fineness
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule BALLMILL::load-increase-throughput-or-grind-fineness-facts
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
=>
(assert

(question (parameter circuit-type)
        (module BALLMILL)
        (the-question
"What circuit type is used?

            O  open
            C  closed

                ==> ")
        (help
"    This information is used to derive other facts."))

(question (parameter discharge-mechanism)
        (module BALLMILL)
        (the-question
"What type of discharge mechanism does the mill have?

            O  overflow
            D  diaphragm

                ==> ")
        (help
"    This information is needed to find the value of wet
    grinding constant which is used in Bond's make-up ball
    size relationship."))

(question (parameter mill-diameter)
        (module BALLMILL)
        (the-question
        "Please enter the internal diameter of the mill, measured
            inside the liner (in meters)?

                ==> ")
        (help
"    The internal dimeter must be measured inside the mill liner
    and must be given in meters."))

(question (parameter make-up-ball-size)
        (module BALLMILL)
      (the-question "What make-up ball size is used (in millimeters)?

                ==> ")
      (help "   "))
```

```
(question (parameter balls-material)
        (module BALLMILL)
        (the-question
"What material are balls made of?

            S  steel
            I  silica

                ==> ")
        (help
"   This information is needed to find the value of wet
    grinding constant which is used in Bond's make-up ball
    size relationship."))

(question (parameter operation-mode)
        (module BALLMILL)
        (the-question
"What mode of grinding operation is used?

            W  wet
            D  dry

                ==> ")
        (help
"   This information is needed to find the value of wet
    grinding constant which is used in Bond's make-up ball
    size relationship."))

(question (parameter selection-function-curve-shape)
        (module BALLMILL)
        (the-question
"What shape does the selection function vs. particle size
                curve have?

            L  linear
            SH  straight line with a small hump at coarse end
            LH  large hump

                ==> ")
        (help
"   Please look at the shape of selection function vs. particle
    size curve and then choose one.")))
(focus QUERY))

(defrule BALLMILL::compute-critical-speed
(parameter (name mill-diameter)(value ?dia))
(test (numberp ?dia))
=>
(bind ?cs (/ 42.3 (sqrt ?dia)))
(assert (parameter (name critical-speed) (value ?cs))))
```

```
(defrule BALLMILL::compute-percent-of-CS
(parameter (name critical-speed)(value ?cs))
(parameter (name mill-speed)(value ?ms))
(test (numberp ?ms))
=>
(bind ?ratio  (* (/ ?ms ?cs) 100))
(assert (parameter (name percent-of-critical-speed)(value ?ratio))))

(defrule BALLMILL::check-mill-speed
(parameter (name percent-of-critical-speed)(value ?pcs))
=>
(if (>= ?pcs 82) then
(assert (conclusion-text (text
"    - The mill speed is apparently too high (above 82% of
      critical speed)."))))
(if (<= ?pcs 65) then
(assert (conclusion-text (text
"    - The mill speed is apparently too low (below 65% of
      critical speed)."))))))

(defrule BALLMILL::assume-steel-balls
(parameter (name balls-material)(value unknown))
=>
(assert (parameter (name balls-material)(value steel))))

(defrule BALLMILL::assume-wet-operation
(parameter (name operation-mode)(value unknown))
=>
(assert (parameter (name operation-mode)(value wet))))

(defrule BALLMILL::assume-overflow-discharge
(parameter (name discharge-mechanism)(value unknown))
=>
(assert (parameter (name discharge-mechanism)(value overflow))))

(defrule BALLMILL::Bond-ball-size-not-exact-a
(parameter (name balls-material)(value unknown))
(parameter (name Bond-make-up-ball-size)(value ?x))
=>
(assert (conclusion-text (text
"    - The ball size calculated using Bond's equation was based
      on the assumption that steel balls are used as grinding
      medium."))))

(defrule BALLMILL::Bond-ball-size-not-exact-b
(parameter (name operation-mode)(value unknown))
(parameter (name Bond-make-up-ball-size)(value ?x))
=>
(assert (conclusion-text (text
"    - The ball size calculated using Bond's equation was based
      on the assumption that the operation mode is wet."))))
```

```
(defrule BALLMILL::Bond-ball-size-not-exact-c
(parameter (name discharge-mechanism)(value unknown))
(parameter (name Bond-make-up-ball-size)(value ?x))
=>
(assert (conclusion-text (text
"    - The ball size calculated using Bond's equation was based
       on the assumption that an overflow discharge mechanism
       is used.""")))

(defrule BALLMILL::K1-steel-wet-overflow
(parameter (name balls-material)(value steel))
(parameter (name operation-mode)(value wet))
(parameter (name discharge-mechanism)(value overflow))
=>
(assert (parameter (name Bond-K)(value 350))))

(defrule BALLMILL::K2-steel-wet-diaphragm
(parameter (name balls-material)(value steel))
(parameter (name operation-mode)(value wet))
(parameter (name discharge-mechanism)(value diaphragm))
=>
(assert (parameter (name Bond-K)(value 330))))

(defrule BALLMILL::K3-steel-dry-diaphragm
(parameter (name balls-material)(value steel))
(parameter (name operation-mode)(value dry))
(parameter (name discharge-mechanism)(value diaphragm))
=>
(assert (parameter (name Bond-K)(value 335))))

(defrule BALLMILL::K4-silica-wet-diaphragm
(parameter (name balls-material)(value silica))
(parameter (name operation-mode)(value wet))
(parameter (name discharge-mechanism)(value diaphragm))
=>
(assert (parameter (name Bond-K)(value 170))))

(defrule BALLMILL::K5-silica-dry-diaphragm
(parameter (name balls-material)(value silica))
(parameter (name operation-mode)(value dry))
(parameter (name discharge-mechanism)(value diaphragm))
=>
(assert (parameter (name Bond-K)(value 175))))

(defrule BALLMILL::assert-mill-speed-question
(parameter (name mill-diameter)(value ~unknown))
=>
(assert

(question (parameter mill-speed)
        (module BALLMILL)
```

```
      (the-question "At what speed is the mill running (in rpm)?

                    ==> ")
      (help
"    Please give the mill speed in rotations per minute, rpm.
    The mill speed is needed to compute the percent of critical
    speed.")))
(focus QUERY))

(defrule BALLMILL::query-Wi-l
(parameter (name mill-speed)(value ~unknown))
=>
(assert
(question (parameter Wi-l)
        (module BALLMILL)
        (the-question "What is the value of laboratory Bond Work index
            (in kWh/t)?

                    ==> ")
      (help
"    Please enter the value of work index (Wi) determined in the
    laboratory.  Proper diagnostic is greatly assisted by comparing
    operating and laboratory work indices.")))
(focus QUERY))

(defrule BALLMILL::query-F80
(parameter (name Wi-l)(value ~unknown))
=>
(assert
(question (parameter F80)
        (module BALLMILL)
      (the-question
"Please enter d80 of the ball mill feed (in microns)?

                    ==> ")
        (help
"    The feed size indicated by d80 is needed to calculate
    ball size.")))
(focus QUERY))

(defrule BALLMILL::assert-ore-specific-gravity-question
(parameter (name F80)(value ~unknown))
=>
(assert
(question (parameter ore-specific-gravity)
        (module BALLMILL)
        (the-question
"What specific gravity does ore have
            (in g per cubic cm)?

                    ==> ")
```

```
      (help
"    The specific gravity of ore is needed to calculate
     ball size.")))
(focus QUERY))

(defrule BALLMILL::predict-Bond-ball-size
(parameter (name mill-diameter)(value ?Dm))
(parameter (name percent-of-critical-speed)(value ?pcs))
(parameter (name Wi-l)(value ?Wi))
(parameter (name F80)(value ?F80))
(parameter (name ore-specific-gravity)(value ?Sg))
(parameter (name Bond-K) (value ?K))
(test (and (numberp ?F80)(numberp ?Sg)(numberp ?K)(numberp ?Wi)(numberp ?Dm)))
=>
(bind ?bs (* 25.4 (sqrt (/ ?F80 ?K))(** (/ (* ?Sg ?Wi) (* ?pcs (sqrt (* 3.281 ?Dm)))) 0.33)))
(assert (parameter (name Bond-make-up-ball-size) (value ?bs))))

(defrule BALLMILL::predict-Azzaroni-ball-size
(parameter (name mill-diameter)(value ?Dm))
(parameter (name mill-speed)(value ?N))
(parameter (name Wi-l)(value ?Wi))
(parameter (name F80)(value ?F80))
(test (and (numberp ?F80)(numberp ?Wi)(numberp ?Dm)(numberp ?N)))
=>
(bind ?bs (* 6.3  (/ (* (** ?F80 0.29)(**  ?Wi 0.4) ) (** (* ?N  ?Dm) 0.25))))
(assert (parameter (name Azzaroni-make-up-ball-size) (value ?bs))))

(defrule BALLMILL::recommend-makeup-ball-size
(parameter (name Azzaroni-make-up-ball-size)(value ?abs))
(parameter (name Bond-make-up-ball-size)(value ?bbs))
=>
(assert (conclusion-string (string
"%n
   - The recommended make-up or top ball size for single size
     recharge is:

      %7.2f mm     (based on Azzaroni's relationship)
      %7.2f mm     (based on Bond's relationship)")
                (arg1  ?abs)
                (arg2  ?bbs))))

(defrule BALLMILL::assert-ball-mill-feed-question
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name selection-function-curve-shape)(value is-a-straight-line))
=>
(assert (question (parameter ball-mill-feed-can-get-coarser)
      (module BALLMILL)
        (the-question
"Can the ball mill feed get much coarser? (y/n)

                ==> ")
```

```
        (help
"    If there is a possibility that the mill feed becomes
    coarser, decreasing the current ball size may cause
    problem.")))
(focus QUERY))

(defrule BALLMILL::ball-size-too-large
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name selection-function-curve-shape)(value is-a-straight-line))
(parameter (name ball-mill-feed-can-get-coarser)(value no))
=>
(assert (parameter (name ball-size-largeness) (value too-large))
       (question (parameter ball-size-increased-in-past)
         (module BALLMILL)
         (the-question
"Was ball size increased in the past to
                improve throughput or grind? (y/n)

                ==> ")
         (help
"    If ball size had been increased before, decreasing it again may
    cause unnecessary oscillations in make-up ball size.")))
(focus QUERY))

(defrule BALLMILL::assert-selection-function-questions
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name selection-function-curve-shape)(value has-a-large-hump))
=>
(assert
       (question (parameter percent-of-discharge-coarser-than-hump-particle-size)
         (module BALLMILL)
         (the-question
"What percent of the ball mill discharge has a size
            larger than that of hump?

                ==> ")
         (help
"    When there is a large hump in the selection function vs. particle
    size curve, then a decision to use bigger balls also depends on whether
    or not there is a large portion of material with a particle size
    coarser than that of corresponding to maximum selection function."))

       (question (parameter hump-selection-function)
         (module BALLMILL)
       (precursors  percent-of-discharge-coarser-than-hump-particle-size is-not unknown)
         (the-question
"Enter the maximum selection function value?

                ==> ")
         (help
"    This information is needed to charectrize the hump
```

```
    largeness."))

      (question (parameter top-size-selection-function)
        (module BALLMILL)
      (precursors  hump-selection-function is-not unknown)
        (the-question
"Enter the top size class selection function?

                    ==> ")
        (help
"    This is information is needed to characterize the
    hump largeness. ")))
(focus QUERY))

(defrule BALLMILL::ball-size-too-small
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name selection-function-curve-shape)(value has-a-large-hump))
(parameter (name percent-of-discharge-coarser-than-hump-particle-size)(value ?per-coarse))
(parameter (name hump-selection-function)(value ?hsf))
(parameter (name top-size-selection-function)(value ?tssf))
(test (and (numberp ?hsf)(numberp ?tssf)(numberp ?per-coarse)))
(test (>= ?per-coarse 20))
(test (< ?tssf (* 0.2 ?hsf)))
=>
(assert (parameter (name ball-size-largeness) (value too-small))
(question (parameter ball-size-decreased-in-past)
        (module BALLMILL)
        (the-question
"Was ball size decreased in the past to improve
                throughput or grind? (y/n)

                ==> ")
        (help
"    If ball size had been decreased in the past before, increasing
    it again may cause unnecessary oscillations in make-up ball size.")))
(focus QUERY))

(defrule BALLMILL::ball-size-near-optimum
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name selection-function-curve-shape)(value has-a-small-hump))
=>
(assert (parameter (name ball-size) (value near-optimum))))

(defrule BALLMILL::ball-size-no-need-to-change
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name ball-size)(value near-optimum))
=>
(assert (conclusion-text (text
"    - There is no serious need to change the make-up ball size.
      However, the impact of any change to the make-up ball size
      can only be indicated by simulation."))))
```

```
(defrule BALLMILL::decrease-ball-size-13-millimeters
(parameter (name ball-size-largeness)(value too-large))
(parameter (name make-up-ball-size)(value ?mbs))
(parameter (name ball-size-increased-in-past)(value no))
(test (numberp ?mbs))
(test (>= ?mbs 38))
=>
(assert (conclusion-text (text
"   - Decrease make-up or top ball size by 13 mm (0.5 inch). This
      can be achieved using a blend of make-up balls. Test the
      effect of this change by NGOTC before real plant exercise."))))

(defrule BALLMILL::decrease-ball-size-6-millimeters
(parameter (name ball-size-largeness)(value too-large))
(parameter (name make-up-ball-size)(value ?mbs))
(parameter (name ball-size-increased-in-past)(value no))
(test (numberp ?mbs))
(test (< ?mbs 38))
=>
(assert (conclusion-text (text
"   - Decrease make-up or top ball size by 6 mm (0.25 inch). Test
      the effect of this change by NGOTC before real plant exercise."))))

(defrule BALLMILL::increase-ball-size-13-millimeters
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name ball-size-largeness)(value too-small))
(parameter (name make-up-ball-size)(value ?mbs))
(parameter (name ball-size-decreased-in-past)(value no))
(test (numberp ?mbs))
(test (< ?mbs 51))
=>
(assert (conclusion-text (text
"   - Increase make-up or top ball size by 13 mm (0.5 inch). Test
      the effect of this change by NGOTC before real plant exercise."))))

(defrule BALLMILL::increase-ball-size-25-millimeters
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name ball-size-largeness)(value too-small))
(parameter (name make-up-ball-size)(value ?mbs))
(parameter (name ball-size-decreased-in-past)(value no))
(test (numberp ?mbs))
(test (>= ?mbs 51))
=>
(assert (conclusion-text (text
"   - Increase make-up or top ball size by 25 mm (1 inch). Test
      the effect of this change with NGOTC before real plant exercise."))))

(defrule BALLMILL::detailed-study-ball-size-is-too-small
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name ball-size-largeness)(value too-small))
```

```
(parameter (name ball-size-decreased-in-past)(value yes))
=>
(assert (conclusion-text (text
"   - It seems that the make-up ball size is too small.
      Since ball size was recently decreased, it is likely
      that the optimum ball size is between the previous
      and existing one."))))

(defrule BALLMILL::detailed-study-ball-size-is-too-large
(parameter (name optimization-objective)(value increase-throughput-or-grind-fineness))
(parameter (name ball-size-largeness)(value too-large))
(parameter (name ball-size-increased-in-past)(value yes))
=>
(assert (conclusion-text (text
"   - It seems that the make-up ball size is too large.
      However, since there has been an attempt to increase
      the ball size before, decreasing it again is not
      likely to improve grinding efficiency, unless feed
      to the mill has become finer or softer."))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; END > optimization objective: increase-throughput-or-grind-fineness
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; START > optimization objective: decrease-operating-costs
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule BALLMILL::load-decrease-operating-cost-facts
(parameter (name optimization-objective) (value decrease-operating-costs))
=>
(assert

(question (parameter mill-filling)
          (module BALLMILL)
          (the-question
"What percentage of the mill volume is filled with
                    the total charge?

                    ==> ")
          (help
"   The mill filling has an optimum ragne."))

(question (parameter mill-diameter)
          (module BALLMILL)
          (the-question
"What is the mill diameter, inside liners (in meters)?

                    ==> ")
          (help
"   This is required to calculate mill filling.")
(precursors mill-filling is unknown))
```

```
(question (parameter charge-to-roof-distance)
        (module BALLMILL)
        (the-question
"What is the distance between the charge surface
            and mill roof, inside liners (in meters)?

                ==> ")
        (help
"   This is required to calculate mill filling.")
(precursors mill-filling is unknown and mill-diameter is-not unknown))

(question (parameter discharge-mechanism)
        (module BALLMILL)
        (the-question
"What type of discharge mechanism does the mill have?

            O overflow
            D diaphragm

                ==> ")
        (help
"   This information is used to derive other facts."))

(question (parameter operation-mode)
        (module BALLMILL)
        (the-question
"What mode of grinding operation is used?

            W wet
            D dry

                ==> ")
        (help
"   This information is used to derive other facts."))

(question (parameter liner-wear)
        (module BALLMILL)
        (the-question "What is the average value of liner wear rate
            (in kg/kWh)?

                ==> ")
        (help
"   The abnormal liner wear rate indicates inefficient grinding
    conditions."))

(question (parameter Wi-l)
        (module BALLMILL)
        (the-question "What is the value of laboratory Bond Work index
            (in kWh/t)?

                ==> ")
```

```
        (help
"    Please enter the value of Wi determined in the laboratory."))

(question (parameter Wi-o)
        (module BALLMILL)
        (the-question "What is the value of operating Work index
            (in kWh/t)?

                    ==> ")
        (help
"    Please enter the value of Wi measured in the plant."))

(question (parameter mill-liner-condition-checked)
        (module BALLMILL)
        (the-question
"Have you checked the mill liner condition? (y/n)

                    ==> ")
        (help
"    The liner condition can affect the transfer of energy from shell
    to balls which then break large particles into smaller ones.
    Therefore, this parameter will affect grinding rate and energy
    consumption in ball mill grinding and must be optimized.")))

(focus QUERY))

(defrule BALLMILL::check-liner
(parameter (name optimization-objective) (value decrease-operating-costs))
(parameter (name mill-liner-condition-checked)(value no))
=>
(assert (conclusion-text (text "    - Check the liner condition."))))

(defrule BALLMILL::wet-liner-wear
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name operation-mode)(value wet))
(parameter (name liner-wear)(value ?lv))
(test (numberp ?lv))
=>
(if (>= ?lv  0.044) then
(assert (conclusion-text (text
"    - It seems that liner wear is too high for this operation.
      This needs to be checked, any way."))))

(defrule BALLMILL::dry-liner-wear
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name operation-mode)(value dry))
(parameter (name liner-wear)(value ?lv))
(test (numberp ?lv))
=>
(if (>= ?lv 0.006) then
(assert (conclusion-text (text
```

```
"    - It seems that liner wear is too high for this operation.
       This needs to be checked, any way.")))))

(defrule BALLMILL::energy-consumption
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name Wi-o)(value ?Wi-o))
(parameter (name Wi-l)(value ?Wi-l))
(test (and (numberp ?Wi-o) (numberp ?Wi-l)))
=>
(bind ?EPI (/ ?Wi-o ?Wi-l))
(if (and (> ?EPI 0.8)(<= ?EPI 1.05)) then
(assert (parameter (name energy-consumption) (value good))))
(if (and (> ?EPI 1.05)(<= ?EPI 1.2)) then
(assert (parameter (name energy-consumption)(value ok))))
(if (> ?EPI 1.2) then
(assert (parameter (name energy-consumption)(value bad)))))

(defrule BALLMILL::assert-energy-consumption-good
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name energy-consumption)(value good))
=>
(assert (conclusion-text (text
"    - The grinding performance, in terms of energy consumption,
       seems very good.")))

(defrule BALLMILL::asset-energy-consumption-bad
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name energy-consumption)(value bad))
=>
(assert (conclusion-text (text
"    - The grinding performance, in terms of energy consumption,
       seems not good. You may need to optimize operation in this
       respect. Significant improvements are possible in this
       situation.")))

(defrule BALLMILL::assert-energy-consumption-Ok
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name energy-consumption)(value ok))
=>
(assert (conclusion-text (text
"    - The grinding performance, in terms of energy consumption,
       seems to be okay. Modest improvements are possible in
       this situation by process optimization efforts.")))

(defrule BALLMILL::determine-laboratory-WI
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name Wi-l)(value unknown|u))
=>
(assert (conclusion-text (text "    - The laboratory Wi must be determined."))))

(defrule BALLMILL::determine-operating-WI
```

```
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name Wi-o)(value unknown|u))
=>
(assert (conclusion-text (text "    - The operating WI must be determined."))))

(defrule BALLMILL::compute-mill-filling
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name mill-diameter)(value ?D))
(parameter (name charge-to-roof-distance)(value ?H))
(test (and (numberp ?D) (numberp ?H)))
=>
(assert (parameter (name mill-filling) (value (- 113  (* (/ ?H ?D) 126))))))

(defrule BALLMILL::over-charge
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name discharge-mechanism)(value overflow))
(parameter (name mill-filling)(value ?v))
(test (numberp ?v))
(test (>= ?v 45))
=>
(assert (conclusion-text (text
"    - Mill filling seems to be high and must be checked.  You might
      be loosing balls at discharge due to the high mill filling."))))

(defrule BALLMILL::under-charge
(parameter (name optimization-objective)(value decrease-operating-costs))
(parameter (name discharge-mechanism)(value overflow))
(parameter (name mill-filling)(value ?v))
(test (numberp ?v))
(test (<= ?v 30))
=>
(assert (conclusion-text (text
"    - Mill filling seems to be low and must be checked."))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; END > optimization objective: decrease-operating-costs
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; START > optimization objective: unknown
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule BALLMILL::optimization-objective-unknown
(parameter (name optimization-objective)(value unknown))
=>
(assert (conclusion-text (text
"    - Having clear plant optimization objectives is important
       since it affects decisions about plant operating changes
      and guides optimization process."))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; END > optimization objective: unknown
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule go-query
=>
(focus QUERY))

(defrule go-conclusion
=>
(focus CONCLUSION))

;=================================================================
=========================
;                    HYDROCYCLONE MODULE
;=================================================================
=========================
(defmodule HYDROCYCLONE (import MAIN deftemplate initial-fact)
                        (import TEMPLATES ?ALL)
                        (import QUERY ?ALL))

(deffacts HYDROCYCLONE::initial-facts

(nonnumparam
        (name classification-objective)
      (values  increase-cut-size decrease-cut-size reduce-water-recovery
            increase-separation-sharpness unknown)
      (menu i d r s u)
        (convert yes))

(nonnumparam
        (name  Plitt-model-fit-done)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name  Plitt-model-fit-optimized)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name  positive-Rf)
      (values yes no)
      (menu y n)
        (convert yes))

(nonnumparam
      (name  satisfactory-optimal-fit)
      (values yes no)
      (menu y n)
        (convert yes))
```

```
(nonnumparam
      (name classification-data-balanced)
      (values yes no)
      (menu y n)
        (convert yes))

(nonnumparam
      (name hump-or-plateau-exists)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name hump-or-plateau-exists-in-double-check)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name heavy-light-phases)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name heavy-light-phases-in-double-check)
        (values yes no)
        (menu y n)
        (convert yes))


(nonnumparam
      (name fish-hook-exists)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name classification-data-available)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name classification-satge)
      (values primary secondary)
        (menu p s)
        (convert yes))

(nonnumparam
      (name pressure-drop)
```

```
      (values too-low normal too-high)
      (menu l n h)
        (convert yes))

(numparam (name  number-of-installed-cyclones) (min 1))
(numparam (name  number-of-operating-cyclones) (min 1))
(numparam (name Rf)(min 0)(max 100))
(numparam (name m)(min 0))

(question (parameter classification-objective)
        (module HYDROCYCLONE)
        (the-question
"What is your classification objective?

              I  increase cut size, d50
              D  decrease d50
              R  reduce water recovery to the cyclone underflow, Rf
              S  increase separation sharpness, m
              U  unknown

                  ==> ")
        (help
"   A pre-defined objective directs decision making regarding
    changes which probably can be made in circuit operation to
    achieve that objective.")))

(defrule increase-cut-size
(parameter (name classification-objective)(value increase-cut-size))
=>
(assert (conclusion-text (text
"    - The cut size can be increased by reducing the apex diameter
        of the cyclone(s).  However, it is recommended to use the
      BMCS to assess the impact of using a smaller apex on the
        classification and full circuit performance."))

        (conclusion-text (text
"    - The cut size can be increased by installing larger cyclones.
      This option, however, is only practical at the design stage.
      For an existing circuit, the BMCS can be used to assess the
      impact of using larger cyclones on the classification and
      full circuit performance."))

        (conclusion-text (text
"    - The cut size can be increased by increasing the inclination of
        cyclones to the vertical to 45 degrees or more.  This option
      is normally practical only at the design stage, and when the
      number of cyclones is small."))

        (conclusion-text (text
"    - Increasing the vortex finder diameter can increase the cut size.
      The BMCS program can be used to assess the impact of using a
```

```
      larger vortex finder diameter on the circuit performance."))

        (conclusion-text (text
"    - Decreasing water addition rate can increase cut size. This is very
      good if too much water goes to downstream."))))

(defrule load-decrease-cut-size-facts
(parameter (name classification-objective)(value decrease-cut-size))
=>
(assert

(question (parameter number-of-operating-cyclones)
          (module HYDROCYCLONE)
          (the-question
"How many cyclones are being operated?

                  ==> ")
        (help
"    The number of operating hydrocyclones can be changed to
    optimize classification efficiency.")))
(focus QUERY))

(defrule decrease-cut-size-by-fewer-cyclones
(parameter (name classification-objective)(value decrease-cut-size))
(parameter (name number-of-operating-cyclones)(value ?noc))
(test (numberp ?noc))
(test (> ?noc 1))
=>
(assert (conclusion-text (text
"    - The cut size can be reduced by switching off a cyclone at
        constant total feed flow rate. It is recommended to use the
      BMCS to assess the impact of this change on the full circuit
        performance."))))

(defrule decrease-cut-size-by-diluted-feed
(parameter (name classification-objective)(value decrease-cut-size))
=>
(assert (conclusion-text (text
"    - The cut size can be reduced by diluting the feed slurry. It
      is recommended to use the BMCS to assess the impact of this
        change on the full circuit performance."))))

(defrule load-reduce-Rf
(parameter (name classification-objective)(value reduce-Rf))
=>
(assert

(question (parameter pressure-drop)
          (module HYDROCYCLONE)
          (the-question
"Is the pressure drop level?
```

```
            L  too low
            N  normal
            H  too high

                ==> ")
        (help
"    Pressure drop is one of the important parameters to decide
    whether or not the number of operating hydrocyclones can be
    changed.")))

(focus QUERY))

(defrule reduce-water-recovery-by-fewer-cyclones
(parameter (name classification-objective)(value reduce-water-recovery))
(parameter (name number-of-operating-cyclones)(value ?noc))
(parameter (name pressure-drop)(value  too-low))
(test (numberp ?noc))
(test (> ?noc 3))
=>
(assert (conclusion-text (text
"    - Water recovery to the cyclone underflow, Rf, can be reduced by
        switching off a cyclone at constant total feed flow rate.  It
      is recommended to use the BMCS to assess the impact of this change
      on full circuit performance.")))

(defrule reduce-water-recovery-by-smaller-apex
(parameter (name classification-objective)(value reduce-water-recovery))
=>
(assert (conclusion-text (text
"    - Water recovery to the cyclone underflow, Rf, can be reduced by
        using smaller apex diameter.  It is recommended to use the BMCS to
      assess the impact of this change on full circuit performance.")))

(defrule reduce-water-recovery-by-larger-vortex-finder
(parameter (name classification-objective)(value reduce-water-recovery))
=>
(assert (conclusion-text (text
"    - Water recovery to the cyclone underflow, Rf, can be reduced by
        using larger vortex finder diameter.  It is recommended to use the
      BMCS to assess the impact of this change on full circuit
        performance.")))

(defrule  increase-separation-sharpness-by-reducing-Rf
(parameter (name classification-objective)(value increase-separation-sharpness))
=>
(assert (conclusion-text (text
"    - The Plitt's separation sharpness can be increased by modifications
      that decrease water recovery to the cyclone underflow or short
      circuiting.  The separation sharpness can be improved by adding
      the number of operating hydrocyclones or increasing pressure drop if
      it is too low.")))
```

```
(defrule  increase-separation-sharpness-by-diluting-feed
(parameter (name classification-objective)(value increase-separation-sharpness))
=>
(assert (conclusion-text (text
"    - In case of excessively high feed solids concentration or high
        slimes concentrations, it is recommended to dilute the feed to
        reduce the viscosity of the fluid.  This can be led to improved
        separation sharpness."))))

(defrule classification-objective-unknown
(parameter (name classification-objective)(value unknown))
=>
(assert

(question (parameter Plitt-model-fit-done)
          (module HYDROCYCLONE)
          (the-question
"Have you fitted Plitt's model to the classification
                    data? (y/n)

                    ==> ")
        (help
"    By fitting Plitt's model to the measured data, classification
    performance indices such as Rf, d50c and m can be determined."))

(question (parameter Plitt-model-fit-optimized)
          (module HYDROCYCLONE)
          (precursors Plitt-model-fit-done is yes)
          (the-question
"Have you optimized the fit? (y/n)

                    ==> ")
        (help
"    Plitt's model must be optimally fit to the measured data, so
    that the lack of fit is minimized and the model parameters are
    the best estimates.  The fit optimization can be done using the
    optimization tool available in commercial software."))

(question (parameter positive-Rf)
          (module HYDROCYCLONE)
        (precursors Plitt-model-fit-optimized is yes)
          (the-question
"Have you obtained a positive Rf? (y/n)

                    ==> ")
        (help
"    To be meaningful, the three parameters of Plitt's model,
    i.e. d50c, m and Rf must be positive.  The first two
    parameters will be normally positive after optimizing
    the fit."))
```

```
(question (parameter satisfactory-optimal-fit)
          (module HYDROCYCLONE)
        (precursors positive-Rf is yes)
          (the-question
"Is the optimized fit satisfactory? (y/n/u)

                  ==> ")
        (help
"   Plitt's model may not satisfactorily fit the measured data
    even when the fit optimized."))

(question (parameter classification-data-balanced)
          (module HYDROCYCLONE)
        (precursors heavy-light-phases is no)
          (the-question
"Did you mass balance the raw classification data? (y/n)

                  ==> ")
        (help
"   It is almost a standard procedure to mass balance
    data before using them for process analysis."))

(question (parameter classification-data-balanced)
          (module HYDROCYCLONE)
        (precursors fish-hook-exists is no)
          (the-question
"Did you mass balance the raw classification data? (y/n)

                  ==> ")
        (help
"   It is almost a standard procedure to mass balance
    data before using them for process analysis."))

(question (parameter hump-or-plateau-exists)
          (module HYDROCYCLONE)
        (precursors satisfactory-optimal-fit is no)
          (the-question
"Does the partition curve have a hump or plateau in
            intermediate size range? (y/n)

                  ==> ")
        (help
"   This unusual partition curves can be observed due to
    the individual mineral classification behaviour."))

(question (parameter heavy-light-phases)
          (module HYDROCYCLONE)
        (precursors hump-or-plateau-exists is yes)
          (the-question
"Does the cyclone feed contain significant
```

```
            heavy and light mineral phases? (y/n)

                    ==> ")
        (help
"   The presence of these phases causes partition curves with
    unusual shapes which cannot be fitted by Plitt's model.
    Plitt's model assumes single-mineral ores."))

(question (parameter fish-hook-exists)
          (module HYDROCYCLONE)
        (precursors hump-or-plateau-exists is no)
          (the-question
"Is there a fish hook at fine end of the
             partition curve? (y/n)

                    ==> ")
        (help
"   The fish hook problem is observed in some partition curves
    which cannot be fitted by Plitt's model."))

(question (parameter classification-data-available)
          (module HYDROCYCLONE)
          (precursors Plitt-model-fit-done is no)
          (the-question
"Is classification data available? (y/n)

                    ==> ")
        (help
"   At least one set of classification data is necessary to
    characterize the cyclone performance. The data set must
    include flow rates, % solids and particle size distribution
    of cyclone underflow and overflow streams."))

(question (parameter classification-stage)
          (module HYDROCYCLONE)
          (the-question
"What classification stage is this?

             P  primary
             S  secondary

                    ==> ")
        (help
"   In some plants a two-stage classification arrangement is
    used. The feed to a secondary cyclone has a narrower
    size distribution than that of a primary cyclone; hence,
    the density of individual minerals has a pronounced effect
    on the classification."))

(question (parameter Rf)
      (module HYDROCYCLONE)
```

```
      (precursors satisfactory-optimal-fit is yes)
      (the-question
"What is the estimated value of the water recovery to
          the cyclone underflow (in %)?

                ==> ")
      (help
"   The water recovery to cyclone underflow is an
    indicatior of short-circuiting or by-pass of solids."))

(question (parameter m)
      (module HYDROCYCLONE)
      (precursors satisfactory-optimal-fit is yes)
      (the-question
"What is the estimated value of the separation sharpness?

                ==> ")
      (help
"   The separation sharpness is an important indicator of
    cyclone efficiency.")))
(focus QUERY))

(defrule fit-Plitt-model
(parameter (name classification-data-available)(value yes))
=>
(assert (conclusion-text (text
"   - Use a speadsheet software such as QuattroPro or Excel
      to fit Plitt's model to data."))))

(defrule do-sampling
(parameter (name classification-data-available)(value no))
=>
(assert (conclusion-text (text
"   - Do a circuit survey around the cyclone(s)."))))

(defrule optimize-Plitt-model-fit
(parameter (name Plitt-model-fit-optimized)(value no))
=>
(assert (conclusion-text (text
"   - The fit must be optimized using a non-linear optimization tool.
      Spreadsheet softwares normally include this function."))))

(defrule negative-Rf-parameter
(parameter (name positive-Rf)(value no))
=>
(assert (conclusion-text (text
"   - When the fit is optimized, the final values of estimated
      parameters, Rf, d50c and m must be positive. If not, this
      can be due to incomplete size distribution information
      of cyclone streams for fine size classes. To solve this
      problem, Rf can be calculated from cyclone overflow
```

```
      and underflow solids flow rate and percent solids information.
      Then, the other two parameters can be estimated using the
      optimization tool.")))

(defrule negative-Rf-parameter-2
(parameter (name positive-Rf)(value no))
=>
(assert (conclusion-text (text
"   - It is recommended to use a wider screening size range in
      next circuit survey so that Rf can be estimated when model
      fitting is optimized.")))

(defrule satisfactory-optimal-fit
(parameter (name satisfactory-optimal-fit)(value unknown))
=>
(assert (conclusion-text (text
"   - To check if the optimal fit is satisfactory or not, you
      can examine the goodness of fit (or the lack of fit)
      criterion and also visually evaluate how close is the fitted
      curve to the measured data. Also, the water recovery to cyclone
      underflow calculated from circulating load, cyclone underflow
      and overflow flow rates and % solids should be close to the
      water recovery to cyclone underflow fitted.")))

(defrule balance-classification-data
(parameter (name classification-data-balanced)(value no))
=>
(assert (conclusion-text (text
"   - It is recommended to use balanced data for classification
      data analysis. A better goodness of fit may be obtained using
      balanced data.")))

(defrule individual-minerals-behaviour
(parameter (name heavy-light-phases)(value yes))
=>
(assert (conclusion-text (text
"   - Individual mineral classification behaviour is
      the possible cause of the lack of fit. The Plitt
      hydrocyclone model can only be used for trending.")))

(defrule repeat-sampling
(parameter (name classification-data-balanced)(value yes))
=>
(assert (conclusion-text (text
"   - It is recommended to check the validity of data used for
      the analysis. It might be necessary to redo sampling
      tests to obtain reliable classification data.")))

(defrule try-fish-hook-model
(parameter (name fish-hook-exists)(value yes))
=>
```

```
(assert (conclusion-text (text
"    - It is recommended to use a fish hook model such as one
     proposed by Finch [1983] to fit this classification data
     set."))))

(defrule Rf-very-poor
(parameter (name Rf)(value ?Rf))
(test (numberp ?Rf))
(test (> ?Rf 50))
=>
(assert (conclusion-text (text
"    - The efficiency of the cyclone operation in terms of
     the amount of water recovered to the cyclone underflow
     is very poor. It is recommended to significantly reduce
     water recovery to the cyclone underflow."))))

(defrule Rf-poor
(parameter (name Rf)(value ?Rf))
(test (numberp ?Rf))
(test (and (> ?Rf 40) (<= ?Rf 50)))
=>
(assert (conclusion-text (text
"    - The efficiency of the cyclone operation in terms of
     the amount of water recovered to the cyclone underflow
     is poor. It is recommended to reduce water recovery
     to the cyclone underflow."))))

(defrule Rf-reasonable
(parameter (name Rf)(value ?Rf))
(test (numberp ?Rf))
(test (and (> ?Rf 30) (<= ?Rf 40)))
=>
(assert (conclusion-text (text
"    - The efficiency of the cyclone operation in terms of
     the amount of water recovered to the cyclone underflow
     is reasonable."))))

(defrule Rf-good
(parameter (name Rf)(value ?Rf))
(test (numberp ?Rf))
(test (and (> ?Rf 20) (<= ?Rf 30)))
=>
(assert (conclusion-text (text
"    - The efficiency of the cyclone operation in terms of
     the amount of water recovered to the cyclone underflow
     is good."))))

(defrule Rf-near-to-rope
(parameter (name Rf)(value ?Rf))
(test (numberp ?Rf))
(test (and (>= ?Rf 10) (<= ?Rf 20)))
```

```
=>
(assert (conclusion-text (text
"    - The amount of water recovered to the cyclone underflow
      is too low. The cyclone operation may be subjected to
      underflow roping. This can be checked visually."))))

(defrule Rf-abnormal
(parameter (name Rf)(value ?Rf))
(test (numberp ?Rf))
(test (< ?Rf 10))
=>
(assert (conclusion-text (text
"    - The amount of water recovered to the cyclone underflow
      is extremely low. This is very unusual with normal
      cyclone operations, and is normally achievable only with
      an underflow valve for producing a product for conveying or
      stockpiling. We recommend that you check the reliability of
      the data"))))

(defrule m-excellent
(parameter (name m)(value ?m))
(test (numberp ?m))
(test (>= ?m 3))
=>
(assert (parameter (name separation-sharpness)(value excellent))))

(defrule ss-excellent
(parameter (name separation-sharpness)(value excellent))
=>
(assert (conclusion-text (text
"    - The cyclone separation sharpness is excellent."))))

(defrule m-poor
(parameter (name m)(value ?m))
(test (numberp ?m))
(test (<= ?m 2))
=>
(assert (parameter (name separation-sharpness)(value poor))))

(defrule separation-sharpness-poor
(parameter (name separation-sharpness)(value poor))
=>
(assert (conclusion-text (text
"    - The cyclone separation sharpness is poor."))

(question (parameter hump-or-plateau-exists-in-double-check)
          (module HYDROCYCLONE)
          (the-question
"Does the partition curve have a hump or plateau in
             intermediate size range? (y/n)
```

```
                ==> ")
        (help
"   This unusual partition curves can be observed due to
    the individual mineral classification behaviour."))

(question (parameter heavy-light-phases-in-double-check)
          (module HYDROCYCLONE)
        (precursors hump-or-plateau-exists-in-double-check is yes)
          (the-question
"Does the cyclone feed contain significant heavy
            and light mineral phases? (y/n)

                ==> ")
        (help
"   The presence of these phases causes partition curves with
    a separation sharpness of ore lower than that of individual
    minerals. ")))

(focus QUERY))

(defrule separation-sharpness-due-to-poor-heavy-light-phases
(parameter (name separation-sharpness)(value poor))
(parameter (name hump-or-plateau-exists-in-double-check)(value yes))
(parameter (name heavy-light-phases-in-double-check)(value yes))
=>
(assert (conclusion-text (text
"   - The cyclone separation sharpness is poor because of
      heavy and light phases."))))

(defrule m-normal
(parameter (name m)(value ?m))
(test (numberp ?m))
(test (and (< ?m 3)(> ?m 2)))
=>
(assert (conclusion-text (text
"   - The cyclone separation sharpness is normal."))))

(defrule go-query
=>
(focus QUERY))

(defrule go-conclusion
=>
(focus CONCLUSION))

;=========================================================
;                    CIRCUIT MODULE
;=========================================================
(defmodule CIRCUIT (import MAIN deftemplate initial-fact)
```

```
(import TEMPLATES ?ALL)(import QUERY ?ALL))

(defrule CIRCUIT::initial-facts
(circuit number 1)
=>
(assert

(nonnumparam
      (name closed-circuit-grinding-is-possible)
      (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name ball-mill-discharge-density-too-low)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name density-control-is-a-problem)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name ball-mill-discharge-size-too-coarse)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name ball-mill-discharge-size-too-wide)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name fresh-feed-coarse)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name fresh-feed-contains-few-fines)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name high-ball-mill-discharge-density-required)
```

```
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name high-ball-mill-discharge-temperature-required)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name higher-capacity-or-finer-grind-required)
        (values yes no)
        (menu y n)
        (convert yes))

(question (parameter ball-mill-discharge-density-too-low)
          (module CIRCUIT)
          (the-question
"Is the ball mill discharge density too low? (y/n)

                  ==> ")
        (help
"    ..."))

(question (parameter density-control-is-a-problem)
          (module CIRCUIT)
        (precursors ball-mill-discharge-density-too-low is yes)
          (the-question
"Is the density control a problem? (y/n)

                  ==> ")
        (help
"    ..."))

(question (parameter high-ball-mill-discharge-density-required)
          (module CIRCUIT)
          (the-question
"Should the ball mill discharge density be
             as high as possible? (y/n)

                  ==> ")
        (help
"    ..."))

(question (parameter high-ball-mill-discharge-temperature-required)
          (module CIRCUIT)
        (precursors  high-ball-mill-discharge-density-required is-not yes)
          (the-question
"Should the ball mill discharge temperature be
             as high as possible? (y/n)
```

```
                ==> ")
        (help
"   ..."))

(question (parameter ball-mill-discharge-size-too-coarse)
          (module CIRCUIT)
          (the-question
"Is the ball mill discharge size distribution
                too coarse? (y/n)

                ==> ")
        (help
"   A classifier such as a hydrocyclone can be used to return
    particles coarser than a desired size to the mill."))

(question (parameter ball-mill-discharge-size-too-wide)
          (module CIRCUIT)
        (precursors  ball-mill-discharge-size-too-coarse is-not yes)
          (the-question
"Is the ball mill discharge size distribution
                too wide? (y/n)

                ==> ")
        (help
"   A classifier such as a hydrocyclone can be used to narrow
    the size distribution of the grinding circuit product."))

(question (parameter higher-capacity-or-finer-grind-required)
          (module CIRCUIT)
        (precursors  ball-mill-discharge-size-too-wide is-not yes)
          (the-question
"Is a higher capacity or a finer grind needed? (y/n)

                ==> ")
        (help
"   A closed grinding circuit can process a higher tonnage or
    produce a finer grind in comparison with a ball mill in an
    open circuit.")))

(defrule run-circuits
(parameter (name current-module)(value CIRCUIT))
=>
(system circuits.exe)
(open circuit.spc cir "r")
(assert (circuit number (read cir)))
(close cir))

(defrule closed-circuit-grinding
(circuit number 1)
=>
(assert
```

```
(question (parameter closed-circuit-grinding-is-possible)
          (module CIRCUIT)
          (the-question
"Can closed circuit grinding be used? (y/n)

                    ==> ")
         (help
"    ...")))
(focus QUERY))

(defrule bmd-too-coarse
(circuit number 1)
(parameter (name ball-mill-discharge-size-too-coarse)(value yes))
(parameter (name closed-circuit-grinding-is-possible)(value yes))
=>
(assert (parameter (name proposed-circuits) (value 5 6))))

(defrule bmd-too-wide
(circuit number 1)
(parameter (name ball-mill-discharge-size-too-wide)(value yes))
(parameter (name closed-circuit-grinding-is-possible)(value yes))
=>
(assert (parameter (name proposed-circuits) (value 5 6))))

(defrule needed-higher-capacity-or-finer-grind
(circuit number 1)
(parameter (name higher-capacity-or-finer-grind-required)(value yes))
(parameter (name closed-circuit-grinding-is-possible)(value yes))
=>
(assert (parameter (name proposed-circuits) (value 5 6))))

(defrule CIRCUIT::query-about-feed
(parameter (name proposed-circuits) (value 5 6))
=>
(assert
(question (parameter fresh-feed-coarse)
          (module CIRCUIT)
          (the-question
"Is the fresh feed too coarse? (y/n)

                    ==> ")
         (help
"    ...")))

(question (parameter fresh-feed-contains-few-fines)
          (module CIRCUIT)
          (the-question
"Does the fresh feed contain few fines? (y/n)

                    ==> ")
         (help
```

```
"    ...")))
(focus QUERY))

(defrule bmd-density-too-low
(circuit number 1)
?f <- (parameter (name proposed-circuits) (value 5 6))
(parameter (name ball-mill-discharge-density-too-low)(value yes))
(parameter (name density-control-is-a-problem)(value yes))
=>
(retract ?f)
(assert (conclusion-text (text
"    - Circuit 6 is proposed as an alternative to the
     current circuit.""")))

(defrule consider-circuit-5
(circuit number 1)
?f <- (parameter (name proposed-circuits) (value 5 6))
(parameter (name fresh-feed-coarse)(value yes))
(parameter (name fresh-feed-contains-few-fines)(value yes))
=>
(retract ?f)
(assert (conclusion-text (text
"    - Circuit 5 is proposed as an alternative to the
     current circuit.""")))

(defrule fresh-feed-not-coarse
(circuit number 1)
?f <- (parameter (name proposed-circuits) (value 5 6))
(parameter (name fresh-feed-coarse)(value no))
(parameter (name fresh-feed-contains-few-fines)(value no))
=>
(retract ?f)
(assert (conclusion-text (text
"    - Circuit 6 is proposed as an alternative to the
     current circuit.""")))

(defrule high-density-bmd-required
(circuit number 1)
(parameter (name high-ball-mill-discharge-density-required)(value yes))
(parameter (name closed-circuit-grinding-is-possible)(value no))
=>
(assert (conclusion-text (text
"    - It is recommended to consider adding grinding aids to
     the circuit.""")))

(defrule high-temp-bmd-required
(circuit number 1)
(parameter (name high-ball-mill-discharge-temperature-required)(value yes))
=>
(assert (conclusion-text (text
"    - It is recommended to consider adding grinding aids to
```

```
      the circuit."))))

(defrule load-cir5-facts
(circuit number 5)
=>
(assert

(nonnumparam
      (name fresh-feed-contains-significant-fines)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name very-sharp-classification-required)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name overgrinding-is-a-problem)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name coarse-material-is-a-problem)
        (values yes no)
        (menu y n)
        (convert yes))

(question (parameter fresh-feed-contains-significant-fines)
          (module CIRCUIT)
         (precursors  very-sharp-classification-required is-not yes)
          (the-question
"Does the fresh feed contain significant fines? (y/n)

                  ==> ")
         (help
"    ..."))

(question (parameter very-sharp-classification-required)
          (module CIRCUIT)
          (the-question
"Is it required to have a very sharp classification? (y/n)

                  ==> ")
         (help
"    ..."))

(question (parameter another-circuit-in-parallel)
```

```
          (module CIRCUIT) -
          (the-question
"Is there another circuit running in parallel? (y/n)

                    = = > ")
         (help
"    ..."))

(question (parameter large-reduction-ratio)
          (module CIRCUIT)
          (the-question
"Is the reduction ratio large? (y/n)

                    = = > ")
         (help
"    ..."))

(question (parameter overgrinding-is-a-problem)
          (module CIRCUIT)
          (the-question
"Is over grinding a problem? (y/n)

                    = = > ")
         (help
"    ..."))

(question (parameter coarse-material-is-a-problem)
          (module CIRCUIT)
        (precursors overgrinding-is-a-problem is-not yes)
          (the-question
"Are there coarse materials in circuit product
              creating metallurgical problems? (y/n)

                    = = > ")
         (help
"    ...")))

(defrule 5-to-6
(circuit number 5)
(parameter (name fresh-feed-contains-significant-fines)(value yes))
(parameter (name very-sharp-classification-required)(value no))
= >
(assert (conclusion-text (text
"    - Circuit 6 is proposed as an alternative to the
      current circuit. Since the fresh feed is very fine
      a pre-classification configuration is prefered."))))

(defrule 5-to-7-8-9
(circuit number 5)
(parameter (name very-sharp-classification-required)(value yes))
(parameter (name overgrinding-is-a-problem)(value no))
```

```
(parameter (name coarse-material-is-a-problem)(value no))
=>
(assert (conclusion-text (text
"    - Circuits 7, 8, and 9 are proposed as alternatives to the
      current circuit.")))

(defrule 5-to-7-8-9-over-grind
(circuit number 5)
(parameter (name very-sharp-classification-required)(value yes))
(parameter (name overgrinding-is-a-problem)(value yes))
=>
(assert (conclusion-text (text
"    - Circuits 7, 8, and 9 are proposed as alternatives to the
      current circuit. However, due to the over grinding problem
      circuit 11 is preferred.")))

(defrule 5-to-7-8-9-coarse
(circuit number 5)
(parameter (name very-sharp-classification-required)(value yes))
(parameter (name coarse-material-is-a-problem)(value yes))
=>
(assert (conclusion-text (text
"    - Circuits 7, 8, and 9 are proposed as alternatives to the
      current circuit. However, due to having coarse material
      in circuit product, circuit 7 is preferred.")))

(defrule assert-circuit-6-questions
(circuit number 6)
=>
(assert

(nonnumparam
      (name fresh-feed-contains-significant-oversize)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name overgrinding-is-a-problem)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name very-sharp-classification-required)
        (values yes no)
        (menu y n)
        (convert yes))

(question (parameter overgrinding-is-a-problem)
          (module CIRCUIT)
```

```
        (precursors  very-sharp-classification-required is yes)
          (the-question
"Is over grinding a problem? (y/n)

                  ==> ")
        (help
"    ..."))

(question (parameter fresh-feed-contains-significant-oversize)
          (module CIRCUIT)
        (precursors  very-sharp-classification-required is-not yes)
          (the-question
"Does the fresh feed contain significant oversize? (y/n)

                  ==> ")
        (help
"    ..."))

(question (parameter very-sharp-classification-required)
          (module CIRCUIT)
          (the-question
"Is it required to have a very sharp classification? (y/n)

                  ==> ")
        (help
"    ...")))
(focus QUERY))

(defrule 6-to-5
(circuit number 6)
(parameter (name very-sharp-classification-required)(value no))
(parameter (name fresh-feed-contains-significant-oversize)(value yes))
=>
(assert (conclusion-text (text
"    - Circuit 5 is proposed as an alternative to the
      current circuit."))))

(defrule 6-no-overgrinding
(circuit number 6)
(parameter (name very-sharp-classification-required)(value yes))
(parameter (name overgrinding-is-a-problem)(value no))
=>
(assert (conclusion-text (text
"    - Circuits 7 and 8 are proposed as alternative to the
      current circuit."))))

(defrule 6-overgrinding
(circuit number 6)
(parameter (name very-sharp-classification-required)(value yes))
(parameter (name overgrinding-is-a-problem)(value yes))
=>
```

```
(assert (conclusion-text (text
"    - Circuits 7 and 8 are proposed as alternative to the
       current circuit. However, dut to the over grinding
       problem, circuit 8 would be preferred."))))

(defrule assert-circuit-7-questions
(circuit number 7)
=>
(assert

(nonnumparam
      (name fresh-feed-contains-significant-fines)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name overgrinding-is-a-problem)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name secondary-cyclone-underflow-density-is-low)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name ball-mill-density-is-too-low)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name primary-cyclone-feed-density-is-high)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
      (name primary-cyclone-efficiency-is-low)
        (values yes no)
        (menu y n)
        (convert yes))

(question (parameter primary-cyclone-efficiency-is-low)
          (module CIRCUIT)
          (the-question
"Is the primary cyclone classification efficiency low? (y/n)
```

```
                ==> ")
        (help
"    ..."))

(question (parameter primary-cyclone-feed-density-is-high)
          (module CIRCUIT)
        (precursors primary-cyclone-efficiency-is-low is yes)
          (the-question
"Is the primary cyclone feed density high? (y/n)

                ==> ")
        (help
"    ..."))

(question (parameter overgrinding-is-a-problem)
          (module CIRCUIT)
          (the-question
"Is over grinding a problem? (y/n)

                ==> ")
        (help
"    ..."))

(question (parameter fresh-feed-contains-significant-fines)
          (module CIRCUIT)
        (precursors overgrinding-is-a-problem is yes)
          (the-question
"Does the fresh feed contain significant fines? (y/n)

                ==> ")
        (help
"    ..."))

(question (parameter secondary-cyclone-underflow-density-is-low)
          (module CIRCUIT)
        (precursors  ball-mill-density-is-too-low is yes)
          (the-question
"Is the secondary cyclone underflow density low? (y/n)

                ==> ")
        (help
"    ..."))

(question (parameter ball-mill-density-is-too-low)
          (module CIRCUIT)
          (the-question
"Is the ball mill density too low? (y/n)

                ==> ")
        (help
"    ...")))
```

```
(focus QUERY))

(defrule 7to8
(circuit number 7)
(parameter (name overgrinding-is-a-problem)(value yes))
(parameter (name fresh-feed-contains-significant-fines)(value no))
=>
(assert (conclusion-text (text
"    - Circuit 8 is proposed as an alternative to the
      current circuit to solve the over grinding problem."))))

(defrule 7to8x
(circuit number 7)
(parameter (name ball-mill-density-is-too-low)(value yes))
(parameter (name secondary-cyclone-underflow-density-is-low)(value yes))
=>
(assert (conclusion-text (text
"    - Circuit 8 is proposed as an alternative to the
      current circuit to solve the density problems."))))

(defrule 7to8xx
(circuit number 7)

(parameter (name primary-cyclone-efficiency-is-low)(value yes))
(parameter (name primary-cyclone-feed-density-is-high)(value yes))
=>
(assert (conclusion-text (text
"    - Circuit 8 is proposed as an alternative to the
      current circuit to solve primary classification
      efficiency and density problems."))))

(defrule 7to11
(circuit number 7)
(parameter (name fresh-feed-contains-significant-fines)(value yes))
(parameter (name overgrinding-is-a-problem)(value yes))
=>
(assert (conclusion-text (text
"    - Circuit 11 is proposed as an alternative to the
      current circuit."))))

(defrule default-conclusion
(circuit number ?x&~1&~5&~6&~7&~8)
=>
(assert (conclusion-text (text
"    - Currently, there are no rules in the knowledge base
      that can be applied to the selected circuit."))))

(defrule go-query
=>
(focus QUERY))
```

```
(defrule go-conclusion
=>
(focus CONCLUSION))

;=========================================================
;                    MODSIM MODULE
;=========================================================
(defmodule MODSIM    (import MAIN deftemplate initial-fact)
                     (import TEMPLATES ?ALL)
                     (import QUERY ?ALL))

(defclass COMMINUTION-UNIT (is-a USER)
 (role abstract)
 (slot manufacturer-name)
 (slot identification-number)
 (slot installation-year)
 (slot electrical-power)
 (slot capacity)
 (slot net-id (create-accessor read-write))
 (slot node (create-accessor read-write)))

(defclass TUMBLING-MILL (is-a COMMINUTION-UNIT)
 (role abstract)
 (slot length)
 (slot diameter))

(defclass BALLMILL (is-a TUMBLING-MILL)
 (role abstract)
 (slot media-type)
 (slot media-size)
 (message-handler grind))

(defmessage-handler BALLMILL grind()
    (system ballmill.exe))

(defclass OVERFLOW-DISCHARGE-BALLMILL (is-a BALLMILL)
 (role concrete)
 (pattern-match reactive))

(defclass HYDROCYCLONE (is-a USER)
 (role concrete)
 (slot manufacturer-name)
 (slot identification-number)
 (slot installation-year)
 (slot net-id (create-accessor read-write))
 (slot node (create-accessor read-write))
 (pattern-match reactive))

(defmessage-handler HYDROCYCLONE classify()
```

```
    (system cyclone.exe))

(defclass JUNCTION (is-a USER)
 (role concrete)
 (pattern-match reactive)
 (slot net-id (create-accessor read-write))
 (slot node (create-accessor read-write))
 (message-handler combine))

(defmessage-handler JUNCTION combine()
    (system junction.exe))

(defclass SPLIT (is-a USER)
 (role concrete)
 (pattern-match reactive)
 (slot net-id (create-accessor read-write))
 (slot node (create-accessor read-write)))

(defmessage-handler SPLIT split()
    (system split.exe))

(defclass FIXCLASS (is-a USER)
 (role concrete)
 (pattern-match reactive)
 (slot net-id (create-accessor read-write))
 (slot node (create-accessor read-write)))

(defmessage-handler FIXCLASS fixclass()
    (system fixclass.exe))

(defclass CONVERGENCE (is-a USER)
 (role concrete)
 (pattern-match reactive)
 (slot net-id (create-accessor read-write))
 (slot node (create-accessor read-write)))

(defmessage-handler CONVERGENCE converge()
    (system converge.exe))

(deffacts MODSIM::initial-facts
(nonnumparam
        (name task-to-do)
        (values  modelling simulation)
        (menu m s)
        (convert yes))

(question (parameter task-to-do)
        (module MODSIM)
        (the-question
"What task do you want to do?
```

```
            M  modelling
            S  simulation

                ==> ")
        (help
"     Modelling and simulation are two different tasks.  To simulate
    any circuit, a valid model of the circuit must be built first.")))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; START > task-to-do: modelling
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule load-modelling-parameters
(parameter (name task-to-do) (value modelling))
=>
(assert

(nonnumparam
        (name study-phase)
        (values preliminary detailed)
        (menu p d)
        (convert yes))

(nonnumparam
        (name one-data-set-available)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name data-set-balanced)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name breakage-function-known)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name selection-function-known)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name selection-function-estimation-method)
        (values sequential-interval-by-interval functional-form-based-search)
        (menu s f)
        (convert yes))
```

```
(nonnumparam
        (name rtd-known)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name Plitt-adjust-factors)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name run-ngotc)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name run-rtdboth)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name selection-function-shows-trend)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name selection-function-shows-noise)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name noise-in-top-size-classes)
        (values yes no)
        (menu y n)
        (convert yes))

(question (parameter breakage-function-known)
          (module MODSIM)
          (the-question
"Is the breakage function of the ore available? (y/n)

                ==> ")
        (help
"   The breakage function is an ore-dependent parameter of ball mill
    classical model.  It most be known in order to fit model to real
```

```
    ball mill."))

(question (parameter selection-function-known)
          (module MODSIM)
          (precursors breakage-function-known is yes and rtd-known is yes)
          (the-question
"Have you estimated the value of selection function
                  for each size class? (y/n)

                  ==> ")
        (help
"    Selection function is required by the BMCS simulator for each ball
     mill unit which exists in the circuit and must be given in the data
     file prepared by the user.  This parameter can be determined by
     running NGOTC program."))

(question (parameter rtd-known)
          (module MODSIM)
          (the-question
"Have you determined parameters of Weller's model
                  that describe residence time distribution (RTD) of
                  solid material flowing through the mill? (y/n)

                  ==> ")
        (help
"    Three parameters of Weller's model,i.e. tau plug flow,
     tau small perfect mixer and tau large perfect mixers are
     required to fit ball mill model."))

(question (parameter one-data-set-available)
          (module MODSIM)
          (the-question
"Is there at least one grinding data set available to estimate
              the selection function(s)? (y/n)

                  ==> ")
        (help
"    At least one grinding data set is needed to estimate a
     mill selection function.")
     (precursors rtd-known is yes and breakage-function-known is yes and
                 selection-function-known is no))

(question (parameter data-set-balanced)
          (module MODSIM)
          (the-question
"Have you already balanced data? (y/n)

                  ==> ")
        (help
"    It is better to use balanced data to estimate mill selection
     functions.")
```

```
    (precursors one-data-set-available is yes))

(question (parameter study-phase)
          (module MODSIM)
          (the-question
"What type of study are you doing now?

              P  preliminary
              D  detailed

                   ==> ")
         (help
"    If the study is in a preliminary stage, some information needed
    for modelling task can be assumed.  If a detailed study is
    excersised, more accurate information might be needed."))

(question (parameter selection-function-shows-trend)
          (module MODSIM)
          (the-question
"Does the selection function vs. particle size show
              a clear trend? (y/n)

                   ==> ")
         (help
"       ")
(precursors selection-function-known is yes))

(question (parameter selection-function-shows-noise)
          (module MODSIM)
          (the-question
"Is there noise in the selection function vs. particle
              size data? (y/n)

                   ==> ")
         (help
"       ")
(precursors selection-function-shows-trend is yes))

(question (parameter noise-in-top-size-classes)
          (module MODSIM)
          (the-question
"Does the noise exist in the top size classes? (y/n)

                   ==> ")
         (help
"       ")
(precursors selection-function-shows-noise is yes))

(question (parameter selection-function-estimation-method)
          (module MODSIM)
          (the-question
```

```
"Which method did you use to estimate the mill
                selection function?

            S  sequential interval-by-interval search
            F  use of functional forms of selection functions

                ==> ")
        (help
"       ")
(precursors selection-function-known is yes and study-phase is detailed)))
(focus QUERY))

(defrule use-typical-bf
(parameter (name task-to-do) (value modelling))
(parameter (name breakage-function-known)(value no))
(parameter (name study-phase) (value preliminary))
=>
(assert (conclusion-text (text
"    - It is recommended to use the breakage function of a similar
      ore for a preliminary work. For very accurate simulations, you
      would better off to determine the breakage function of the ore
      using representative samples of the ore."))))

(defrule do-rough-sf-bf
(parameter (name task-to-do) (value modelling))
(parameter (name breakage-function-known)(value no))
(parameter (name study-phase) (value preliminary))
=>
(assert (conclusion-text (text
"    - Although the actual ore breakage function is unavailable,
      it is still possible to estimate the selection function using
      typical ore breakage functions. For a detailed study,
      however, you may need to measure the breakage function."))))

(defrule do-rough-sf-rtd
(parameter (name task-to-do) (value modelling))
(parameter (name rtd-known)(value no))
(parameter (name study-phase) (value preliminary))
=>
(assert (conclusion-text (text
"    - Although the actual RTD parameters are unavailable,
      it is still possible to estimate the selection function
      using typical values. For a detailed study, however, you
      may need to measure RTD."))))

(defrule do-bf-tests
(parameter (name task-to-do) (value modelling))
(parameter (name breakage-function-known)(value no))
(parameter (name study-phase) (value detailed))
=>
(assert (conclusion-text (text
```

```
"    - It is recommended to determine the breakage function
       of the ore using representative samples of the ore and
       laboratory procedures.")))

(defrule  do-mass-balance
(parameter (name task-to-do) (value modelling))
(parameter (name selection-function-known)(value no))
(parameter (name data-set-balanced)(value no))
=>
(assert (conclusion-text (text
"    - It is recommended to use mass balanced data for selection
       function estimation.  You must use a mass balance software
       to first adjust raw data.")))

(defrule  post-to-do-sf
(parameter (name task-to-do) (value modelling))
(parameter (name breakage-function-known)(value yes))
(parameter (name rtd-known)(value yes))
(parameter (name data-set-balanced)(value yes))
(parameter (name selection-function-known)(value no))
=>
(post
"    - The NGOTC program must be run to back-calculate the selection
       function based on the available data set.  The user, however,
       must be familiar to use the program.")

(assert (ask-to-run ngotc)))

(defrule  use-typical-rtd
(parameter (name task-to-do) (value modelling))
(parameter (name rtd-known)(value no))
(parameter (name study-phase) (value preliminary))
=>
(assert (conclusion-text (text
"    - It is recommended to use typical values for Weller's
       model parameters such as tau PF=0.1, tau SPM=0.1 and
       tau LPM=0.7 at a standard ball mill feed rate.")))

(defrule  do-tracer-test
(parameter (name task-to-do) (value modelling))
(parameter (name rtd-known)(value no))
(parameter (name study-phase)(value detailed))
=>
(assert (conclusion-text (text
"    - It is recommended to do plant tracer tests to determine
       RTD model parameters.")))

(defrule ngotc
(ask-to-run ngotc)
=>
(assert
```

```
(question (parameter run-ngotc)
          (module MODSIM)
          (the-question
"Would you like the NGOTC program to be run? (y/n)

                  ==> ")
         (help
"    If you answer yes, GCOC will automatically start the NGOTC
   program for you.")))
(focus QUERY))

(defrule run-ngotc-program
(parameter (name run-ngotc) (value yes))
=>
(system ngotc.exe))

(defrule check-sf
(parameter (name task-to-do) (value modelling))
(parameter (name selection-function-shows-trend) (value no))
=>
(assert (conclusion-text (text
"    - The estimated selection function may not be valid.
      Normally, a selection function vs. particle size curve
      shows a linear trend at fine size range followed by a
      non-linear trend at coarse sizes. It is recommended to
      check the validity of the selection function before
      using it for the circuit simulations."))))

(defrule sf-ok
(parameter (name task-to-do) (value modelling))
(parameter (name selection-function-shows-trend) (value yes))
(parameter (name selection-function-shows-noise) (value no))
=>
(assert (conclusion-text (text
"    - The estimated selection function seems to be valid.
      Normally, if selection function vs. particle size curve
      has a clear trend and there is no significant noise in
      data particularly at coarse size classes, it indicates a
      reliable estimated selection function."))))

(defrule check-screening
(parameter (name task-to-do) (value modelling))
(parameter (name selection-function-shows-trend) (value yes))
(parameter (name selection-function-shows-noise) (value yes))
(parameter (name noise-in-top-size-classes)(value yes))
=>
(assert (conclusion-text (text
"    - The estimated selection function is valid for fine size
      classes. However, for the top size classes, the selection
      function values may be uncertain and erratic due to screening
      errors, if there is very little mass in top size classes."))))
```

```
(defrule small-screening-errors
(parameter (name task-to-do) (value modelling))
(parameter (name selection-function-shows-trend) (value yes))
(parameter (name selection-function-shows-noise) (value yes))
(parameter (name noise-in-top-size-classes)(value no))
=>
(assert (conclusion-text (text
"   - If the noise level is low and distributed over the full
      size size range, the estimated selection function is still
      valid. However, for simulation purposes, it is better to
      smooth the selection function values by the spline curve
      fitting algorithm."))))

(defrule use-functional-forms
(parameter (name task-to-do) (value modelling))
(parameter (name selection-function-estimation-method) (value sequential-interval-by-interval))
(parameter (name study-phase)(value detailed))
=>
(assert (conclusion-text (text
"   - As this is a detailed study, it is recommended to use other
      selection function estimation methods as well. For example,
      (1) use more than one data set (2) methods based on assumed
      functional forms of selection functions can be used. The best
      selection function vector then can be found by the analysis
      of results from various methods."))))

(defrule use-sequential-interval-by-interval
(parameter (name task-to-do) (value modelling))
(parameter (name selection-function-estimation-method) (value functional-form-based-search))
(parameter (name study-phase)(value detailed))
=>
(assert (conclusion-text (text
"   - As this is a detailed study, it is recommended to use other
      selection function estimation methods as well. For example,
      (1) use more than of data set (2) sequential interval-by-
      interval search method can be used. The best selection function
      vector then can be found by the analysis of results from various
      methods."))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; END > task-to-do: modelling
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; START > task-to-do: simulation
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule MODSIM::load-simulation-parameters
(parameter (name task-to-do) (value simulation))
=>
(assert
```

```
(nonnumparam
        (name modelling-done)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name user-datafile)
        (values yes no)
        (menu y n)
        (convert yes))

(nonnumparam
        (name run-bmcs)
        (values yes no)
        (menu y n)
      (convert yes))

(question (parameter modelling-done)
          (module MODSIM)
          (the-question
"Have you already done the modelling step? (y/n)

                  ==> ")
        (help
"    The BMCS simulator must be calibrated and tested before it can
    be used for any circuit simulation.")))
(focus QUERY))

(defrule modelling-first
(parameter (name modelling-done)(value no))
=>
(post
"    You need to build a model of your circuit by calibrating
    BMCS simulator. In modelling step, various model parameters
    must be estimated.")
(assert (modelling first)))

(defrule do-modelling
(modelling first)
=>
(assert
(nonnumparam
        (name shift-to-modelling)
        (values yes no)
        (menu y n)
        (convert yes))

(question (parameter shift-to-modelling)
          (module MODSIM)
          (the-question
```

```
"Would you like to shift focus to modelling? (y/n)

                    ==> ")
          (help
"    If you want to do modelling task answer yes.")))
(focus QUERY))

(defrule go-modelling
?f <- (parameter (name task-to-do) (value simulation))
(parameter (name shift-to-modelling)(value yes))
=>
(retract ?f)
(assert (parameter (name task-to-do) (value modelling))))

(defrule do-simulation
(parameter (name modelling-done)(value yes))
=>
(post
"    The BMCS program can be run now to simulate the grinding circuit.
    It is expected that the user be familiar with the program and be
    able to prepare the correct data file.")
(assert (ask-to-run bmcs)))

(defrule bmcs
(ask-to-run bmcs)
=>
(assert
(question (parameter run-bmcs)
          (module MODSIM)
          (the-question
"Would you like to start a simulation trial? (y/n)

                    ==> ")
          (help
"    GCOC will automatically start a pre-simulation program in case of
    a yes answer.")))
(focus QUERY))

(defrule run-presim
(parameter (name run-bmcs)(value yes))
=>
(system presim.exe)
(open circuit.spc i "r")
(bind  ?cirspc (read i))
(close i)
(if (eq ?cirspc exit)
  then
    (printout t crlf crlf crlf
    "    This circuit cannot be simulated with the current version of BMCS.")
    (printout t crlf crlf "    Please press any key to continue ...")
    (system presskey)
```

```
   else
    (if (eq (check-simulation-status simstat.lst) exit)
        then (assert (simulation status exit))
        else (load-facts cirfact.lib)
            (assert (presim done)))))

(defrule get-cirno-convnodenum
?f1 <- (presim done)
=>
(retract ?f1)
(open circuit.spc i "r")
(open cvrg.lst cvrg "r")
(bind  ?cirno (read i))
(bind  ?convrgnodenums (read cvrg))
(close i)
(close cvrg)
(assert (circuit number ?cirno))
(assert (convergnodenums  ?convrgnodenums))
(assert (current-iteration  1))
(printout t crlf crlf crlf crlf crlf))

(defrule create-instances
(circuit number ?cirno)
(circuit ?cirno ?node ?nodetype ?id)
=>
(if (eq ?nodetype 1) then (make-instance (sym-cat "BALLMILL-" ?id) of
OVERFLOW-DISCHARGE-BALLMILL (node ?node) (net-id ?id)))
(if (eq ?nodetype 2) then (make-instance (sym-cat "HYDROCYCLONE-" ?id) of HYDROCYCLONE
(node ?node) (net-id ?id)))
(if (eq ?nodetype 3) then (make-instance (sym-cat "JUNCTION-" ?id) of JUNCTION (node ?node)
(net-id ?id)))
(if (eq ?nodetype 4) then (make-instance (sym-cat "SPLIT-" ?id) of SPLIT (node ?node) (net-id ?id)))
(if (eq ?nodetype 5) then (make-instance (sym-cat "FIXCLASS-" ?id) of FIXCLASS (node ?node)
(net-id ?id)))
(if (eq ?nodetype 100) then (make-instance (sym-cat "CONVERGENCE-" ?id) of CONVERGENCE
(node ?node) (net-id ?id))))

(defrule do-iteration
(current-iteration ?iter)
(not (simulation completed))
=>
(printout t crlf "     ITERATION " ?iter)
(assert (current-node 1)))

(defrule  call-node-obj-1
(current-node ?node)
?o1 <- (object (is-a OVERFLOW-DISCHARGE-BALLMILL)(node ?node))
=>
(send (instance-name ?o1) grind)
(if (eq (check-simulation-status simstat.lst) exit) then
        (assert (simulation status exit))
```

```
else (assert (next node))))

(defrule  call-node-obj-2
(current-node ?node)
?o1 <- (object (is-a HYDROCYCLONE)(node ?node))
=>
(send (instance-name ?o1) classify)
(if (eq (check-simulation-status simstat.lst) exit) then
        (assert (simulation status exit))
else (assert (next node))))

(defrule  call-node-obj-3
(current-node ?node)
?o1 <- (object (is-a JUNCTION)(node ?node))
=>
(send (instance-name ?o1) combine)
(if (eq (check-simulation-status simstat.lst) exit) then
        (assert (simulation status exit))
else (assert (next node))))

(defrule  call-node-obj-4
(current-node ?node)
?o1 <- (object (is-a SPLIT)(node ?node))
=>
(send (instance-name ?o1) split)
(if (eq (check-simulation-status simstat.lst) exit) then
        (assert (simulation status exit))
else (assert (next node))))

(defrule  call-node-obj-5
(current-node ?node)
?o1 <- (object (is-a CONVERGENCE)(node ?node))
=>
(send (instance-name ?o1) converge)
(if (eq (check-simulation-status simstat.lst) exit) then
        (assert (simulation status exit))
else (assert (next node))))

(defrule next-node
?f1 <- (current-node ?node)
(circuit number ?cirno)
(circuit ?cirno number-of-nodes ?n)
(next node)
=>
(retract ?f1)
(if (< ?node ?n) then (assert (current-node =(+ ?node 1))) else (assert (check convergence))))

(defrule check-convergence
?f1 <- (check convergence)
?f2 <- (current-iteration ?iter)
=>
```

```
(retract ?f1 ?f2)
(system simcont.exe)
(open simstat.lst sim "r")
(bind ?status (read sim))
(if (eq ?status completed) then (assert (simulation completed)) (system repgen.exe)
 else (assert (current-iteration =(+ ?iter 1))))
(close sim))

(defrule simulation-stop
(simulation status exit)
=>
(assert (conclusion-text (text
"    - The simulation data file must be corrected before trying again."))))

(defrule  do-sampling
(parameter (name calibration-done)(value no))
(parameter (name two-data-sets)(value no))
=>
(assert (conclusion-text (text
"    - To build a grinding circuit model at least two data sets
      are needed.  One data set is required for the estimation
      of unit model parameters and another one is needed to
      validate the model built based on the first data set.  It
      is recommended that plant sampling campaigns to be done
      for calibration of the BMCS simulator."))))

(defrule  estimate-selection-function
(parameter (name calibration-done)(value no))
(parameter (name two-data-sets)(value yes))
(parameter (name selection-function) (value no))
=>
(post
"    The NGOTC program must be run to back-calculate selection
    function based on an available data set.  The user, however,
    must be familiar to use the program.")
(assert (ask-to-run ngotc)))

(defrule  no-circuit-data
(current-node ?node)
(circuit number ?cirno)
(not (circuit ?cirno ?node ?node-type ?id))
(not (nodenums ?cirno ?nodenums))
=>
(post
"    The facts regarding selected circuit was not found in the
    knowledge base.  Please assert the required facts prior to
    load knowledge base."))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  END > task-to-do: simulation
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(defrule go-query
=>
(focus QUERY))

(defrule go-conclusion
=>
(focus CONCLUSION))

;=========================================================
;                    CONCLUSION MODULE
;=========================================================
(defmodule CONCLUSION (import MAIN deftemplate initial-fact)
                      (import TEMPLATES ?ALL))

(deffunction CONCLUSION::printheader()
        (system cls)
        (printout t crlf "
    ****************************************************************
                              CONCLUSIONS
    ****************************************************************"crlf))

(defrule CONCLUSION::check
(not(reached-conclusion yes))
(or (conclusion-text (text ?text))(conclusion-string (string ?string)))
=>
(assert (reached-conclusion yes)))

(defrule header
(reached-conclusion yes)
=>
(printheader)
(printout t "    The system reached to the following conclusions:" crlf)
(assert (print conclusions)))

(defrule CONCLUSION::print-text
(print conclusions)
(conclusion-text (text ?text))
=>
(printout t crlf crlf ?text crlf)
(printout t crlf "     ==> ")
(system presskey))

(defrule CONCLUSION::print-string
(print conclusions)
(conclusion-string (string ?string) (arg1 ?arg1) (arg2 ?arg2))
=>
(format t ?string ?arg1 ?arg2)
(printout t crlf "     ==> ")
(system presskey))
```

```
(defrule no-conclusion-header
(not(reached-conclusion yes))
=>
(printheader)
(printout t crlf "     No conclusion was found.")
(printout t crlf crlf"     Please press any key to continue ... ")
(system presskey))

(defrule go-RESET
=>
(focus RESET))

;=========================================================
==========================
;                          RESET MODULE
;=========================================================
==========================
(defmodule RESET (import MAIN deftemplate initial-fact)
                 (import TEMPLATES ?ALL)
                 (import FUNCTIONS deffunction yes-or-no-p)
                 (import QUERY ?ALL))

(defrule RESET::reset-and-clear
=>
(if (yes-or-no-p "     The current consultation session is terminated.

    Would you like to start a new session? (y/n)

    ==> ")
then (reset) else (exit)))
```