Nonlinear, dispersive, shallow-water waves developed by a moving bed

By MARC VILLENEUVE

Department of Civil Engineering & Applied Mechanics

McGill University, Montreal

July 1989

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master in Engineering.

Abstract

Landslides and avalanches plunging into lakes or reservoirs located in mountainous regions can generate large waves which can result in loss of life and significant property damage. As is the case with tsunamis generated in the ocean by underwater seismic activity, landslide induced water waves result from the motion of a solid boundary of the fluid. The present study deals with the mathematical modelling of water waves developed in a channel by a moving bed. A set of depth-averaged governing equations is derived to predict the evolution of the free surface resulting from a predetermined bed motion. These equations, which constitute a generalization of the Boussinesq system for waves over a flat bed, include both nonlinear and dispersive effects. Numerical solutions are obtained by using the finite difference method coupled with a Flux Corrected Transport (FCT) algorithm. The resulting model is used to predict the waves resulting from simple bed motions. The numerical results show an excellent agreement with corresponding experimental results obtained by J. Sander at ETH, Zürich.

Sommaire

En région montagneuse, les glissements de terrain et avalanches plongeant dans les lacs ou réservoirs peuvent provoquer des ondes de forte amplitude constituant une menace pour les populations et les propriétés avoisinantes. Tout comme les tsunamis provoqués dans l'océan par une activité sismique sous-marine, les ondes induites dans l'eau par des glissements de terrain résultent d'un mouvement d'une des frontières solides du fluide La présente étude porte sur la modélisation mathématique d'intumescences engendrées dans un canal par un mouvement du fond. Des équations intégrées par rapport à la profondeur sont développées en vue de prédire l'évolution de la surface libre suite à un déplacement prédéterminé du lit. Ces équations constituent une généralisation de la formulation de Boussinesq s'appliquant aux ondes propagées sur un fond plat Elles incluent à la fois des effets non-linéaires et de dispersion. Des solutions numériques sont obtenues à l'aide de la méthode des différences finies associée à un algorithme de 'Flux Corrected Transport' (FCT). Le modèle résultant est utilisé pour prédire les ondes engendrées par de simples mouvements du fond. Les résultats numériques reproduisent très bien des résultats expérimentaux correspondants obtenus par J. Sander à ETH, Zurich.

Acknowledgements

The author is grateful to Dr. Stuart B. Savage for his help and guidance and to the Natural Sciences and Engineering Research Council of Canada (NSERC) for financial support of his work He also wishes to thank Dr. K. Hutter and J. Sander of the Laboratory of Hydraulics, Hydrology and Glaciology (VAW) of the Swiss Federal Institute of Technology (ETH) in Zürich for their kind help and cooperation and the Director Dr D. Vischer for permitting the unpublished results of J. Sander to be quoted in this thesis.

Contents

Abstract i
Acknowledgements ii
List of Figures v
List of Tables vi

1 Introduction 1
1.1 Theoretical background 1
1.2 Thesis' outline 7

2 Governing equations 10
2.1 Derivation of the governing equations 10
2.2 Scaling of the governing equations 15

3 Numerical solutions 21
3.1 The finite difference method 21
3.1.1 Eulerian description 22
3.1.2 Lagrangian description 24
3.1.3 Explicit and implicit formulations 26
3.2 Finite difference formulation of the governing equations 28
3.2.1 Choice of the flow field description 28
3.2.2 Eulerian scheme 29
3.2.3 Lagrangian scheme 32
3.3 Initial and boundary conditions 34
3.3.1 Initial conditions 34
3.3.2 Boundary conditions 35
3.4 The Flux Corrected Transport (FCT) method 36

4 Computations and experiments 41
4.1 Types of bed motion considered 41
4.1.1 Moving wall 42
4.1.2 Submerged wedge 42
4.1.3 Moving shelf 44
4.1.4 Rotating plate 44
4.1.5 Moving wedge 45
4.2 Experimental set-up 45
4.2.1 Rotating plate 45
4.2.2 Lateral displacements 46
4.3 Computer models 47
4.3.1 General description 47
4.3.2 Program WALL 49
4.3.3 Program SUBWED 50
4.3.4 Program SHELF 51

4.3.5 Program ROPLATE 51
4.3.6 Program WEDGE 51

5 Comparison of results **53**
5.1 Moving wall 53
5.2 Submerged wedge 56
5.3 Moving shelf 59
5.4 Rotating plate 62
5.5 Moving wedge 64

6 Conclusions **67**

References **69**

Appendices
A Validity of the shallow-water approximation A-1
B Derivation of the governing equations using an expansion method . . B-1
C Computer programs C-1
D Supplementary results D-1

List of Figures

2.1 Definition sketch for waves generated by a moving bed. 10
3 1 The Eulerian finite difference problem. 22
3.2 The Lagrangian finite difference problem. 24
3 3 The Lagrangian grid. 25
3.4 The explicit form 26
3.5 The fully implicit form 27
3.6 The Crank-Nicolson form 27
3.7 Eulerian description of the flow field 29
3.8 Lagrangian description of the flow field. 33
4.1 Types of bed motion considered. 43
5.1a Wave generated by a moving wall. 53
5.1b Wave generated by a moving wall (comparison with experiment) . 55
5.2a Wave generated by a submerged wedge. 56
5.2b Wave generated by a submerged wedge (comparison with experiment) 58
5.3a Wave generated by a moving shelf. 60
5.3b Wave generated by a moving shelf (comparison with experiment). 61
5.4a Wave generated by a rotating plate 62
5.4b Wave generated by a rotating plate (comparison with experiment) 63
5 5a Wave generated by a moving wedge 65
5.5b Wave generated by a moving wedge (comparison with experiment) 66

List of Tables

4 1 Types of bed motion considered and corresponding computer programs 48

1 INTRODUCTION

1.1 Theoretical background

In mountainous regions, rockslides, snow or ice avalanches and calving glaciers sometimes enter lakes and water supply reservoirs. In doing so they often generate large waves which can result in loss of life and significant property damage. One of the greatest disasters of that type occured in the Vaiont Valley, Italy in 1963 where a wave generated by a landslide into a reservoir completely destroyed a town and claimed over 2,000 lives. Similar events dating to, at least, 1731 also occured in many fjords of Norway and caused hundreds of fatalities. Among other well known examples are the 1792 catastrophe of Shimbara Bay, Japan; the 1850 and 1905 glacier falls of Disenchantment Bay and the 1958 rockslide of Lituya Bay, Alaska, and finally the 1971 landslide in Lake Yanahuin, Peru. The combined death toll from all these disasters exceeds 20,000 people (Slingerland & Voight 1979). Huber (1982) has given details of five rock avalanches and a bankslide that occured in Swiss lakes between 1923 and 1974; all resulted in serious damage to the shoreline structures.

Landslide generated waves in lakes and reservoirs have much in common with tsunamis which are oceanic waves resulting from underwater seismic activity (see Voit (1987) for a recent review of tsunamis). Three stages of development are usually associated with both of these phenomena:

1. Generation of an initial wave by a motion of the fluid's boundary
2. Free propagation in water of approximately constant depth.
3. Propagation of the wave in coastal waters of nonuniform depth where the shallowness results in amplification and strong deformation of the wave profile prior to the runup on beaches.

One of the important problems arising in the theoretical treatment of these three stages is the choice of approximations to obtain appropriate mathematical models

Peregrine (1972) reviews important water wave equations and their underlying approximations.

The equations of the linear (or 'small-amplitude') theory are obtained by linearization of the free surface boundary conditions; they are only valid for waves propagating in water having a depth of the same order as the wavelength (i.e. deep water waves) and an amplitude negligible compared to the water depth (i.e. small-amplitude waves). These waves have a tendency to spread out as they propagate, an effect known as frequency dispersion. In one spatial dimension, if we denote the respective positions of the free surface and the bed by η and h, we can use the velocity potential ϕ to write :

$$\nabla^2\phi = 0 \tag{1.1a}$$

$$\eta_t = \phi_y \text{ and } \phi_t + g\eta = 0 \text{ on } y = 0 \tag{1.1b}$$

$$\phi_x h_x + \phi_y = 0 \text{ on } y = -h(x) \tag{1.1c}$$

If on the other hand we only know that the water depth to wavelength ratio is small (i.e. shallow-water, or 'long' waves), it can be assumed that the pressure distribution is hydrostatic or equivalently that the horizontal velocity is uniform with depth. The resulting nonlinear equations form the basis of the 'finite-amplitude, shallow-water theory':

$$\eta_t + [(h+\eta)\overline{u}]_x = 0 \tag{1.2a}$$

$$\overline{u}_t + \overline{u}\overline{u}_x + g\eta_x = 0 \tag{1.2b}$$

where $\overline{u}$ represents the uniform horizontal velocity of the fluid. Waves predicted by that theory steepen as they propagate under the influence of nonlinearity; this phenomenon is sometimes called amplitude dispersion.

If in addition to the shallow-water approximation, it is assumed that the amplitude to water depth ratio is of small (but not negligible) order, equations can be derived in which the nonlinear effect of amplitude dispersion is approximately balanced by the linear effect of frequency dispersion. The resulting weakly nonlinear, dispersive equations are called the Boussinesq equations. The typical solution of these equations

is the solitary wave which is a single wave of elevation propagating without change in form. For waves propagating in water of unit depth, the Boussinesq system (Witham 1974) is

$$\eta_t + [(h+\eta)\overline{u}]_x = 0 \tag{1.3a}$$

$$\overline{u}_t + \overline{u}\overline{u}_x + g\eta_x = \frac{1}{3}\overline{u}_{xxt} \tag{1.3b}$$

Finally, the neglect of both the nonlinear and dispersive terms in the above theories leads to the 'linearized long-wave theory':

$$\eta_t + (h\overline{u})_x = 0 \tag{1.4a}$$

$$\overline{u}_t + g\eta_x = 0. \tag{1.4b}$$

In the following text, the term 'linear theory' will only refer to the linearized, small-amplitude equations and not to the linearized long-wave equations.

Since they represent the most direct threat to shoreline populations and structures, the shoaling and runup of landslide generated waves and tsunamis have received much attention. This third stage of development involves waves in shallow water and clearly requires a nonlinear theory. In most cases, the incoming wave is modelled as a solitary wave propagating over a nonuniform (usually uniformly sloping) bed. Weakly nonlinear, dispersive equations were derived by Peregrine (1967). These depth-averaged equations are a generalization of the Boussinesq system for waves over a flat bed. Higher order equations have also been derived (Seabras-Santos, Renouard & Temperville 1987).

As it approaches the shoreline, the wave front steepens under the influence of shallowness. The treatment of the actual runup depends on the wave characteristics and on the beach geometry. If the wave slope is small—as it is for most tsunamis and landslide generated waves—or if the beach slope is large, the wave will not have a tendency to break. If the beach slope steepness does not exceed a certain limit, the runup can then be described analytically by uniformly valid solutions of the finite-amplitude, shallow-water (Airy) equations (Carrier & Greenspan 1958, Spielvogel

1975). Otherwise, it can be modelled by numerical solutions of equations of the Boussinesq type (Pedersen & Gjevik 1983). For steeper wave slopes or for very mild beach slopes, the incoming wave will break. Peregrine (1983) reviews the case of breaking waves.

The linear theory and the Boussinesq system have both been used to model the free propagation stage in water of approximately constant depth. The respective validity of these models is highly dependent on the characteristics of the initially generated waves and on the water depth of the region in which they subsequently propagate.

The choice of an appropriate model for the generation stage still remains an open question. Until recently, before the availability of numerical solutions, the linear theory was almost the inevitable choice. Therefore, it had to be assumed that the amplitude of the generated wave was very small relative to both the water depth and the wavelength. The generation of the wave was often represented as an initial disturbance (elevation or depression) of the free surface imposed at the end of a constant depth channel (Kranzer & Keller 1959, Wiegel et al. 1970, Noda 1971). That initial condition is equivalent to the vertically falling box model used by Noda (1970) to simulate a vertical rockfall into a channel or to the impulsive upthrust of a bed's segment used by Hammack (1973) in the study of tsunami generation. In all cases, the resulting theoretical problem was similar to the well known Cauchy-Poisson problem (Wehausen & Laitonen 1960). More recently, Hunt (1988) used the linear theory to model landslide generated waves by injecting an instantaneous point source of fluid through the bottom of a channel; his results were very close to those of Noda (1970). Solutions of the linear theory are integrals which are obtained by various transform methods. These integrals are usually very difficult to evaluate numerically due to the oscillatory nature of the integrands and the usual procedure is to apply asymptotic methods (stationary phase and steepest descent) which yield solutions which are valid only for large time and large distances from the source of generation.

Wiegel et al. (1970) compared their solutions with the experiments of Prins (1958)

and also performed their own experiments to simulate vertical rockfalls by dropping vertically falling boxes at the end of a channel. Noda (1970) compared his theoretical results with these box drop experiments. In both cases, it was found that the linear theory was only valid for small height and length of the initial disturbance (Wiegel et al. 1970) or equivalently for small length and vertical displacements of the falling box (Noda 1970). For larger dimensions, nonlinear effects became noticeable as solitary waves and bores started to develop. Noda (1970) also simulated horizontal landslides by the horizontal translation of a vertical wall. The corresponding experiments of Das & Wiegel (1972) showed the results of the linear theory to be only valid for small displacements of the wall. The above conclusions were to be expected since the linear theory is only applicable in the case of small-amplitude waves which in turn can only be generated by small displacements or dimensions of the moving boundaries.

However, since most of the above theoretical results were only valid for large values of time and distance, it was not possible to determine if the linear theory only became invalid in the free propagation stage or right from the start of the generation. Hammack (1973) looked at this problem in the context of tsunamis generated by a positive or negative vertical displacement of a segment of the bed. He considered separately the ranges of validity of the linear theory in the generation and propagation stages. By scaling the fundamental equations of motion, he found that, for impulsive bed motions, the linear theory was only correct for generation by very small bed displacements. For slow bed motions, he showed the linear theory to be valid for arbitrary large bed displacements. Using Ursell number considerations, he also claimed that the Boussinesq system should be preferred to the linear theory for the free propagation stage during which nonlinear effects accumulate with time. The Ursell number is a dimensionless number measuring the relative importance of nonlinear to linear wave effects (Ursell 1953).

From the above, it appears that one of the main disadvantages of the linear theory is that it only possesses narrow ranges of validity and thus can only describe very limited generating conditions. Under the influence of frequency dispersion, waves

generated and propagated using the linear theory eventually become independent of the geometry and time history of the generating process and only depend on the volume of water displaced originally (Hunt 1988). That description may account for waves created by rapid rockfalls of small dimensions but certainly does not describe properly large progressive bankslides in which the generated wave often propagates without change in shape, almost as a solitary wave (Huber 1982). Furthermore, the complexity of the integral solutions even for simple bed motions precludes the application of the linear theory to more realistic descriptions of the moving bed. Sabatier (1983) used the linear theory for waves generated by ground motions on a slope.

One of the first attempts to use numerical solutions of nonlinear wave theories in the description of the generation stage was made by Hwang & Divoky (1970). They used the finite-amplitude, shallow-water equations in two spatial dimensions to model the tsunami which resulted from the Alaskan Good Friday earthquake of 1964. The effect of the bed motion was incorporated directly in the depth-averaged continuity equation. In that respect the numerical treatment of the governing equations possesses the advantage of allowing one to treat nonlinearity and more complex bed motions with little increase in difficulty over that corresponding to the use of the linear wave equations. That is of considerable importance since, as pointed out by Hwang & Divoky (1970), one of the fundamental problems in the generation phase is the accurate estimation of the bed motion, not only in its permanent displacement (as implied by impulsive linear theories), but also in its complicated time history. Comparison of their numerical results with limited field data showed an encouraging agreement.

Raney & Butler (1976) used a similar procedure to study landslide generated waves but modified the governing equations to include forcing functions accounting for the volume displacement, viscous drag and "form" drag exerted by a landslide on the water. Their numerical solutions were only presumed good for the leading wave since their governing equations did not properly account for wave runup and

reflection.

The finite-amplitude, shallow-water theory has the advantage over the linear theory of allowing for the propagation of larger amplitude waves. However, it can hardly be used in the free propagation stage since it ignores frequency dispersion and ultimately leads to an infinite wave steepness. Therefore, as with the case of the linear theory, the nonlinear nondispersive theory suffers from a narrow range of applicability.

The same can also be said about the linearized long-wave equations (Tuck & Hwang 1972) which can be used to some extent in the generation stage but rapidly become invalid.

The logical alternative to the above theories is to assume that the amplitude to depth ratio is small and derive model equations which combine the linear effect of frequency dispersion and the nonlinear effect of amplitude dispersion. Wu (1981) derived weakly nonlinear, dispersive (Boussinesq) equations including the effects of the moving bed. These equations would appear to constitute a very good model for the complete description of the generation and propagation stages. Unfortunately, Wu did not present numerical results for actual moving bed problems.

1.2 Thesis' outline

The following thesis is concerned with the mathematical modelling of water waves developed by a moving bed. It was performed at McGill University in conjunction with corresponding experimental work carried out at the Laboratory of Hydraulics, Hydrology and Glaciology directed by Dr. Daniel Vischer and affiliated to the Swiss Federal Institute of Technology in Zürich. The experiments were made by Johannes Sander under the direction of Drs. Kolumban Hutter and Daniel Vischer.

The basic idea was to check the validity of a depth-averaged, weakly nonlinear and dispersive (i.e. Boussinesq) approach to the numerical modelling of water waves induced by landslides and avalanches plunging into alpine lakes and water supply reservoirs. The logical starting point was to consider generation and propagation in

one spatial dimension before studying more realistic problems in two dimensions.

The present study deals primarily with waves generated at the upstream end of a long channel. The generation of the wave by the incoming material is simulated by the motion of an impervious boundary at the end of the channel. The above schematizations lead to the development of a model which has the advantage of being applicable not only to landslide generated waves but to all problems involving the generation of water waves under the action of moving solid boundaries.

The thesis is divided in six sections. In §2, the fundamental equations governing the motion of an inviscid, incompressible and irrotational fluid are integrated over the water depth. The resulting depth-averaged equations give the free surface disturbance $\eta(x,t)$ and the depth-averaged fluid velocity $\overline{u}(x,t)$ resulting from a known bed history $h(x,t)$. These high-order equations are a generalization of those derived by Su & Gardner (1969) for waves over a steady, uniform bed and by Seabra-Santos, Renouard & Temperville (1987) for waves over a steady, nonuniform bed. The validity of the shallow-water assumption in problems involving a moving bed is discussed in Appendix A.

Scaling parameters pertaining to the wave characteristics are introduced in §2.2 in order to reduce the governing equations to a 'Boussinesq' formulation in which nonlinear wave effects are approximately balanced by dispersive effects. The resulting equations are identical to those derived by Wu (1981). Appendix B outlines a derivation of these equations using a more direct pertubation method.

Finite difference schemes are developed in §3 to solve the depth-averaged continuity and momentum equations. The basic algorithm consists of a simple three step scheme in which both explicit and implicit finite difference approximations are used. The Eulerian specification of the flow field is used except in problems involving a laterally moving waterline. These problems are more easily treated with the help of the Lagrangian formulation.

Initial conditions and boundary conditions applying to the various types of bed motions are described in §3.3. A Flux Corrected Transport (FCT) algorithm is pre-

sented in §3.4 in order to eliminate spurious oscillations from the computed free surface profiles.

Five simple moving bed geometries are modelled both numerically and experimentally:

1. Moving vertical wall,
2. Moving submerged wedge,
3. Moving shelf,
4. Rotating plate,
5. Moving wedge.

The above bed motions are presented and described in §4.1. Apart from the 'rotating plate' wave generating mechanism, the four remaining bed motions arise from the horizontal translation of a wavemaker having a variable geometry.

The experimental set-up used by Sander (1988) is briefly described in §4.2 and corresponding computer models are introduced in §4.3. Appendix C gives the listing of the Pascal source codes of the programs.

Comparisons between the numerical and experimental results are treated in §5. In every cases, the agreement between computed and measured wave heights is excellent and confirms the validity of the weakly nonlinear, dispersive numerical model for the simulation of water waves developed by a moving bed.

2 GOVERNING EQUATIONS

2.1 Derivation of the governing equations

We start by considering an incompressible, inviscid and irrotational fluid bounded by a free surface $y = \eta(x,t)$ and a bed $y = -h(x,t)$ with the position of the free surface at rest being $y = 0$. The total water depth $h + \eta$ is denoted by H (Figure 2.1). Our objective in the study of water waves developed by a moving bed consists in

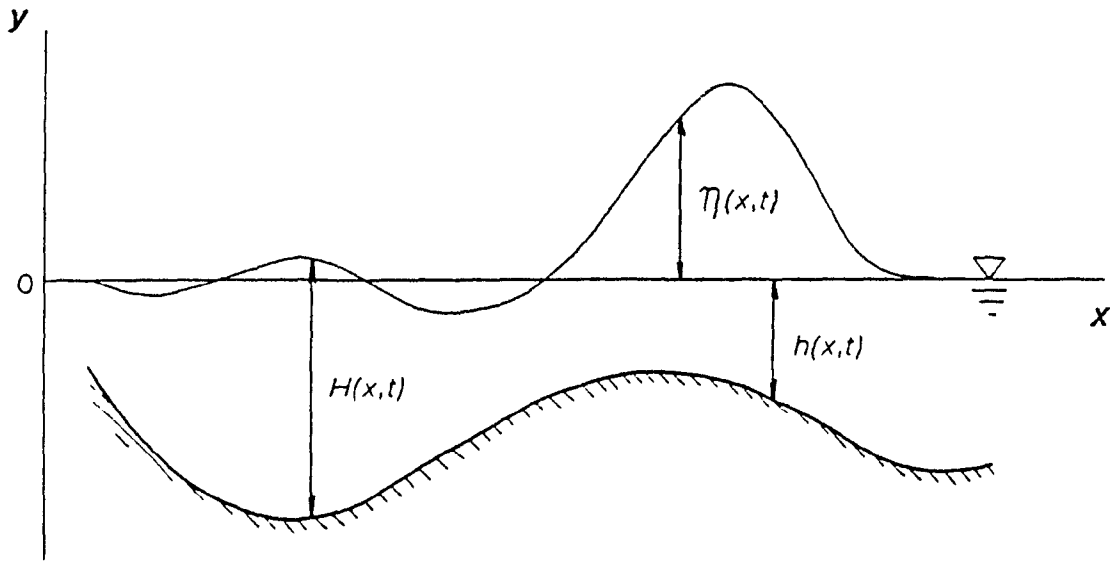


Figure 2.1: Definition sketch for waves generated by a moving bed.

determining the deformation $\eta(x,t)$ of the free surface resulting from a prescribed motion $h(x,t)$ of the bed. The following non-dimensional variables are used:

$$(x, y, \eta, h) = h_o^{-1}(x', y', \eta', h'),$$

$$(u, v) = (gh_o)^{-\frac{1}{2}}(u', v'),$$

$$p = p'/(\rho g h_o),$$

$$t = (g/h_o)^{\frac{1}{2}}t',$$

where the primes denote dimensional (i.e. physical) variables and h_o is a characteristic value for the water depth. The horizontal and vertical components of the fluid velocity

are respectively $u = u(x, y, t)$ and $v = v(x, y, t)$ The pressure at any point in the fluid is given by $p(x, y, t)$. The constant fluid density is ρ and the gravitational acceleration g acts vertically downward. The fundamental two-dimensional equations of motion are then:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{2.1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x}, \tag{2.2}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} - 1, \tag{2.3}$$

$$\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} = 0. \tag{2.4}$$

Equation (2.1) is the continuity equation for an incompressible fluid. Equations (2.2) and (2.3) are Euler's momentum equations in the x and y directions respectively and (2.4) is the irrotationality condition. Equations (2.1)–(2.4) are subject to the kinematic boundary conditions

$$v_s = \frac{\partial \eta}{\partial t} + u_s\frac{\partial \eta}{\partial x}, \tag{2.5}$$

$$v_b = -\frac{\partial h}{\partial t} - u_b\frac{\partial h}{\partial x}, \tag{2.6}$$

and the stress free surface condition

$$p_s = 0 \tag{2.7}$$

where the subscripts s and b respectively indicate quantities evaluated at the free surface and at the bed. The boundary conditions (2.5) and (2.6) state that a fluid particle located at the free surface (or at the bed) subsequently moves along with the free surface (or the bed). The boundary condition (2.7) simply implies that there is no pressure acting on the free surface.

Integration of the continuity equation (2.1) in the vertical direction, from the bed to the free surface, yields

$$v_s - v_b = u_s\frac{\partial \eta}{\partial x} + u_b\frac{\partial h}{\partial x} - \frac{\partial}{\partial x}\int_{-h}^{\eta} u\, dy \tag{2.8}$$

in which the kinematic boundary conditions (2.5) and (2.6) can be substituted to give

$$\frac{\partial}{\partial t}(h+\eta)+\frac{\partial}{\partial x}[(h+\eta)\overline{u}]=0 \tag{2.9}$$

where

$$\overline{u}(x,t)=\frac{1}{H}\int_{-h}^{\eta}u(x,y,t)\,dy. \tag{2.10}$$

Equation (2.9) is the depth-averaged continuity equation. For a known bed history $h(x,t)$, it gives the evolution of the free surface η, provided we know the depth-averaged velocity $\overline{u}$. Note that (2.9) is exact since no approximation was made in its derivation.

For shallow-water wave propagation, it is usually assumed that the horizontal velocity of the fluid is approximately uniform with depth:

$$u(x,y,t)\simeq\overline{u}(x,t). \tag{2.11}$$

By integrating the continuity equation (2.1) using (2.11) and the kinematic boundary condition (2.6), we can obtain a corresponding approximation for the vertical fluid velocity:

$$v(x,y,t)\simeq-\frac{\partial h}{\partial t}-\frac{\partial}{\partial x}(h\overline{u})-y\frac{\partial\overline{u}}{\partial x}. \tag{2.12}$$

In view of deriving an evolution equation for $\overline{u}$, we substitute the approxiomation (2.11) in the x-momentum equation (2.2):

$$\frac{\partial\overline{u}}{\partial t}+\overline{u}\frac{\partial\overline{u}}{\partial x}+\frac{\partial p}{\partial x}=0. \tag{2.13}$$

Equation (2.13) can be integrated from $y=-h$ to η with the boundary conditions (2.5)-(2.7) to give

$$\frac{\partial}{\partial t}(H\overline{u})+\frac{\partial}{\partial x}[H(\overline{u}^2+\overline{p})]=p_b\frac{\partial h}{\partial x} \tag{2.14}$$

where the depth-averaged pressure is

$$\overline{p}(x,t)=\frac{1}{H}\int_{-h}^{\eta}p(x,y,t)\,dy. \tag{2.15}$$

It is important to note that the depth-averaging of the complete x-momentum equation (2.2), without the approximation (2.11), yields

$$\frac{\partial}{\partial t}(H\overline{u}) + \frac{\partial}{\partial x}[H(\overline{u}^2 + \overline{p}) + H(\overline{u^2} - \overline{u}^2)] = p_b \frac{\partial h}{\partial x} \tag{2.16}$$

with

$$\overline{u^2}(x,t) = \frac{1}{H}\int_{-h}^{\eta} u^2(x,y,t)\,dy. \tag{2.17}$$

Therefore the validity of the approximation (2.11) which led to equation (2.14) implies that the term $H(\overline{u^2}-\overline{u}^2)$ is of small order compared to the remaining terms of equation (2.16). That requirement is always satisfied for a flat bed, but in the case of a moving bed, it imposes certain restraints on the bed slope and velocity (Appendix A)

An expression for the pressure is obtained by integration of the y-momentum equation (2.3) with the dynamic boundary condition (2.6):

$$p = (\eta - y) + \int_y^{\eta} \frac{dv}{dt}\,dy \tag{2.18}$$

where

$$\frac{d}{dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y}$$

which can be approximated as:

$$\frac{d}{dt} \simeq \frac{\partial}{\partial t} + \overline{u}\frac{\partial}{\partial x} + v\frac{\partial}{\partial y}.$$

The second term on the right hand side of (2.18) represents the correction to the hydrostatic pressure. Using the approximation (2.12) for v, we can write:

$$\frac{dv}{dt} = -\frac{dh_t}{dt} - \frac{d(h\overline{u})_x}{dt} - y\frac{d\overline{u}_x}{dt} - v\overline{u}_x \tag{2.19}$$

or

$$\frac{dv}{dt} = -\beta - \gamma - y\alpha$$

where

$$\alpha = \frac{d\overline{u}_x}{dt} - \overline{u}_x^2, \tag{2.20a}$$

$$\beta = \frac{d(h\overline{u})_x}{dt} - (h\overline{u})_x\overline{u}_x, \tag{2.20b}$$

$$\gamma = \frac{dh_t}{dt} - h_t \overline{u}_x. \tag{2.20c}$$

Substitution of (2.19) in (2.18) gives

$$p = (\eta - y)(1 - \beta - \gamma) - \left(\frac{\eta^2 - y^2}{2}\right)\alpha. \tag{2.21}$$

The pressure at the bed and the depth-averaged pressure are then:

$$p_b = H(1 - \beta - \gamma) - \left(\frac{\eta^2 - h^2}{2}\right)\alpha \tag{2.22}$$

$$\overline{p} = \frac{H}{2}(1 - \beta - \gamma) - \frac{1}{H}\left(\frac{\eta^3}{3} + \frac{\eta^2 h}{2} - \frac{h^3}{6}\right)\alpha. \tag{2.23}$$

Finally, substitution of (2.22)-(2.23) in (2.14) gives:

$$\frac{\partial}{\partial t}(H\overline{u}) + \frac{\partial}{\partial x}\left[H\overline{u}^2 + \frac{H^2}{2}(1 - \beta - \gamma) - \left(\frac{\eta^3}{3} + \frac{\eta^2 h}{2} - \frac{h^3}{6}\right)\alpha\right] = \frac{\partial h}{\partial x}\left[H(1 - \beta - \gamma) - \left(\frac{\eta^2 - h^2}{2}\right)\alpha\right]. \tag{2.24}$$

Equation (2.24) together with the continuity equation (2.9) constitute a set of depth-averaged equations governing the amplitude η and velocity $\overline{u}$ of waves developed by a moving bed.

If the bed is not moving, $h_t = 0$ and (2.24) can be simplified. The term γ defined by (2.20c) vanishes and β in (2.20b) reduces to:

$$\beta = \overline{u}_t h_x + \overline{u}^2 h_{xx} + \overline{u}\overline{u}_x h_x + h\alpha.$$

The resulting equation is then:

$$\frac{\partial}{\partial t}(H\overline{u}) + \frac{\partial}{\partial x}\left[H\overline{u}^2 + \frac{H^2}{2}(1 - \phi) - \frac{H^3}{3}\alpha\right] = \frac{\partial h}{\partial x}\left[H(1 - \phi) - \frac{H^2}{2}\alpha\right] \tag{2.25}$$

where

$$\phi = \beta - h\alpha = \overline{u}_t h_x + \overline{u}\overline{u}_x h_x + \overline{u}^2 h_{xx}.$$

Equation (2.25) is the same as the equation derived by Seabra-Santos, Renouard & Temperville (1987).

For waves over a flat, horizontal bed, γ is again zero and since $h_x = 0$, we have $\beta = h\alpha$. The governing equation (2. 24) becomes

$$\frac{\partial}{\partial t}(H\overline{u}) + \frac{\partial}{\partial x}\left(H\overline{u}^2 + \frac{H^2}{2} - \frac{H^3}{3}\alpha\right) = 0 \qquad (2.26)$$

which is identical to the "correction" equation to the shallow-water theory derived by Su & Gardner (1969).

2.2 Scaling of the governing equations

Compared with corresponding 'Boussinesq' formulations, the governing equations (2.24)–(2.26) derived in the previous section include extra terms which are of higher order. However, for most practical problems, these terms are so small that they can be expected to have little or no effect on the resulting solutions. If appropriate scaling parameters are introduced to represent the respective order of magnitude of the problem's variables, these higher order terms can be identified and eliminated. Such a procedure not only gives insight into the relative importance of the various terms in the governing equations but also facilitates the computations of numerical solutions by greatly reducing the length of the equations that have to be programmed.

Two scaling factors are usually associated with water wave theories. The water depth to wavelength ratio is represented by σ while ϵ denotes the wave amplitude to water depth ratio:

$$\sigma = \frac{h_o}{\lambda},$$
$$\epsilon = \frac{a}{h_o}.$$

The relative size of these parameters is incorporated in the well known Ursell number

$$Ur = \frac{\epsilon}{\sigma^2} = \frac{a\lambda^2}{h_o^3}$$

which measures the importance of nonlinear to linear effects in wave propagation (Ursell 1953). Thus $Ur \ll 1$ for linear waves, while $Ur \gg 1$ for fully nonlinear (Airy) waves. Our interest lies in weakly nonlinear, dispersive (Boussinesq) waves for which $Ur \sim 1$. We therefore assume $\epsilon \sim \sigma^2$ in what follows.

For all types of long waves, since the variations of x and t are both small in terms of the wavelength, we can write

$$(x,t) = \frac{1}{\sigma}(x^*,t^*) \tag{2.27a}$$

where the asterisks denote non-dimensional, scaled variables. The relation (2.27a) implies that the change in the scaled variables is of $O(1)$ for a significant change in the wave. Furthermore, weakly nonlinear, dispersive, shallow-water waves also have a small amplitude to depth ratio. The appropriate scaling for η and $\overline{u}$ is then

$$(\eta,\overline{u}) = \epsilon(\eta^*,\overline{u}^*) \tag{2.27b}$$

which states that a small change in wave amplitude or in depth-averaged velocity becomes $O(1)$ in the scaled variables.

In the case of a moving bed, scaling factors can also be determined for the bed velocity h_t and slope h_x. Clearly, since $\eta = O(\epsilon)$ and $h = O(1)$, we have $H = O(1)$. Equation (2.27) can be substituted in the depth-averaged continuity equation (2.9) to give

$$\frac{\partial h}{\partial t} + \epsilon\left(\overline{u}^*\frac{\partial h}{\partial x} + \sigma h\frac{\partial \overline{u}^*}{\partial x^*}\right) + \epsilon^2\sigma\left(\overline{u}^*\frac{\partial \eta^*}{\partial x^*} + \eta^*\frac{\partial \overline{u}^*}{\partial x^*}\right) = 0.$$

For the above equation to be consistent, we will therefore impose

$$\frac{\partial h}{\partial t} = \epsilon\sigma(h_t)^* \tag{2.28}$$

and

$$\frac{\partial h}{\partial x} = \sigma(h_x)^* \tag{2.29}$$

which leads to the following non-dimensional, scaled depth-averaged continuity equation:

$$(h_t)^* + \frac{\partial \eta^*}{\partial t^*} + h\frac{\partial \overline{u}^*}{\partial x^*} + \overline{u}^*(h_x)^* + \epsilon\left(\overline{u}^*\frac{\partial \eta^*}{\partial x^*} + \eta^*\frac{\partial \overline{u}^*}{\partial x^*}\right) = 0. \tag{2.30}$$

With the help of the continuity equation (2.9), the momentum equation (2.24) can be rearranged as :

$$H\left(\frac{\partial \overline{u}}{\partial t} + \overline{u}\frac{\partial \overline{u}}{\partial x}\right) = H\frac{\partial H}{\partial x}(-1+\beta+\gamma) + \frac{H^2}{2}\frac{\partial}{\partial x}(\beta+\gamma)$$

$$+\left[\frac{\partial}{\partial x}\left(\frac{\eta^3}{3}+\frac{\eta^2 h}{2}-\frac{h^3}{6}\right)-\frac{\partial h}{\partial x}\left(\frac{\eta^2-h^2}{2}\right)\right]\alpha$$
$$+\left(\frac{\eta^3}{3}+\frac{\eta^2 h}{2}-\frac{h^3}{6}\right)\frac{\partial \alpha}{\partial x}+\frac{\partial h}{\partial x}[H(1-\beta-\gamma)]$$

which simplifies to

$$\frac{\partial \overline{u}}{\partial t}+\overline{u}\frac{\partial \overline{u}}{\partial x}+\frac{\partial \eta}{\partial x} = \frac{H}{2}\frac{\partial}{\partial x}(\beta+\gamma)+\frac{1}{H}\left(\frac{\eta^3}{3}+\frac{\eta^2 h}{2}-\frac{h^3}{6}\right)\frac{\partial \alpha}{\partial x} \qquad (2.31)$$
$$+\frac{\partial \eta}{\partial x}(\eta\alpha+\beta+\alpha).$$

The terms α, β and γ defined by (2.20) can be scaled as follows :

$$\alpha^* = (\epsilon\sigma^2)\overline{u}^*_{x^*t^*}+(\epsilon^2\sigma^2)(\overline{u}^*\overline{u}^*_{x^*x^*}-\overline{u}^{*2}_{x^*}) \qquad (2.32a)$$

$$\beta^* = (\epsilon\sigma^2)(h\overline{u}^*)_{x^*t^*}+(\epsilon^2\sigma^2)[\overline{u}^*(h\overline{u}^*)_{x^*x^*}-(h\overline{u}^*)_{x^*}\overline{u}^*_{x^*}] \qquad (2.32b)$$

$$\gamma^* = (\epsilon\sigma^2)(h_t)^*_{t^*}+(\epsilon^2\sigma^2)[\overline{u}^*(h_t)^*_{x^*}-(h_t)^*\overline{u}^*_{x^*}] \qquad (2.32c)$$

and substitution of (2.27) and (2.32) in equation (2.31) gives the following scaled momentum equation :

$$(\epsilon\sigma)\frac{\partial \overline{u}^*}{\partial t^*}+(\epsilon^2\sigma)\overline{u}^*\frac{\partial \overline{u}^*}{\partial x^*}+(\epsilon\sigma)\frac{\partial \eta^*}{\partial x^*}=\sigma\frac{(h+\epsilon\eta^*)}{2}\frac{\partial}{\partial x^*}[(\epsilon\sigma^2(h_t)^*_{t^*}+O(\epsilon^2\sigma^2)]$$
$$+\frac{\sigma}{H}\left(\epsilon^3\frac{\eta^{*3}}{3}+\epsilon^2\frac{\eta^{*2}h}{2}-\frac{h^3}{6}\right)\frac{\partial}{\partial x^*}[(\epsilon\sigma^2)\overline{u}^*_{x^*t^*}+O(\epsilon^2\sigma^2)]$$
$$+\epsilon\sigma\frac{\partial \eta^*}{\partial x^*}[\epsilon\eta^*(\epsilon\sigma^2\overline{u}^*_{x^*t^*})+O(\epsilon^2\sigma^2)]$$
$$+\epsilon\sigma\frac{\partial \eta^*}{\partial x^*}[(\epsilon\sigma^2)(h_t)^*_{t^*}+(\epsilon\sigma^2)(h\overline{u}^*)_{x^*t^*}+O(\epsilon^2\sigma^2)].$$

Keeping only the terms larger than $O(\epsilon^2\sigma^3)$, we obtain

$$(\epsilon\sigma)\frac{\partial \overline{u}^*}{\partial t^*}+(\epsilon^2\sigma)\overline{u}^*\frac{\partial \overline{u}^*}{\partial x^*}+(\epsilon\sigma)\frac{\partial \eta^*}{\partial x^*} = (\epsilon\sigma^3)\frac{h}{2}[(h_t)^*+(h\overline{u}^*)_{x^*}]_{t^*x^*}$$
$$-(\epsilon\sigma^3)\frac{h^2}{6}\overline{u}^*_{x^*t^*x^*}+O(\epsilon^2\sigma^3)$$

in which it was assumed that $\frac{h^3}{H}=h^2$. Finally, we can go back to the original non-dimensional variables to get

$$\overline{u}_t+\overline{u}\overline{u}_x+\eta_x = \frac{h}{2}[h_t+(h\overline{u})_x]_{tx}-\frac{h^2}{6}\overline{u}_{xtx}+O(\epsilon^2\sigma^3)$$

$$
\begin{aligned}
&= \frac{h^2}{3}\overline{u}_{xtx} + \frac{h}{2}h_{xx}\overline{u}_t + \frac{h}{2}h_x(\overline{u}_{xt} + \overline{u}_{tx}) \\
&+\frac{h}{2}h_{ttx} + \frac{h}{2}(h_{xt} + h_{tx})\overline{u}_x + \frac{h}{2}h_{xtx}\overline{u} \\
&+\frac{h}{2}h_t\overline{u}_{xx} + O(\epsilon^2\sigma^3).
\end{aligned} \tag{2.33}
$$

Equation (2.33) constitutes a 'low-order' representation of the 'complete' momentum equation (2.24). In fact, (2.33) together with the continuity equation (2.9) represent the Boussinesq formulation of the equations governing waves generated by a moving bed. The assumptions made in the derivation of (2.33) imply that the waves of interest are nonlinear, dispersive, shallow-water waves. Equation (2.9) and (2.33) therefore constitute a generalisation of the well-known Boussinesq system for waves over a flat bed.

It is important to note that, since x and t are both assumed to be of the same order, the operator $\frac{\partial^2}{\partial x \partial t}$ in equation (2.33) can be replaced by $\frac{\partial^2}{\partial t \partial x}$ which gives the following equation :

$$
\begin{aligned}
\overline{u}_t + \overline{u}\overline{u}_x + \eta_x &= \frac{h}{2}[h_t + (h\overline{u})_x]_{xt} - \frac{h^2}{6}\overline{u}_{xxt} + O(\epsilon^2\sigma^3) \\
&= \frac{h^2}{3}\overline{u}_{xxt} + \frac{h}{2}h_{xx}\overline{u}_t + hh_x\overline{u}_{xt} \\
&+\frac{h}{2}h_{txt} + hh_{xt}\overline{u}_x + \frac{h}{2}h_{xxt}\overline{u} \\
&+\frac{h}{2}h_t\overline{u}_{xx} + O(\epsilon^2\sigma^3).
\end{aligned} \tag{2.34}
$$

The order of appearance of the x and t partial derivatives only depends on the procedure used to obtain the final governing equation. In the derivation of (2.33), we assumed $u(x, y, t) \simeq \overline{u}(x, t)$ and depth-averaged the resulting x-momentum equation (2.13). All these terms of equation (2.24) (or equation 2.33) which would not appear in the standard Airy equation and which represent the deviation of the pressure from the hydrostatic state thus came from the depth-integration of the horizontal pressure gradient $\frac{\partial p}{\partial x}$.

However a different procedure can be used. We can start by stating that, to an order $O(\epsilon\sigma)$, the vertical velocity $v(x, y, t)$ is given exactly by the expression on the

right hand side of approximation (2.12), i.e.

$$v(x,y,t) = -\frac{\partial h}{\partial t} - \frac{\partial}{\partial x}(h\overline{u}) - y\frac{\partial \overline{u}}{\partial x} + O(\epsilon^2\sigma). \tag{2.35}$$

Then from the irrotationality condition (2.4) (which was not used in the derivation of (2.33)) we can deduce (Appendix A):

$$u = \overline{u} - \left(y + \frac{h}{2}\right)[h_t + (h\overline{u})_x]_x - \left(\frac{y^2}{2} + \frac{h^2}{6}\right)\overline{u}_{xx} + O(\epsilon^2\sigma^2). \tag{2.36}$$

Furthermore, the expression (2.21) for the pressure derived earlier can be written as

$$p = (\eta - y) + y[h_t + (h\overline{u})_x]_t + \frac{y^2}{2}\overline{u}_{xt} + O(\epsilon^2\sigma^2). \tag{2.37}$$

Substitution of (2.35)–(2.37) in the x-momentum equation (2.2) gives

$$\begin{aligned}\overline{u}_t + \overline{u}\overline{u}_x + \eta_x \;=\; & \left(y + \frac{h}{2}\right)[h_t + (h\overline{u})_x]_{xt} + \left(\frac{y^2}{2} + \frac{h^2}{6}\right)\overline{u}_{xxt} \\ & -y[h_t + (h\overline{u})_x]_{tx} - \frac{y^2}{2}\overline{u}_{xtx} + O(\epsilon^2\sigma^3)\end{aligned}$$

which implies that $\frac{\partial^2}{\partial t \partial x} = \frac{\partial^2}{\partial x \partial t}$ and leads to equation (2.33) or (2.34). The above procedure is incorporated more systematically in a perturbation approach of the problem in which the basic variables are expanded in terms of the small parameters ϵ and σ. Wu (1981) used such a procedure to obtain an equation identical to (2.34) Appendix B shows a derivation of (2.34) using an expansion method.

Since the dispersive term usually appears as $\overline{u}_{xxt}$ in the Boussinesq system, we will henceforth used the formulation (2.34). It is worth noticing that, altough terms of $O(\epsilon^2\sigma^3)$ were left out from (2.34), the last term $\frac{h}{2}h_t\overline{u}_{xx}$ is also $O(\epsilon^2\sigma^3)$ according to the scales introduced at the beginning of this section. That contradiction originates in the term $(h\overline{u})_{xxt}$ which is $O(\epsilon\sigma^3)$ but gives rise to $\frac{h}{2}h_t\overline{u}_{xx}$, a term of $O(\epsilon^2\sigma^3)$, when the partial differentiation is carried out. Therefore the presence of a term of $O(\epsilon^2\sigma^3)$ in the final equation is only due to a slight scaling inconsistency and for that reason, the term $\frac{h}{2}h_t\overline{u}_{xx}$ is not eliminated from the final equation.

As it was done for the 'high-order' equation (2.24), we can reduce the governing equation for the cases of nonuniform and flat beds. If the bed is sloping but not

moving, equation (2.34) reduces to

$$\begin{aligned} \overline{u}_t + \overline{u}\overline{u}_x + \eta_x &= \frac{h}{2}[(h\overline{u})_x]_{xt} - \frac{h^2}{6}\overline{u}_{xxt} + O(\epsilon^2\sigma^3) \\ &= \frac{h^2}{3}\overline{u}_{xxt} + \frac{h}{2}h_{xx}\overline{u}_t + hh_x\overline{u}_{xt} + O(\epsilon^2\sigma^3) \end{aligned} \tag{2.38}$$

as derived by Peregrine (1967) in the study of long waves on beaches. If we also restrict the bed to a zero slope, we obtain the basic Boussinesq equation:

$$\overline{u}_t + \overline{u}\overline{u}_x + \eta_x = \frac{h^2}{3}\overline{u}_{xxt} + O(\epsilon^2\sigma^3). \tag{2.39}$$

In the next section, equation (2.34) together with the continuity equation (2.9) will be used to model problems of waves generated by a moving bed.

3 NUMERICAL SOLUTIONS

Because of the nonlinear and dispersive terms appearing in the governing equations obtained in §2.2, it is extremely difficult (if not impossible) in general to derive analytical solutions to those equations. For that reason, we have to resort to a numerical method to obtain approximate solutions for the amplitude η and velocity $\overline{u}$ appearing in the depth-averaged continuity (2.9) and momentum (2.34) equations

3.1 The finite difference method

The finite difference method is one of the oldest numerical methods used in the solution of differential equations. Because of its simplicity and reliability, it still constitutes a very widespread tool for the numerical solution of fluid flow problems For the sake of completeness, a brief elementary review of various aspects of the finite difference method follows.

The basis of the finite difference method consists in replacing a function defined over a continuous solution domain by approximations of that function evaluated at discrete locations of the solution domain called grid points. These approximations may then be used to express the derivatives of the function in terms of algebraic equations called finite difference equations. The important consequence of the finite difference method is therefore the possibility to transform one or more differential equations into an equivalent algebraic problem.

In the present problem, we are concerned with the approximation of dependent flow variables such as η, $\overline{u}$ and h which are functions of spatial position x and time t. Thus, we may consider an arbitrary function $f(x,t)$ which is assumed to be the solution of a boundary value problem defined over a domain $0 \leq x \leq x_N$ and $0 \leq t \leq t_P$. The starting point of the finite difference method is to replace this continuous solution domain by a discrete set of points denoted by (x_i, t_j) such that $0 \leq x_i \leq x_N$ and $0 \leq t_j \leq t_P$ where $i = 0, 1, \ldots, N$ and $j = 0, 1, \ldots, P$ (Figure 3.1). The solution domain thus takes the appearance of a grid and for that reason the points (x_i, t_j) are

called grid points. The mesh size in the t-direction is constant and denoted by Δt so that $t_P = P\Delta t$. The spatial mesh size is denoted by Δx_i, a distance which may vary with time. The finite difference approximation of $f(x,t)$ evaluated at the grid point (i,j) is denoted by f_i^j.

3.1.1 Eulerian description

Most of the finite difference models used in fluid mechanics and hydraulics involve an Eulerian description of the flow field. That implies that the flow variables are calculated at fixed locations in the solution domain and therefore that the grid points spacing in the x-direction does not change with time. We will only consider the case where the spatial mesh size is constant, i. e. $\Delta x_i = \Delta x$ such that $x_N = N\Delta x$ (Figure 3.1)

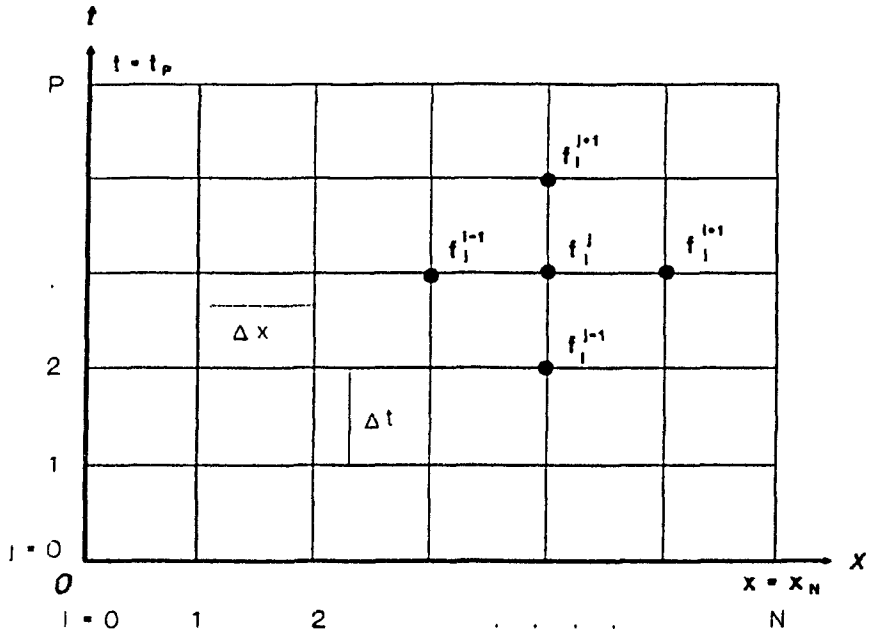


Figure 3.1: The Eulerian finite difference problem.

At any given time, if $f(x,t)$ has continuous spatial derivatives, it can be approximated in the neighborhood of $x = x_i$ by the following Taylor series:

$$f(x,t) = f_i + (x - x_i)\frac{\partial f}{\partial x_i} + \frac{(x - x_i)^2}{2}\frac{\partial^2 f}{\partial x_i^2} + \frac{(x - x_i)^3}{3!}\frac{\partial^3 f}{\partial x_i^3} + \cdots$$

where f_i denotes the finite difference approximation of $f(x_i, t)$. Using this Taylor series, we can write the following approximation for $f(x_{i+1}, t)$:

$$f_{i+1} = f_i + \Delta x \frac{\partial f}{\partial x_i} + \frac{\Delta x^2}{2} \frac{\partial^2 f}{\partial x_i^2} + \frac{\Delta x^3}{3!} \frac{\partial^3 f}{\partial x_i^3} + O(\Delta x^4) \tag{3.1}$$

The series (3.1) directly gives a finite difference approximation for $\frac{\partial f}{\partial x}$ evaluated at $x = x_i$:

$$\frac{\partial f}{\partial x_i} = \frac{f_{i+1} - f_i}{\Delta x} + O(\Delta x) \tag{3.2}$$

In a similar way, we can approximate $f(x_{i-1}, t)$ by the series

$$f_{i-1} = f_i - \Delta x \frac{\partial f}{\partial x_i} + \frac{\Delta x^2}{2} \frac{\partial^2 f}{\partial x_i^2} - \frac{\Delta x^3}{3!} \frac{\partial^3 f}{\partial x_i^3} + O(\Delta x^4) \tag{3.3}$$

from which we obtain

$$\frac{\partial f}{\partial x_i} = \frac{f_i - f_{i-1}}{\Delta x} + O(\Delta x) \tag{3.4}$$

Equations (3.2) and (3.4) are respectively called forward and backward difference equations. In both cases the approximation involves an error term of $O(\Delta x)$. The magnitude of this error term is reduced if we subtract (3.3) from (3.1) to obtain

$$\frac{\partial f}{\partial x_i} = \frac{f_{i+1} - f_{i-1}}{2\Delta x} + O(\Delta x^2) \tag{3.5}$$

which is a central difference representation of $\frac{\partial f}{\partial x}$. A finite difference approximation for $\frac{\partial^2 f}{\partial x^2}$ at $x = x_i$ can be obtained by addition of (3.1) and (3.3) :

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} + O(\Delta x^2) \tag{3.6}$$

We follow a similar procedure to derive finite difference approximations for the time derivatives of $f(x, t)$. At any given location in space, approximations for $f(x, t_{j+1})$ and $f(x, t_j)$ in the neighborhood of $t = t_{j+\frac{1}{2}}$ can be written as

$$f^{j+1} = f^{j+\frac{1}{2}} + (\Delta t/2) \frac{\partial f}{\partial t_{j+\frac{1}{2}}} + \frac{(\Delta t/2)^2}{2} \frac{\partial^2 f}{\partial t_{j+\frac{1}{2}}^2} + O(\Delta t^3) \tag{3.7}$$

and

$$f^j = f^{j+\frac{1}{2}} - (\Delta t/2) \frac{\partial f}{\partial t_{j+\frac{1}{2}}} + \frac{(\Delta t/2)^2}{2} \frac{\partial^2 f}{\partial t_{j+\frac{1}{2}}^2} + O(\Delta t^3) \tag{3.8}$$

Subtraction of (3.8) from (3 7) gives an approximation of $\frac{\partial f}{\partial t}$ evaluated at $t = t_{j+\frac{1}{2}}$:

$$\frac{\partial f}{\partial t_{j+\frac{1}{2}}} = \frac{f^{j+1} - f^j}{\Delta t} + O(\Delta t^2) \tag{3.9}$$

and addition of the two series gives :

$$\frac{\partial^2 f}{\partial t^2_{j+\frac{1}{2}}} = \frac{f^{j+1} - 2f^{j+\frac{1}{2}} + f^j}{(\Delta t/2)^2} + O(\Delta t^2) \tag{3.10}$$

3.1.2 Lagrangian description

In the Lagrangian description of the flow field, the flow variables are not evaluated in a fixed reference frame but rather at the instantaneous locations of the moving fluid particles. In the finite difference method, that implies that the grid moves with the fluid particles and therefore that the spatial mesh size varies with time, i.e. $\Delta x_i = \Delta x_i(t) = \frac{1}{2}(x_{i+1} - x_{i-1})$ for $i = 1, 2, \ldots, N-1$. Depending on the motion of the fluid, the space-time grid can then take the distorted appearance of figure 3.2.

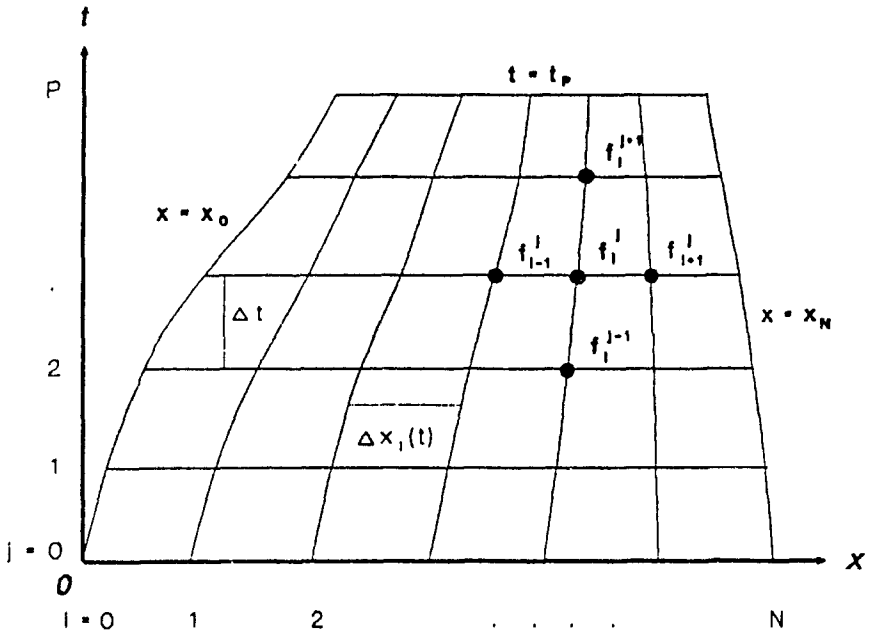


Figure 3.2: The Lagrangian finite difference problem.

If at in given time we let $\Delta x = x_{i+1} - x_i$ and $\Delta x' = x_i - x_{i-1}$ (Figure 3.3), we can approximate $f(x_{i+1}, t)$ and $f(x_{i-1}, t)$ by power series about $x = x_i$:

$$f_{i+1} = f_i + \Delta x \frac{\partial f}{\partial x_i} + \frac{\Delta x^2}{2} \frac{\partial^2 f}{\partial x_i^2} + O(\Delta x^3) \tag{3.11}$$

$$f_{i-1} = f_i + \Delta x' \frac{\partial f}{\partial x_i} + \frac{\Delta x'^2}{2} \frac{\partial^2 f}{\partial x_i^2} + O(\Delta x'^3) \tag{3.12}$$

Subtraction of (3.12) from (3.11) gives

$$\begin{aligned} f_{i+1} - f_{i-1} &= (\Delta x + \Delta x') \frac{\partial f}{\partial x_i} + \frac{1}{2}(\Delta x^2 - \Delta x'^2) \frac{\partial^2 f}{\partial x_i^2} + O[\max(\Delta x^3, \Delta x'^3)] \\ &= 2\Delta x_i \frac{\partial f}{\partial x_i} + \frac{1}{2}(2\delta x)(2\Delta x_i) \frac{\partial^2 f}{\partial x_i^2} + O[\max(\Delta x^3, \Delta x'^3)] \end{aligned}$$

where $\delta x = \frac{1}{2}(\Delta x - \Delta x')$ (Figure 3.3). The central difference formula for the first

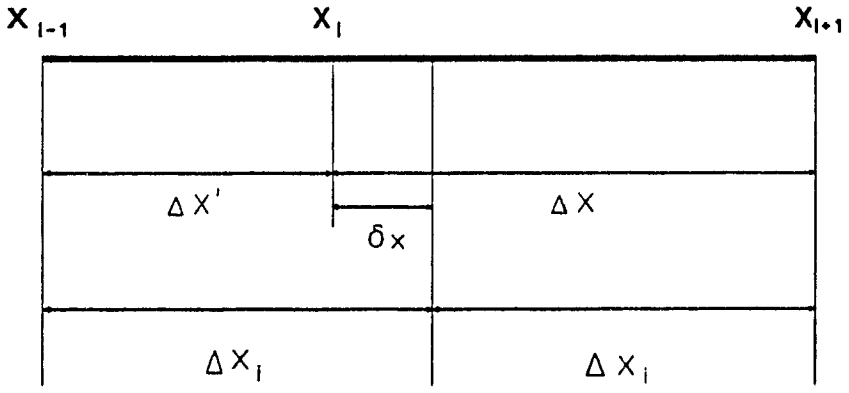


Figure 3.3: The Lagrangian grid.

order spatial derivative at $x = x_i$ is then

$$\frac{\partial f}{\partial x_i} = \frac{f_{i+1} - f_{i-1}}{2\Delta x_i} + O[\max(\delta x, \Delta x^2, \Delta x'^2)] \tag{3.13}$$

The error term in (3.13) is a function of the grid deformation. For a fixed grid of uniform mesh size, $\delta x = 0$, $\Delta x' = \Delta x$ and (3.13) is identical to (3.5). However as the distortion of the grid increases, the approximation (3.13) increasingly looses accuracy (Boris 1981). Addition of (3.11) and (3.12) gives

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{f_{i+1} - 2f_i + f_{i-1}}{\frac{1}{2}(\Delta x^2 + \Delta x'^2)} + \cdots \tag{3.14}$$

which is also subject to an error term depending on the grid deformation.

For the time derivatives, a procedure similar to the one used in the Eulerian case is followed. The important difference is that the resulting derivatives are not evaluated locally but rather following the fluid. They are total (or material) derivatives and are expressed as :

$$\frac{df_i}{dt_{j+\frac{1}{2}}} = \frac{f_i^{j+1} - f_i^j}{\Delta t} + O(\Delta t^2) \tag{3.15}$$

and

$$\frac{d^2 f_i}{dt_{j+\frac{1}{2}}} = \frac{f_i^{j+1} - 2f_i^{j+\frac{1}{2}} + f_i^j}{(\Delta t/2)} + O(\Delta t^2) \tag{3.16}$$

3.1.3 Explicit and implicit formulations

When the finite difference method is applied to time-dependent problems, the resulting algebraic equations may either have an explicit or an implicit solution. Therefore a finite difference scheme can be classified as explicit or implicit according to the algebraic problem to which it leads.

The explicit form may be represented schematically by figure 3.4 in which the dots and crosses indicate grid points needed in the computations of spatial and temporal derivatives respectively. In that type of scheme, the calculation of the solution at the 'new' time is only based on the solution at the previous time.

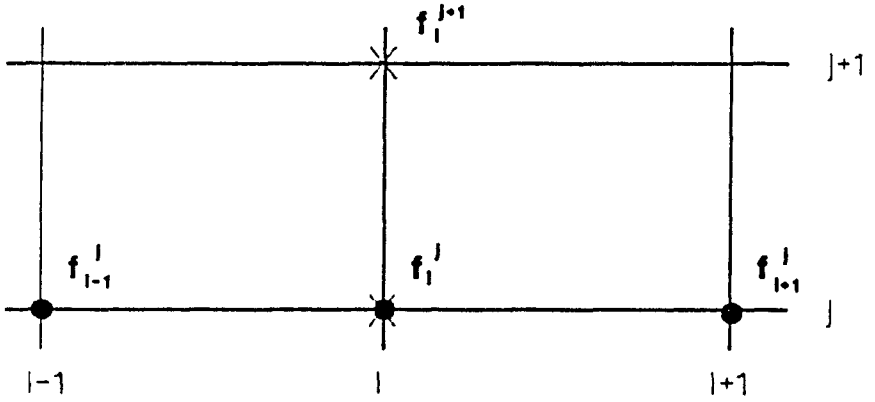


Figure 3.4: The explicit form.

In an implicit scheme, the unknown solution at the new time not only depends on the solution at the previous time but also on values of the function at this new time. This type of scheme leads to a system of equations which must be solved simultaneously. Figure 3.5 represents a fully implicit scheme in which all the derivatives are

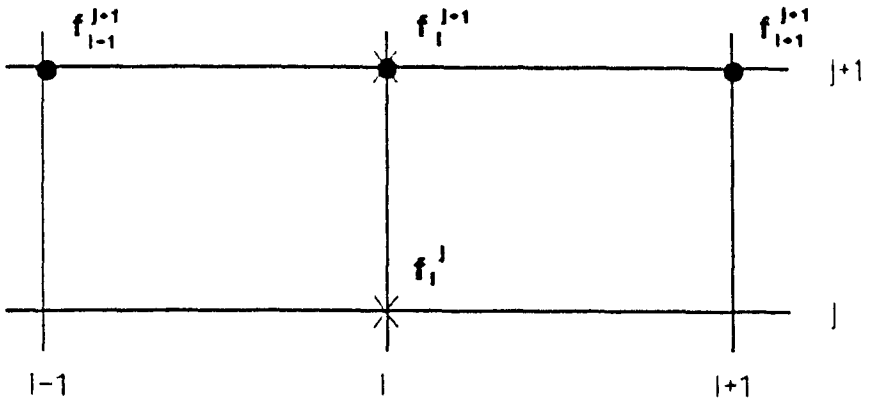


Figure 3.5: The fully implicit form.

evaluated at the advanced time. Figure 3.6 shows an implicit scheme proposed by Crank and Nicolson in 1947. The derivatives are now evaluated at the intermediate time and calculated by averaging values at the previous and advanced times.

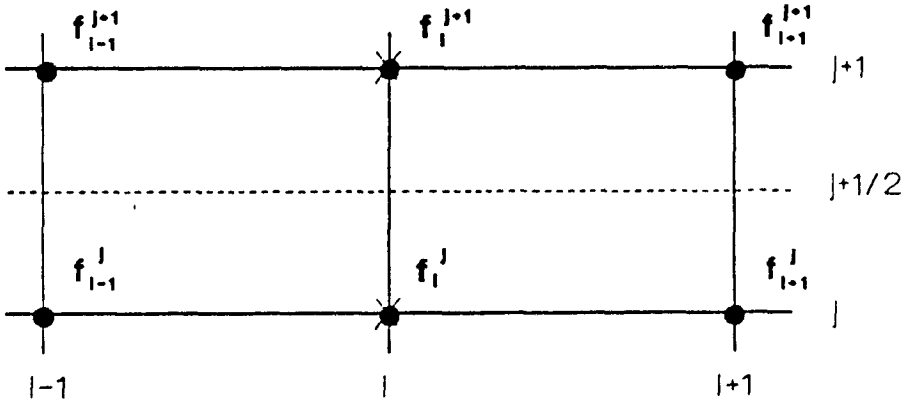


Figure 3.6: The Crank-Nicolson form.

Explicit schemes have the advantage of being simple to formulate but they are subject to stability conditions which impose certain limitations on the mesh size. On

the other hand, implicit schemes can be shown to be unconditionally stable but their formulation is often complex and requires longer computation times.

In the next section the equations derived in §2.2 will be formulated in terms of explicit and implicit finite difference methods.

3.2 Finite difference formulation of the governing equations

We will now use the basic principles developed in §3.1 to obtain finite difference formulations for the governing continuity and momentum equations.

3.2.1 Choice of the flow field description

The first question which must be answered in the application of the finite difference method to the governing equations (2.9) and (2.34) is whether to use the Eulerian or Lagrangian specification of the flow field.

Equations (2.9) and (2.34) were derived from the fundamental Eulerian equations of motion (2.1 to 2.3). It would therefore appear natural to use the Eulerian finite difference method introduced in §3.1.1. As already pointed out, that method implies that the spatial grid points are not associated with the moving fluid particles (Lagrangian specification) but rather with evenly spaced fixed locations in the flow field.

However, in some moving bed problems, the water surface may intersect the moving segment of the bed at a point of zero depth. In such problems, the upstream boundary of the flow field is constituted by a moving waterline. The 'moving wedge' problem mentioned in §1 and presented in §4.1 is an example of such a case. The major disadvantage implied in the Eulerian treatment of these problems is that the number of grid points involved in the description of the free surface may constantly vary. That creates difficulties in the computer coding of the scheme and almost invariably requires interpolation or extrapolation procedures. The above remarks could also be made about the Eulerian treatment of wave run-up on beaches in which

various methods must be used to overcome the difficulties associated with the shoreline boundary (see for example Hibberd & Peregrine 1979).

The obvious alternative in problems involving a moving waterline is to use a Lagrangian description of the flow field. In that way, the location of that point of zero depth is easily followed since it always corresponds to one of the moving grid points.

We will now describe both the Eulerian and the Lagrangian finite difference formulations of the governing equations (2.9) and (2.34).

3.2.2 Eulerian scheme

The functions η, $\overline{u}$ and h are all specified on the same spatial grid points. Figure 3.7 shows the arrangment of the flow field at any given time. Since we are only dealing with equations in one spatial dimension, the grid points actually correspond to vertical interfaces separating fluid elements bounded by the bed and the free surface

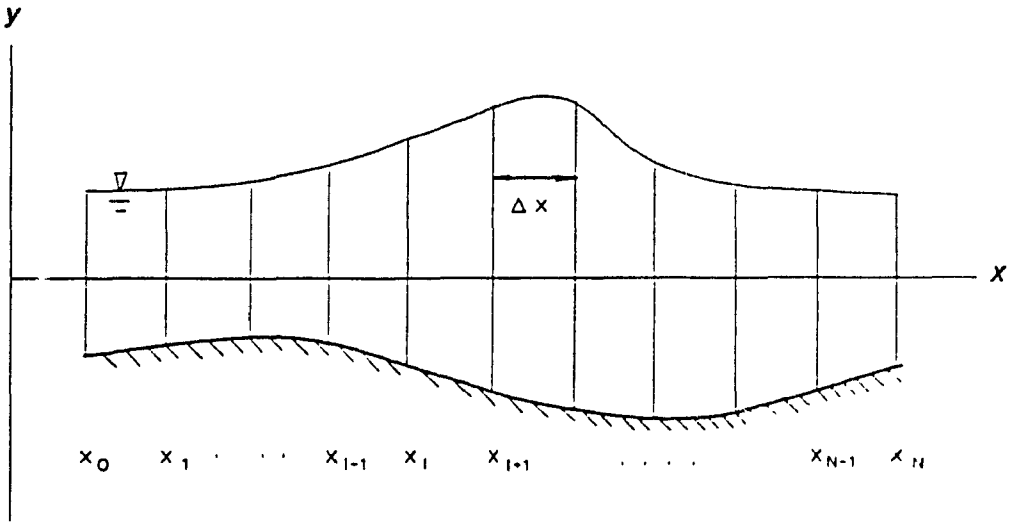


Figure 3.7: Eulerian description of the flow field.

The proposed numerical scheme is divided in three main parts :

1. Explicit solution of the depth-averaged continuity equation to obtain the

predicted free surface position.

2. Implicit (Crank-Nicolson) solution of the momentum equation (2.34) to obtain the depth-averaged velocity.

3. Explicit solution of the depth-averaged continuity equation to obtain the *corrected* free surface position.

As a prerequisite to the application of the above scheme, we assume that initial conditions at $t = 0$ and boundary conditions at $x = x_0$ and $x = x_N$ are known for the free surface position η and the depth-averaged fluid velocity $\overline{u}$. Furthermore, we assume that the bed variation $h(x,t)$ is known everywhere in the solution domain.

We now look more closely into each of the above 3 steps :

- *Explicit calculation of the predicted free surface position* $\tilde{\eta}_i^{j+1}$.

The depth-averaged continuity equation (2.9) can be written as

$$\frac{\partial \eta}{\partial t} = -(h+\eta)\frac{\partial \overline{u}}{\partial x} - \overline{u}\frac{\partial \eta}{\partial x} - \overline{u}\frac{\partial h}{\partial x} - \frac{\partial h}{\partial t} \tag{3.17}$$

Using the finite-difference formulae developed in §3.1.1, we approximate (3.17) as :

$$\begin{aligned} \tilde{\eta}_i^{j+1} = \eta_i^j \; &- \; \Delta t\left[\frac{1}{2}(h_i^{j+1} + h_i^j) + \eta_i^j\right]\frac{\overline{u}_{i+1}^j - \overline{u}_{i-1}^j}{2\Delta x} \\ &- \; \Delta t\overline{u}_i^j\left(\frac{\eta_{i+1}^j - \eta_{i-1}^j}{2\Delta x}\right) \\ &- \; \Delta t\overline{u}_i^j\left(\frac{h_{i+1}^{j+1} - h_{i-1}^{j+1} + h_{i+1}^j - h_{i-1}^j}{4\Delta x}\right) \\ &- \; (h_i^{j+1} - h_i^j). \end{aligned} \tag{3.18}$$

For known values of η and $\overline{u}$ at time $t = t_j$, equation (3.18) can be used to calculate explicitly predicted values of the free surface position at time $t = t_{j+1}$ denoted by $\tilde{\eta}_i^{j+1}$ where $i = 1, 2, \ldots, N-1$.

- *Implicit calculation of the depth-averaged velocity* $\overline{u}_i^{j+1}$.

With the help of the finite-difference formulae introduced in §3.1.1 and the Crank-Nicolson scheme described in §3.1.3, the depth-averaged momentum equation (2.34) can be written as :

$$\mathrm{v1}_i\overline{u}_{i-1}^{j+1} + \mathrm{v2}_i\overline{u}_i^{j+1} + \mathrm{v3}_i\overline{u}_{i+1}^{j+1} = \mathrm{rhs}_i \tag{3.19}$$

where

$$\begin{aligned}
\mathrm{v1}_i &= -\frac{\overline{u}_i^j}{4\Delta x} - A + B + \frac{h_i^{j+1}}{4\Delta x}C - \frac{h_i^{j+1}}{2\Delta x^2}\overline{u}_i^j m_i^{j+1} - \frac{h_i^{j+1}}{4\Delta x^2}F \\
\mathrm{v2}_i &= \frac{1}{\Delta t} + 2A - \frac{D}{\Delta t} - \frac{h_i^{j+1}}{4}E + \frac{h_i^{j+1}}{\Delta x^2}\overline{u}_i^j m_i^{j+1} + \frac{h_i^{j+1}}{2\Delta x^2}F \\
\mathrm{v3}_i &= \frac{\overline{u}_i^j}{4\Delta x} - A - B - \frac{h_i^{j+1}}{4\Delta x}C - \frac{h_i^{j+1}}{2\Delta x^2}\overline{u}_i^j m_i^{j+1} - \frac{h_i^{j+1}}{4\Delta x^2}F
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{rhs}_i = \quad & \overline{u}_{i-1}^j\left(\frac{\overline{u}_i^j}{4\Delta x} - A + B - \frac{h_i^j}{4\Delta x}C + \frac{h_i^j}{2\Delta x^2}\overline{u}_i^j m_i^j + \frac{h_i^j}{4\Delta x^2}F\right) \\
+ \quad & \overline{u}_i^j\left(\frac{1}{\Delta t} + 2A - \frac{D}{\Delta t} - \frac{h_i^j}{4}E - \frac{h_i^j}{\Delta x^2}\overline{u}_i^j m_i^j - \frac{h_i^j}{2\Delta x^2}F\right) \\
+ \quad & \overline{u}_{i+1}^j\left(-\frac{\overline{u}_i^j}{4\Delta x} - A - B + \frac{h_i^j}{4\Delta x}C + \frac{h_i^j}{2\Delta x^2}\overline{u}_i^j m_i^j + \frac{h_i^j}{4\Delta x^2}F\right) \\
- \quad & F + [(h_i^{j+1} + h_i^j)(r_{i+1}^j - r_{i-1}^j)]/(8\Delta x^2)
\end{aligned}$$

Also

$$\begin{aligned}
A &= \frac{1}{6}[(h_i^j)^2 + (h_i^{j+1})^2)] \\
B &= \frac{1}{4\Delta x}[h_i^j(h_{i+1}^j - h_{i-1}^j) + h_i^{j+1}(h_{i+1}^{j+1} - h_{i-1}^{j+1})] \\
C &= \frac{1}{2\Delta x\Delta t}(h_{i+1}^{j+1} - h_{i-1}^{j+1} - h_{i+1}^j + h_{i-1}^j) \\
D &= \frac{1}{4\Delta x^2}[h_i^j(h_{i+1}^j - 2h_i^j + h_{i-1}^j) + h_i^{j+1}(h_{i+1}^{j+1} - 2h_i^{j+1} + h_{i-1}^{j+1})] \\
E &= \frac{1}{\Delta x^2\Delta t}[(h_{i+1}^{j+1} - 2h_i^{j+1} + h_{i-1}^{j+1}) - (h_{i+1}^j - 2h_i^j + h_{i-1}^j)] \\
F &= \frac{1}{4\Delta x}(\eta_{i+1}^{j+1} - \eta_{i-1}^{j+1} + \eta_{i+1}^j - \eta_{i-1}^j) \\
m_i^j &= \frac{1}{2\Delta x}(h_{i+1}^j - h_{i-1}^j) \\
r_i^j &= \frac{1}{(\Delta t/2)^2}(h_i^{j+1} - 2h_i^{j+\frac{1}{2}} + h_i^j)
\end{aligned}$$

The only unknown in (3.19) is the velocity field $\overline{u}_i^{j+1}$ for $i = 1, 2, \ldots, N-1$ which obviously cannot be calculated explicitly. By rewriting (3.19) in matrix form as

$$V\overline{U}^{j+1} = \text{RHS}$$

or

$$\begin{pmatrix} \text{v}2_1 & \text{v}3_1 & 0 & \cdots & 0 \\ \text{v}1_2 & \text{v}2_2 & \text{v}3_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \text{v}1_{N-2} & \text{v}2_{N-2} & \text{v}3_{N-2} \\ 0 & \cdots & 0 & \text{v}1_{N-1} & \text{v}2_{N-1} \end{pmatrix} \begin{pmatrix} \overline{u}_1^{j+1} \\ \overline{u}_2^{j+1} \\ \vdots \\ \overline{u}_{N-2}^{j+1} \\ \overline{u}_{N-1}^{j+1} \end{pmatrix} = \begin{pmatrix} \text{rhs}_1 - \text{v}1_1\overline{u}_0^{j+1} \\ \text{rhs}_2 \\ \vdots \\ \text{rhs}_{N-2} \\ \text{rhs}_{N-1} - \text{v}3_{N-1}\overline{u}_N^{j+1} \end{pmatrix},$$

we see that this velocity field is the solution of a system of linear non-homogeneous equations having a tridiagonal coefficient matrix. That type of system can be solved by a straightforward elimination procedure.

- *Explicit calculation of the corrected free surface position* η_i^{j+1}.

In this last step, we calculate a corrected value η_i^{j+1} for the free surface position by using the velocity field $\overline{u}_i^{j+1}$ calculated in the second step. Equation (2.9) is solved explicitly as

$$\begin{aligned} \eta_i^{j+1} = \eta_i^j &- \Delta t \left[\frac{1}{2}(h_i^{j+1} + h_i^j) + \eta_i^j\right] \frac{\overline{u}_{i+1}^{j+1} - \overline{u}_{i-1}^{j+1} + \overline{u}_{i+1}^j - \overline{u}_{i-1}^j}{4\Delta x} \\ &- \Delta t \left(\frac{\overline{u}_i^{j+1} + \overline{u}_i^j}{2}\right) \frac{\eta_{i+1}^j - \eta_{i-1}^j}{2\Delta x} \\ &- \Delta t \left(\frac{\overline{u}_i^{j+1} + \overline{u}_i^j}{2}\right) \frac{h_{i+1}^{j+1} - h_{i-1}^{j+1} + h_{i+1}^j - h_{i-1}^j}{4\Delta x} \\ &- (h_i^{j+1} - h_i^j). \end{aligned} \tag{3.20}$$

3.2.3 Lagrangian scheme

Figure (3.7) shows a Lagrangian configuration of the flow field. The important difference with respect to the Eulerian representation of Figure (3.6) is that the fluid interfaces on which $\eta, \overline{u}$ and h are specified are now allowed to move.

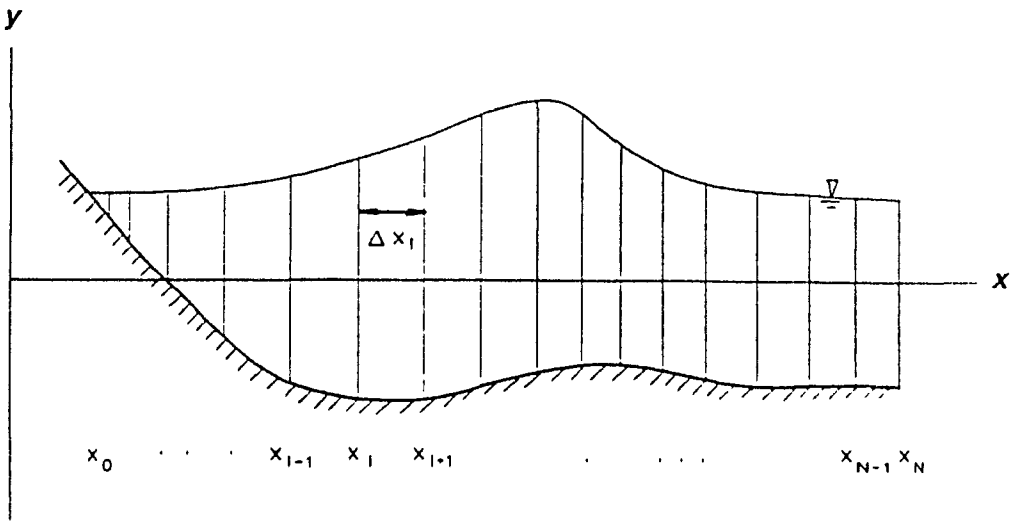


Figure 3.8: Lagrangian description of the flow field.

As mentioned before, the depth-averaged continuity and momentum equations (2.9) and (2.34) constitute Eulerian formulations. By introducing the material derivative (or 'derivative following the fluid')

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \overline{u}\frac{\partial}{\partial x}$$

and applying it to the free surface position η

$$\frac{d\eta}{dt} = \frac{\partial \eta}{\partial t} + \overline{u}\frac{\partial \eta}{\partial x} \tag{3.21a}$$

and to the depth-averaged velocity $\overline{u}$

$$\frac{d\overline{u}}{dt} = \frac{\partial \overline{u}}{\partial t} + \overline{u}\frac{\partial \overline{u}}{\partial x}, \tag{3.21b}$$

we can rewrite (2.9) and (2.34) as :

$$\frac{d\eta}{dt} = -(h+\eta)\overline{u}_x - \frac{dh}{dt} \tag{3.22}$$

$$\begin{aligned} \frac{d\overline{u}}{dt} &= -\eta_x + \frac{h^2}{3}\frac{d\overline{u}_{xx}}{dt} \\ &\quad + hh_x\frac{d\overline{u}_x}{dt} + \frac{h}{2}\frac{d^2h_x}{dt^2} + h\overline{u}_x\frac{dh_x}{dt} + \frac{h}{2}\overline{u}_{xx}\frac{dh}{dt} \end{aligned} \tag{3.23}$$

Equation (3.22) is exact while (3.23) is valid up to order $O(\epsilon^2\sigma^3)$. For waves over a flat bed, equation (3.22) and (3.23) reduce to :

$$\frac{d\eta}{dt} = -(h+\eta)\overline{u}_x \tag{3.24}$$

$$\frac{d\overline{u}}{dt} = -\eta_x + \frac{h^2}{3}\frac{d\overline{u}_{xx}}{dt} \tag{3.25}$$

Taking the material derivative of x, we also obtain

$$\frac{dx}{dt} = \overline{u} \tag{3.26}$$

which can be used to define the position interfaces x_i $(i = 0, 1, \ldots, N)$ arising from the Lagrangian finite difference description of the flow field.

For the finite difference approximation of (3.22) to (3.25), a three step procedure very similar to the one described in §3.2.2 is used. The only modifications introduced by the use of a moving grid are the replacement of the constant Δx by a variable mesh size Δx_i and the displacement of the computation grid calculated from the finite difference approximation of (3.26):

$$x_i^{j+1} = x_i^j + \Delta t[\theta\overline{u}_i^{j+1} + (1-\theta)\overline{u}_i^j] \tag{3.27}$$

where $0 \leq \theta \leq 1$.

3.3 Initial and boundary conditions

As mentioned before, the numerical solution of the governing equations requires the knowledge of initial and boundary conditions for η and $\overline{u}$. Some of these conditions are obvious and arise directly from the physics of the problem while others must be derived indirectly by using the governing equations.

3.3.1 Initial conditions

In all the problems to be treated in §4, the initial conditions correspond to still water. Thus

$$\eta(x, 0) = 0 \tag{3.28}$$

and

$$\overline{u}(x,0) = 0 \tag{3.29}$$

for all x. In finite difference form, the conditions (3.28) and (3.29) become

$$\eta_i^0 = 0 \tag{3.30}$$

$$\overline{u}_i^0 = 0 \tag{3.31}$$

for $i = 0, 1, 2, \ldots, N$.

3.3.2 Boundary conditions

In problems where the flow field is bounded by a vertical wall at $x = x_0$, we have the following boundary condition for the depth-averaged fluid velocity

$$\overline{u}(x_0, t) = \overline{u}_w(t) \tag{3.32}$$

where $\overline{u}_w(t)$ is the horizontal velocity of the wall. In finite difference, we have

$$\overline{u}_0^j = \overline{u}_w(t_j). \tag{3.33}$$

If the wall is not moving, $\overline{u}_w = 0$ and

$$\overline{u}_0^j = 0. \tag{3.34}$$

In all the cases to be treated in §4, the flow field is bounded at all times by a fixed vertical wall at the downstream end $x = x_N$ such that

$$\overline{u}_N^j = 0 \tag{3.35}$$

for $j = 0, 1, 2, \ldots, P$.

The free surface position η at $x = x_0$ and $x = x_N$ is unknown for $t > 0$. Therefore, unlike the case of the velocity, simple Dirichlet boundary conditions cannot be deduced directly. However, by combining the velocity boundary conditions with the depth-averaged momentum equation (2.34), we can derive boundary conditions of

the Neumann type for η. For example if $\overline{u}(x_0,t) = 0$, equation (2.34) evaluated at $x = x_0$ becomes

$$\eta_x(x_0,t) = \left[hh_{xt}\overline{u}_x + \frac{h}{2}h_t\overline{u}_{xx} + \frac{h}{2}h_{txt}\right]_{x=x_0}. \tag{3.36}$$

Equation (3.36) can be further simplified in cases where the bed acceleration at $x = x_0$ is equal to zero :

$$\eta_x(x_0,t) = \left[hh_{xt}\overline{u}_x + \frac{h}{2}h_t\overline{u}_{xx}\right]_{x=x_0}. \tag{3.37}$$

Furthermore, if the bed slope is constant at $x = x_0$, we have

$$\eta_x(x_0,t) = \left[\frac{h}{2}h_t\overline{u}_{xx}\right]_{x=x_0} \tag{3.38}$$

and finally, for a non moving bed at $x = x_0$:

$$\eta_x(x_0,t) = 0. \tag{3.39}$$

If however, $\overline{u}(x_0,t) = \overline{u}_w(t)$, for the case of the non moving bed at $x = x_0$, we can write :

$$\eta_x(x_0,t) = -\frac{d\overline{u}_w(t)}{dt}. \tag{3.40}$$

At the downstream end $x = x_N$, we will always have :

$$\eta_x(x_N,t) = 0 \tag{3.41}$$

The finite difference approximations of equations (3.36)-(3.41) will be used together with the other initial and boundary conditions in the numerical solution of the various problems presented in §4.

3.4 The Flux Corrected Transport (FCT) method

In §4, specific moving bed problems will be solved with the help of the finite difference approximations derived in §3.2 for the governing equations. In most cases, these bed motions will be such that strong deformations of the free surface position will occur over relatively short distances. In other words, considerable variations in η (and consequently on $\overline{u}$) will take place over a very small number of grid points.

It is a well known fact that in finite difference methods the treatment of solutions with sharp gradients may lead to accuracy and/or stability problems. Standard techniques available to circumvent these difficulties include grid refinment in the regions of sharp gradients and the introduction of artificial viscosity. However, one of the most effective technique is certainly the so-called Flux Corrected Transport (FCT) method which was initially developped by Boris & Book (1973,1975,1976) for the numerical treatment of shock waves in compressible flows. In view of applying it to the treatment of the problems presented in §4, we now give a brief summary of that method.

The first important thing to note is that the FCT method is not a numerical scheme in itself but rather a procedure for handling steep gradients solutions calculated by means of standard numerical techniques such as the finite difference method. We here follow the general definition of FCT given by Zalesak (1979).

We start by considering the following conservation equation

$$w_t + f_x = 0 \tag{3.42}$$

where w and f are both functions of the independent variables x and t. A finite difference approximation to w is said to be in conservative or "flux" form if it can be written as :

$$w_i^{j+1} = w_i^j - \Delta x_i^{-1}[F_{i+(1/2)} - F_{i-(1/2)}]$$

where w and f are defined on spatial grid points x_i and temporal grid points t_j. The spatial mesh size is given by $\Delta x_i = \frac{1}{2}(x_{i+1} - x_{i-1})$. The terms $F_{i+(1/2)}$ and $F_{i-(1/2)}$ are called transportive fluxes and their functional dependence on f is related to the specific finite difference scheme which is used to approximate (3.42). For example, in the case of an Eulerian (fixed grid) Crank-Nicolson scheme, we would have

$$F_{i+(1/2)} = \frac{\Delta t}{4}[f_{i+1}^{j+1} + f_{i+1}^j]$$

It can be shown (Boris & Book 1973) that in the regions of steep gradients in w, finite difference approximations to (3.42) may suffer from two possible problems

First, if the particular integration scheme which is used is of high order (order 2 or above, e.g. leapfrog, Crank-Nicolson, ...), the solution w will show dispersive 'ripples'. On the other hand, if a low order scheme (e.g. Lax-Friedrichs, donor cell, .) is used, the solution will not be affected by ripples but will suffer from excessive numerical diffusion. A similar result is also obtained if a zeroth order artificial viscosity is added to a high order scheme.

The basic idea behind FCT is the combination of the best aspects of the above two schemes: the accuracy of a high order scheme coupled with the diffusive properties of a low order scheme. The originality of the method is that it constructs a net transportive flux point by point as a weighted average of a flux computed by a low order scheme and a flux computed by a high order scheme. This weighting procedure favors the use of the high order flux except in regions of steep gradients where the arising dispersive 'ripples' are attenuated by increasing the contribution of the low order flux. Therefore FCT can be thought of as a selective application of numerical diffusion which is limited to steep regions of the solution domain in which that diffusion is most needed. That constitute a major advantage over most artificial viscosity methods in which the solution is usually diffused uniformly over the whole domain.

The formal FCT procedure can be defined as follows (Zalesak 1979) :

1. Compute $F^H_{i+(1/2)}$, the transportive flux as given by a high order scheme.

2. Compute $F^L_{i+(1/2)}$, the transportive flux as given by a low order scheme guaranteed to give monotonic (ripple-free) solutions.

3. Define the "antidiffusive flux" :

$$A_{i+(1/2)} \equiv F^H_{i+(1/2)} - F^L_{i+(1/2)}.$$

4. Compute the updated low order ("transported and diffused") solution :

$$w^{td}_i = w^j_i - \Delta x_i^{-1}[F^L_{i+(1/2)} - F^L_{i-(1/2)}].$$

5. Limit the antidiffusive flux by using

$$A^c_{i+(1/2)} = C_{i+(1/2)} A_{i+(1/2)} \text{ with } 0 \leq C_{i+(1/2)} \leq 1$$

in a manner such that w^{j+1} as computed in (6) is free of extrema not found in w^{td} or w^j.

6. Compute the final solution by using the limited antidiffusive fluxes calculated in (5) :

$$w_i^{j+1} = w_i^{td} - \Delta x_i^{-1} [A^c_{i+(1/2)} - A^c_{i-(1/2)}].$$

The crucial part of the FCT method is obviously the choice of an appropriate selection rule for limiting the antidiffusive fluxes in (5). This important flux limitation or "flux correction" step determines the respective contribution of the dispersive (high order) and diffusive (low order) effects in the final solution.

For the problems of §4, the simple FCT algorithm proposed by Peyret & Taylor (1983) will be used to improve the numerical solution of the depth-averaged continuity equation :

1. Compute η_i^{j+1} by the high order method described in §3.2.

2. Compute a low order (transported and diffused) estimation $\overline{\eta}_i^{j+1}$ of the solution by using

$$\overline{\eta}_i^{j+1} = \eta_i^{j+1} + \sigma[\eta_{i+1}^{j+1} - 2\eta_i^{j+1} + \eta_{i-1}^{j+1}].$$

where σ is an artificial viscosity coefficient.

3. Compute the limited antidiffusive flux $A^c_{i+(1/2)}$ by using the following selection rule :

$$A^c_{i+(1/2)} = S \max[0, \min(S\Delta_{i-(1/2)}, |A^1_{i+(1/2)}|, S\Delta_{i+(3/2)})]$$

with

$$\begin{aligned} A^1_{i+(1/2)} &= \frac{1}{8}(\eta_{i+1}^{j+1} - \eta_i^{j+1}) \\ S &= \text{sign}(\Delta_{i+(1/2)}) \\ \Delta_{i+(1/2)} &= \overline{\eta}_{i+1}^{j+1} - \overline{\eta}_i^{j+1}. \end{aligned}$$

4 Compute the final solution

$$\eta_i^{j+1} = \overline{\eta}_i^{j+1} - (A^c_{i+(1/2)} - A^c_{i-(1/2)}).$$

Although it may not be obvious at first sight, the above algorithm constitutes a specific example of the application of the formal procedure described previously.

4 COMPUTATIONS AND EXPERIMENTS

As pointed out in §1, there are numerous instances in which water waves can be generated in a water body by the motion of the underlying bed. Natural phenomena such as landslides, rockfalls and submarine earthquakes as well as underwater volcanic eruptions or explosions are a few examples of the many possible sources of such wave formation. One of the important questions arising in the modelling (physical or numerical) of these problems lies in the proper choice of a wave generation mechanism which must be as close as possible to reality.

In problems of water waves generated in a channel by underwater earthquakes (tsunamis) or explosions, the wave generation mechanism usually consists in the sudden vertical upthrust of a segment of the bed (Kranzer & Keller 1959; Hwang & Divoky 1970; Hammack 1973). This disturbed segment remains under water at all times.

Waves generated by landslides or other types of sliding material are often simulated by two distinct wave generation mechanisms: the horizontal motion of a vertical wall and the vertical fall of a box (Wiegel et al. 1970; Noda 1970; Das & Wiegel 1972). In both cases, the disturbed segment is located at the end of a channel and intersects the water surface. The logic behind this type of modelling is that the actual wave generation mechanism probably consists of a mixed horizontal-vertical type of motion which lies somwhere between the fully horizontal and vertical generation modes.

In this section, we present five cases of wave generation by a moving bed and describe their corresponding experimental and numerical models.

4.1 Types of bed motion considered

The present study was made in collaboration with workers at the Laboratory of Hydraulics, Hydrology and Glaciology (VAW) of the Swiss Federal Institute of Technology (ETH) in Zurich.

The five following numerical simulations were conceived in conjunction with iden-

tical laboratory experiments in order to permit a direct comparison of computational and experimental results. The experiments were performed at VAW in 1987–88 by Johannes Sander under the direction of Drs. K.Hutter and D.Vischer.

4.1.1 Moving wall

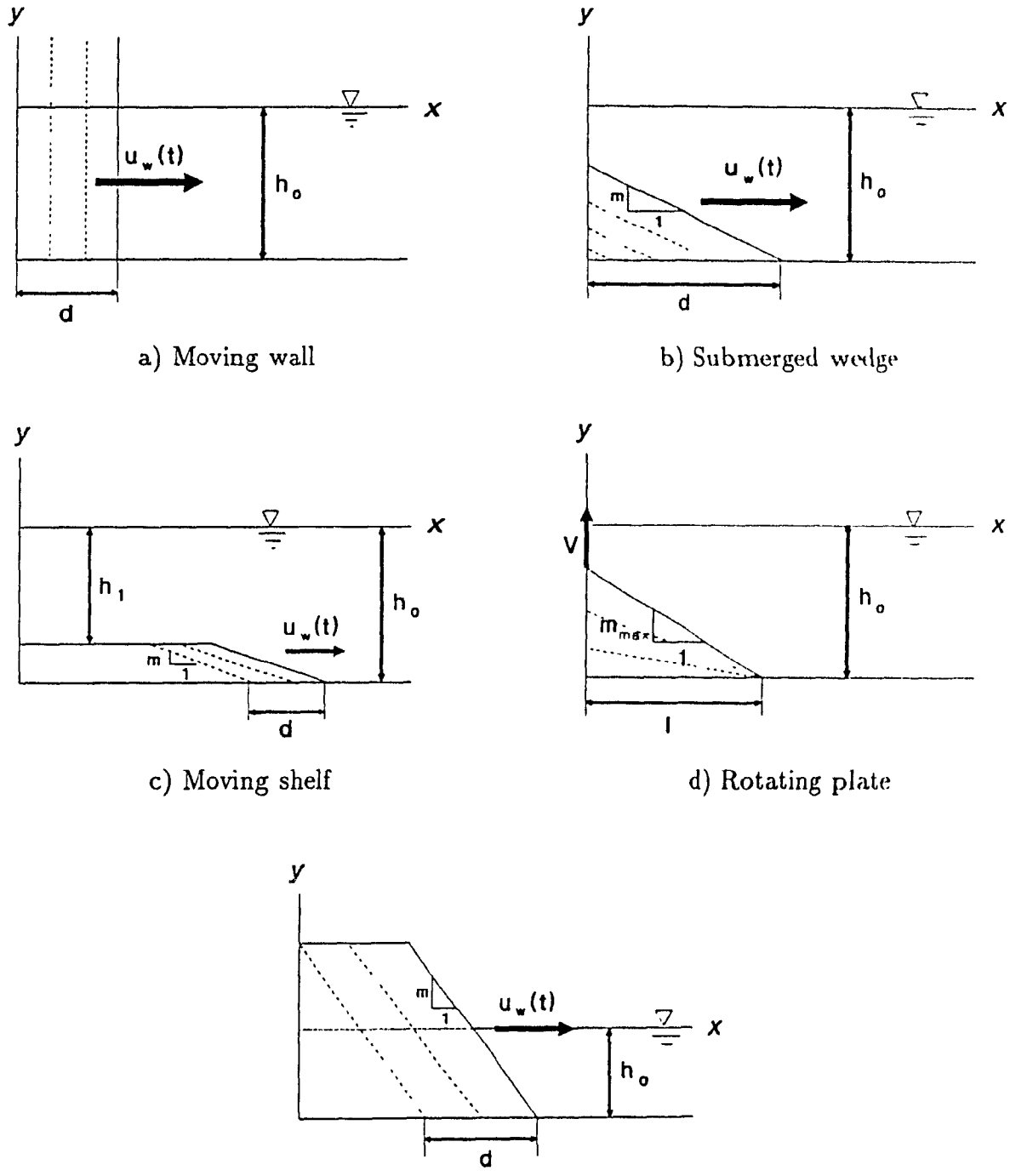
Formally speaking, the horizontal motion of a vertical wall at the end of a long channel does not constitute an example of wave generation by a moving bed. Indeed, the modelling of that problem by the Boussinesq theory does not require the use of the general equation (2.34) but can rather be handled by the simple Boussinesq equation (2.39) for waves over a flat bed with appropriate boundary conditions to model the moving wall.

However, we will still treat this case for two important reasons. Firstly, it constitutes an obvious starting point in the modelling of landslide generated waves. Secondly, that representation not only gives a rough simulation of the actual problem but it also can be easily implemented in an experimental or numerical model.

Figure 4.1a shows that arrangement which from now on will be referred to as the 'moving wall'. The wave generation simply arises from the displacement of the wall over a distance d with a velocity $\overline{u}_w(t)$ during a time interval $0 \leq t \leq t_f$.

4.1.2 Submerged wedge

The second wave generator which will be considered consists of the horizontal displacement of an inclined plate of constant slope m at the end of a channel (Figure 4.1b) At $t = 0$, the channel depth is constant. This wedge shape then moves over a distance d into the channel with an horizontal velocity $\overline{u}_w(t)$. The total duration t_f of the displacement is always chosen such that the resulting bed deformation does not reach the level of the undisturbed free surface. For that reason, we will refer to this wave generating device as the 'submerged wedge'. The fact that the moving bed never intersects the undisturbed free surface greatly simplifies the numerical solution of the problem by avoiding the treatment of a moving waterline.



a) Moving wall

b) Submerged wedge

c) Moving shelf

d) Rotating plate

e) Moving wedge

Figure 4.1: Types of bed motion considered.

Compared to the moving wall, the submerged wedge certainly constitutes a better representation of the entry of a landslide in water. The model was inspired by the probable configuration of a mass of granular material which is inclined at its angle of repose after entering the water.

4.1.3 Moving shelf

This wave generating device constitutes an extension of the previous case, the only differences being that the wedge now posseses a predefined height less than the undisturbed water depth and that its motion takes place some distance away from the upstream end of the channel (Figure 4.1c). The depth above the shelf is denoted by h_1. The resulting arrangment is referred to as the 'moving shelf'. This model can probably simulate the advance of a long strip of material along the bed of the channel. Contrary to the two previous wave generating devices, the moving shelf could probably be useful in the modelling of tsunamis.

The model was conceived as an extension of the shelf topography which was often used in the study of solitary wave propagation (Madsen & Mei 1969; Johnson 1972; Helfrich & Melville 1986).

4.1.4 Rotating plate

The 'rotating plate' wave generating device consists of a plate which is rotated about a fixed location $x = l$ at the bottom of the channel (Figure 4.1d). The plate initially coincides with the bed of the channel and is then lifted with a constant vertical velocity V at $x = 0$ until the slope reaches a maximum value m_{max} at $t = t_f$ which is always chosen such that the deformed bed does not go beyond the undisturbed free surface level. Unlike the previous wave generators, the rotating plate does not involve a lateral translation of the wave generating device in the x-direction. Furthermore, its resulting motion implies a strong vertical component.

The rotating plate could well represent the accumulation at the bottom of the channel of a granular mass which does not travel laterally but rather builds up into

a sloping heap.

4.1.5 Moving wedge

Compared to the four other wave generating devices, the 'moving wedge' (Figure 4.1e) probably constitutes the best model for landslide generated waves. It involves the motion of an inclined wall spanning accross the whole water depth. The horizontal bed velocity is denoted by $\overline{u}_w(t)$ and its constant slope by m. The resulting bed deformation is similar to the case of the submerged wedge except that it now intersects the free surface. As pointed in §3.2.2, the moving wedge model introduces the difficulty associated with the numerical treatment of the constantly moving point of intersection between the bed and the free surface.

4.2 Experimental set-up

As mentioned earlier, the five wave generating devices described above were modelled experimentally at the Laboratory of Hydraulics, Hydrology and Glaciology (VAW) in Zurich, Switzerland (Sander 1988).

The channel used for the experiments was 16 m long and had a cross section of 0.3 m × 0.3 m. The bottom and back walls were made of 1 cm thick dark grey PVC sheets while the front was made of a 1 cm thick plexiglass wall to permit direct observation of the propagated waves. Two types of wave generating mechanisms were used.

4.2.1 Rotating plate

For the rotating plate generator, a flat plate 30 to 50 cm long located at the upstream end of the channel was hinged to a cylindrical bearing at the bed. The plate was rotated around that hinge by the vertical traction of an electrical motor. At $x = 0$, the water was bounded by a vertical plate which was free to move in the vertical direction as the rotating plate was set in motion.

4.2.2 Lateral displacements

For all the remaining wave generating devices, a common basic mechanism was used. A piston driven by an electrical stepping motor was installed at the upstream end of the channel. A gearing with cross-like slot guides was used to transform the rotation of the motor into the horizontal translation of two straight parallel bars. Different piston geometries were attached to the extremity of these bars to make up the various wave generating devices.

For the moving wall, a simple vertical plate was used. In the case of the moving wedge, it was replaced by inclined plates with slopes of 15, 30 or 45°. The submerged wedge generator was obtained by placing a moveable vertical partition above the inclined plate at $x = 0$. For the moving shelf, a long, shallow rectangular box was installed in front of the vertical wall piston. Again the water was bounded at $x = 0$ by a vertical partition

The electrical stepping motor turns at a constant frequency. By neglecting its initial acceleration and final deceleration, it can be shown that the rotation of the motor gives rise to an horizontal piston velocity which has a sinusoidal variation in time (Sander 1988). The piston velocity can be expressed as :

$$\overline{u}_w(t) = v^* \sin\left\{\pi[\tau_1 + \frac{(\tau_2 - \tau_1)}{t_f}t]\right\} \tag{4.1}$$

where t represents the non-dimensional time and t_f is the total duration of the bed motion. The maximum piston velocity is denoted by v^*. The parameters τ_1 and τ_2 depend on the specific wave generating device which is used. In all experiments, $\overline{u}_w(t)$ is non-negative at all times (i.e. no pulsating motion of the piston). For the moving wall, moving shelf and moving wedge, we always have $\tau_1 = 0$ and $\tau_2 = 1$. For the submerged wedge, values of $\tau_1 = \frac{1}{4}, \frac{1}{3}$ and $\tau_2 = \frac{3}{4}, \frac{2}{3}$ are also used.

The resulting horizontal displacement d of the wave generating device was always predefined in each experiment. By integration of (4.1),

$$d = \int_0^{t_f} \overline{u}_w(t)\, dt = v^* \int_0^{t_f} \sin\left\{\frac{\pi}{t_f}[t_f\tau_1 + (\tau_2 - \tau_1)t]\right\} dt$$

from which we can deduce the following expression for the total duration of the bed motion :

$$t_f = \frac{\pi(\tau_2 - \tau_1)d}{v^*(\cos\pi\tau_1 - \cos\pi\tau_2)} \tag{4.2}$$

Equation (4.2) will be useful in the development of the computer models which will be presented in the next subsection.

4.3 Computer models

Five computer programs were developed in order to model the various cases of waves induced by a moving bed described in §4.1. We now give an overview of these programs' general characteristics and a description of their source code.

4.3.1 General description

All the computations were performed on IBM PC or compatibles computers of the XT and AT type.

The programs were edited, compiled and run with the help of the TURBO PASCAL package (versions 3.0 and 4.0). The graphical output was produced by PLOT-CALL 1.0.

Depending on the extent of the solution domain, the total computation time for each problem ranged from 3 to 10 hours.

The source codes of the five programs are listed in Appendix C along with the definition of the variables and of the constants used to specify the initial parameters of the problems. Table 4.1. gives the name of these programs and the wave generating mechanisms to which they correspond. All these programs have the same main structure, which can be divided in 3 basic parts.

A first set of procedures specifies the initial and boundary conditions for the dependent variables η and $\overline{u}$. The initial conditions always correspond to a still water surface and are given by equations (3.30) and (3.31). The boundary conditions depend on the geometry and the type of displacement of each particular problem (see

§3.3.2).

Type of wave generation	*Name of program*
Moving wall	WALL
Submerged wedge	SUBWED
Moving shelf	SHELF
Rotating plate	ROPLATE
Moving wedge	WEDGE

Table 4.1. Types of bed motion considered and corresponding computer programs

A second group of procedures insures the proper mathematical modelling of the evolution $h(x,t)$ of the moving bed. That part of the program is very important since the bed displacement history directly influences the characteristics of the resulting wave profile

The core of the programs involves the application of the numerical methods of §3 to the solution of the governing equations. This segment of the programs therefore performs the actual calculation of η and $\overline{u}$ resulting from a known bed motion with appropriate initial and boundary conditions

For the reasons already pointed out in §3.2.1, both the Eulerian and the Lagrangian finite difference schemes are used. The Lagrangian scheme is applied to the moving wall (program WALL) and the moving wedge problems (program WEDGE), which are the two cases involving a point of intersection between the water surface and the wave generating device. The treatment of the three other cases (programs SUBWED, SHELF, ROPLATE) relies on Eulerian schemes.

In the case of Lagrangian computations, a fourth set of procedures was developed for the task of reallocating the moving spatial grid points to their new location at every time step.

In order to avoid possible accuracy or stability problems, the FCT scheme de-

scribed in §3.4 was implemented in each program to eliminate non-physical ripples in the calculated wave profiles. The scheme proved to be necessary in all the problems treated except for the moving wall where a satisfactory solution was obtained whithout the use of FCT.

We now proceed to a separate description of each program.

4.3.2 Program WALL

The WALL program models the moving wall problem. It is listed in Appendix C on pages C-2 to C-7 The main program is relatively short and relies on the use of various procedures.

The procedure 'Initial_Cond' creates an initial spatial grid having a constant mesh size 'dx'. It also specifies on each grid points ($i = 0, 1, 2, \ldots, n$) initial values 'etha1[i]' and 'u1[i]' for the free surface position and the depth-averaged velocity which correspond to equations (3.30) and (3.31), i.e. fluid at rest.

The velocity boundary conditions are implemented by the procedure 'U_Bc' which uses equations (3.33) and (3.35). The free surface boundary conditions are based on equations (3.40) and (3.41) and calculated by the procedure 'Etha_Bc'

The horizontal motion of the wall is modelled by the functions 'Vpiston' and 'Accpiston' which respectively give the velocity and acceleration of the wall at any given time. The function 'Vpiston' is based on equation (4.1) with $\tau_1 = 0$ and $\tau_2 = 1$

To obtain the numerical solutions of the governing equations, the three step scheme proposed in §3.2 is incorporated in three distinct procedures.

The procedure 'Continuity_Predictor' calculates a predicted value 'etha2[i]' ($i = 1, 2, \ldots, n-1$) for the free surface position using equation (3.24) with the 'old' values 'u1[i]' of the velocity field. The procedure 'Momentum' solves equation (3.25) by using an implicit (Crank-Nicolson) finite difference scheme.

The updated velocity field 'u2[i]' ($i = 0, 1, 2, \ldots, n-1$) is obtained as the solution of a tridiagonal matrix system by the procedure 'Tridiag'.

Finally, the procedure 'Continuity_Corrector' also solves (3.24) but uses the values

'u2[i]' of the updated velocity field.

At each time step, the updated location 'x2[i]' of the moving grid points is determined by means of two procedures : 'Approx_Interfaces' and 'Interfaces'. The first one gives the approximate location of the grid points (or fluid interfaces) as given by equation (3.27) with $\theta = 0$. The procedure 'Interfaces' is used after the 'Momentum' procedure and solves equation (3.24) with $\theta = 1/2$.

4.3.3 Program SUBWED

This program uses an Eulerian finite-difference scheme to model the waves generated by a moving, submerged wedge. It is listed on pages C-8 to C-17 of Appendix C.

The main program begins by the imposition of initial conditions and velocity boundary conditions via the procedures 'Initial_Cond' and 'U_Bc'. The velocity boundary conditions (3.34) and (3.35) are used.

The free surface boundary conditions are handled by the procedures 'Etha_Bc_Pred' and 'Etha_Bc_Corr' in accordance with equations (3.38) and (3.41).

The evolution of the moving bed is treated by the procedures 'Bed_is_Moving' and 'Move_Bed' which are based on equation (4.1). The procedure 'Stationary_Bed' is used to specify the topography of the bed once its motion has stop.

The continuity equation is again solved in two steps by the procedure 'Continuity_Predictor' which uses the finite-difference expression (3.18) and by 'Continuity_Corrector' which uses (3.20).

The updated velocity field 'u2[i]' is calculated by the procedure 'Momentum' in accordance with the implicit (Crank-Nicolson) scheme presented in §3.2.1). To reduce the computation time, two distinct procedures are used separately for calculations during the bed motion (procedure 'Momentum_Fxt') and after the bed has come to rest (procedure 'Momentum_Fx'). Both procedures use the 'Tridiag' matrix solver.

The procedure FCT corrects the updated free surface position 'etha2[i]' by using the Flux Corrected Transport (FCT) algorithm presented in §3.4.

4.3.4 Program SHELF

This program modelling the generation of a wave by the motion of a shelf is listed in Appendix C, pages C-18 to C-27.

The source code is almost identical to SUBWED. The only differences are in the specification of the bed motion and in the free surface boundary conditions calculated by the procedure 'Etha_Bc'. The upstream and downstream boundaries now bo'' consists of fixed vertical walls over a non-moving bed and the conditions (3.39) and (3.41) are used.

4.3.5 Program ROPLATE

This program models the rotating plate problem described in §4.1.4. The listing of ROPLATE can be found on pp. C-28 to C-37. Again, this program closely resembles the other two (SUBWED and SHELF) using an Eulerian scheme.

The main difference is now in the specification of the bed motion which does not imply an horizontal sinusoidal velocity as given by (4.1) but rather a vertical pull of constant velocity (Figure 4.2a).

The upstream free surface boundary condition at is now given by equation (3.37)

4.3.6 Program WEDGE

This program (pp. C-38 to C-47) treats the moving wedge problem of §4.1.5. As mentioned before, it uses a Lagrangian finite difference scheme. It constitutes a generalisation of the program WALL in which the complete governing equations (3.22) (3.23) are used instead of (3.24)-(3.25).

The upstream boundary condition, now consists in a laterally moving point of zero depth. Instead of using the equations derived in §3.3.2, we simply express the free surface boundary condition at this point by the equation

$$\eta(x_0, t) = -h(x_0, t)$$

which is used in procedure 'Etha_Bc'.

Furthermore, the upstream velocity boundary condition is now calculated by a simple extrapolation (procedure 'U_Bc_Corr') of the values of the velocity field at the grid points $i = 1$ and 2.

The modelling of the moving bed is now done by the procedure 'Find_Depth'. Unlike the previous cases, that procedure allows the use of an interpolating polynomial in order to smooth off the transition between the sloping segment and the flat part of the bed (Figure 4.1c). The interpolating polynomial is given by (Pedersen & Gjevik 1983) :

$$p(s) = \frac{1}{32}(s^6 - 5s^4 + 15s^2 - 16s + 5) \tag{4.3}$$

where

$$s = \frac{m}{l_t}(x_i - x_b)$$

and

x_b = position of the foot of the wedge,

m = slope of the wedge,

l_t = required length of the smoothed transition.

This feature was added in order to investigate the effect of the slope discontinuity on the evaluation of the first order spatial derivative $\frac{\partial h}{\partial x}$ at $x = x_b$.

5 COMPARISON OF RESULTS

Graphical outputs obtained from the five computer models described earlier will now be presented and compared with corresponding experimental results.

5.1 Moving wall

Figure 5.1a gives a representation in the x-t plane of a wave generated by the lateral motion of a vertical wall. The wave profiles were computed with the program WALL based on the Lagrangian scheme described in §3.2.3. The initial spatial mesh size was set to $\Delta x = 0.25$ and the temporal mesh size to $\Delta t = 0.25$. The FCT algorithm was not used in the numerical solution. For the case shown, the motion of the wall was governed by equation (4.1) with $\tau_1 = 0$, $\tau_2 = 1$ and $v^* = 0.2$. The wall moved from $x = 0$ to $x = 1.67$ in a time $t_f = 13.12$ as calculated from equation (4.2). The instantaneous position of the wall is indicated by a line which becomes parallel to the time axis for $t \geq 13.12$.

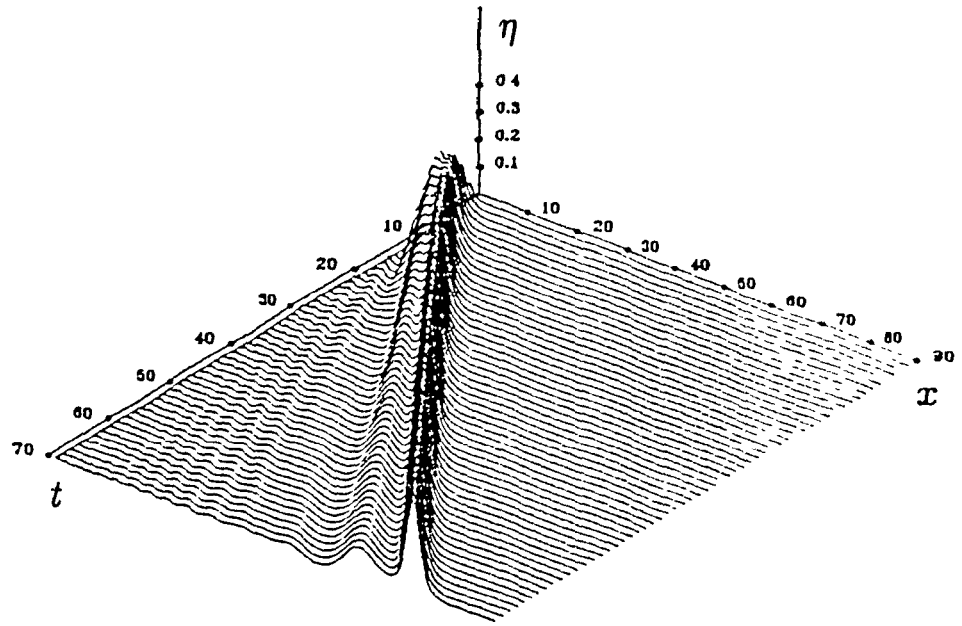


Figure 5.1a: Wave generated by a moving wall.

As described in §2.1, all the variables were non-dimensionalized according to a characteristic depth h_o which was chosen to correspond to the free surface level at rest

Figure 5.1b shows a comparison of the above numerical output with results obtained from a channel experiment at VAW-ETH. The channel had an initial depth water depth of $h_o = 15$ cm. The parameters governing the motion of the wall were all identical to the ones used in the computation. Results are presented as wave height recordings measured at eight locations along the channel. The experimental results are represented by a solid line and the computational ones by a dashed line.

It appears that the general agreement between computed and measured wave profiles is quite good. The free surface disturbance initially takes the form of a single hump As it travels down the channel, the initial wave splits into a large leading wave followed by a train of smaller, dispersive oscillations of decreasing amplitude. It can be shown (Sander 1988) that the leading wave matches almost exactly the solitary wave solution of the Boussinesq equation.It is worthwile noticing that the numerical model not only predicts the shape of that leading wave but it also gives a very good representation of the trailing dispersive waves.

However, there is a slight difference between computed and observed peak amplitudes Although the calculated peak of the initial disturbance (gauge 1) is very close to the measured value, it appears that further downstream, the calculated amplitude is slightly larger. That small difference could probably be attributed to the neglect in the mathematical model of the dissipation effects associated with the water viscosity and the wall friction in the channel.

The agreement between computed and measured values indicates that our model can account properly for wave amplitudes in the order of 25% of the undisturbed water depth. Such values of the relative wave amplitude could probably not be treated using the 'small amplitude' theory. Figure D.1 in Appendix D shows a comparison between experimental and computational results for which a reasonable agreement was obtained even for a leading wave amplitude of more than 0.5.

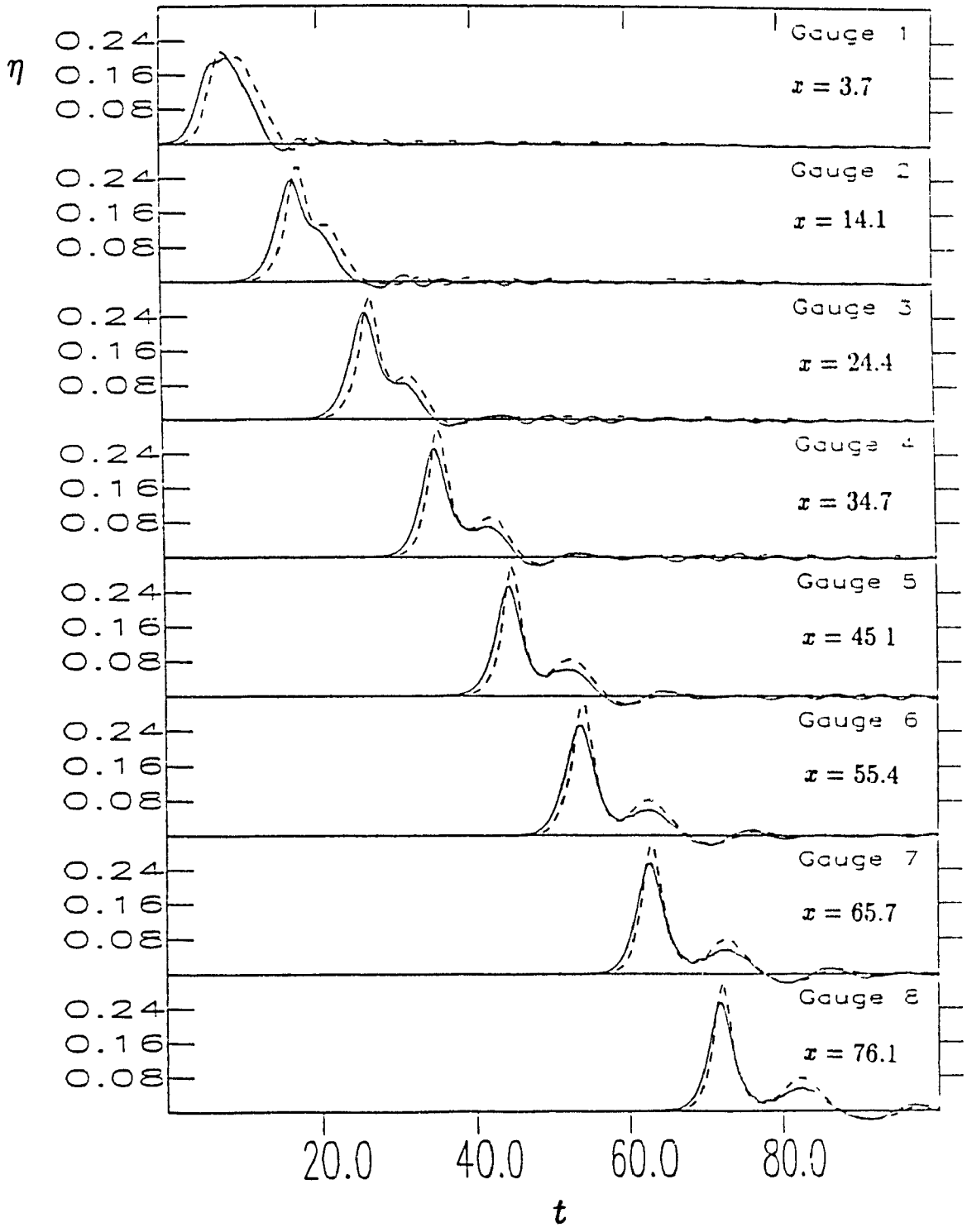


Figure 5.1b: Wave generated by a moving wall. Maximum horizontal velocity of the wall $v^* = 0.2$; total displacement $d = 1.67$; duration of the motion $t_f = 13.12$ Experiment ——, computation - - -.

5.2 Submerged wedge

The so-called 'submerged wedge' problem was the first moving bed problem to be treated in this study. For that reason, numerous computations were performed and supplementary results are presented in Appendix D.

Figure 5.2a represents the propagation of a wave generated by a moving submerged wedge having a slope of 0.268 (15°) and a sinusoidal velocity described by equation (4.1) with $\tau_1 = 0$, $\tau_2 = 1$ and $v^* = 0.2$. The foot of the wedge moves from $x = 0$ to $x = 1.67$ in a time t_f of 13.12. The wave generation and propagation were computed by the program SUBWED. The Eulerian scheme of §3.2.2 was used with a constant spatial mesh size of 0.25 and a temporal mesh size of 0.25.

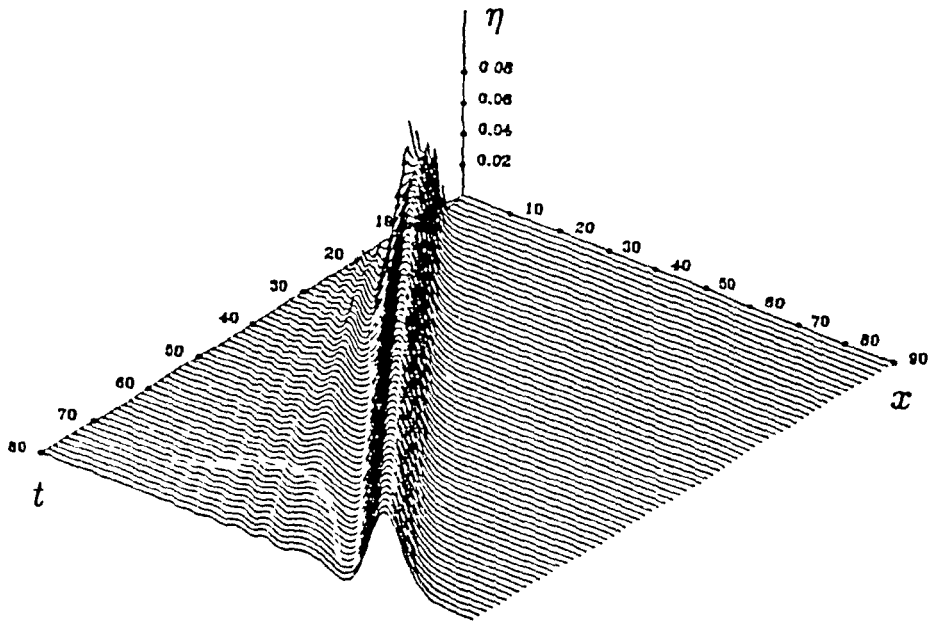


Figure 5.2a: Wave generated by a submerged wedge.

Unlike the case of the moving wall which we just described, the submerged wedge problem uses the FCT algorithm of §3.4. As already mentioned, the necessity of FCT in the submerged wedge problem (as well as in the three remaining cases involving a motion of the bed) can be explained by the fact that sharp variations in the bed elevation (and consequently on η and $\overline{u}$) occur over only a few spatial grid points

during the generation phase. That explains why FCT was not needed in the moving wall problem which was simply modelled by the Boussinesq system for waves over a flat bed with appropriate boundary conditions to handle the moving wall.

Figure D.2b in Apppendix D shows computational results obtained without the use of FCT for the bed motion conditions stated earlier. When compared to the solid line of figure 5.2b which was obtained from measurements in a channel experiment, we see that the computed wave profiles of figure D.2b are unsatisfactory. A train of non-physical oscillations is trailing the leading wave. These 'wiggles' are especially important in wave profiles computed during the generation phase.

The results of a computation using the FCT algorithm with a diffusion constant σ of 0.2 are shown in figure 5.2a and represented by a dashed line in figure 5.2b (these results are also shown on figure D.2a but with a different scale).

The agreement between computed and measured wave wave heights (Figure 5.2b) is even better than in the case of the moving wall. The calculated peak amplitudes now closely match the experimental values. Although the dissipative effects associated with the experiment are still probably present, it would seem that a corresponding diffusion is introduced in the computed profiles by the use of FCT.

Again, we observe that our model permits appropriate representation of both the leading wave and the trailing sequence of smaller dispersive waves. To reinforce that statement, computations were carried out using the linearized long-wave equations (1.4) and the shallow-water, finite-amplitude equations (1.2) in non-dimensional form (i.e. g replaced by 1). The results are shown in Appendix D in figures D.2c and D.2d. We observe that both theories are unable to predict the occurence of the trailing dispersive waves. Furthermore, in both cases, the calculated leading wave is valid only for a short time. As it travels downstream, its amplitude becomes overestimated and, in the case of the finite-amplitude equations (Figure D.2d), it suffers from an excessive steepening.

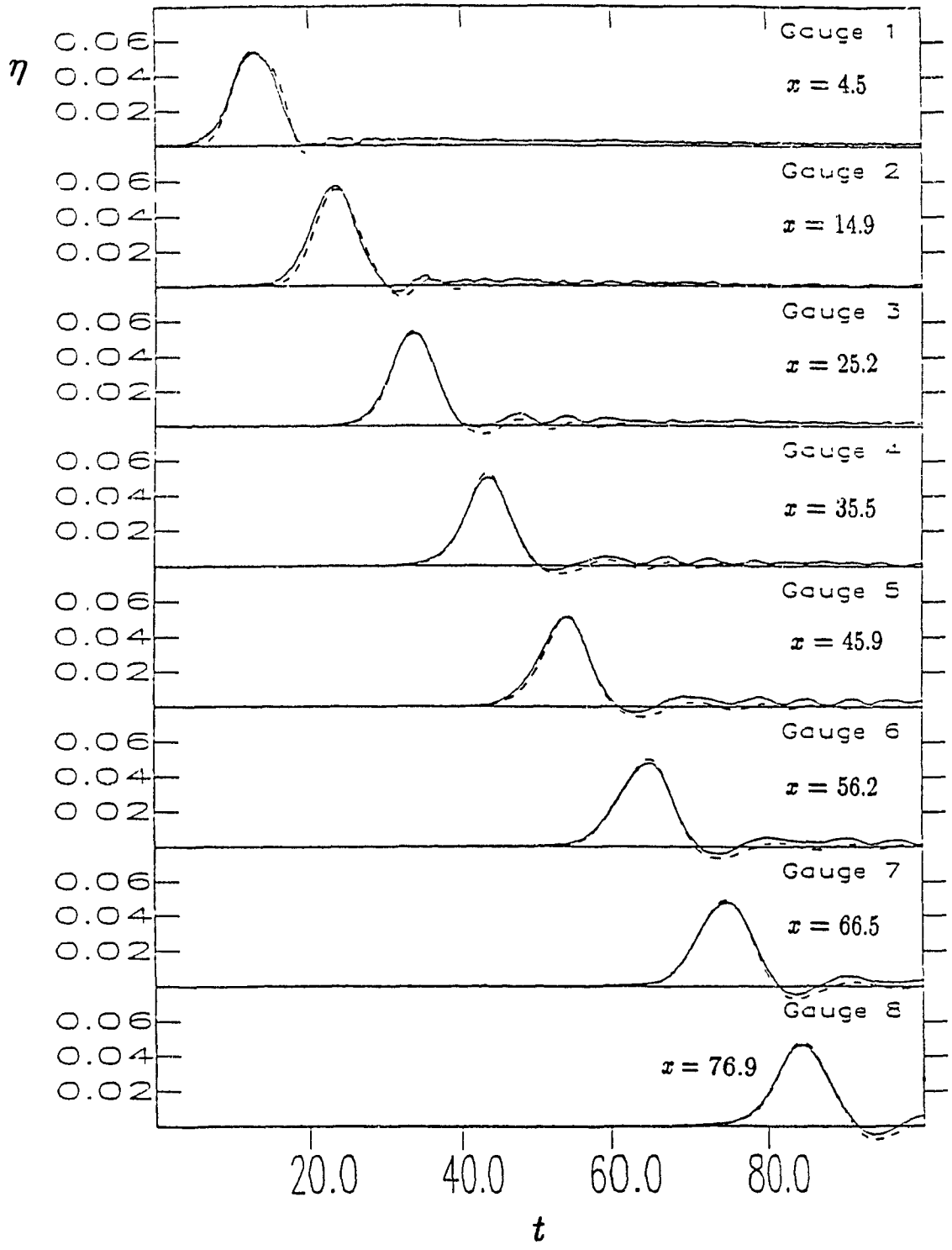


Figure 5.2b: Wave generated by a submerged wedge. Slope $m = 0.268$; maximum horizontal velocity of the wedge $v^* = 0.2$; total displacement $d = 1.67$; duration of the bed motion $t_f = 13.12$. Experiment ——,computation - - -.

In order to confirm the validity of our weakly nonlinear, dispersive approach, the results of two other simulations are presented. Figures D.2e and D.2f both involve the same displaced bed volume as the case we already described (Figure D.2a). The wave profiles of figure D.2e were obtained by reducing the amplitude v^* of the sinusoidal wedge velocity from 0.2 to 0.1 and by increasing the duration t_f of the bed motion from 13.12 to 26.23 in order to obtain the same displaced volume. Similarly, in figure D.2f, v^* was kept at 0.2 but the slope was increased from 0.268 (15°) to 0.577 (30°) thus reducing the motion time from 13.12 to 8.94. The conclusion which can be drawn from comparison of figures D.2a, D.2e and D.2f is that the far field behaviour of the wave depends on the bed motion history and not only on its final displacement. This fact contradicts the conclusion which can be derived from the small-amplitude theory (Hunt 1988) and which states that, under the influence of dispersion, all waves created by the same resultant bed displacement should ultimately become identical.

Supplementary results are presented on figures D.2g, D.2h and D.2i to confirm the validity of the model.

5.3 Moving shelf

As mentioned in §4.1.3, the 'moving shelf' wave generator is similar to the case of the submerged wedge. The interest of this wave generator lies in the fact that the initial free surface disturbance is induced some distance away from the origin. Therefore the moving shelf could probably be useful in the study of waves generated away from the shoreline such as tsunamis.

Computations and corresponding experiments were performed to model the propagation of a wave generated by the motion of a shelf having the following characteristics:

- depth of water above the shelf $h_1 = \frac{2}{3}h_o$ with $h_o = 15$ cm in the experiment,
- slope of the depth transition $m = 0.577$,
- initial position of the top of the transition $= 5$,

- distance travelled by the shelf $d = 1.67$,
- duration of motion $t_f = 13.12$.

The resulting wave profile is shown in the x-t plane in figure 5.3a. The numerical results are compared with experimental values in figure 5.3b. The computations were performed with the program SHELF. The spatial and temporal mesh size were again set to 0.25. Flux Corrected Transport was used with $\sigma = 0.2$.

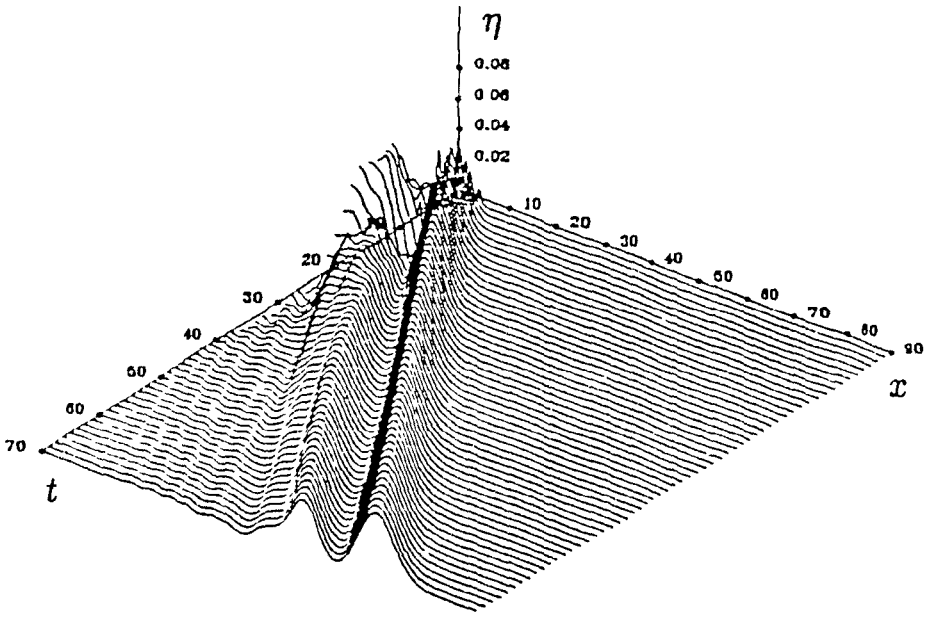


Figure 5.3a. Wave generated by a moving shelf.

The initial disturbance is split up into a right running and a left running wave. As it reaches the origin, the left running wave is reflected and starts to trail the right running wave. Figure 5.3b indicates a very good agreement between computed and experimental values.

The only observable difference is in the amplitude of the trough which follows the reflected wave. The calculated amplitude is slightly higher than the observed one. However, as time increases, the numerical and experimental wave heights become equal (see gauges 3 to 6).

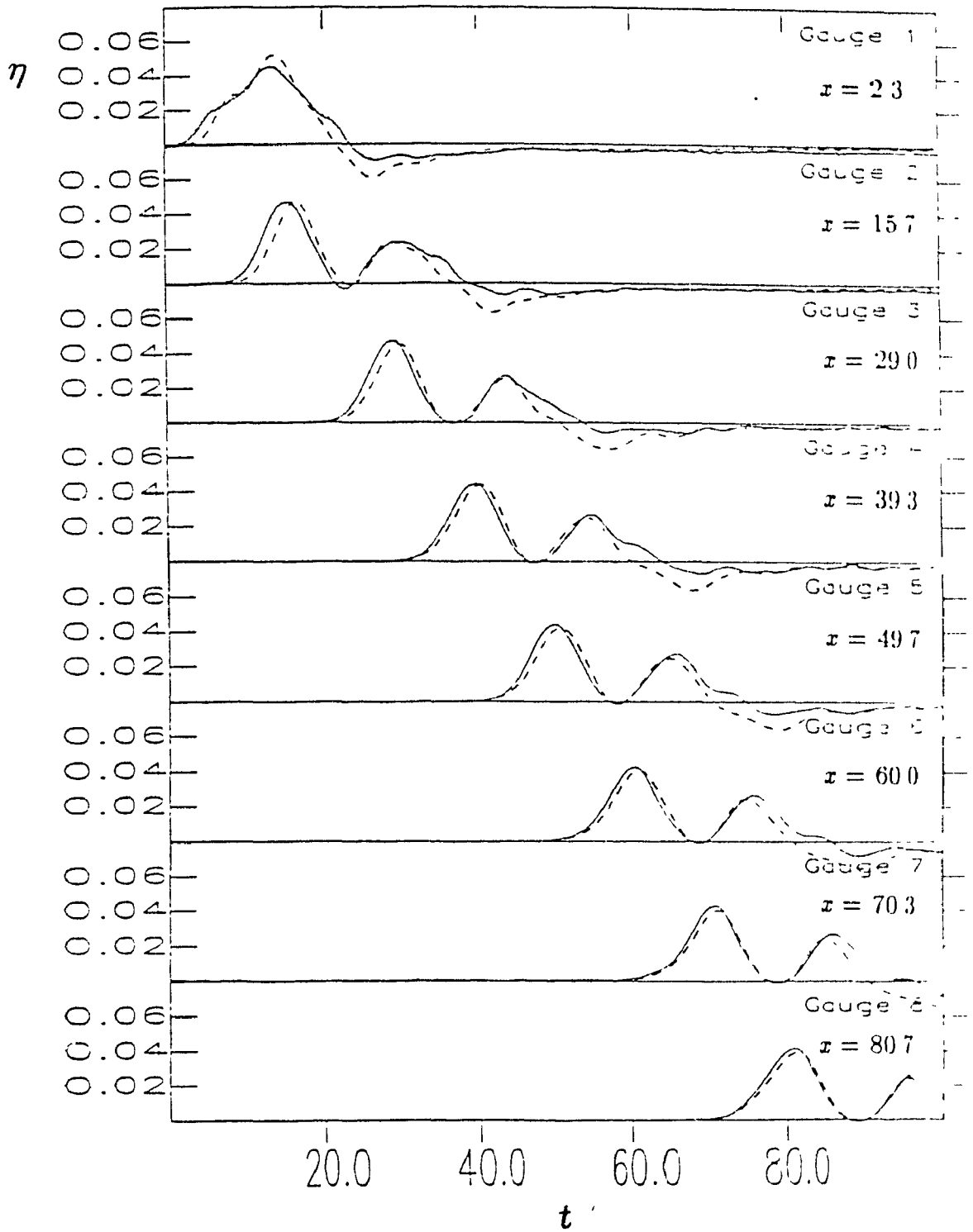


Figure 5.3b: Wave generated by a moving shelf. Slope of the depth transition $m = 0.577$; depth above the shelf $h_1 = \frac{2}{3}h_o$, maximum horizontal velocity $u^* = 0.2$ total displacement $d = 1.67$; duration of the bed motion $t_f = 13.12$ Experiment ——,computation - - -

5.4 Rotating plate

The 'rotating plate' problem is the only one which does not involve a sinusoidal motion of the wave generator. The motion here simply consists of the rotation of a flat plate resulting from a pull of constant velocity applied at $x = 0$.

The program ROPLATE was used to calculate a numerical solution for the wave height η. The results are shown in figure 5.4a and compared with experimental data in figure 5.4b. In both cases, the plate had a length $l = 3.33$ and was rotated for $t_f = 8$ with a vertical velocity $V = 0.1$ at $x = 0$.

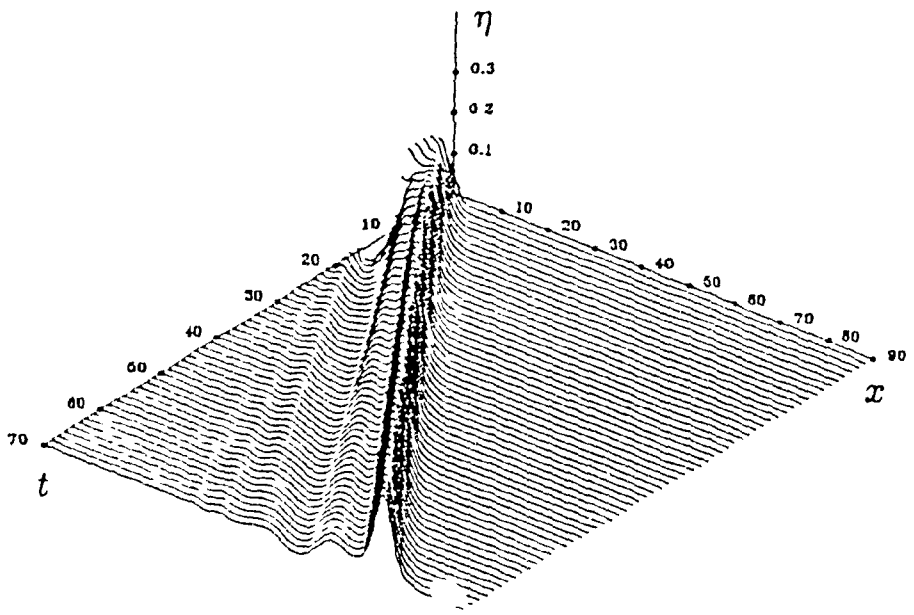


Figure 5.4a. Wave generated by a rotating plate.

In spite of the apparent simplicity of the wave generating device, the agreement between computed and observed wave heights is not as good as in the previous cases. Figure 5.4b shows that the computed wave amplitude (dashed line) is always slightly larger than the experimental values (solid line) even in the initial generation stage. Therefore, unlike the case of the moving wall, the difference in amplitude cannot be simply attributed to dissipative effects in the wave channel.

It can also be observed that at gauge 1 ($x = 10.3$), the observed disintegration

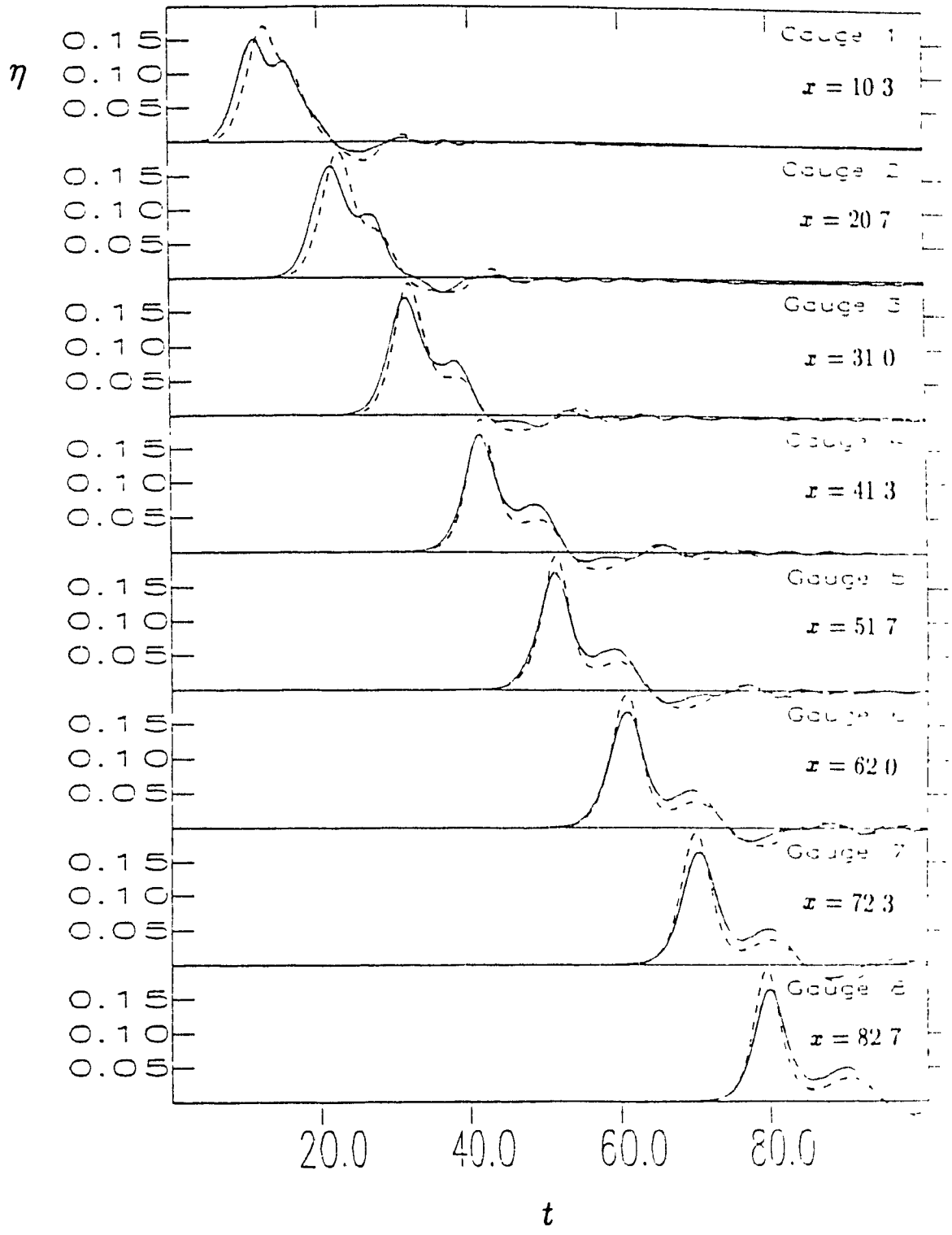


Figure 5.4b: Wave generated by a rotating plate. Length of the plate $l = 3.33$ vertical pull velocity $V = 0.1$ at $x = 0$, duration of the bed motion $t_f = 8.0$, final slope $m_{max} = 0.24$. Experiment ——,computation - - -.

of the initial disturbance into shorter humps is not clearly defined in the computed results

The above comments can also be applied to the results shown on figures D.3a and D.3b of Appendix D

Two reasons may explain these small discrepancies. First, it is possible that the imposition of a constant and impulsive pull velocity at the end of the plate causes slight instabilities in the numerical model. Secondly, it appears that in all the simulated rotating plate problems, the final slope was rather low (≈ 0.2). Thus the rotation of the plate mainly induces a vertical motion component to the water layer above it. In these conditions, the depth-averaged velocity assumption made in the governing equations is perhaps inappropriate. Nevertheless, apart from these minor differences, the correspondence between numerical and experimental results remains quite satisfactory. As seen in figure 5.4b, the model gives a good representation of both the leading and trailing waves

5.5 Moving wedge

The last moving bed problem that was considered involves the generation of a wave by the lateral displacement of an inclined wall intersecting the water surface.

As mentioned earlier, the treatment of this problem is complicated by the presence of a moving waterline. The program WEDGE was used to compute the resulting wave profiles. The FCT algorithm was again used in this case. Without FCT, the computed solutions suffered from severe instabilities.

Figures 5.5a and 5.5b give the evolution of a wave created by the motion of a wedge of slope $m = 0.268$ which moves with a sinusoidal velocity from $x = 0$ to $x = 1.67$ m a time of 13.12. In figure 5.5b, the instantaneous position of the moving waterline is indicated by a curve in the x-t plane.

It appears that, during the initial rise of water, the x-coordinate of the waterline remains at $x = 0$. As soon as the wave leaves the origin, the water depth comes

back to its undisturbed level as indicated by the position of the waterline which stays constant at $x \approx 1.67$.

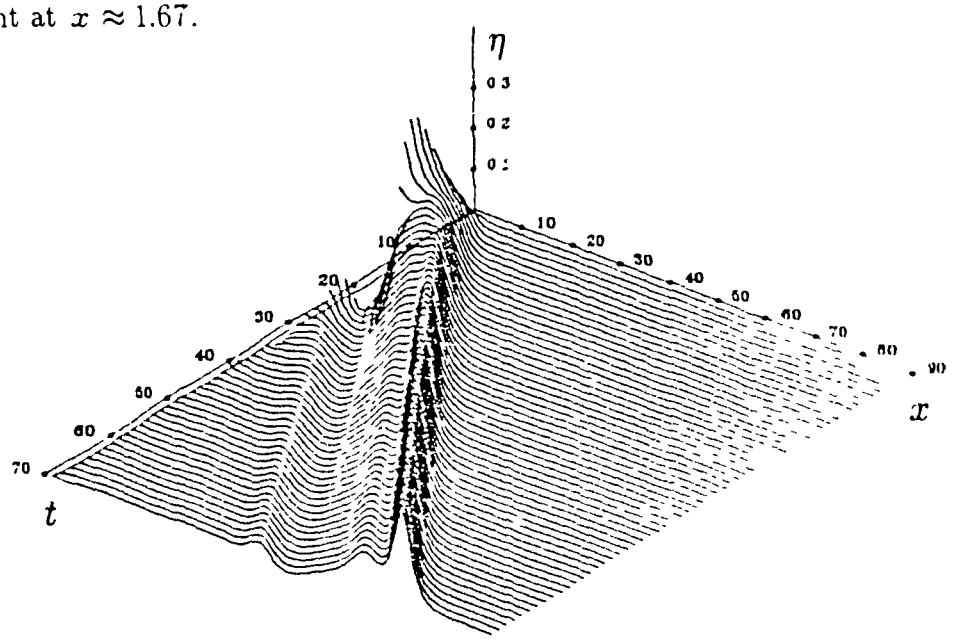


Figure 5.5a: Wave generated by a moving wedge

As before, the numerical output of figure 5.5a is compared with equivalent experimental results in figure 5.5b. The agreement is excellent. The computed amplitude and wavelength of both the leading and trailing waves match almost perfectly the experimental values.

The effect of the bed slope discontinuity on the evaluation of the derivative $h_{,x}$ was studied by replacing that discontinuity by a smooth transition as given by equation (4.3). No improvement was observed in the computed solution.

Another simulation was made to investigate the validity of the model in cases involving a steeper wedge slope. Figure D.4 shows calculated and measured wave heights for a moving wedge of slope $m = 1$. As in the previous case, the computations closely match the experiment.

The above results show that the Lagrangian treatment of the moving waterline gives unexpectedly good results in spite of its relative simplicity. Furthermore, they confirm the validity of the proposed model for the accurate descripiton of waves generated by moving boundaries.

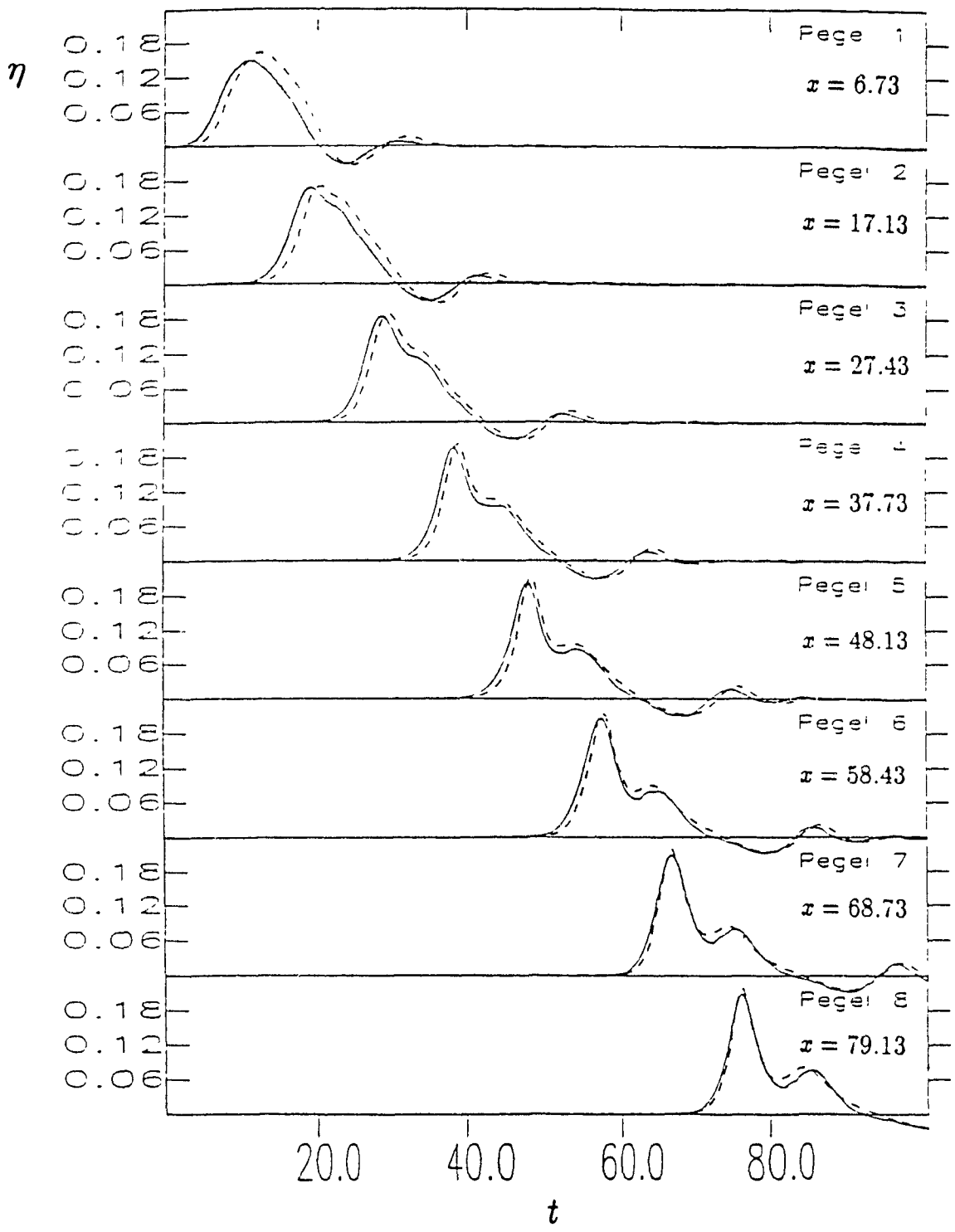


Figure 5.5b Wave generated by a moving wedge. Slope $m = 0.268$; maximum horizontal velocity of the wedge $v^* = 0.2$; total displacement $d = 1.67$; duration of the bed motion $t_f = 13.12$. Experiment ——,computation - - -.

6 CONCLUSIONS

Natural phenomena involving the generation and propagation of water waves developed by moving boundaries sometimes can constitute a serious threat to both coastal populations and man-made structures. Landslide induced water waves in lakes and tsunamis are well documented examples of this type of problem.

In the preceding sections, a mathematical model was developed for the modelling of water waves generated by a moving bed. As a starting point to more realistic representations, only one-dimensional, frictionless problems were considered.

The model relies on a set of depth-averaged continuity and momentum equations which constitute a generalization of the Boussinesq equations accounting for the unsteady and nonuniform bed topography. This theory is limited to wave propagation in shallow water but it presents the advantage of combining, in an approximate balance—the two fundamental effects of nonlinearity (or amplitude dispersion) and dispersion (or frequency dispersion). This feature eliminates questions about the range of applicability of the theory and in that respect constitute a major advantage over small-amplitude and finite-amplitude (Airy) theories.

A simple numerical solution to the governing equations was obtained by means of finite difference approximations. In order to increase the accuracy, the computed wave profiles were treated with a simple Flux Corrected Transport (FCT) algorithm. That scheme was not only useful in upgrading the accuracy of the solutions, it also lead to accurate solutions in cases where a stable solution could not even be obtained without FCT.

The numerical results presented in §5 showed an excellent agreement with corresponding experimental results. The amplitude, wave length and velocity of both the leading wave and dispersive trailing waves were very well predicted by the model (except perhaps in the case of the rotating plate for which the shallow-water approximation $u(x,y,t) \simeq \overline{u}(x,t)$ was maybe inappropriate).

Therefore, we conclude that the nonlinear, dispersive, shallow water wave math

ematical model presented herein constitutes a very good representation for the generation and propagation of waves developed by a moving bed.

Further studies could be directed towards the treatment of problems in two spatial dimensions and on a more realistic representation of the moving bed in which the moving solid boundaries could be replaced by a sliding granular material.

REFERENCES

BORIS, J.P. 1981 A one-dimensional Lagrangian code for nearly incompressible flow In *Finite-difference Techniques For Vectorized Fluid Dynamics Calculations* (ed D.L. Book), 75–97.

BORIS, J.P. & BOOK, D.L 1973 Flux-Corrected Transport. I. SHASTA, a fluid transport algorithm that works. *J. Comput. Phys.* **11**, 38–69.

BORIS, J.P.,BOOK, D.L. & HAIN, K. 1975 Flux-Corrected Transport. II Generalizations of the method *J. Comput. Phys.* **18**, 248–283.

BORIS, J.P. & BOOK, D.L. 1976 Flux-Corrected Transport. III. Minimal error FCT algorithms. *J. Comput. Phys.* **20**, 397–431.

CARRIER,G.F. & GREENSPAN,H.P. 1958 Water waves of finite amplitude on a sloping beach. *J. Fluid Mech.* **4**, 97–109.

DAS, M.M. & WIEGEL, R.L. 1972 Waves generated by horizontal motion of a wall *J. Waterw. Harbors Div. ASCE.* **98**, No. WW1, 49–65

FRIEDRICHS, K.O 1948 On the derivation of the shallow water theory (appendix to 'The formation of breakers and bores' by J.J Stoker. *Commun Appl Math* **1**, 81–85.

HAMMACK, J L. 1973 A note on tsunamis: their generation and propagation in an ocean of uniform depth. *J. Fluid Mech.* **60**, 769–799.

HELFRICH, K.R. & MELVILLE, W.K. 1986 On long nonlinear internal waves over slope-shelf topography. *J. Fluid Mech.* **167**, 285–308.

HIBBERD, S. & PEREGRINE, D H. 1979 Surf and run-up on a beach a uniform bore. *J. Fluid Mech.* **95**, 323–345.

HUBER, A. 1982 Impulse waves in Swiss lakes as a result of rock avalanches and landslides. Experimental results for the prediction of the characteristic numbers of these waves. *Quatorzième Congrès des Grands Barrages*, Rio De Janeiro, 1982 455–475.

HUNT, B. 1988 Water waves generated by distant lanslides. *J. Hyd Res* **26**, 307 322

HWANG,L.S. & DIVOKY, D. 1970 Tsunami generation *J Geophys Res* **75** 6802–6817.

KELLER, J.B. 1948 The solitary wave and periodic waves in shallow water *Commun. Appl Math* **1**, 323–339

JOHNSON, R.S. 1972 Some numerical solutions of a variable-coefficient Korteweg-de Vries equation (with applications to solitary wave development on a shelf). *J. Fluid Mech.* **54**, 81-91.

KRANZER, H.C. & KELLER, J.B. 1959 Water waves produced by explosions. *J. Appl. Phys.* **30**, 398-407.

MADSEN, O S & MEI, C.C. 1969 The transformation of a solitary wave over an uneven bottom. *J. Fluid Mech.* **39**, 781-791.

NODA, E. 1970 Water waves generated by landslides. *J. Waterw. Harbors Div. ASCE* **96**, No. WW4,835-855.

NODA, E.K 1971 Water waves generated by a local surface disturbance. *J. Geophys. Res.* **76**, 7389-7400.

PEDERSEN, G. & GJEVIK, B. 1983 Run-up of solitary waves. *J. Fluid Mech.* **135**. 283-299.

PEREGRINE, D H 1967 Long waves on a beach. *J. Fluid Mech.* **27**, 815-827.

PEREGRINE, D.H. 1972 Equations for water waves and the approximation behind them In *Waves on Beaches and Resulting Sediment Transport* (ed. R.E. Meyer), Academic Press, 95-121.

PEREGRINE, D.H 1983 Breaking waves on beaches. *Ann. Rev. Fluid Mech.* **15**, 149-178.

PEYRET, R. & TAYLOR, T.D. 1983 *Computational Methods for Fluid Flow.* Springer-Verlag, 104-105.

PRINS, J.E. 1958 Characteristics of waves generated by a local disturbance. *Trans. Am Geophys Union* **39**, 865-874.

RANEY, D C. & BUTLER, H L. 1976 Landslide generated water wave model. *J. Hyd. Div. ASCE.* **102**,No.HY9, 1269-1282.

SABATIER, P.C. 1983 On water waves produced by ground motions. *J. Fluid Mech.* **126**, 27-58.

SANDER, J. 1988 Waves in lakes generated by rockslides. Internal report, Laboratory of Hydraulics, Hydrology and Glaciology (VAW) of the Swiss Federal Institute of Technology (ETH), Zürich.

SEABRA-SANTOS,F.G.,RENOUARD,D.P. & TEMPERVILLE, A.M. 1987 Numerical and experimental study of the transformation of a solitary wave over a shelf or isolated obstacle. *J. Fluid Mech.* **176**, 117-134.

SLINGERLAND, R.L. & VOIGHT, B. 1979 Occurences, properties, and predictive models of landslide-generated water waves. In *Rockslides and Avalanches, Vol 2* (ed. B. Voight),Elsevier, 317–397.

SPIELVOGEL, L.Q. 1975 Single-wave run-up on sloping beaches. *J. Fluid Mech.* **74**, 685–694.

SU, C.H. & GARDNER, C.S. 1969 Korteweg-de Vries equation and generalizations III. Derivation of the Korteweg-de Vries equation and Burgers equations *J Math. Phys.* **10**, 536–539.

TUCK, E.O. & HWANG, L S. 1972 Long wave generation on a sloping beach *J Fluid Mech.* **51**, 449–461.

URSELL, F. 1953 The long-wave paradox in the theory of gravity waves *Proc. Camb Phil. Soc.* **49**, 685–694.

VOIT, S.S. 1987 Tsunamis. *Ann. Rev. Fluid Mech.* **19**, 217–236.

WIEGEL, R.L. ET AL 1970 Water waves generated by landslides in reservoirs. *J Waterw. Harbors Div. ASCE* **96**, No.WW2, 307–333.

WEHAUSEN,J.V. & LAITONEN,E.V. 1960 Surface waves. In *Encyclopedia of Physics, Vol.9* (ed. S. Flugge), Springer-Verlag, 617-619.

WHITHAM, G.B. 1974 *Linear and Nonlinear Waves* Wiley-Interscience, New-York

WU, T.Y. 1981 Long waves in ocean and coastal waters. *J. Eng. Mech. Div ASCE* **107**, No. EM3, 501–522.

ZALESAK, S.T. 1979 Fully multidimensional Flux-Corrected Transport algorithms for fluids. *J. Comput. Phys* **31**, 335–362.

APPENDIX A

Validity of the shallow-water approximation

As pointed out in §2.1, the depth-averaging of the x-momentum equation shows that the 'shallow-water' approximation—which merely consists in replacing $u(x,y,t)$ by $\overline{u}(x,t)$—is only valid if $H(\overline{u^2}-\overline{u}^2)$ is small compared to the remaining terms of the equation

$$\frac{\partial}{\partial t}(H\overline{u})+\frac{\partial}{\partial x}[(H(\overline{u}^2+\overline{p})+H(\overline{u^2}-\overline{u}^2)]=p_b\frac{\partial h}{\partial x}. \tag{A1}$$

In other words, for $\overline{u^2}-\overline{u}^2$ to be negligible, we must show that it is of higher order than the expressions developed for p_b and $\overline{p}$. From (2.22) and (2.23), we see that the order of p_b and $\overline{p}$ is determined by the order of the factors α, β and γ (equation 2.20) in terms of which they are expressed. In the scaled equations (2.32) for α^*,β^* and γ^*, we saw that the higher order terms were $O(\epsilon^2\sigma^2)$. We therefore must determine under what conditions the difference $\overline{u^2}-\overline{u}^2$ becomes smaller than $O(\epsilon^2\sigma^2)$.

We first rewrite the approximation (2.12) for the vertical velocity as :

$$v\simeq -[h_t+\overline{u}h_x+(y+h)\overline{u}_x]. \tag{A2}$$

Combining (A2) with the irrotationality condition

$$\frac{\partial u}{\partial y}=\frac{\partial v}{\partial x}, \tag{A3}$$

we obtain

$$\frac{\partial u}{\partial y}=-(y+h)\overline{u}_{xx}-D \tag{A4}$$

where

$$D=h_{tx}+2h_x\overline{u}_x+h_{xx}\overline{u}.$$

Integration of (A4) from y to η gives:

$$u=u_s+\overline{u}_{xx}\left[\frac{H^2}{2}-\frac{(y+h)^2}{2}\right]+D(\eta-y) \tag{A5}$$

and from the definition (2.10) of the depth-averaged velocity, we have :

$$\overline{u}=\frac{1}{H}\int_{-h}^{\eta}u\,dy=u_s+\overline{u}_{xx}\frac{H^2}{3}+D\frac{H}{2}. \tag{A6}$$

Subtracting (A6) from (A5), we get

$$u - \overline{u} = \overline{u}_{xx}\left[\frac{H^2}{6} - \frac{(y+h)^2}{2}\right] + D\left[\frac{H}{2} - (y+h)\right]. \tag{A7}$$

Note that (A7) can be reduced to

$$u = \overline{u} - \left(y + \frac{h}{2}\right)[h_t + (h\overline{u})_x]_x - \left(\frac{y^2}{2} + \frac{h^2}{6}\right)\overline{u}_{xx} + O(\epsilon^2\sigma^2)$$

as in equation (2.36). Squaring u in equation (A7) gives

$$\begin{aligned} u^2 = \overline{u}^2 \quad &+ \quad 2\overline{u}\overline{u}_{xx}\left[\frac{H^2}{6} - \frac{(y+h)^2}{2}\right] + 2\overline{u}D\left[\frac{H}{2} - (y+h)\right] \\ &+ \quad \overline{u}_{xx}^2\left[\frac{H^4}{36} - \frac{H^2(y+h)^2}{6} + \frac{(y+h)^4}{4}\right] \\ &+ \quad 2\overline{u}_{xx}D\left[\frac{H^3}{12} - \frac{H(y+h)^2}{4} - \frac{H^2(y+h)}{6} + \frac{(y+h)^3}{2}\right] \\ &+ \quad D^2\left[\frac{H^2}{4} - \frac{H}{H}(y+h) + (y+h)^2\right] \end{aligned} \tag{A8}$$

and applying the definition (2.17) of $\overline{u^2}$ to (A8), we finally obtain

$$\overline{u^2} - \overline{u}^2 = \overline{u}_{xx}^2\frac{H^4}{45} + \overline{u}_{xx}D\frac{H^3}{12} + D^2\frac{H^2}{12}. \tag{A9}$$

For waves over a flat horizontal bed, $D = 0$ and (A9) reduces to

$$\overline{u^2} - \overline{u}^2 = \overline{u}_{xx}^2\frac{H^4}{45} \tag{A10}$$

as derived by Su & Gardner (1969). The difference $\overline{u^2} - \overline{u}^2$ as given by (A10) is $O(\epsilon^2\sigma^4)$ and is always negligible relative to the $O(\epsilon^2\sigma^2)$ pressure terms. Therefore, we conclude that there is no restriction on the applicability of the 'shallow-water' assumption $u(x, y, t) \simeq \overline{u}(x, t)$ for long waves over a flat bed.

However, if the bed is in motion, D is non-zero and depends on the slope and velocity of the bed. Assuming the orders of magnitude of the slope $\frac{\partial h}{\partial x}$ and velocity $\frac{\partial h}{\partial t}$ to be unknowns, we can use the scaling factors defined by (2.27) to write

$$D^* = -[\sigma\nu(h_t)_{x^*}^* + 2\epsilon\sigma\mu(h_x)^*\overline{u}_{x^*}^* + \epsilon\sigma\mu(h_x)_{x^*}^*\overline{u}^*] \tag{A11}$$

where it is assumed that μ and ν respectively represent the order of magnitude of the slope and velocity of the bed, i. e.

$$\frac{\partial h}{\partial x} = O(\mu),$$

$$\frac{\partial h}{\partial t} = O(\nu)$$

As mentionned before, the first term on the right hand side of (A9) will always be smaller than $O(\epsilon^2\sigma^2)$. Therefore, for the difference $\overline{u^2} - \overline{u}^2$ to be negligible, we need

$$\overline{u}_{12} D \frac{H^3}{12} < O(\epsilon^2\sigma^2) \tag{A12}$$

and

$$D^2 \frac{H^2}{12} < O(\epsilon^2\sigma^2). \tag{A13}$$

Using the scales defined in (2.27) and (A11), we see that the conditions (A12) and (A13) respectively imply :

$$\left.\begin{array}{r}(\epsilon\sigma^3)\nu \\ (\epsilon^2\sigma^3)\mu\end{array}\right\} < \epsilon^2\sigma^2$$

and

$$\left.\begin{array}{r}\sigma^2\nu^2 \\ (\epsilon^2\sigma^2)\mu^2 \\ (\epsilon\sigma^2)\mu\nu\end{array}\right\} < \epsilon^2\sigma^2$$

In the above inequalities, if we assume that the Ursell number is approximately equal to 1, i.e. $\epsilon \sim \sigma^2$, the most stringent restrictions on the bed's slope and velocity are found to be:

$$\mu < 1 \text{ and } \nu < \epsilon$$

The slope of the bed must then be

$$\frac{\partial h}{\partial x} < O(1) \tag{A14}$$

and its velocity

$$\frac{\partial h}{\partial t} < O(\epsilon) \tag{A15}$$

We conclude that for long waves generated by a moving bed, the approximation $u(x,y,t) \simeq \overline{u}(x,t)$ is only valid if the conditions (A14) and (A15) are satisfied. Nevertheless, the very wide range of realistic bed motions that can be described within the limits of (A14) and (A15) justifies the elimination of the term $H(\overline{u^2} - \overline{u}^2)$ from the governing momentum equation (2.24).

Finally, we observe that these conditions are satisfied by the scales

$$\frac{\partial h}{\partial x} = O(\sigma) \tag{A16}$$

$$\frac{\partial h}{\partial t} = O(\epsilon\sigma) \tag{A17}$$

which were assumed in the reduction of the governing equations to a 'Boussinesq' order in §2.2. With (A16) and (A17), the term D given by (A11) becomes $O(\epsilon\sigma^2)$ and from (A9) we therefore obtain

$$\overline{u^2} - \overline{u}^2 = O(\epsilon^2\sigma^4)$$

as in Wu (1981).

APPENDIX B

Derivation of the governing equations using an expansion method

The momentum equation (2.34) of 'Boussinesq' order was obtained by scaling and eliminating the higher-order terms from the 'complete' depth-averaged equation (2.24).

An alternative procedure can be used to derive equation (2.34) directly from the fundamental equations of motion. The method basically consists in expanding the dependent variables of the problem in terms of the small parameters σ and ϵ defined in §2.2. These expansions are then substituted in the equations of motion and the boundary conditions, and coefficients of like powers of σ and ϵ are equated. The original problem is then effectively split into simpler sub-problems each associated with a specific order of magnitude indicated by the power of the small factors σ and ϵ. The solution to the exact hydrodynamic problem can then be approximated by the sum of the solutions to these sub-problems, from order zero up to the required order.

The above procedure, which is based on the introduction of one or more small parameter(s) in view of simplifying the solution of a complex problem, is called a *perturbation method*. The formal application of perturbation methods in the theory of shallow-water waves has been described by Friedrichs (1948) and Keller (1948). We here extend the procedure described by Peregrine (1967) to account for the effects of moving bed. The following derivation is limited to two spatial dimensions, although the treatment of three dimensions does not present further difficulties.

We recall from §2.1 the fundamental equations of motion in dimensionless form (2.1) - (2.4).

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{B1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x}, \tag{B2}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} - 1, \tag{B3}$$

$$\frac{\partial u}{\partial y} = \frac{\partial v}{\partial x} \tag{B4}$$

and write the boundary conditions (2.5)-(2.7) as :

$$v = \frac{\partial \zeta}{\partial t} + u\frac{\partial \zeta}{\partial x} \text{ at } y = \zeta(x,t), \tag{B5}$$

$$v = -\frac{\partial h}{\partial t} - u\frac{\partial h}{\partial x} \text{ at } y = -h(x,t), \tag{B6}$$

$$p = 0 \text{ at } y = \zeta(x,t) \tag{B7}$$

where the wave amplitude is now denoted by ζ (as opposed to η in §2). Using the kinematic boundary conditions (B5) and (B6), we integrate the continuity equation (B1) over the depth to obtain

$$\frac{\partial \zeta}{\partial t} + \frac{\partial Q}{\partial x} = -\frac{\partial h}{\partial t} \tag{B8}$$

where

$$Q = \int_{-h}^{\eta} u\, dy \tag{B9}$$

As in §2.2, the independent variables x and t are scaled as

$$(x,t) = \frac{1}{\sigma}(x_1, t_1) \tag{B10}$$

where the scaled variables are here denoted by the subscript 1 (an asterisk was used in §2.2). We also assume that

$$\frac{\partial h}{\partial t} = \left(\frac{\partial h}{\partial t}\right)_0 + \epsilon\left(\frac{\partial h}{\partial t}\right)_1 + \epsilon^2\left(\frac{\partial h}{\partial t}\right)_2 \tag{B11}$$

The dependent variables ζ, u, p and Q are expanded as :

$$\zeta = \zeta_0 + \epsilon\zeta_1 + \epsilon^2\zeta_2 + \tag{B12a}$$

$$u = u_0 + \epsilon u_1 + \epsilon^2 u_2 + \cdots \tag{B12b}$$

$$p = p_0 + \epsilon p_1 + \epsilon^2 p_2 + \cdots \tag{B12c}$$

$$Q = Q_0 + \epsilon Q_1 + \epsilon^2 Q_2 + \cdots \tag{B12d}$$

and for (B1) to be consistent, we write

$$v = \sigma(v_0 + \epsilon v_1 + \cdots). \tag{B12e}$$

At order zero (i.e. $\sigma = \epsilon = 0$), the solution will be taken to be still water, so that

$$p_0 = -y \tag{B13}$$

which implies that ζ_0, u_0, v_0, $(\frac{\partial h}{\partial t})_0$ and Q_0 are all equal to zero. The still water solution was implicitely included in the derivation used in §2.2 when we wrote $\overline{u} = O(\epsilon)$, $\eta = O(\epsilon)$ and $h_t = O(\epsilon\sigma)$. Note that if (B11) and the expansions (B12) are actually substituted in the equations (B1)-(B9) the zero-order equations turn out to be

$$\frac{\partial \zeta_0}{\partial t_1} + \frac{\partial}{\partial x_1}[(h+\zeta_0)u_0] = -\left(\frac{\partial h}{\partial t}\right)_0,$$

$$\frac{\partial u_0}{\partial t_1} + u_0\frac{\partial u_0}{\partial x_1} + \frac{\partial \zeta_0}{\partial x_1} = 0$$

which are the well-known Airy (finite-amplitude, shallow-water) equations.

The first order solution is derived as follows. At $O(\epsilon) = O(\sigma^2)$, the irrotationality condition (B1) becomes

$$\frac{\partial u_1}{\partial y} = 0$$

since $v_0 = 0$. Hence,

$$u_1 = u_1(x_1, t_1) \tag{B14}$$

and therefore, from (B9) :

$$Q_1 = hu_1. \tag{B15}$$

The $O(\epsilon)$ y-momentum equation is

$$\frac{\partial p_1}{\partial y} = 0$$

and from the stress free boundary condition (B7)

$$p_0 + \epsilon p_1 = 0 \text{ at } y = \zeta_0 + \epsilon\zeta_1,$$

we obtain

$$p_1 = \zeta_1. \tag{B16}$$

Substitution of (B14)-(B16) in (B2) and (B8) gives the following first-order equations

$$\frac{\partial \zeta_1}{\partial t_1} + \frac{\partial (h u_1)}{\partial x_1} = - \left(\frac{\partial h}{\partial t} \right)_1 , \tag{B17}$$

$$\frac{\partial u_1}{\partial t_1} + \frac{\partial \zeta_1}{\partial x_1} = 0 \tag{B18}$$

which are the linearized long-wave equations. The first-order vertical velocity component v_1 is deduced by integrating the first-order continuity equation (B1) with respect to y:

$$v_1 = - y \frac{\partial u_1}{\partial x_1} + V_1(x_1, t_1)$$

where $V_1(x_1, t_1)$ is an arbitrary function arising from the integration. Using the first order kinematic boundary condition at the bed

$$v_1 = - \left(\frac{\partial h}{\partial t} \right)_1 - u_1 \frac{\partial h}{\partial x_1} \text{ at } y = - h,$$

we deduce

$$V_1 = - \left(\frac{\partial h}{\partial t} \right)_1 - u_1 \frac{\partial h}{\partial x_1} - h \frac{\partial u_1}{\partial x_1}$$

and can then write

$$v_1 = - \left(\frac{\partial h}{\partial t} \right)_1 - \frac{\partial (h u_1)}{\partial x_1} - y \frac{\partial u_1}{\partial x_1} \tag{B19}$$

The above procedure is repeated in order to derive the second-order equations. At $O(\epsilon^2) = O(\epsilon \sigma^2)$, irrotationality (B4) gives

$$\frac{\partial u_2}{\partial y} = \frac{\partial v_1}{\partial x_1}$$

in which (B19) can be substituted to obtain

$$\frac{\partial u_2}{\partial y} = - \frac{\partial}{\partial x_1} \left(\frac{\partial h}{\partial t} \right)_1 - \frac{\partial^2 (h u_1)}{\partial x_1^2} - y \frac{\partial^2 u_1}{\partial x_1^2}.$$

Integration with respect to y leads to

$$u_2 = U_2(x_1, t_1) - y \frac{\partial}{\partial x_1} \left(\frac{\partial h}{\partial t} \right)_1 - y \frac{\partial^2 (h u_1)}{\partial x_1^2} - \frac{y^2}{2} \frac{\partial^2 u_1}{\partial x_1^2} \tag{B20}$$

where $U_2(x_1, t_1)$ is an arbitrary function of integration. The $O(\epsilon^2)$ y-momentum equation

$$\frac{\partial p_2}{\partial y} = -\frac{\partial v_1}{\partial t_1}$$

now includes the effect of the local vertical acceleration of the fluid. Hence, from (B19), we have

$$\frac{\partial p_2}{\partial y} = \frac{\partial}{\partial t_1}\left(\frac{\partial h}{\partial t}\right)_1 + \frac{\partial^2 (h u_1)}{\partial t_1 \partial x_1} + y\frac{\partial^2 u_1}{\partial t_1 \partial x_1}$$

and integration with respect to y gives

$$p_2 = \eta_2 + y\frac{\partial}{\partial t_1}\left(\frac{\partial h}{\partial t}\right)_1 + y\frac{\partial^2 (h u_1)}{\partial t_1 \partial x_1} + \frac{y^2}{2}\frac{\partial^2 u_1}{\partial t_1 \partial x_1} \tag{B21}$$

in which the boundary condition (B7)

$$p = p_0 + \epsilon p_1 + \epsilon^2 p_2 = 0 \text{ at } y = \zeta_0 + \epsilon\zeta_1 + \epsilon^2\zeta_2$$

has been used to determine the arbitrary function arising from the integration. From the definition (B9) of Q, we also have

$$\epsilon^2 Q_2 = \int_0^{\epsilon\zeta_1} \epsilon u_1\, dy + \int_{-h}^{0} \epsilon^2 u_2\, dy$$

so that, from (B20)

$$Q_2 = \zeta_1 u_1 + h U_2 + \frac{h^2}{2}\frac{\partial}{\partial x_1}\left[\left(\frac{\partial h}{\partial t}\right)_1 + \frac{\partial (h u_1)}{\partial x_1}\right] - \frac{h^3}{6}\frac{\partial^2 u_1}{\partial x_1^2}. \tag{B22}$$

Substitution of (B20) (B22) in (B2) and (B8) gives the following second order equations

$$\frac{\partial \zeta_2}{\partial t_1} + \frac{\partial Q_2}{\partial x_1} = -\left(\frac{\partial h}{\partial t}\right)_2, \tag{B23}$$

$$\frac{\partial U_2}{\partial t_1} + u_1\frac{\partial u_1}{\partial x_1} + \frac{\partial \zeta_2}{\partial x_1} = 0. \tag{B24}$$

As pointed out by Peregrine (1967), the first-order linearized long-wave equations (B17) and (B18) are only valid for small values of t_1. After only a short propagation time, the second order effects of amplitude and frequency dispersion included in (B23) and (B21) effectively become first-order effects. The final governing equations must

therefore include "first-order variables incorporating second-order terms". For the wave amplitude, we let

$$\eta = \epsilon\zeta_1 + \epsilon^2\zeta_2 \tag{B25}$$

For the horizontal fluid velocity, we introduce the depth averaged velocity which we define as

$$\begin{aligned}\overline{u} &= \frac{1}{h+\eta}(\epsilon Q_1 + \epsilon^2 Q_2) \\ &= \epsilon u_1 + \epsilon^2\left\{U_2 + \frac{h}{2}\frac{\partial}{\partial x_1}\left[\left(\frac{\partial h}{\partial t}\right)_1 + \frac{\partial(hu_1)}{\partial x_1}\right] - \frac{h^2}{6}\frac{\partial^2 u_1}{\partial x_1^2}\right\}.\end{aligned} \tag{B26}$$

Note that the choice of $\overline{u}$ to incorporate second-order effects is not the only alternative. The final governing equations can also be formulated in terms of the velocity at the bed (Madsen & Mei 1969) or by means of the velocity at $y = 0$.

$$u(x,0,t) = \epsilon u_1 + \epsilon^2 U_2.$$

Combining (B17) and (B18) with (B23) (B26), and going back to the original variables, we finally obtain

$$\frac{\partial\eta}{\partial t} + \frac{\partial}{\partial x}[(h+\eta)\overline{u}] = -\frac{\partial h}{\partial t}, \tag{B27}$$

$$\frac{\partial\overline{u}}{\partial t} + \overline{u}\frac{\partial\overline{u}}{\partial x} + \frac{\partial\eta}{\partial x} = \frac{h}{2}\frac{\partial^2}{\partial t\partial x}\left[\frac{\partial h}{\partial t} + \frac{\partial(h\overline{u})}{\partial x}\right] - \frac{h^2}{6}\frac{\partial^3\overline{u}}{\partial t\partial x^2}. \tag{B28}$$

Equations (B27) and (B28) are respectively identical to equations (2.9) and (2.31) derived in §2. The depth-averaged continuity equation (B27) is exact while the depth-averaged momentum equation (B28) has an error term of order $O(\epsilon^2\sigma^3)$

APPENDIX C

Computer programs

Program WALL

Purpose of the program

Modelling of nonlinear, dispersive, shallow-water waves generated in a channel by a moving vertical wall.

Definition of the constants

```
        n : Total number of spatial grid points (i = 0,1,...n).
       dx : Initial spatial mesh size at t = 0.
       dt : Temporal mesh size.
   totald : Total length of the channel.
     tmax : Total duration of the simulation.
    vstar : Amplitude of the sinusoidal wall velocity.
    lastT : Time at which the motion of the wall stops.
    lastX : Final x-coordinate of the moving wall.
g1,...,g8 : Location of the eight gauges measuring wave amplitude.
```

Definition of the global variables

```
    etha1 : Height of the free surface above its undisturbed level at
            time t.
    etha2 : Height of the free surface above its undisturbed level at
            time t + dt.
       u1 : Depth-averaged velocity at time t.
       u2 : Depth-averaged velocity at time t + dt.
       x1 : Position of the spatial grid points at time t.
       x2 : Position of the spatial grid points at time t + dt.
        t : Time.
        i : Loop counter.
     iter : Number of time steps involved in the computation.
gauge1...
   gauge8 : Grid points corresponding to the gauges' location.
     disk : Text file used to store the results.
```

Program listing

```
Program WALL (input,output);

uses crt,graph3;

{*************Experimental conditions : WAVE.DAT 139*******************}

CONST
         n = 500;
        dx = 0.25;
        dt = 0.25;
    totald = 125;
      tmax = 125;
     vstar = 0.2;
     lastT = 13.12;
     lastX = 1.67;
        g1 = 3.7;
        g2 = 14.1;
        g3 = 24.4;
        g4 = 34.7;
        g5 = 45.1;
        g6 = 55.4;
        g7 = 65.7;
        g8 = 76.1;

TYPE
   vector = array [0..n] of real;

VAR
              etha1,etha2,u1,u2 : vector;
                          x1,x2 : vector;
                              t : real;
                        i,iter : integer;
    gauge1,gauge2,gauge3,gauge4 : integer;
    gauge5,gauge6,gauge7,gauge8 : integer;
                           disk : text;


Function VPISTON(t:real):real;

begin
   if t <= lastT
      THEN
        vpiston:= vstar * sin(pi*t/lastT)
      ELSE
        vpiston:= 0.0;
end;
```

```
Function ACCPISTON(t:real):real;

begin
   if t <= lastT
      THEN
        accpiston:= (vpiston(t+dt)-vpiston(t-dt))/2/dt
      ELSE
        accpiston:= 0.0;
end;


Procedure INITIAL_CONDITIONS;

begin
   for i:= 0 to n do etha1[i]:= 0.0;

   for i:= 0 to n do u1[i]:= 0.0;

   for i:= 0 to n do x1[i]:= i*dx;

end;


Procedure CONTINUITY_PREDICTOR;

begin
   for i:= 1 to n-1 do
       etha2[i]:= etha1[i] - dt*(1+etha1[i])*(u1[i+1]-u1[i-1])
                                             /(x1[i+1]-x1[i-1]);
end;


Procedure CONTINUITY_CORRECTOR;

begin
   for i:= 1 to n-1 do
       etha2[i]:= etha1[i] - dt*(1+0.5*etha1[i]+0.5*etha2[i])
                           *0.5*((u1[i+1]-u1[i-1])/(x1[i+1]-x1[i-1])
                                +(u2[i+1]-u2[i-1])/(x2[i+1]-x2[i-1]) );
end;



Procedure ETHA_BC;

begin
   etha2[0]:= etha2[1] + dx*ACCPISTON(t);
   etha2[n]:= etha2[n-1];
end;
```

```
Procedure TRIDIAG(a,b,c:vector; var x:vector; rhs:vector; first,last:integer);

var
   mult : real;
      i : integer;

begin
   for i:= first+1 to last do
       begin
          mult:= a[i]/b[i-1];
          b[i]:= b[i] - mult * c[i-1];
          rhs[i]:= rhs[i] - mult * rhs[i-1];
       end;
   x[last]:= rhs[last]/b[last];
   for i:= last-1 downto first do x[i]:= ( rhs[i] - c[i]*x[i+1] ) / b[i];
end;



Procedure MOMENTUM;

var diag1,diag2,diag3,rhs : vector;

begin
   for i:= 1 to n-1 do
       begin
          diag1[i]:= -2/3/(sqr(x2[i+1]-x2[i])+sqr(x2[i]-x2[i-1]));
          diag2[i]:= 1 + 4/3/(sqr(x2[i+1]-x2[i])+sqr(x2[i]-x2[i-1]));
          diag3[i]:= -2/3/(sqr(x2[i+1]-x2[i])+sqr(x2[i]-x2[i-1]));

          if i = 1 THEN
             rhs[i]:= u1[i] - 2/3/(sqr(x1[i+1]-x1[i])+sqr(x1[i]-x1[i-1]))
                              *(u1[i+1]-2*u1[i]+u1[i-1])
                            - dt*0.5
                            *( (etha2[i+1]-etha2[i-1])/(x2[i+1]-x2[i-1])
                              +(etha1[i+1]-etha1[i-1])/(x1[i+1]-x1[i-1]) )
                            - diag1[i]*VPISTON(t)
                  ELSE
             rhs[i]:= u1[i] - 2/3/(sqr(x1[i+1]-x1[i])+sqr(x1[i]-x1[i-1]))
                              *(u1[i+1]-2*u1[i]+u1[i-1])
                            - dt*0.5
                            *( (etha2[i+1]-etha2[i-1])/(x2[i+1]-x2[i-1])
                              +(etha1[i+1]-etha1[i-1])/(x1[i+1]-x1[i-1]) );
       end;
   TRIDIAG(diag1,diag2,diag3,u2,rhs,1,n-1);
end;



Procedure APPROX_INTERFACES;

begin
   for i:= 0 to n do x2[i]:= x1[i] + dt*u1[i];
end;
```

```
Procedure U_BC;

begin
   u2[0]:= VPISTON(t);
   u2[n]:= 0.0;
end;


Procedure INTERFACES;

begin
   x2[0]:= x1[0] + 0.5*dt*(u1[0]+u2[0]);
   for i:= 1 to n do
       begin
          x2[i]:= x1[i] + 0.5*dt*(u1[i]+u2[i]);
          if (x2[i-1]<=g1)  and (x2[i]>g1)
             then if (g1-x2[i-1]) <= (x2[i]-g1) then gauge1:= i-1
                                                else gauge1:= i;

          if (x2[i-1]<=g2)  and (x2[i]>g2)
             then if (g2-x2[i-1]) <= (x2[i]-g2) then gauge2:= i-1
                                                else gauge2:= i;

          if (x2[i-1]<=g3)  and (x2[i]>g3)
             then if (g3-x2[i-1]) <= (x2[i]-g3) then gauge3:= i-1
                                                else gauge3:= i;

          if (x2[i-1]<=g4)  and (x2[i]>g4)
             then if (g4-x2[i-1]) <= (x2[i]-g4) then gauge4:= i-1
                                                else gauge4:= i;

          if (x2[i-1]<=g5)  and (x2[i]>g5)
             then if (g5-x2[i-1]) <= (x2[i]-g5) then gauge5:= i-1
                                                else gauge5:= i;

          if (x2[i-1]<=g6)  and (x2[i]>g6)
             then if (g6-x2[i-1]) <= (x2[i]-g6) then gauge6:= i-1
                                                else gauge6:= i;

          if (x2[i-1]<=g7)  and (x2[i]>g7)
             then if (g7-x2[i-1]) <= (x2[i]-g7) then gauge7:= i-1
                                                else gauge7:= i;

          if (x2[i-1]<=g8)  and (x2[i]>g8)
             then if (g8-x2[i-1]) <= (x2[i]-g8) then gauge8:= i-1
                                                else gauge8:= i;

       end;
end;

Procedure DRAWBED;

var x1 : real;

begin
   HiRes;
   GraphWindow(50,40,589,159);
   GoToXY(8,23);writeln('TIME = ',t:5:2,'
   piston VELOCITY = ',VPISTON(t):5:3,'    ACC = ',ACCPISTON(t):5:3);

   draw(round(x2[0]/totald*540),0,round(x2[0]/totald*540),100,1);

   x1:= lastX/totald * 540;
   draw(round(x1),100,540,100,1);
end;
```

```
Procedure DRAWSURFACE(v:vector);

var x1,y1 : real;

begin
   for i:= 0 to n do
       begin
          x1:= x2[i]/totald * 540;
          y1:= v[i] * 60;
          plot(round(x1),70-round(y1),1);
       end;
end;



BEGIN   {main}
   ClrScr;
   t:=0.0;
   iter:= 0;
   vel:= 0.0;

   assign(disk,'B:RES139.DOC');
   rewrite(disk);

   INITIAL_CONDITIONS;

   While t < tmax do
      begin
         t:= t + dt;
         iter:= iter+1;

         CONTINUITY_PREDICTOR;
         ETHA_BC;
         APPROX_INTERFACES;
         U_BC;
         MOMENTUM;
         MOVE_CELLS;
         CONTINUITY_CORRECTOR;
         ETHA_BC;

         DRAWBED;
         DRAWSURFACE(etha2);

         if (iter mod 2) = 0 THEN
             begin
                writeln(disk,iter div 2);
                writeln(disk,etha2[gauge1]);
                writeln(disk,etha2[gauge2]);
                writeln(disk,etha2[gauge3]);
                writeln(disk,etha2[gauge4]);
                writeln(disk,etha2[gauge5]);
                writeln(disk,etha2[gauge6]);
                writeln(disk,etha2[gauge7]);
                writeln(disk,etha2[gauge8]);
                writeln(disk);
             end;
```

```
            for i:= 0 to n do etha1[i]:= etha2[i];
            for i:= 0 to n do u1[i]:= u2[i];
            for i:= 0 to n do x1[i]:= x2[i];

        end;
    close(disk);
END.
```

Program SUBWED

Purpose of the program

Modelling of nonlinear, dispersive, shallow-water waves generated in a channel by the motion of a submerged wedge.

Definition of the constants

n : Total number of spatial grid points (i = 0,1,...n).
ni : n - 1.
dx : Spatial mesh size (constant).
dt : Temporal mesh size.
tmax : Total duration of the simulation.
endmotion : Total duration of the bed motion.
vstar : Amplitude of the horizontal, sinusoidal velocity of the wave generating device.
slope : Constant slope of the submerged wedge.
xsin : Parameter defining the sinusoidal velocity of the bed, xsin = τ_2-τ_1.
eu : Implicitness parameter for the depth-averaged velocity.
en : Implicitness parameter for the wave amplitude.
eh : Implicitness parameter for the bed position.

Definition of the global variables

h1 : Position of the bed at time t.
hmid : Position of the bed at time t + dt/2.
h2 : Position of the bed at time t + dt.
etha1 : Height of the free surface above its undisturbed level at time t.
etha2 : Height of the free surface above its undisturbed level at time t + t.
u1 : Depth-averaged velocity at time t.
u2 : Depth-averaged velocity at time t + dt.
diag1...
diag3 : Diagonals of the tridiagonal matrix system (section 3.2.2).
rhs : Right hand side of the tridiagonal matrix system (section 3.2.2).
dhdt : Bed velocity.
acc : Bed acceleration.
slope1 : Slope of the bed at time t.
slope2 : Slope of the bed at time t + dt.
A...F : Groupings of variables used in the implicit solution of the momentum equation.
t : Time.

```
      vel1 : Horizontal velocity of the wedge at time t.
    velmid : Horizontal velocity of the wedge at time t + dt/2.
      vel2 : Horizontal velocity of the wedge at time t + dt.
      disp : Displacement of the wedge between t and t + dt.
     displ : Displacement of the wedge between t and t + dt/2.
 totaldisp : Cumulative displacement of the wedge at time t + dt.
totaldispl : Cumulative displacement of the wedge at time t + dt/2.
         i : Loop counter.
      iter : Number of time steps involved in the computation.
    bottom : Grid point corresponding to the foot of the wedge at t + dt.
   bottoml : Grid point corresponding to the foot of the wedge at t + dt/2.
      disk : Text file used to store the results.
```

Program listing

```
Program SUBWED (input,output);

uses crt,graph3;

{***************Experimental conditions : WAVE.DAT 107**********************}

CONST
           n = 600;
          n1 = 599;
          dx = 0.25;
          dt = 0.25;
        tmax = 125;
   endmotion = 13.116;
       vstar = 0.2;
       slope = 0.268;
        xsin = 1.0;  {  x range of sinus velocity  }
          eu = 0.5;
          en = 0.5;
          eh = 0.5;

TYPE
      vector = array [0..n] of real;

VAR
               h1,h2,hmid,etha1,etha2,u1,u2 : vector;
                     diag1,diag2,diag3,rhs : vector;
                    dhdt,acc,slope1,slope2 : vector;
                                A,B,C,D,E,F : real;
                                          t : real;
                              vel1,velmid,vel2 : real;
            disp,displ,totaldisp,totaldispl : real;
                      i,bottom,bottoml,iter : integer;
                                       disk : text;
```

```
Procedure U_BC;

begin
   u2[0]:= 0.0;
   u2[n]:= 0.0;
end;


Procedure INITIAL_CONDITIONS;

begin
   for i:= 0 to n do
       begin
          h1[i]:= 1.0;
          ethai[i]:= 0.0;
          ui[i]:= 0.0;
       end;
end;


Procedure MOVE_BED;

begin
   for i:= 0 to bottom do
       h2[i]:= 1.0 - slope * (totaldisp-i*dx);
   for i:= 0 to bottom1 do
       hmid[i]:= 1.0 - slope * (totaldisp1-i*dx);

   for i:= bottom+1 to n do
       h2[i]:= 1.0;
   for i:= bottom1+1 to n do
       hmid[i]:= 1.0;
end;



Procedure BED_IS_MOVING;

begin
   vel1:= vstar * sin( 0.5*pi - (0.5*endmotion-(t-dt))*xsin/endmotion*pi );
   velmid:= vstar * sin( 0.5*pi - (0.5*endmotion-(t-0.5*dt))*xsin/endmotion*pi );
   vel2:= vstar * sin( 0.5*pi - (0.5*endmotion-t)*xsin/endmotion*pi );

   disp1:= 0.5*(vel1+velmid) * dt/2;
   totaldisp1:= totaldisp + disp1;
   bottom1:= trunc(totaldisp1/dx);

   disp:= 1/3*(vel1+velmid+vel2) * dt;
   totaldisp:= totaldisp + disp;
   bottom:= trunc(totaldisp/dx);

   MOVE_BED;
```

```
      if bottom = 0
         THEN
           begin
              acc[0]:= (h2[0]-2*hmid[0]+h1[0]) /sqr(0.5*dt);
              for i:= 1 to n do acc[i]:= 0.0;
           end
         ELSE
           begin
              for i:= 0 to bottom-1 do
                  acc[i]:= (h2[i]-2*hmid[i]+h1[i]) /sqr(0.5*dt);
              acc[bottom]:= acc[bottom-1];
              for i:= bottom+1 to n do
                  acc[i]:= 0.0;
           end;

     for i:= 0 to n do
         begin
            if h1[i] = 1.0 THEN slope1[i]:= 0.0
                           ELSE slope1[i]:= slope;

            if h2[i] = 1.0 THEN slope2[i]:= 0.0
                           ELSE slope2[i]:= slope;

            dhdt[i]:= (h2[i]-h1[i])/dt;
         end;
  end;



  Procedure STATIONARY_BED;

  begin
     if (t-endmotion) <= dt then
        begin
           sound(100);
           delay(1000);
           NoSound;
           for i:= 0 to n do acc[i]:= 0.0;
           for i:= 0 to n do
               begin
                  if h2[i] = 1.0
                     THEN
                       begin
                          slope1[i]:= 0.0;
                          slope2[i]:= 0.0;
                       end
                     ELSE
                       begin
                          slope1[i]:= slope;
                          slope2[i]:= slope;
                       end;

                  dhdt[i]:= 0.0;
               end;
        end;
  end;
```

```
Procedure CONTINUITY_PREDICTOR;

begin
   for i:= 1 to n1 do
       begin
          etha2[i]:= etha1[i] - dt*( ( 0.5*(h1[i]+h2[i])+etha1[i] )
                                              * (u1[i+1]-u1[i-1])/2/dx
                                   + u1[i]*(  0.5*(slope1[i]+slope2[i])
                                            + (etha1[i+1]-etha1[i-1])/2/dx )
                                   + dhdt[i]  );
       end;
end;


Procedure ETHA_BC_PRED;

begin
   if t <= endmotion
      THEN
        etha2[0]:= etha2[1] - dx * (h2[0]/2*(h2[0]-h1[0])/dt
                                   *(u1[2]-2*u1[1]+u1[0])/sqr(dx))
      ELSE
        etha2[0]:= etha2[1];
   etha2[n]:= etha2[n1];
end;


Procedure CONTINUITY_CORRECTOR;

begin
   for i:= 1 to n1 do
       begin
          etha2[i]:= etha1[i] - dt*( ( 0.5*(h1[i]+h2[i])+etha1[i] )
                                    *0.25*(u2[i+1]-u2[i-1]+u1[i+1]-u1[i-1])/dx
                                   +0.5*(u1[i]+u2[i])
                                       * ( 0.5*(slope1[i]+slope2[i])
                                            + (etha1[i+1]-etha1[i-1])/2/dx )
                                   + dhdt[i]  );
       end;
end;


Procedure ETHA_BC_CORR;

begin
   if t <= endmotion
      THEN
        etha2[0]:= etha2[1] - dx * (h2[0]/2*(h2[0]-h1[0])/dt
                                   *(u2[2]-2*u2[1]+u2[0])/sqr(dx))
      ELSE
        etha2[0]:= etha2[1];
   etha2[n]:= etha2[n1];
end;
```

```
Procedure TRIDIAG(a,b,c:vector; var x:vector; rhs:vector; first,last:integer);

var
   mult : real;
      i : integer;

begin
   for i:= first+1 to last do
       begin
          mult:= a[i]/b[i-1];
          b[i]:= b[i] - mult * c[i-1];
          rhs[i]:= rhs[i] - mult * rhs[i-1];
       end;
   x[last]:= rhs[last]/b[last];
   for i:= last-1 downto first do x[i]:= ( rhs[i] - c[i]*x[i+1] ) / b[i];
end;



Procedure MOMENTUM_Fxt;

begin
   for i:= 1 to n1 do
       begin
          u2[i]:= u1[i];

          A:= ( (1-eh)*sqr(h1[i]) + eh*sqr(h2[i]) )/3/sqr(dx)/dt;
          B:= ( (1-eh)*h1[i]*slope1[i] + eh*h2[i]*slope2[i] )/2/dx/dt;
          C:= (slope2[i] - slope1[i])/dt;

          D:= (1-eh)*h1[i]/2*(h1[i+1]-2*h1[i]+h1[i-1])/sqr(dx)
                + eh*h2[i]/2*(h2[i+1]-2*h2[i]+h2[i-1])/sqr(dx);
          E:= ((h2[i+1]-2*h2[i]+h2[i-1])-(h1[i+1]-2*h1[i]+h1[i-1]))/sqr(dx)/dt;

          F:= ((1-en)*(etha1[i+1]-etha1[i-1])+en*(etha2[i+1]-etha2[i-1]))/2/dx;


          diag1[i]:= -eu*u2[i]/2/dx - A + B + eu*h2[i]/2/dx * C
                     - eu*h2[i]/2/sqr(dx)*dhdt[i];
          diag2[i]:= 1/dt + 2*A - 1/dt*D - eu*h2[i]/2*E
                     + eu*h2[i]/sqr(dx)*dhdt[i];
          diag3[i]:=  eu*u2[i]/2/dx - A - B - eu*h2[i]/2/dx * C
                     - eu*h2[i]/2/sqr(dx)*dhdt[i];

          rhs[i]:= u1[i-1] *((1-eu)*u1[i]/2/dx - A + B - (1-eu)*h1[i]/2/dx*C
                             +(1-eu)*h1[i]/2/sqr(dx)*dhdt[i] )
                   + u1[i] *(1/dt + 2*A - 1/dt*D + (1-eu)*h1[i]/2*E
                             -(1-eu)*h1[i]/sqr(dx)*dhdt[i] )
                  +u1[i+1] *(-(1-eu)*u1[i]/2/dx - A - B +(1-eu)*h1[i]/2/dx*C
                             +(1-eu)*h1[i]/2/sqr(dx)*dhdt[i] )
                   - F
                   + ((1-eh)*h1[i]+eh*h2[i])/2 * (acc[i+1]-acc[i-1])/2/dx;
       end;
   TRIDIAG(diag1,diag2,diag3,u2,rhs,1,n1);
end;
```

```
Procedure MOMENTUM_Fx;

begin
   for i:= 1 to n1 do
       begin
          u2[i]:= u1[i];

          A:= sqr(h2[i])/3/sqr(dx)/dt;
          B:= h2[i]*slope2[i]/2/dx/dt;
          D:= h2[i]/2*(h2[i+1]-2*h2[i]+h2[i-1])/sqr(dx);
          F:= ((1-en)*(etha1[i+1]-etha1[i-1])+en*(etha2[i+1]-etha2[i-1]))/2/dx;

          diag1[i]:= -eu*u2[i]/2/dx - A + B;
          diag2[i]:= 1/dt + 2*A - 1/dt*D;
          diag3[i]:=  eu*u2[i]/2/dx - A - B;

          rhs[i]:=   u1[i-1] *((1-eu)*u1[i]/2/dx - A + B)
                   + u1[i] *(1/dt + 2*A - 1/dt*D)
                   + u1[i+1] *(-(1-eu)*u1[i]/2/dx - A - B)
                   - F;

       end;
   TRIDIAG(diag1,diag2,diag3,u2,rhs,1,n1);
end;


Procedure MOMENTUM;

begin
   if t <= endmotion THEN MOMENTUM_Fxt
                     ELSE MOMENTUM_Fx;
end;



Procedure DRAWBED(v:vector; no_points:integer);

var x1,x2,y1,y2 : real;
          color : integer;

begin
   HiRes;
   GraphWindow(50,40,589,159);
   GoToXY(35,20);writeln('t = ',t:6:3);
{   for i:= 0 to no_points do draw(round(i/no_points*540),90,
                                  round(i/no_points*540),20,1);}
   for i:= 0 to no_points-1 do
       begin
          x1:= i/no_points * 540;
          y1:= v[i] * 30;
          x2:= (i+1)/no_points * 540;
          y2:= v[i+1] * 30;
          draw(round(i/no_points*540),60,round(i/no_points*540),0,1);
          draw(round(x1),60+round(y1),round(x2),60+round(y2),1);
       end;
end;
```

```
Procedure DRAWSURFACE(v:vector; no_points:integer);

var x1,x2,y1,y2 : real;

begin
   for i:= 0 to no_points-1 do
       begin
          x1:= i/no_points * 540;
          y1:= v[i] * 300;
          x2:= (i+1)/no_points * 540;
          y2:= v[i+1] * 300;
          draw(round(x1),60-round(y1),round(x2),60-round(y2),1);
       end;
end;



Procedure FCT(var vt:vector);

const sigma = 0.2;

var
        vtd,delta,Dc : vector;
     D1,sign,max,min : real;

begin
   delta[0]:= 0.0;   delta[n-1]:= 0.0;
   for i:= 1 to n-1 do
       vtd[i]:= vt[i] + sigma*(vt[i+1] - 2.0*vt[i] + vt[i-1]);
   for i:= 1 to n-2 do
       delta[i]:= vtd[i+1] - vtd[i];
   for i:= 1 to n-2 do
       begin
          if delta[i] > 0.0 then sign:= 1.0;
          if delta[i] = 0.0 then sign:= 0.0;
          if delta[i] < 0.0 then sign:=-1.0;
          D1:= (vt[i+1] - vt[i])/8.0;
          if sign*delta[i-1] <= sign*delta[i+1]
                        then   min:= sign*delta[i-1]
                        else   min:= sign*delta[i+1];
          if abs(D1) <= min
                        then   min:= abs(D1);
          if 0.0 >= min
                        then max:= 0.0
                        else max:= min;
          Dc[i]:= sign*max;
       end;
   for i:= 2 to n-2 do
       vt[i]:= vtd[i] - (Dc[i] - Dc[i-1]);
end;
```

```
BEGIN
   ClrScr;
   t:= 0.0;
   iter:= 0;
   bottom:= 0;
   totaldisp:= 0.0;
   bottom1:= 0;
   totaldisp1:= 0.0;

   assign(disk,'B:107.DOC');
   rewrite(disk);

   U_BC;
   INITIAL_CONDITIONS;

   DRAWBED(h1,n);
   DRAWSURFACE(etha1,n);

   While t < tmax do
       Begin
          iter:= iter + 1;
          t:= t + dt;

          if t <= endmotion THEN BED_IS_MOVING
                            ELSE STATIONARY_BED;

          CONTINUITY_PREDICTOR;
          ETHA_BC_PRED;
          MOMENTUM;
          CONTINUITY_CORRECTOR;
          ETHA_BC_CORR;
          FCT(etha2);

          DRAWBED(h2,10);
          DRAWSURFACE(etha2,10);

          if (iter mod 2) = 0 THEN
             begin
                writeln(disk,iter div 2);
                writeln(disk,etha2[20]);
                writeln(disk,etha2[60]);
                writeln(disk,etha2[120]);
                writeln(disk,etha2[180]);
                writeln(disk,etha2[240]);
                writeln(disk,etha2[300]);
                writeln(disk,etha2[360]);
                writeln(disk,etha2[420]);
                writeln(disk);
             end;
```

```
            For i:= 0 to n do
                Begin
                   etha1[i]:= etha2[i];
                   u1[i]:= u2[i];
                End;

            For i:= 0 to n do h1[i]:= h2[i];

        End;
    close(disk);
END.
```

Program SHELF

Purpose of the program

Modelling of nonlinear, dispersive, shallow-water waves generated in a channel by the motion of a shelf.

Definition of the constants

n	: Total number of spatial grid points (i = 0,1,...n).
n1	: n - 1.
dx	: Spatial mesh size (constant).
dt	: Temporal mesh size.
tmax	: Total duration of the simulation.
endmotion	: Total duration of the bed motion.
vstar	: Amplitude of the horizontal, sinusoidal velocity of the wave generating device.
slope	: Constant slope of the depth transition between the shelf and the channel bottom.
xsin	: Parameter defining the sinusoidal velocity of the bed, xsin = τ_2-τ_1.
initp	: Initial x-coordinate of the top of the depth transition.
hbox	: Height of the shelf.
eu	: Implicitness parameter for the depth-averaged velocity.
en	: Implicitness parameter for the wave amplitude.
eh	: Implicitness parameter for the bed position.

Definition of the global variables

h1	: Position of the bed at time t.
hmid	: Position of the bed at time t + dt/2.
h2	: Position of the bed at time t + dt.
etha1	: Height of the free surface above its undisturbed level at time t.
etha2	: Height of the free surface above its undisturbed level at time t + dt.
u1	: Depth-averaged velocity at time t.
u2	: Depth-averaged velocity at time t + dt.
diag1...	
diag3	: Diagonals of the tridiagonal matrix system (section 3.2.2).
rhs	: Right hand side of the tridiagonal matrix system (section 3.2.2).
dhdt	: Bed velocity.
acc	: Bed acceleration.

```
     slope1 : Slope of the bed at time t.
     slope2 : Slope of the bed at time t + dt.
      A...F : Groupings of variables used in the implicit solution of the
              momentum equation.
          t : Time.
       vel1 : Horizontal velocity of the shelf at time t.
     velmid : Horizontal velocity of the shelf at time t + dt/2.
       vel2 : Horizontal velocity of the shelf at time t + dt.
       disp : Displacement of the shelf between t and t + dt.
      disp1 : Displacement of the shelf between t and t + dt/2.
  totaldisp : Cumulative displacement of the shelf at time t + dt.
 totaldisp1 : Cumulative displacement of the shelf at time t + dt/2.
     shelfd : Depth of the fluid above the shelf (= 1.0-hbox).
          i : Loop counter.
       iter : Number of time steps involved in the computation.
     bottom : Grid point corresponding to the lower end of the depth
              transition at time t + dt.
    bottom1 : Grid point corresponding to the lower end of the depth
              transition at time t + dt/2.
        top : Grid point corresponding to the higher end of the depth
              transition at time t + dt.
       top1 : Grid point corresponding to the higher end of the depth
              transition at time t + dt/2.
       disk : Text file used to store the results.
```

Program listing

```
Program SHELF (input,output);

uses crt,graph3;

{*****************Experimental conditions : WAVE.DAT 72**********************}

CONST
           n = 500;
          n1 = 499;
          dx = 0.25;
          dt = 0.25;
        tmax = 120;
   endmotion = 13.0;
       vstar = 0.2;
       slope = 0.577;
        xsin = 1.0;
       initp = 5.0;
        hbox = 0.33333;
          eu = 0.5;
          en = 0.5;
          eh = 0.5;

TYPE
      vector = array [0..n] of real;
```

```
VAR
                h1,h2,hmid,etha1,etha2,u1,u2 : vector;
                      diag1,diag2,diag3,rhs : vector;
                     dhdt,acc,slope1,slope2 : vector;
                                A,B,C,D,E,F : real;
                                          t : real;
                           vel1,velmid,vel2 : real;
              disp,displ,totaldisp,totaldispl : real;
                                    shelfd : real;
              i,bottom,bottom1,top,top1,iter : integer;
                                     disk : text;


Procedure U_BC;

begin
   u2[0]:= 0.0;
   u2[n]:= 0.0;
end;


Procedure INITIAL_CONDITIONS;

begin
   for i:= 0 to n do
       begin
          etha1[i]:= 0.0;
          u1[i]:= 0.0;
       end;

   for i:= 0 to top do h1[i]:= shelfd;
   for i:= top+1 to bottom do h1[i]:= 1.0 - slope * (totaldisp-i*dx);
   for i:= bottom+1 to n do h1[i]:= 1.0;
end;


Procedure MOVE_BED;

begin
   for i:= 0 to top do h2[i]:= shelfd;
   for i:= top+1 to bottom do h2[i]:= 1.0 - slope * (totaldisp-i*dx);
   for i:= bottom+1 to n do h2[i]:= 1.0;

   for i:= 0 to top1 do hmid[i]:= shelfd;
   for i:= top1+1 to bottom1 do hmid[i]:= 1.0 - slope * (totaldispl-i*dx);
   for i:= bottom1+1 to n do hmid[i]:= 1.0;
end;


Procedure BED_IS_MOVING;

begin
   vel1:= vstar * sin( 0.5*pi - (0.5*endmotion-(t-dt))*xsin/endmotion*pi );
   velmid:= vstar * sin( 0.5*pi - (0.5*endmotion-(t-0.5*dt))*xsin/endmotion*pi );
   vel2:= vstar * sin( 0.5*pi - (0.5*endmotion-t)*xsin/endmotion*pi );
```

```
    displ:= 0.5*(vel1+velmid) * dt/2;
    totaldispl:= totaldisp + displ;
    bottoml:= trunc(totaldispl/dx);
    topl:= trunc((totaldispl-hbox/slope)/dx);

    disp:= 1/3*(vel1+velmid+vel2) * dt;
    totaldisp:= totaldisp + disp;
    bottom:= trunc(totaldisp/dx);
    top:= trunc((totaldisp-hbox/slope)/dx);

    MOVE_BED;

    for i:= 0 to n do acc[i]:= (h2[i]-2*hmid[i]+h1[i])/sqr(dt);

    for i:= 0 to n do
        begin
           if (h1[i] = shelfd) or (h1[i] = 1.0)
              THEN slope1[i]:= 0.0
              ELSE slope1[i]:= slope;

           if (h2[i] = shelfd) or (h2[i] = 1.0)
              THEN slope2[i]:= 0.0
              ELSE slope2[i]:= slope;

           dhdt[i]:= (h2[i]-h1[i])/dt;
        end;
end;


Procedure STATIONARY_BED;

begin
   if (t-endmotion) <= dt then
      begin
         sound(100);
         delay(1000);
         NoSound;
         for i:= 0 to n do acc[i]:= 0.0;
         for i:= 0 to n do
            begin
               if (h2[i] = shelfd) or (h2[i] = 1.0)
                  THEN
                    begin
                       slope1[i]:= 0.0;
                       slope2[i]:= 0.0;
                    end
                  ELSE
                    begin
                       slope1[i]:= slope;
                       slope2[i]:= slope;
                    end;

               dhdt[i]:= 0.0;
            end;
      end;
end;
```

```
Procedure CONTINUITY_PREDICTOR;

begin
   for i:= 1 to n1 do
       begin
          etha2[i]:= etha1[i] - dt*( ( 0.5*(h1[i]+h2[i])+etha1[i] )
                                              * (u1[i+1]-u1[i-1])/2/dx
                                   + u1[i]*(  0.5*(slope1[i]+slope2[i])
                                            + (etha1[i+1]-etha1[i-1])/2/dx )
                                   + dhdt[i]  );
       end;
end;



Procedure CONTINUITY_CORRECTOR;

begin
   for i:= 1 to n1 do
       begin
          etha2[i]:= etha1[i] - dt*( ( 0.5*(h1[i]+h2[i])+etha1[i] )
                                    *0.25*(u2[i+1]-u2[i-1]+u1[i+1]-u1[i-1])/dx
                                   +0.5*(u1[i]+u2[i])
                                       * ( 0.5*(slope1[i]+slope2[i])
                                            + (etha1[i+1]-etha1[i-1])/2/dx )
                                   + dhdt[i]  );
       end;
end;



Procedure ETHA_BC;

begin
   etha2[0]:= etha2[1];
   etha2[n]:= etha2[n1];
end;



Procedure TRIDIAG(a,b,c:vector; var x:vector; rhs:vector; first,last:integer);

var
   mult : real;
      i : integer;

begin
   for i:= first+1 to last do
       begin
          mult:= a[i]/b[i-1];
          b[i]:= b[i] - mult * c[i-1];
          rhs[i]:= rhs[i] - mult * rhs[i-1];
       end;
   x[last]:= rhs[last]/b[last];
   for i:= last-1 downto first do x[i]:= ( rhs[i] - c[i]*x[i+1] ) / b[i];
end;
```

```
Procedure MOMENTUM_Fxt;

begin
   for i:= 1 to n1 do
       begin
          u2[i]:= u1[i];

          A:= ( (1-eh)*sqr(h1[i]) + eh*sqr(h2[i]) )/3/sqr(dx)/dt;
          B:= ( (1-eh)*h1[i]*slope1[i] + eh*h2[i]*slope2[i] )/2/dx/dt;
          C:= (slope2[i] - slope1[i])/dt;

          D:= (1-eh)*h1[i]/2*(h1[i+1]-2*h1[i]+h1[i-1])/sqr(dx)
                + eh*h2[i]/2*(h2[i+1]-2*h2[i]+h2[i-1])/sqr(dx);
          E:= ((h2[i+1]-2*h2[i]+h2[i-1])-(h1[i+1]-2*h1[i]+h1[i-1]))/sqr(dx)/dt;

          F:= ((1-en)*(etha1[i+1]-etha1[i-1])+en*(etha2[i+1]-etha2[i-1]))/2/dx;


          diag1[i]:= -eu*u2[i]/2/dx - A + B + eu*h2[i]/2/dx * C
                     - eu*h2[i]/2/sqr(dx)*dhdt[i];
          diag2[i]:= 1/dt + 2*A - 1/dt*D - eu*h2[i]/2*E
                     + eu*h2[i]/sqr(dx)*dhdt[i];
          diag3[i]:=  eu*u2[i]/2/dx - A - B - eu*h2[i]/2/dx * C
                     - eu*h2[i]/2/sqr(dx)*dhdt[i];

          rhs[i]:= u1[i-1] *((1-eu)*u1[i]/2/dx - A + B - (1-eu)*h1[i]/2/dx*C
                             +(1-eu)*h1[i]/2/sqr(dx)*dhdt[i] )
                   + u1[i] *(1/dt + 2*A - 1/dt*D + (1-eu)*h1[i]/2*E
                             -(1-eu)*h1[i]/sqr(dx)*dhdt[i] )
                  +u1[i+1] *(-(1-eu)*u1[i]/2/dx - A - B +(1-eu)*h1[i]/2/dx*C
                             +(1-eu)*h1[i]/2/sqr(dx)*dhdt[i] )
                   - F
                   + ((1-eh)*h1[i]+eh*h2[i])/2 * (acc[i+1]-acc[i-1])/2/dx;
       end;
   TRIDIAG(diag1,diag2,diag3,u2,rhs,1,n1);
end;



Procedure MOMENTUM_Fx;

begin
   for i:= 1 to n1 do
       begin
          u2[i]:= u1[i];

          A:= sqr(h2[i])/3/sqr(dx)/dt;
          B:= h2[i]*slope2[i]/2/dx/dt;
          D:= h2[i]/2*(h2[i+1]-2*h2[i]+h2[i-1])/sqr(dx);
          F:= ((1-en)*(etha1[i+1]-etha1[i-1])+en*(etha2[i+1]-etha2[i-1]))/2/dx;

          diag1[i]:= -eu*u2[i]/2/dx - A + B;
          diag2[i]:= 1/dt + 2*A - 1/dt*D;
          diag3[i]:=  eu*u2[i]/2/dx - A - B;
```

```
            rhs[i]:=   u1[i-1] *((1-eu)*u1[i]/2/dx - A + B)
                     + u1[i] *(1/dt + 2*A - 1/dt*D)
                     + u1[i+1] *(-(1-eu)*u1[i]/2/dx - A - B)
                     - F;

        end;
    TRIDIAG(diag1,diag2,diag3,u2,rhs,1,n1);
end;



Procedure MOMENTUM;

begin
    if t <= endmotion THEN MOMENTUM_Fxt
                      ELSE MOMENTUM_Fx;
end;



Procedure DRAWBED(v:vector; no_points:integer);

var x1,x2,y1,y2 : real;
           color : integer;

begin
    HiRes;
    GraphWindow(50,40,589,159);
    GoToXY(35,20);writeln('t = ',t:6:3);
{    for i:= 0 to no_points do draw(round(i/no_points*540),90,
                                    round(i/no_points*540),20,1);}
    for i:= 0 to no_points-1 do
        begin
           x1:= i/no_points * 540;
           y1:= v[i] * 30;
           x2:= (i+1)/no_points * 540;
           y2:= v[i+1] * 30;
           draw(round(x1),60+round(y1),round(x2),60+round(y2),1);
        end;
end;



Procedure DRAWSURFACE(v:vector; no_points:integer);

var x1,x2,y1,y2 : real;

begin
    for i:= 0 to no_points-1 do
        begin
           x1:= i/no_points * 540;
           y1:= v[i] * 300;
           x2:= (i+1)/no_points * 540;
           y2:= v[i+1] * 300;
           draw(round(x1),60-round(y1),round(x2),60-round(y2),1);
        end;
end;
```

```
Procedure FCT(var vt:vector);

const sigma = 0.2;

var
        vtd,delta,Dc : vector;
     D1,sign,max,min : real;

begin
   delta[0]:= 0.0;   delta[n-1]:= 0.0;
   for i:= 1 to n-1 do
       vtd[i]:= vt[i] + sigma*(vt[i+1] - 2.0*vt[i] + vt[i-1]);
   for i:= 1 to n-2 do
       delta[i]:= vtd[i+1] - vtd[i];
   for i:= 1 to n-2 do
       begin
          if delta[i] > 0.0 then sign:= 1.0;
          if delta[i] = 0.0 then sign:= 0.0;
          if delta[i] < 0.0 then sign:=-1.0;
          D1:= (vt[i+1] - vt[i])/8.0;
          if sign*delta[i-1] <= sign*delta[i+1]
                        then   min:= sign*delta[i-1]
                        else   min:= sign*delta[i+1];
          if abs(D1) <= min
                        then   min:= abs(D1);
          if 0.0 >= min
                        then max:= 0.0
                        else max:= min;
          Dc[i]:= sign*max;
       end;
   for i:= 2 to n-2 do
       vt[i]:= vtd[i] - (Dc[i] - Dc[i-1]);
end;



BEGIN   (main)
   ClrScr;
   t:= 0.0;
   iter:= 0;
   shelfd:= 1.0 - hbox;
   totaldisp:= initp + hbox/slope;
   totaldisp1:= totaldisp;
   bottom:= trunc(totaldisp/dx);
   bottom1:= bottom;
   top:= trunc(initp/dx);
   top1:= top;

   assign(disk,'B:RES72.DOC');
   rewrite(disk);

   U_BC;
   INITIAL_CONDITIONS;

   DRAWBED(h1,n);
   DRAWSURFACE(ethai,n);
```

```
    While t < tmax do
        Begin
           iter:= iter + 1;
           t:= t + dt;

           if t <= endmotion THEN BED_IS_MOVING
                             ELSE STATIONARY_BED;

           CONTINUITY_PREDICTOR;
           ETHA_BC;
           MOMENTUM;
           CONTINUITY_CORRECTOR;
           ETHA_BC;
           FCT(etha2);

           DRAWBED(h2,n);
           DRAWSURFACE(etha2,n);

           if (iter mod 2) = 0 THEN
              begin
                 writeln(disk,iter div 2);
                 writeln(disk,etha2[9]);
                 writeln(disk,etha2[63]);
                 writeln(disk,etha2[116]);
                 writeln(disk,etha2[157]);
                 writeln(disk,etha2[199]);
                 writeln(disk,etha2[240]);
                 writeln(disk,etha2[281]);
                 writeln(disk,etha2[323]);
                 writeln(disk);
              end;


           For i:= 1 to n1 do
               Begin
                  etha1[i]:= etha2[i];
                  u1[i]:= u2[i];
               End;

           For i:= 0 to n1 do h1[i]:= h2[i];

        End;
     close(disk);
END.
```

Program ROPLATE

Purpose of the program

Modelling of nonlinear, dispersive, shallow-water waves generated in a channel by the motion of a rotating plate.

Definition of the constants

n	: Total number of spatial grid points (i = 0,1,...n).
n1	: n - 1.
dx	: Spatial mesh size (constant).
dt	: Temporal mesh size.
tmax	: Total duration of the simulation.
endmotion	: Total duration of the bed motion.
vertvel	: Constant vertical velocity applied to the plate at x = 0.
pivloc	: x-coordinate of the point around which the plate rotates.
eu	: Implicitness parameter for the depth-averaged velocity.
en	: Implicitness parameter for the wave amplitude.
eh	: Implicitness parameter for the bed position.

Definition of the global variables

h1	: Position of the bed at time t.
h2	: Position of the bed at time t + dt.
etha1	: Height of the free surface disturbance at time t.
etha2	: Height of the free surface disturbance at time t + dt.
u1	: Depth-averaged velocity at time t.
u2	: Depth-averaged velocity at time t + dt.
diag1... diag3	: Diagonals of the tridiagonal matrix system (section 3.2.2).
rhs	: Right hand side of the tridiagonal matrix system (section 3.2.2).
dhdt	: Bed velocity.
slope1	: Slope of the bed at time t.
slope2	: Slope of the bed at time t + dt.
A...F	: Groupings of variables used in the implicit solution of the momentum equation.
t	: Time.
i	: Loop counter.
iter	: Number of time steps involved in the computation.
pivot	: Spatial grid point corresponding to x = pivloc.
disk	: Text file used to store the results.

Program listing

```
Program ROTATING_PLATE(input,output);

uses crt,graph3;

{***************Experimental conditions : WAVE.DAT 232******************}


CONST
           n = 600;
          n1 = 599;
          dx = 0.25;
          dt = 0.25;
        tmax = 125;
   endmotion = 8.0;
     vertvel = 0.1;
      pivloc = 3.3333333;
          eu = 0.5;
          en = 0.5;
          eh = 0.5;

TYPE
      vector = array [0..n] of real;

VAR
                    h1,h2,etha1,etha2,u1,u2 : vector;
                         slope1,slope2,dhdt : vector;
                      diag1,diag2,diag3,rhs : vector;
                                          t : real;
                                A,B,C,D,E,F : real;
                               i,iter,pivot : integer;
                                       disk : text;

Procedure U_BC;

begin
   u2[0]:= 0.0;
   u2[n]:= 0.0;
end;


Procedure INITIAL_CONDITIONS;

begin
   for i:= 0 to n do
       begin
          h1[i]:= 1.0;
          etha1[i]:= 0.0;
          u1[i]:= 0.0;
       end;
end;
```

```
Procedure RAISE_BED;

begin
   for i:= 0 to pivot do
       h2[i]:= 1.0 - (pivloc - i*dx) * vertvel*t / pivloc;
   for i:= pivot+1 to n do
       h2[i]:= 1.0;

   for i:= 1 to ni do
       begin
          slope1[i]:= (h1[i+1] - h1[i-1]) / (2.0*dx);
          slope2[i]:= (h2[i+1] - h2[i-1]) / (2.0*dx);
          dhdt[i]:= (h2[i] - h1[i]) / dt;
       end;
end;



Procedure STATIONARY_BED;

begin
   if (t-endmotion) <= dt then
      begin
         sound(100);
         delay(1000);
         NoSound;
         for i:= 0 to n do
             begin
                slope1[i]:= slope2[i];
                dhdt[i]:= 0.0;
             end;
      end;
end;



Procedure CONTINUITY_PREDICTOR;

begin
   for i:= 1 to ni do
       etha2[i]:= etha1[i] - dt*( ( 0.5*(h1[i]+h2[i])+etha1[i] )
                                          * 0.5*(u1[i+1]-u1[i-1])/dx
                                + u1[i]*(  0.5*(slope1[i]+slope2[i])
                                         + 0.5*(etha1[i+1]-etha1[i-1])/dx )
                                + dhdt[i]  );
end;
```

```
Procedure ETHA_BC_PRED;

var
   term1,term2 : real;

begin
   if t <= endmotion
      THEN
        begin
           term1:= h2[0] * (u1[1]-u1[0])/dx
                        * (h2[1]-h2[0]-h1[1]+h1[0])/dx/dt;
           term2:= h2[0]/2 * (h2[0]-h1[0])/dt
                           * (u1[2]-2*u1[1]+u1[0])/sqr(dx);
           etha2[0]:= etha2[1] - dx * (term1+term2);
        end
      ELSE
        etha2[0]:= etha2[1];
   etha2[n]:= etha2[n1];
end;


Procedure CONTINUITY_CORRECTOR;

begin
   for i:= 1 to n1 do
       etha2[i]:= etha1[i] - dt*( ( 0.5*(h1[i]+h2[i])+etha1[i] )
                                   *0.25*(u2[i+1]-u2[i-1]+u1[i+1]-u1[i-1])/dx
                                  +0.5*(u1[i]+u2[i])
                                      * ( 0.5*(slope1[i]+slope2[i])
                                           + 0.5*(etha1[i+1]-etha1[i-1])/dx )
                                  + dhdt[i]  );
end;


Procedure ETHA_BC_CORR;

var
   term1,term2 : real;

begin
   if t <= endmotion
      THEN
        begin
           term1:= h2[0] * (u2[1]-u2[0])/dx
                        * (h2[1]-h2[0]-h1[1]+h1[0])/dx/dt;
           term2:= h2[0]/2 * (h2[0]-h1[0])/dt
                           * (u2[2]-2*u2[1]+u2[0])/sqr(dx);
           etha2[0]:= etha2[1] - dx * (term1+term2);
        end
      ELSE
        etha2[0]:= etha2[1];
   etha2[n]:= etha2[n1];
end;
```

```
Procedure TRIDIAG(a,b,c:vector; var x:vector; rhs:vector; first,last:integer);

var
   mult : real;
      i : integer;

begin
   for i:= first+1 to last do
       begin
          mult:= a[i]/b[i-1];
          b[i]:= b[i] - mult * c[i-1];
          rhs[i]:= rhs[i] - mult * rhs[i-1];
       end;
   x[last]:= rhs[last]/b[last];
   for i:= last-1 downto first do x[i]:= ( rhs[i] - c[i]*x[i+1] ) / b[i];
end;



Procedure MOMENTUM_Fxt;

begin
   for i:= 1 to n1 do
       begin
          u2[i]:= u1[i];

          A:= ( (1-eh)*sqr(h1[i]) + eh*sqr(h2[i]) )/3/sqr(dx)/dt;
          B:= ( (1-eh)*h1[i]*slope1[i] + eh*h2[i]*slope2[i] )/2/dx/dt;
          C:= (slope2[i] - slope1[i])/dt;

          D:= (1-eh)*n1[i]/2*(h1[i+1]-2*h1[i]+h1[i-1])/sqr(dx)
                + eh*h2[i]/2*(h2[i+1]-2*h2[i]+h2[i-1])/sqr(dx);
          E:= ((h2[i+1]-2*h2[i]+h2[i-1])-(h1[i+1]-2*h1[i]+h1[i-1]))/sqr(dx)/dt;

          F:= ((1-en)*(etha1[i+1]-etha1[i-1])+en*(etha2[i+1]-etha2[i-1]))/2/dx;


          diag1[i]:= -eu*u2[i]/2/dx - A + B + eu*h2[i]/2/dx * C
                     - eu*h2[i]/2/sqr(dx)*dhdt[i];
          diag2[i]:= 1/dt + 2*A - 1/dt*D - eu*h2[i]/2*E
                     + eu*h2[i]/sqr(dx)*dhdt[i];
          diag3[i]:=  eu*u2[i]/2/dx - A - B - eu*h2[i]/2/dx * C
                     - eu*h2[i]/2/sqr(dx)*dhdt[i];

          rhs[i]:= u1[i-1] *((1-eu)*u1[i]/2/dx - A + B - (1-eu)*h1[i]/2/dx*C
                             +(1-eu)*h1[i]/2/sqr(dx)*dhdt[i] )
                   + u1[i] *(1/dt + 2*A - 1/dt*D + (1-eu)*h1[i]/2*E
                             -(1-eu)*h1[i]/sqr(dx)*dhdt[i] )
                  +u1[i+1] *(-(1-eu)*u1[i]/2/dx - A - B +(1-eu)*h1[i]/2/dx*C
                             +(1-eu)*h1[i]/2/sqr(dx)*dhdt[i] )
                   - F;
       end;
   TRIDIAG(diag1,diag2,diag3,u2,rhs,1,n1);
end;
```

```
Procedure MOMENTUM_Fx;

begin
   for i:= 1 to n1 do
       begin
          u2[i]:= u1[i];

          A:= sqr(h2[i]) / (3*sqr(dx)*dt);
          B:= slope2[i]*h2[i] / (4*dx*dt);
          D:= h2[i]/2*(h2[i+1]-2*h2[i]+h2[i-1])/sqr(dx);
          F:= ((1-en)*(etha1[i+1]-etha1[i-1])+en*(etha2[i+1]-etha2[i-1]))/2/dx;

          diag1[i]:= -eu*u2[i]/2/dx - A + B;
          diag2[i]:= 1/dt + 2*A - 1/dt*D;
          diag3[i]:=  eu*u2[i]/2/dx - A - B;

          rhs[i]:=   u1[i-1] *((1-eu)*u1[i]/2/dx - A + B)
                   + u1[i] *(1/dt + 2*A - 1/dt*D)
                   + u1[i+1] *(-(1-eu)*u1[i]/2/dx - A - B)
                   - F;

       end;
   TRIDIAG(diag1,diag2,diag3,u2,rhs,1,n1);
end;


Procedure MOMENTUM;

begin
   if t <= endmotion then MOMENTUM_Fxt
                     else MOMENTUM_Fx;
end;



Procedure DRAWBED(v:vector);

var x1,x2,y1,y2 : real;

begin
   HiRes;
   GraphWindow(50,40,589,159);
   GoToXY(35,20);writeln('t = ',t:5:3);
   for i:= 0 to n1 do
       begin
          x1:= (i-1)/(n1) * 540;
          y1:= v[i] * 30;
          x2:= i/(n1) * 540;
          y2:= v[i+1] * 30;
          draw(round(x1),60+round(y1),round(x2),60+round(y2),1);
       end;
end;
```

```
Procedure DRAWSURFACE(v:vector);

var x1,x2,y1,y2 : real;

begin
   for i:= 0 to n1 do
       begin
          x1:= (i-1)/(n1) * 540;
          y1:= v[i] * 300;
          x2:= i/(n1) * 540;
          y2:= v[i+1] * 300;
          draw(round(x1),60-round(y1),round(x2),60-round(y2),1);
       end;
end;



Procedure FCT(var vt:vector);

const sigma = 0.2;

var
        vtd,delta,Dc : vector;
     D1,sign,max,min : real;

begin
   delta[0]:= 0.0;    delta[n-1]:= 0.0;
   for i:= 1 to n-1 do
       vtd[i]:= vt[i] + sigma*(vt[i+1] - 2.0*vt[i] + vt[i-1]);
   for i:= 1 to n-2 do
       delta[i]:= vtd[i+1] - vtd[i];
   for i:= 1 to n-2 do
       begin
          if delta[i] > 0.0 then sign:= 1.0;
          if delta[i] = 0.0 then sign:= 0.0;
          if delta[i] < 0.0 then sign:=-1.0;
          D1:= (vt[i+1] - vt[i])/8.0;
          if sign*delta[i-1] <= sign*delta[i+1]
                        then   min:= sign*delta[i-1]
                        else   min:= sign*delta[i+1];
          if abs(D1) <= min
                        then   min:= abs(D1);
          if 0.0 >= min
                        then max:= 0.0
                        else max:= min;
          Dc[i]:= sign*max;
       end;
  for i:= 2 to n-2 do
       vt[i]:= vtd[i] - (Dc[i] - Dc[i-1]);
end;
```

```
BEGIN {main}
   t:= 0.0;
   iter:= 0;
   pivot:= trunc(pivloc/dx);

   assign(disk,'B:RES232.DOC');
   rewrite(disk);

   U_BC;
   INITIAL_CONDITIONS;

   DRAWBED(h1);
   DRAWSURFACE(etha1);

   While t < tmax do
       Begin
          iter:= iter + 1;
          t:= t + dt;

          if t <= endmotion then RAISE_BED
                            else STATIONARY_BED;

          CONTINUITY_PREDICTOR;
          ETHA_BC_PRED;
          MOMENTUM;
          CONTINUITY_CORRECTOR;
          ETHA_BC_CORR;
          FCT(etha2);

          DRAWBED(h2);
          DRAWSURFACE(etha2);

          if (iter mod 2) = 0 THEN
             begin
                writeln(disk,iter div 2);
                writeln(disk,etha2[62]);
                writeln(disk,etha2[124]);
                writeln(disk,etha2[186]);
                writeln(disk,etha2[248]);
                writeln(disk,etha2[310]);
                writeln(disk,etha2[372]);
                writeln(disk,etha2[434]);
                writeln(disk,etha2[496]);
                writeln(disk);
             end;

          For i:= 0 to n do
              Begin
                 etha1[i]:= etha2[i];
                 u1[i]:= u2[i];
              End;

          for i:= 0 to n do h1[i]:= h2[i];

       End;
   close(disk);
END.
```

Program WEDGE

Purpose of the program

Modelling of nonlinear, dispersive, shallow-water waves generated in a channel by a moving inclined wall (moving wegde).

Definition of the constants

n	: Total number of spatial grid points (i = 0,1,...n).
dx	: Initial spatial mesh size at t = 0.
dt	: Temporal mesh size.
totald	: Total length of the channel.
tmax	: Total duration of the simulation.
vstar	: Amplitude of the sinusoidal wall velocity.
slope	: Constant slope of the moving wedge.
lastT	: Time at which the motion of the wedge stops.
lastX	: Final x-coordinate of the moving wall.
smooth	: Parameter controlling the lenght of the smoothed transition between the inclined wall and the channel bottom.
g1,...,g8	: Location of the eight gauges measuring wave amplitude.

Definition of the global variables

etha1	: Height of the free surface above its undisturbed level at time t.
etha2	: Height of the free surface above its undisturbed level at time t + dt.
u1	: Depth-averaged velocity at time t.
u2	: Depth-averaged velocity at time t + dt.
h1	: Position of the bed at time t.
hmid	: Position of the bed at time t + dt/2.
h2	: Position of the bed at time t + dt.
x1	: Position of the spatial grid points at time t.
xmid	: Position of the spatial grid points at time t + dt/2.
x2	: Position of the spatial grid points at time t + dt.
dhdx1	: Slope of the bed at time t.
dhdxmid	: Slope of the bed at time t + dt/2.
dhdx2	: Slope of the bed at time t + dt.
t	: Time.
vel	: Horizontal velocity of the bed.
totaldisp	: Horizontal displacement of the bed
i	: Loop counter.
iter	: Number of time steps involved in the computation.

```
gauge1...
   gauge8 : Grid points corresponding to the gauges  location.
     disk : Text file used to store the results.
```

Program listing

```
Program WEDGE (input,output);

uses crt,graph3;

{********************Experimental conditions : WAVE.DAT39**********************}

CONST
       n = 500;
      dx = 0.25;
      dt = 0.25;
  totald = 125;
    tmax = 125.0;
   vstar = 0.2;
   slope = 0.268;
   lastT = 13.116;
   lastX = 1.67;
  smooth = 0.25;      {   2*smooth/slope = lenght of smoothed transition }
      g1 = 6.73;
      g2 = 17.13;
      g3 = 27.43;
      g4 = 37.73;
      g5 = 48.13;
      g6 = 58.43;
      g7 = 68.73;
      g8 = 79.13;

TYPE
   vector = array [0..n] of real;

VAR
       etha1,etha2,u1,u2,h1,h2 : vector;
                         x1,x2 : vector;
                  dhdx1,dhdx2 : vector;
            hmid,xmid,dhdxmid : vector;
              t,vel,totaldisp : real;
                         i,iter : integer;
   gauge1,gauge2,gauge3,gauge4 : integer;
   gauge5,gauge6,gauge7,gauge8 : integer;
                          disk : text;
```

```
Function p(s:real):real;     {interpolating polynomial    }

begin
   p:= 1/32 * (s*s*s*s*s*s - 5*s*s*s*s*s + 15*sqr(s) - 16*s + 5);
end;


Procedure FIND_DEPTH (t:real; x:vector; var h:vector; var dhdx:vector);

var
  wedgebottom,beginsmooth,endsmooth : real;

begin
   if t <= lastT
      THEN
        begin
           totaldisp:= vstar*lastT/pi * ( 1 - cos (pi*t/lastT) );
           vel:= vstar * sin(pi*t/lastT);
        end
      ELSE
        begin
           totaldisp:= 2*vstar*lastT/pi;
           vel:= 0.0;
        end;

   wedgebottom:= 1/slope + totaldisp;
   beginsmooth:= wedgebottom - smooth/slope;
   endsmooth:= wedgebottom + smooth/slope;

   for i:= 0 to n do
       begin
          if x[i] <= beginsmooth
             THEN
               h[i]:= 1 - (wedgebottom-x[i])*slope
             ELSE
               if x[i] >= endsmooth
                  THEN
                    h[i]:= 1.0
                  ELSE
                    h[i]:= 1 - smooth * p((x[i]-wedgebottom)*slope/smooth);
      end;

   for i:= 1 to n-1 do dhdx[i]:= (h[i+1]-h[i-1])/(x[i+1]-x[i-1]);
end;


Procedure INITIAL_CONDITIONS;

begin
   for i:= 0 to n do x1[i]:= i*dx;
   for i:= 0 to n do etha1[i]:= 0.0;
   for i:= 0 to n do u1[i]:= 0.0;
   FIND_DEPTH(0,x1,h1,dhdx1);
end;
```

```
Procedure APPROX_INTERFACES;

begin
   for i:= 0 to n do
       begin
          xmid[i]:= x1[i] + dt/2*u1[i];
          x2[i]:= x1[i] + dt*u1[i];
       end;
end;


Procedure CONTINUITY_PREDICTOR;

begin
   for i:= 1 to n-1 do
       begin
          etha2[i]:= etha1[i] - (h2[i]-h1[i])
                              - dt * (etha1[i]+h1[i])
                                   *  (u1[i+1]-u1[i-1])/(x1[i+1]-x1[i-1]);
       end;
end;



Procedure CONTINUITY_CORRECTOR;

begin
   for i:= 1 to n-1 do
     begin
       etha2[i]:= etha1[i] - (h2[i]-h1[i])
                           - dt * 0.5*(etha1[i]+etha2[i]+h1[i]+h2[i])
                                * 0.5 *( (u1[i+1]-u1[i-1])/(x1[i+1]-x1[i-1])
                                        +(u2[i+1]-u2[i-1])/(x2[i+1]-x2[i-1]));
     end;
end;



Procedure FS_BC;

begin
   etha2[0]:= - h2[0];
   etha2[n]:= etha2[n-1];
end;



Procedure U_BC_PRED;

begin
   u2[0]:= u1[0];
   u2[n]:= 0;
end;
```

```
Procedure TRIDIAG(a,b,c:vector; var x:vector; rhs:vector; first,last:integer);

var
    mult : real;
       i : integer;

begin
    for i:= first+1 to last do
        begin
           mult:= a[i]/b[i-1];
           b[i]:= b[i] - mult * c[i-1];
           rhs[i]:= rhs[i] - mult * rhs[i-1];
        end;
    x[last]:= rhs[last]/b[last];
    for i:= last-1 downto first do x[i]:= ( rhs[i] - c[i]*x[i+1] ) / b[i];
end;



Procedure MOMENTUM;

var
   diag1,diag2,diag3,rhs : vector;
     dx1,dx2,dx1sq,dx2sq : real;
                 A,B,C,D : real;

begin
   for i:= 1 to n-1 do
       begin
          dx1:= x1[i+1]-x1[i-1];
          dx2:= x2[i+1]-x2[i-1];
          dx1sq:= 0.5*(sqr(x1[i+1]-x1[i])+sqr(x1[i]-x1[i-1]));
          dx2sq:= 0.5*(sqr(x2[i+1]-x2[i])+sqr(x2[i]-x2[i-1]));

          A:= 1/6*(sqr(h1[i])+sqr(h2[i]));
          B:= 0.5*(h1[i]*dhdx1[i]+h2[i]*dhdx2[i]);
          C:= 0.5*(dhdx2[i]-dhdx1[i]);
          D:= 0.25*(h2[i]-h1[i]);

          diag1[i]:= -A/dx2sq + B/dx2 + C/dx2*h2[i] - D/dx2sq*h2[i];
          diag2[i]:= 1 + 2*A/dx2sq + 2*D/dx2sq*h2[i];
          diag3[i]:= -A/dx2sq - B/dx2 - C/dx2*h2[i] - D/dx2sq*h2[i];

          if i = 1 THEN
             rhs[i]:= - dt*0.5
                        *( (etha2[i+1]-etha2[i-1])/(x2[i+1]-x2[i-1])
                          +(etha1[i+1]-etha1[i-1])/(x1[i+1]-x1[i-1]) )

              +u1[i-1]* ( -A/dx1sq + B/dx1 - C/dx1*h1[i] + D/dx1sq*h1[i])
              +u1[i]  * (1 + 2*A/dx1sq - 2*D/dx1sq*h1[i] )
              +u1[i+1]* ( -A/dx1sq - B/dx1 + C/dx1*h1[i] + D/dx1sq*h1[i])

              + 0.25*(h1[i]+h2[i])*(dhdx2[i]-2*dhdxmid[i]+dhdx1[i])/dt
              - diag1[i]*u2[0]
```

```
                    ELSE
                rhs[i]:= - dt*0.5
                           *( (etha2[i+1]-etha2[i-1])/(x2[i+1]-x2[i-1])
                             +(etha1[i+1]-etha1[i-1])/(x1[i+1]-x1[i-1]) )

                  +u1[i-1]* ( -A/dx1sq + B/dx1 - C/dx1*h1[i] + D/dx1sq*h1[i])
                  +u1[i]   * (1 + 2*A/dx1sq - 2*D/dx1sq*h1[i] )
                  +u1[i+1]* ( -A/dx1sq - B/dx1 + C/dx1*h1[i] + D/dx1sq*h1[i])

                 + 0.25*(h1[i]+h2[i])*(dhdx2[i]-2*dhdxmid[i]+dhdx1[i])/dt;
        end;
    TRIDIAG(diag1,diag2,diag3,u2,rhs,1,n-1);
end;


Procedure U_BC_CORR;

begin
    u2[0]:= u2[1] - (x2[1]-x2[0])/(x2[2]-x2[1]) * (u2[2] - u2[1]);
    u2[n]:= 0.0;
end;


Procedure INTERFACES;

begin
    x2[0]:= x1[0] + 0.5*dt*(u1[0]+u2[0]);
    for i:= 1 to n do
        begin
           x2[i]:= x1[i] + 0.5*dt*(u1[i]+u2[i]);

           if (x2[i-1]<=g1)  and (x2[i]>g1)
              then if (g1-x2[i-1]) <= (x2[i]-g1) then gauge1:= i-1
                                                 else gauge1:= i;
           if (x2[i-1]<=g2)  and (x2[i]>g2)
              then if (g2-x2[i-1]) <= (x2[i]-g2) then gauge2:= i-1
                                                 else gauge2:= i;
           if (x2[i-1]<=g3)  and (x2[i]>g3)
              then if (g3-x2[i-1]) <= (x2[i]-g3) then gauge3:= i-1
                                                 else gauge3:= i;
           if (x2[i-1]<=g4)  and (x2[i]>g4)
              then if (g4-x2[i-1]) <= (x2[i]-g4) then gauge4:= i-1
                                                 else gauge4:= i;
           if (x2[i-1]<=g5)  and (x2[i]>g5)
              then if (g5-x2[i-1]) <= (x2[i]-g5) then gauge5:= i-1
                                                 else gauge5:= i;
           if (x2[i-1]<=g6)  and (x2[i]>g6)
              then if (g6-x2[i-1]) <= (x2[i]-g6) then gauge6:= i-1
                                                 else gauge6:= i;
           if (x2[i-1]<=g7)  and (x2[i]>g7)
              then if (g7-x2[i-1]) <= (x2[i]-g7) then gauge7:= i-1
                                                 else gauge7:= i;
           if (x2[i-1]<=g8)  and (x2[i]>g8)
              then if (g8-x2[i-1]) <= (x2[i]-g8) then gauge8:= i-1
                                                 else gauge8:= i;
        end;
end;
```

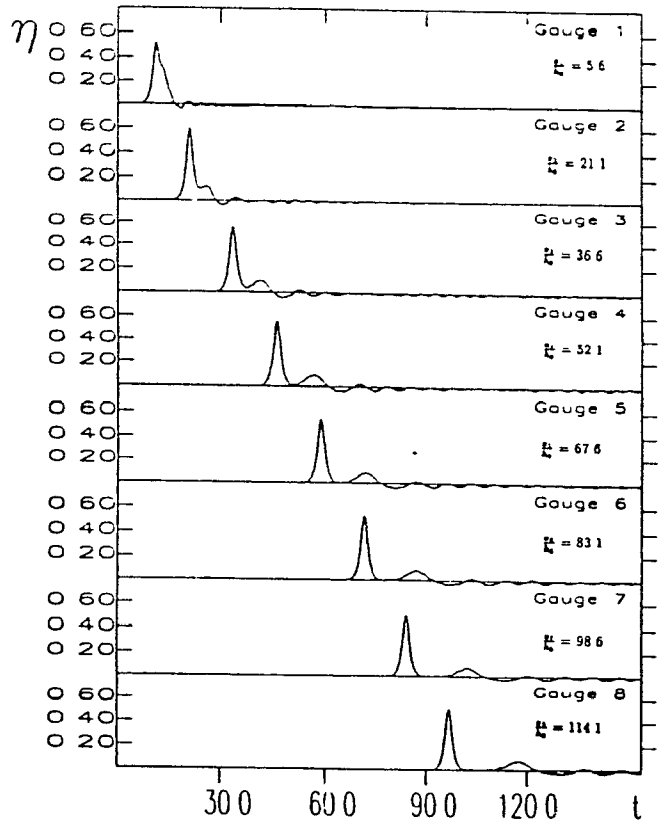
```
Procedure DRAWBED;

var x1,x2,y1,y2 : real;

begin
   HiRes;
   GraphWindow(50,20,589,159);
   GoToXY(8,23);
   writeln('TIME = ',t:5:2,'     wedge VELOCITY = ',
   vel:5:3,'    slope = ',slope:6:3);

   x1:= totaldisp/(1/slope+totald)*540;
   draw(0,10,round(x1),10,1);
   x2:= (totaldisp+2/slope)/(totald+1/slope)*540;
   draw(round(x1),10,round(x2),130,1);
   draw(round(x2),130,540,130,1);

end;


Procedure DRAWSURFACE(x,v:vector);

const pred_amp = 0.2;

var x1,y1 : real;

begin
   for i:= 0 to n do
       begin
          x1:= (x[i]+1/slope)/(totald+1/slope) * 540;
          y1:= v[i] * 60;
          plot(round(x1),70-round(y1),1);
       end;
end;



Procedure FCT(var vt:vector);

const sigma = 0.2;

var
        vtd,delta,Dc : vector;
     D1,sign,max,min : real;

begin
   vtd[0]:= vt[0];        vtd[n]:= vt[n];
   for i:= 1 to n-1 do
       vtd[i]:= vt[i] + sigma*(vt[i+1] - 2.0*vt[i] + vt[i-1]);
   for i:= 1 to n do
       delta[i]:= vtd[i] - vtd[i-1];
   for i:= 2 to n-1 do
       begin
          if delta[i] > 0.0 then sign:= 0;
          if delta[i] = 0.0 then sign:= 0.0;
          if delta[i] < 0.0 then sign:=-1.0;
          D1:= (vt[i] - vt[i-1])/8.0;
```

```
            if sign*delta[i-1] <= sign*delta[i+1]
                              then   min:= sign*delta[i-1]
                              else   min:= sign*delta[i+1];
            if abs(Di) <= min
                              then   min:= abs(Di);
            if 0.0 >= min
                              then max:= 0.0
                              else max:= min;
            Dc[i]:= sign*max;
         end;
      for i:= 2 to n-2 do
         vt[i]:= vtd[i] - (Dc[i+1] - Dc[i]);
   end;



   BEGIN  {main}
      ClrScr;
      t:= 0.0;
      iter:= 0;

      assign(disk,'B:RES39.DOC');
      rewrite(disk);

      INITIAL_CONDITIONS;

      DRAWBED;
      DRAWSURFACE(x1,etha1);

      While t < tmax do
         begin
            t:= t + dt;
            iter:= iter + 1;

            APPROX_INTERFACES;
            FIND_DEPTH(t,x2,h2,dhdx2);
            FIND_DEPTH(t-dt/2,xmid,hmid,dhdxmid);

            CONTINUITY_PREDICTOR;
            FS_BC;
            U_BC_PRED;
            MOMENTUM;
            U_BC_CORR;
            INTERFACES;
            FIND_DEPTH(t,x2,h2,dhdx2);
            CONTINUITY_CORRECTOR;
            FS_BC;
            FCT(etha2);



            DRAWBED;
            DRAWSURFACE(x2,etha2);
```

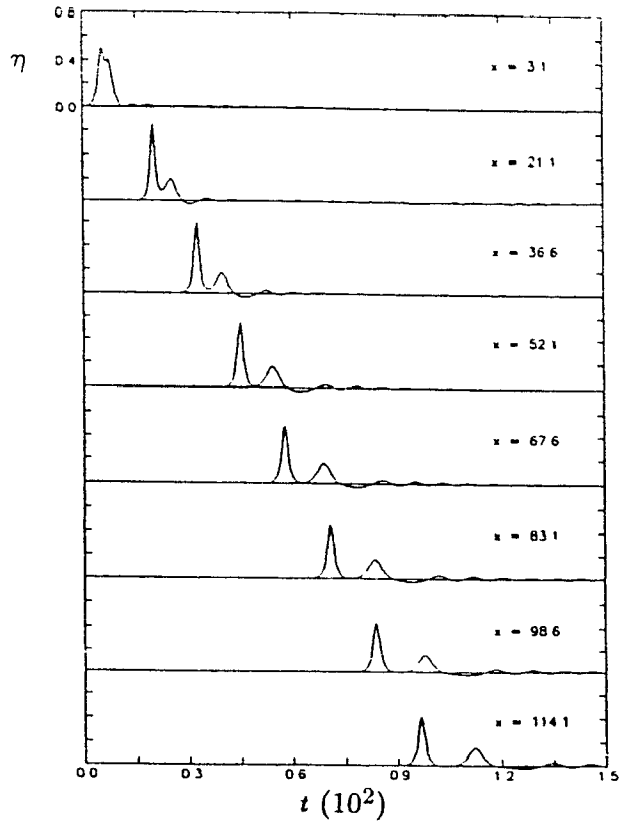
```
        if (iter mod 2) = 0 THEN
            begin
                writeln(disk,iter div 2);
                writeln(disk,etha2[gauge1]);
                writeln(disk,etha2[gauge2]);
                writeln(disk,etha2[gauge3]);
                writeln(disk,etha2[gauge4]);
                writeln(disk,etha2[gauge5]);
                writeln(disk,etha2[gauge6]);
                writeln(disk,etha2[gauge7]);
                writeln(disk,etha2[gauge8]);
                writeln(disk);
            end;

        for i:= 0 to n do
            begin
                etha1[i]:= etha2[i];
                h1[i]:= h2[i];
                x1[i]:= x2[i];
                dhdx1[i]:= dhdx2[i];
                u1[i]:= u2[i];
            end;
    end;
  close(disk);
END.
```

APPENDIX D

Supplementary results



a) Experiment



b) Computation

Figure D.1: Wave generated by a moving wall. Maximum horizontal velocity of the wall $v^* = 0.4$; total displacement $d = 2.5$; duration of the motion $t_f = 9.82$. *Note : The time coordinate corresponding to $t = 0$ may slightly differ in the computations and experiments.*

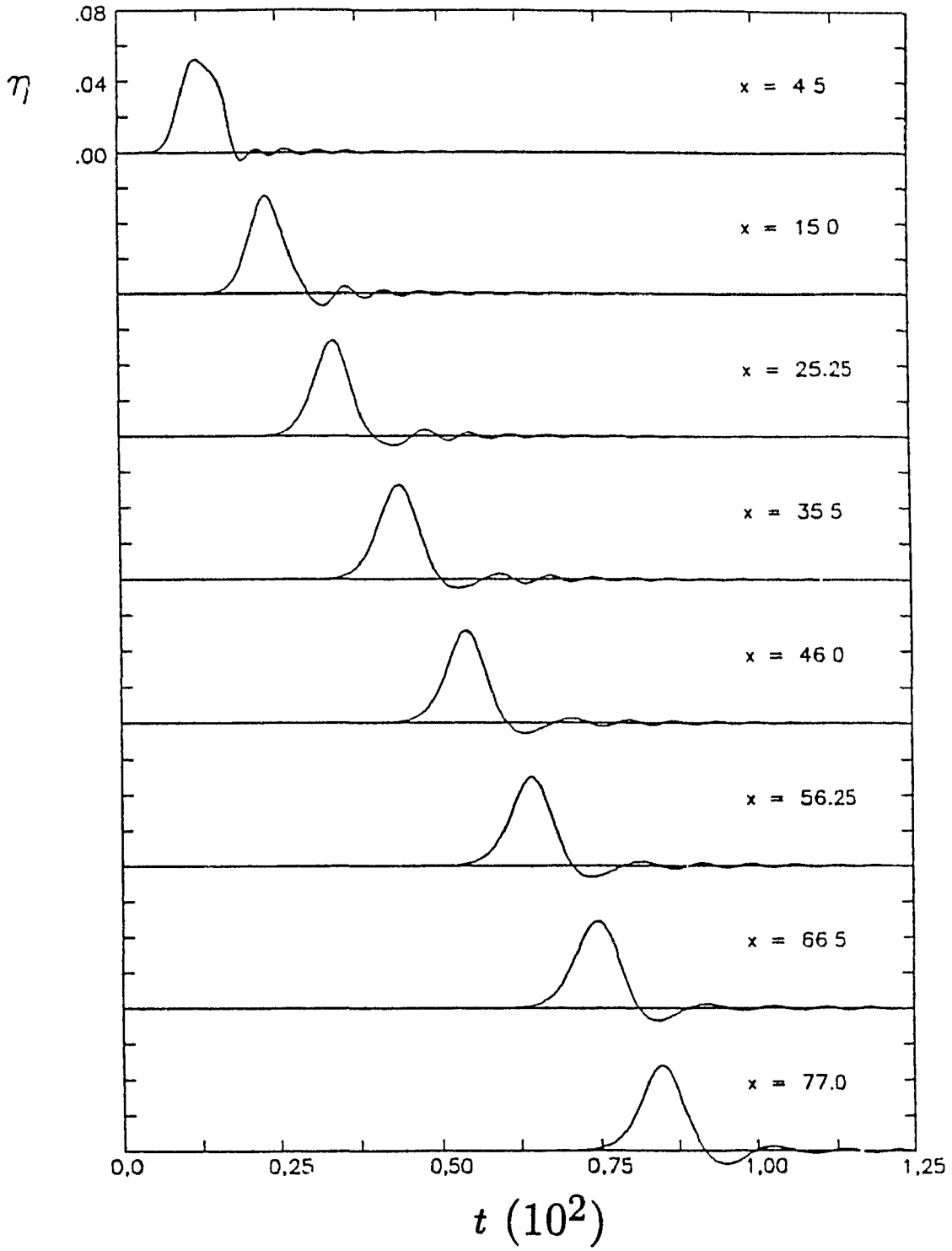


Figure D.2a: Wave generated by a submerged wedge. Slope $m = 0.268$;maximum horizontal velocity of the wedge $v^* = 0.2$; total displacement $d = 1.67$; duration of the bed motion $t_f = 13.12$.

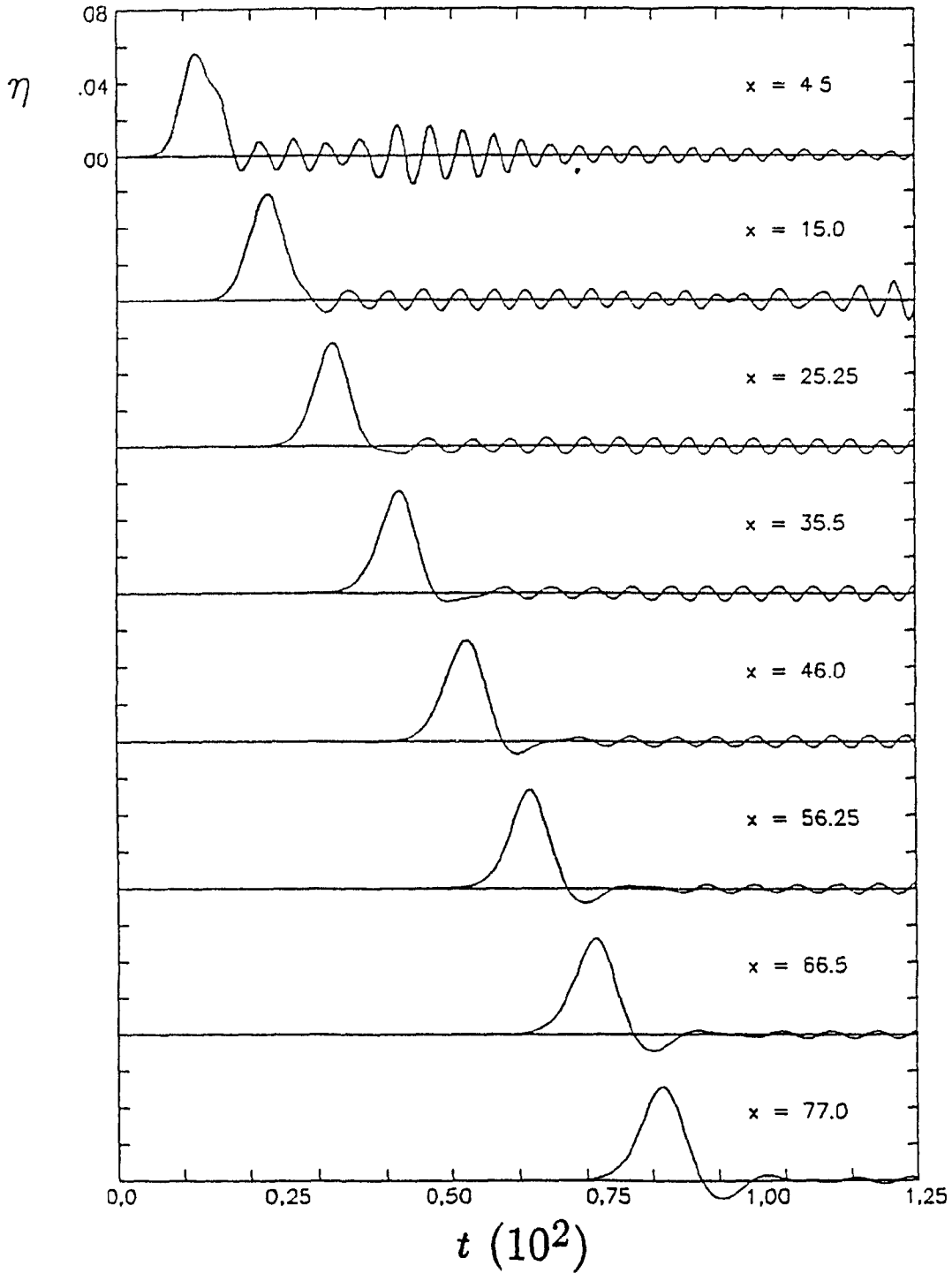


Figure D.2b: Wave generated by a submerged wedge. Results obtained without the use of the FCT algorithm.

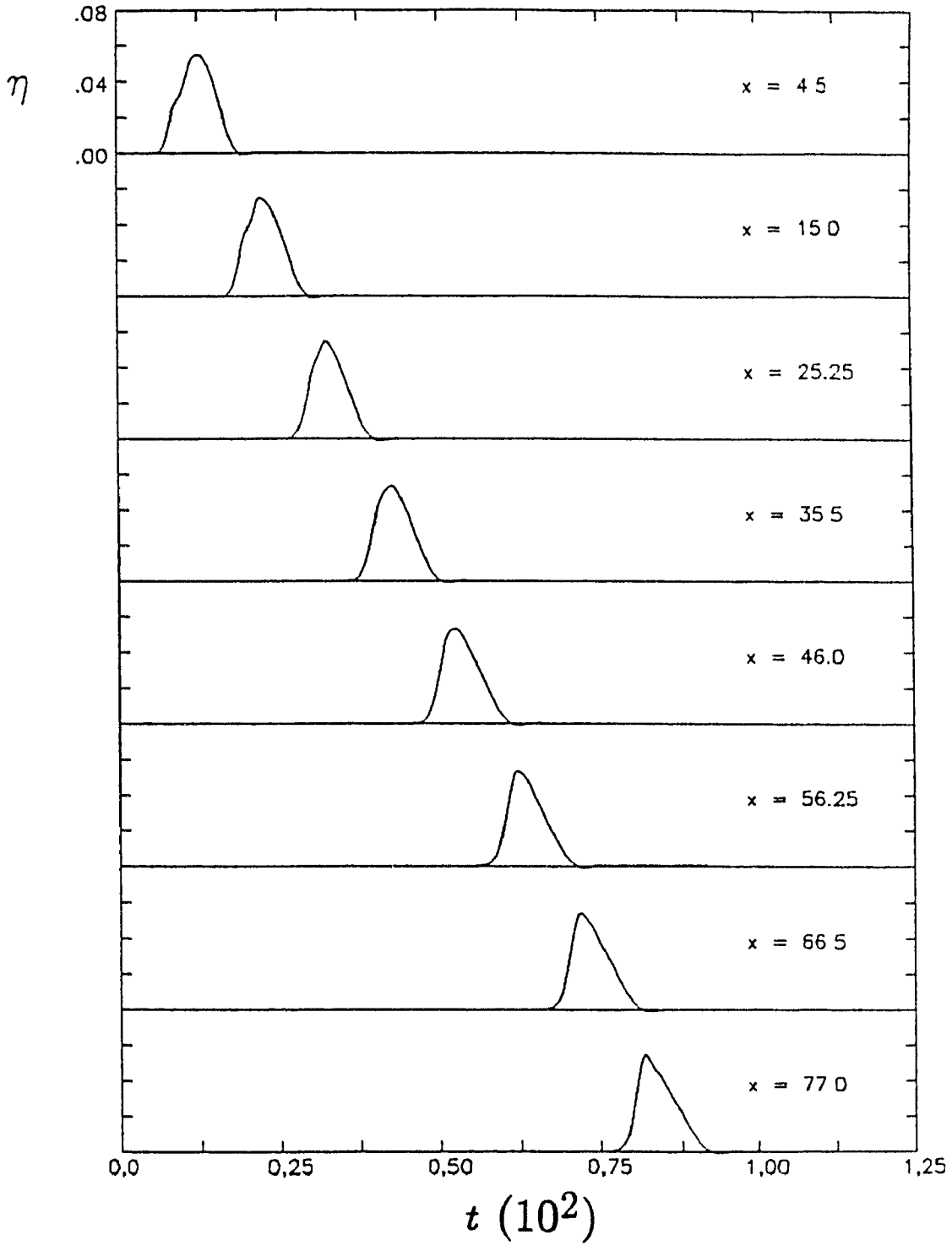


Figure D.2c: Wave generated by a submerged wedge. Computations based on the linearized long-wave theory.

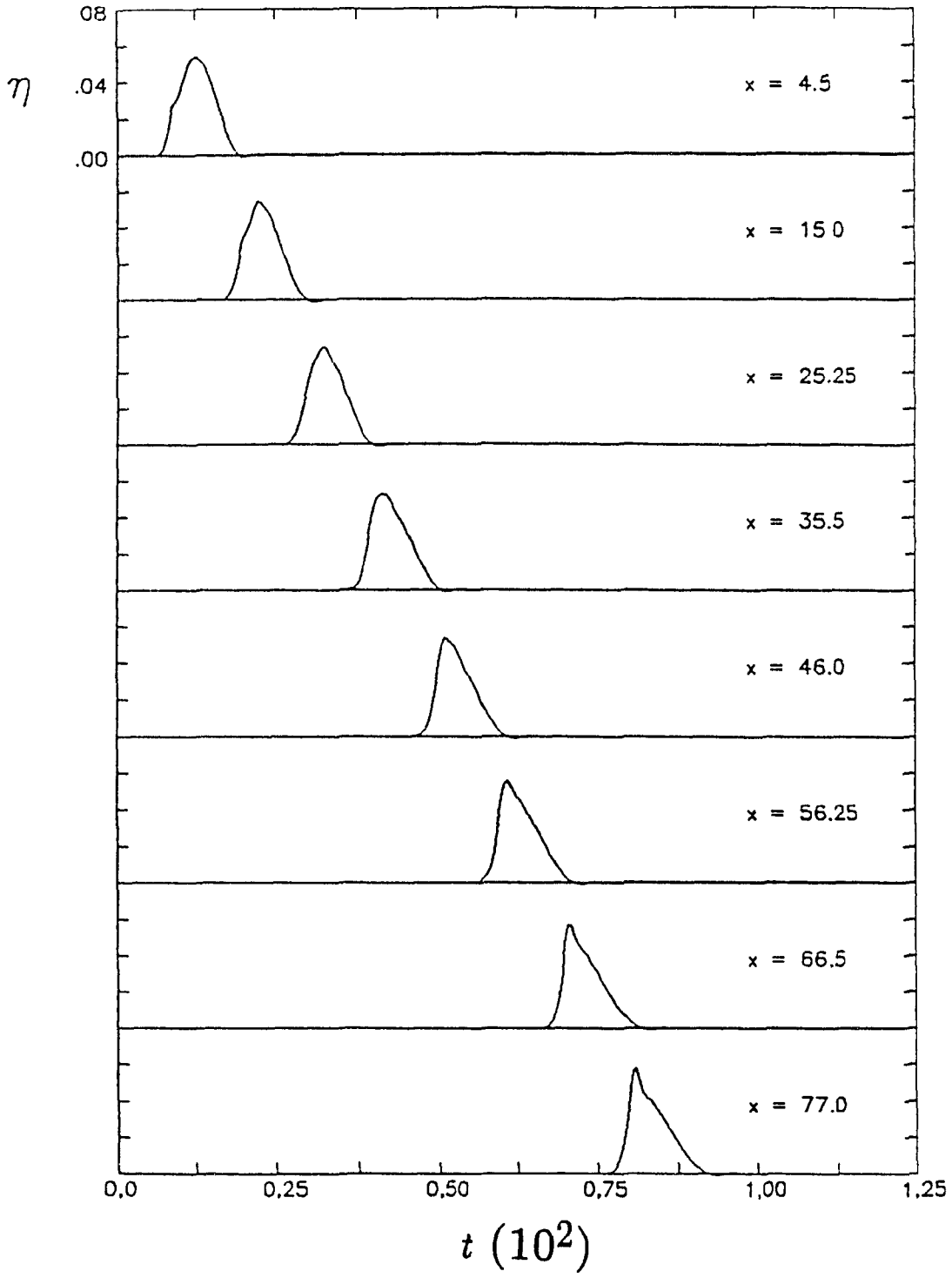


Figure D.2d: Wave generated by a submerged wedge. Computations based on the shallow-water, finite-amplitude theory.

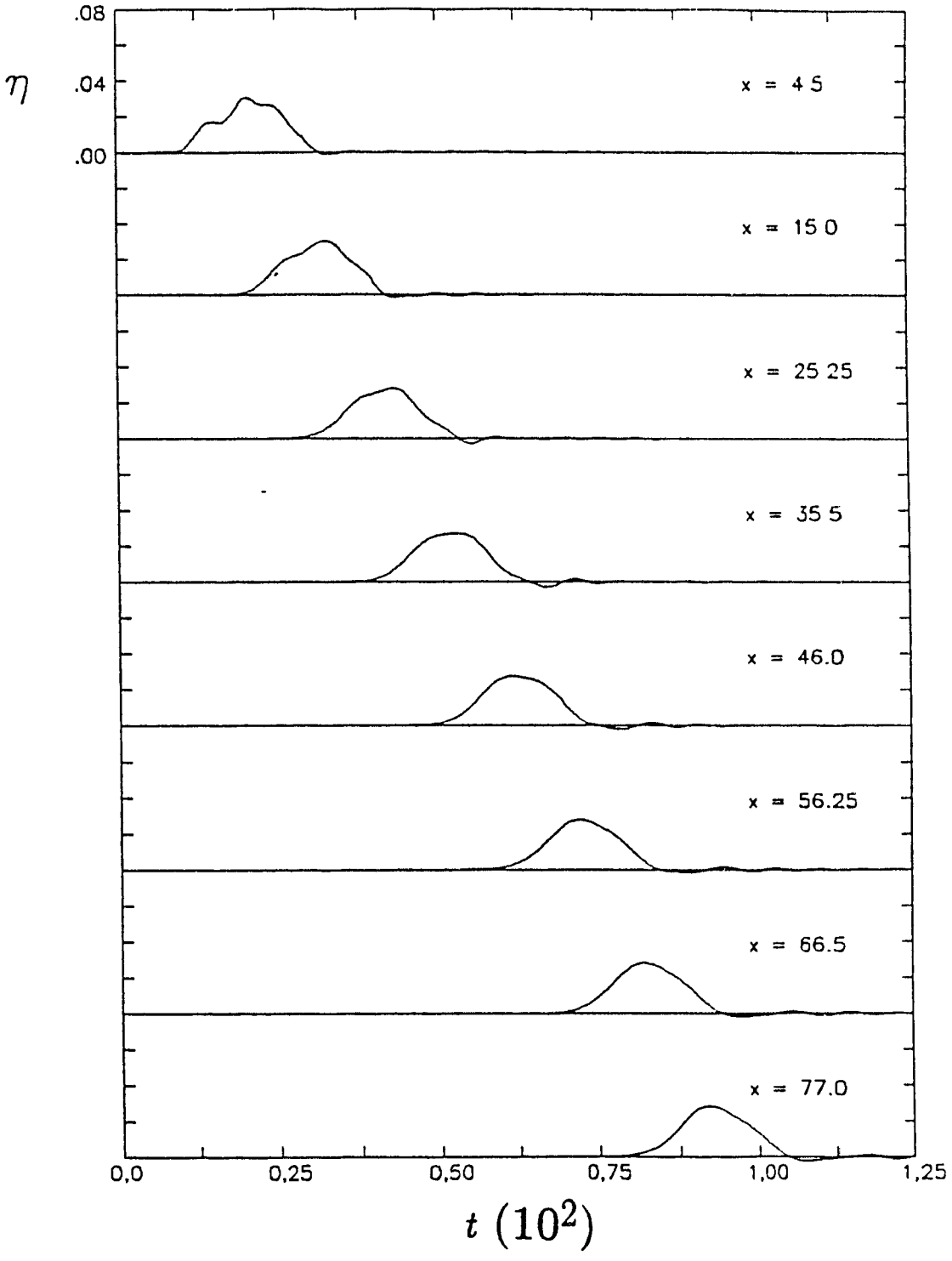


Figure D.2e: Wave generated by a submerged wedge. Slope $m = 0.268$;maximum horizontal velocity of the wedge $v^* = 0.1$; total displacement $d = 1.67$; duration of the bed motion $t_f = 26.23$.

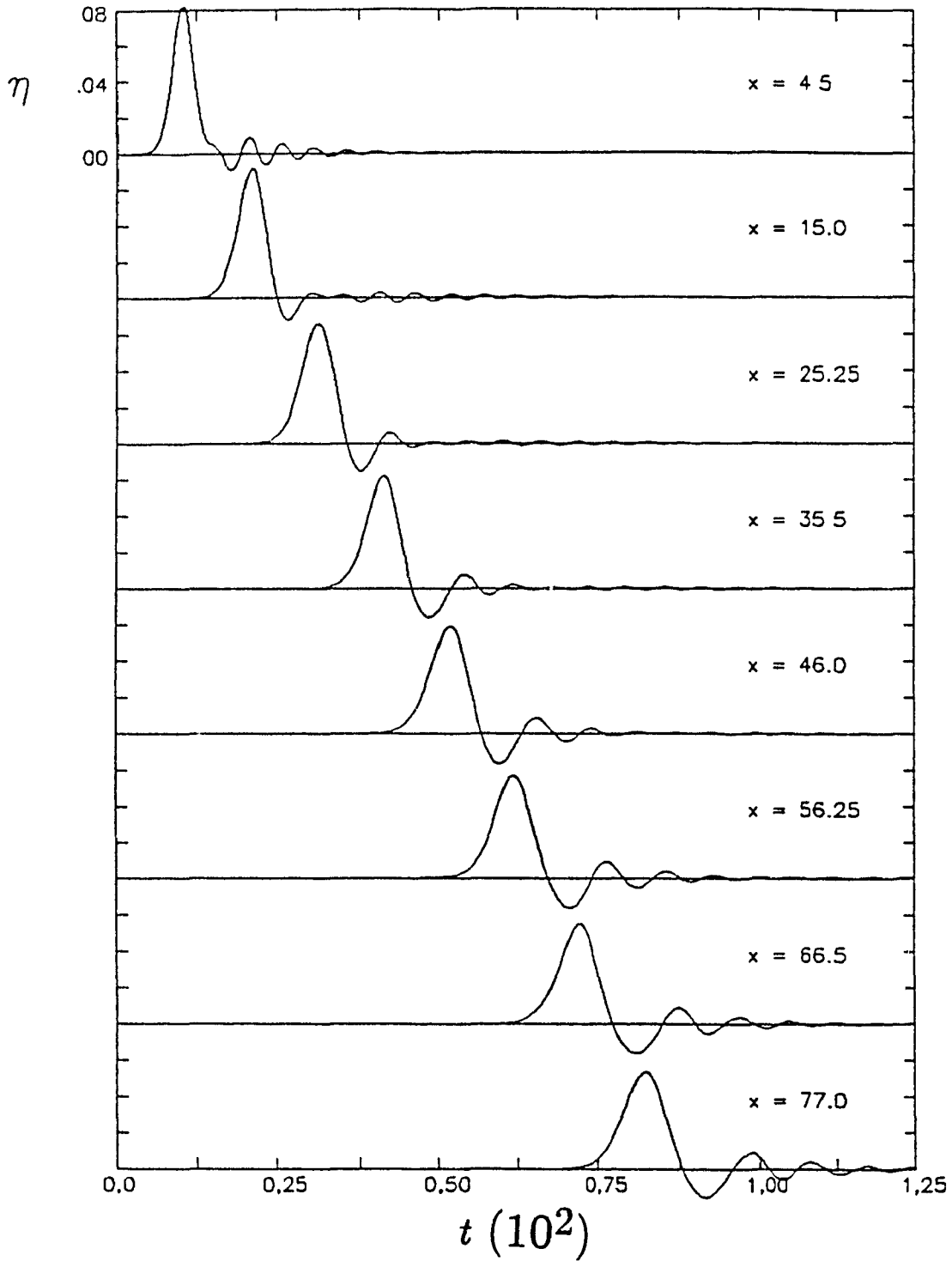
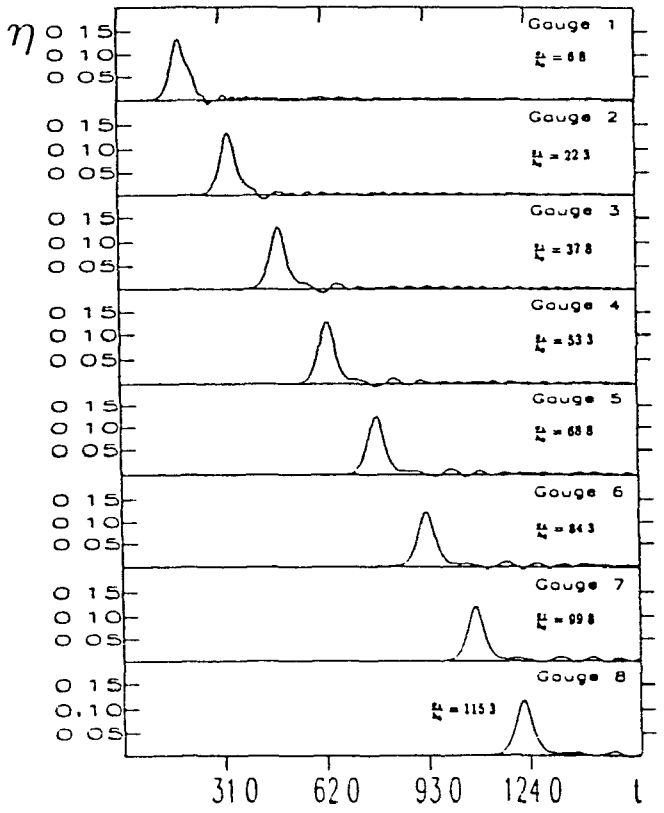
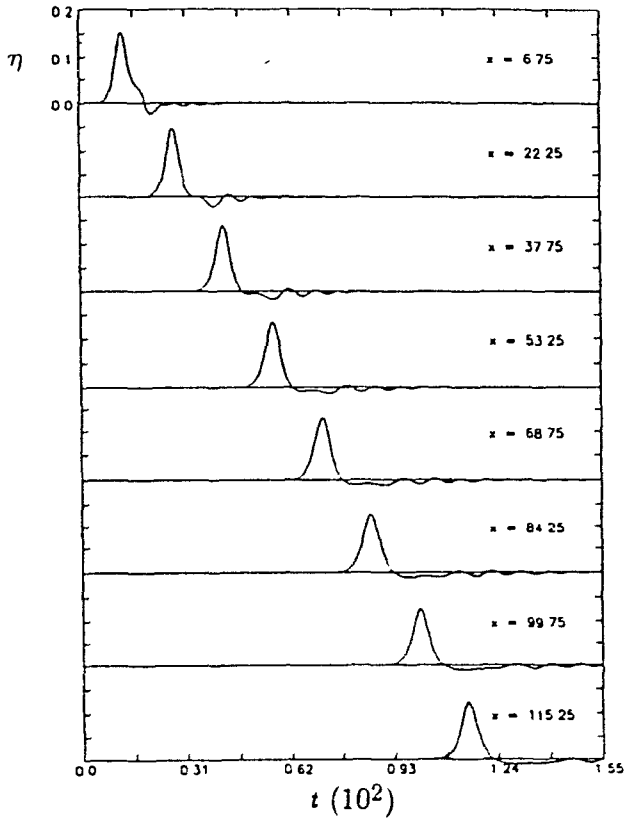


Figure D.2f: Wave generated by a submerged wedge. Slope $m = 0.577$;maximum horizontal velocity of the wedge $v^* = 0.2$; total displacement $d = 1.14$; duration of the bed motion $t_f = 8.94$.

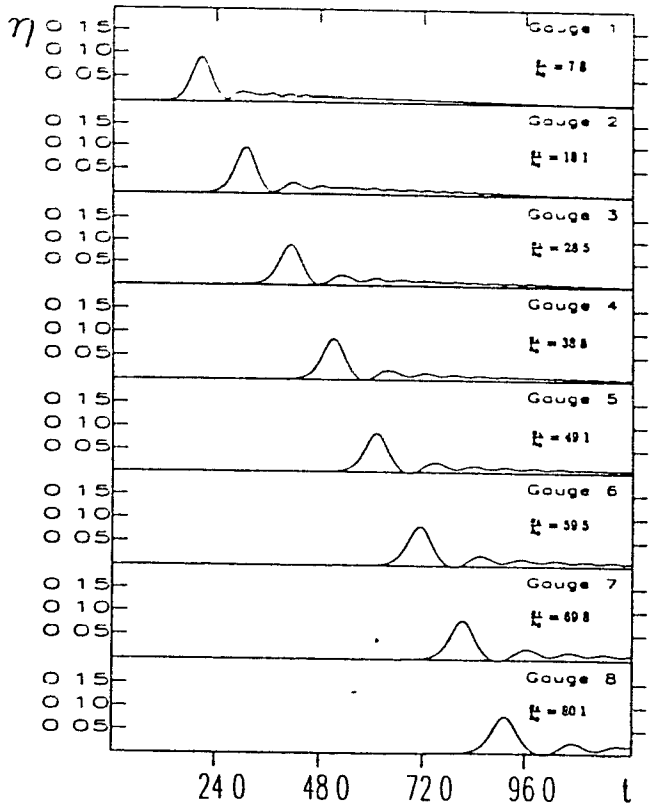


a) Experiment

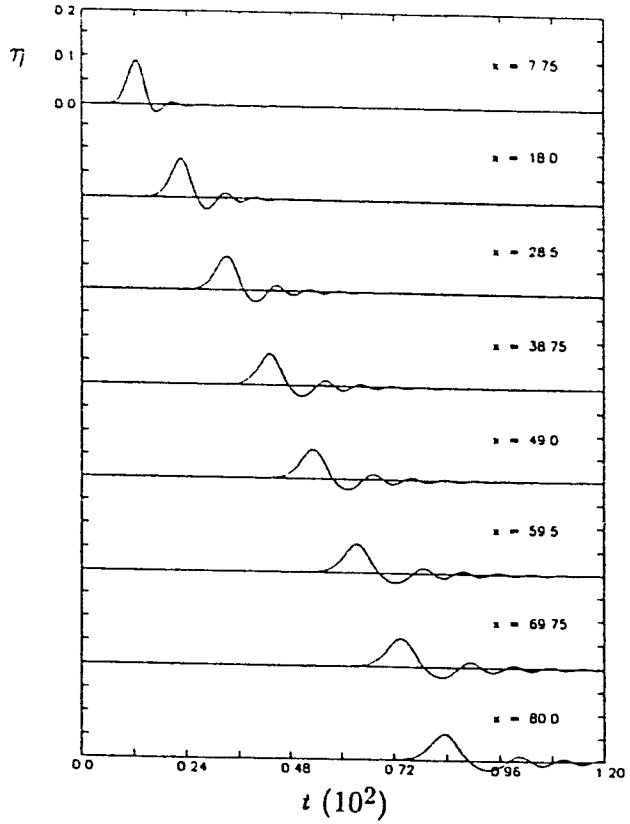


b) Computation

Figure D 2g: Wave generated by a submerged wedge. Slope $m = 0.268$;maximum horizontal velocity of the wedge $v^* = 0\ 4$; total displacement $d = 2\ 5$, duration of the bed motion $t_f = 9.75$

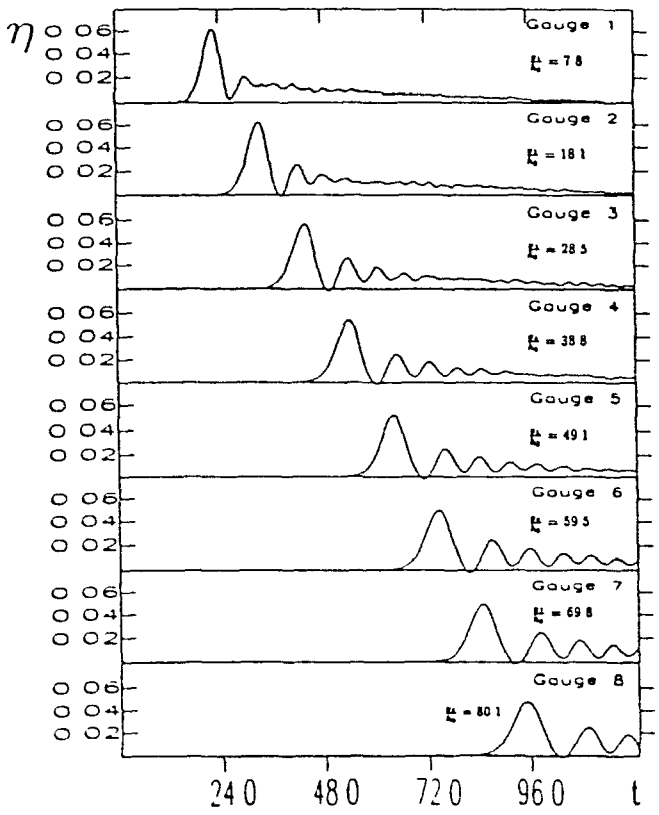


a) Experiment

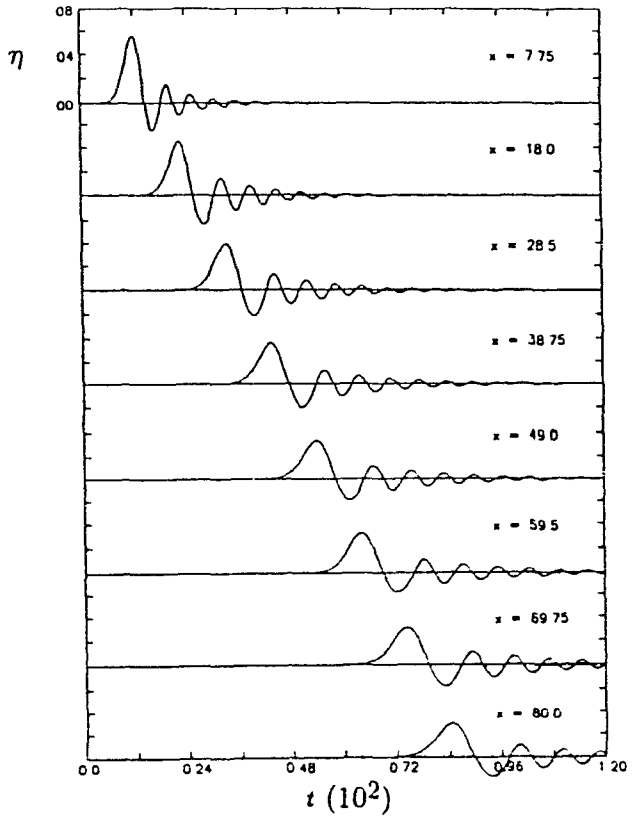


b) Computation

Figure D.2h: Wave generated by a submerged wedge. Slope $m = 0.577$;maximum horizontal velocity of the wedge $v^* = 0.2$; total displacement $d = 1.18$; duration of the bed motion $t_f = 6.18$.

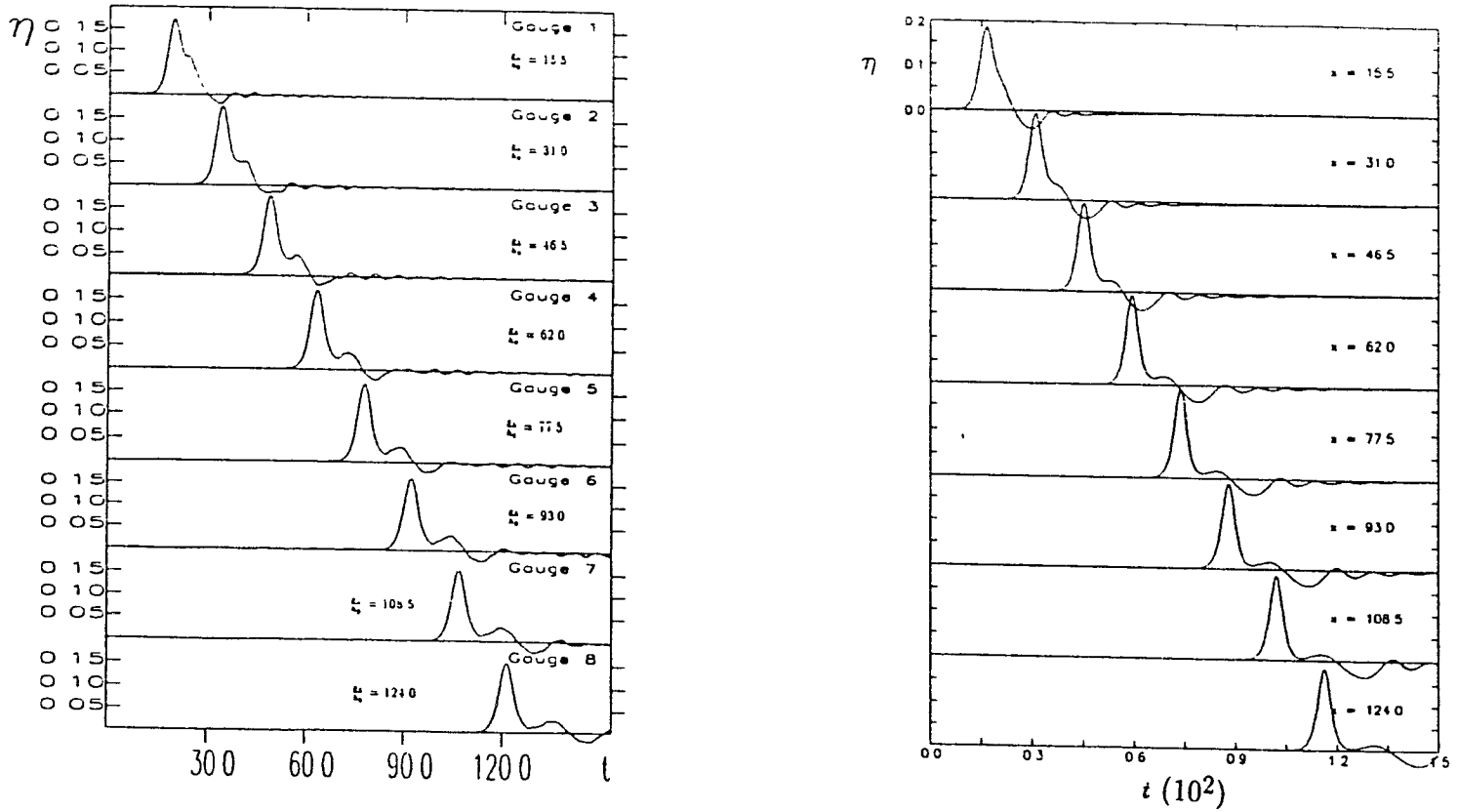


a) Experiment



b) Computation

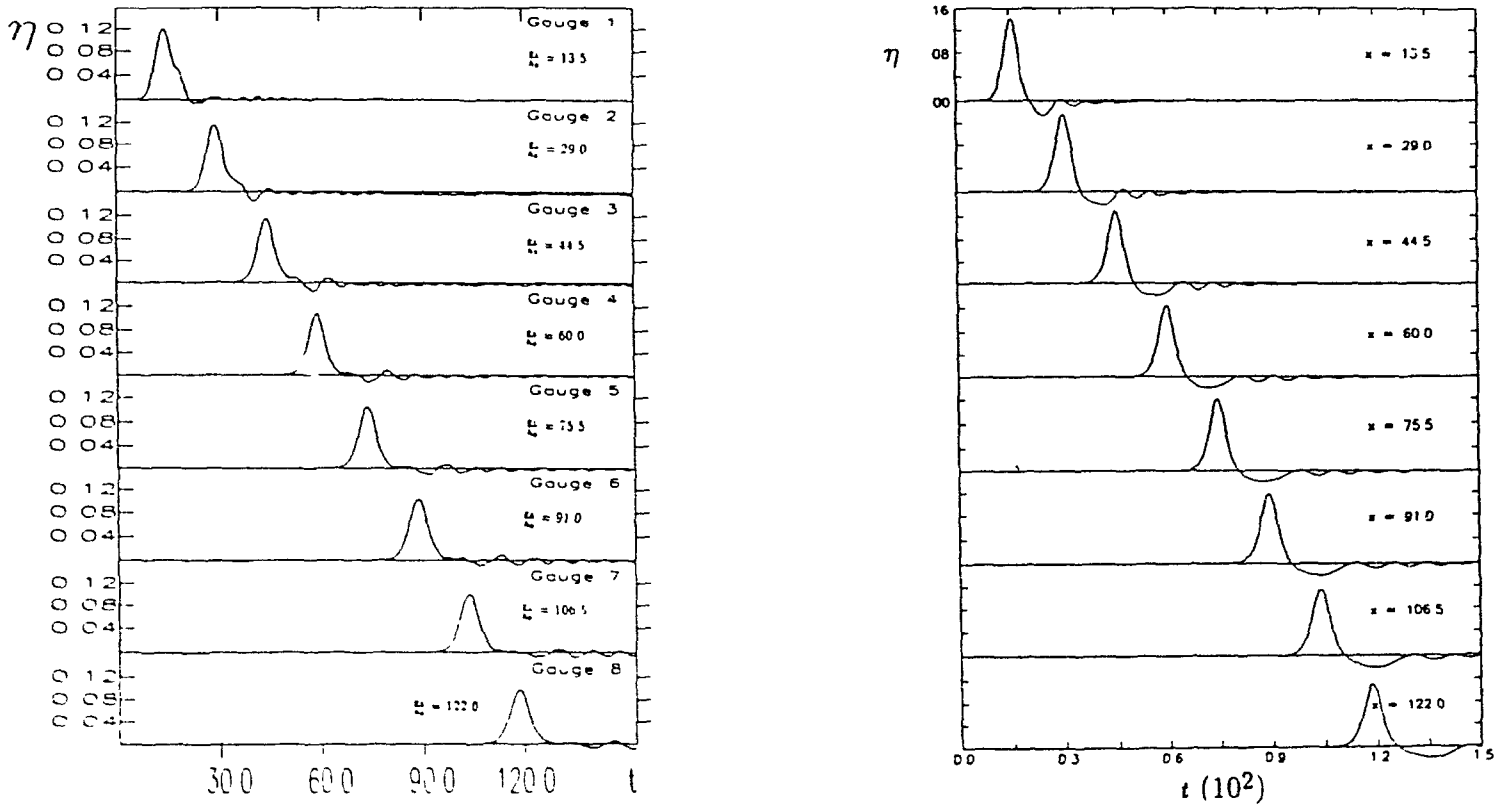
Figure D.2i: Wave generated by a submerged wedge Slope $m = 0.577$;maximum horizontal velocity of the wedge $v^* = 0.2$; total displacement $d = 0.85$, duration of the bed motion $t_f = 4.75$.



a) Experiment

b) Computation

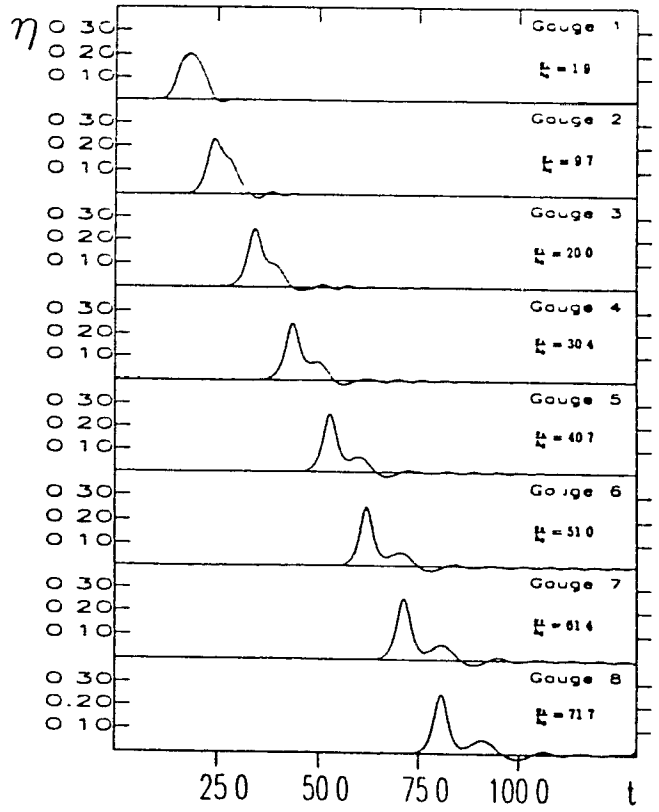
Figure D.3a: Wave generated by a rotating plate. Length of the plate $l = 5$;vertical pull velocity $V = 0.1$ at $x = 0$; duration of the bed motion $t_f = 5.0$; final slope $m_{max} = 0.1$.



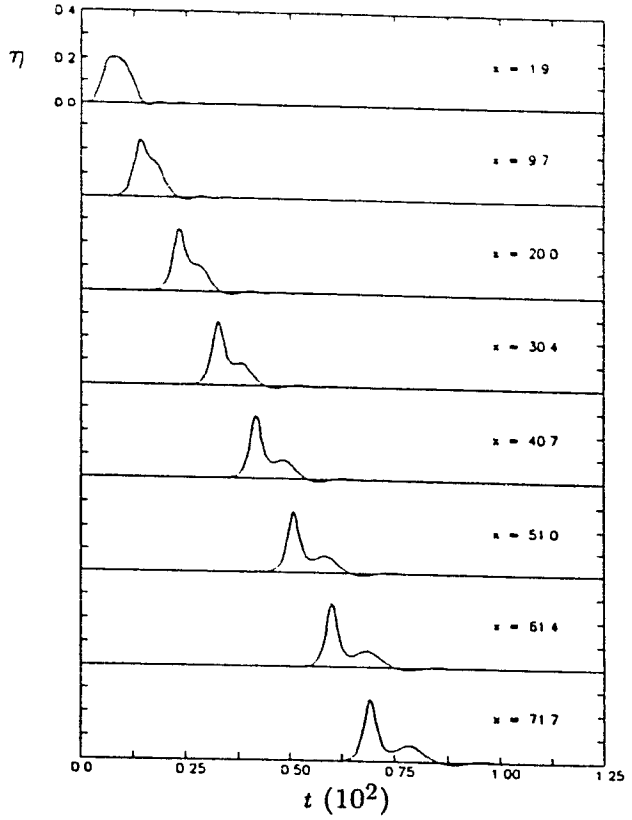
a) Experiment

b) Computation

Figure D 3b Wave generated by a rotating plate Length of the plate $l = 3$;vertical pull velocity $V = 0.1$ at $x = 0$; duration of the bed motion $t_f = 5.0$; final slope $m_{max} = \frac{5}{30}$



a) Experiment



b) Computation

Figure D.4: Wave generated by a moving wedge. Slope $m = 1$;maximum horizontal velocity of the wedge $v^* = 0.2$; total displacement $d = 1.67$; duration of the bed motion $t_f = 13.12$.