Speed Comparison of Solution Methods for the Obstacle Problem

Rebecca M. Carrington
Department of Mathematics and Statistics
McGill University, Montréal

August 2017

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science.

Acknowledgments

The work of this thesis would not have been possible without the persistent support of my advisor, Dr. Adam Oberman. I am very grateful for the time he invested in this project and the opportunity of working with him. His vast knowledge of many areas of applied mathematics was invaluable.

I am indebted to the greater body of the Mathematics and Statistics department for supporting me and helping me learn throughout the last few years. I want to thank the National Science and Engineering Research Council (NSERC) for supporting me with a PGS-M scholarship this past year.

Finally, I am immensely grateful for the continued support of my parents and brother throughout this journey.

But now, Lord, what do I look for?

My hope is in you.

Psalm 39:7

Abstract

Obstacle problems can be solved iteratively. Several solution methods are implemented and their speeds are compared. Speed is measured both in terms of the number of iterations required to converge and the average CPU time needed for one iteration for each method. The implementation is done using MATLAB for problems in one and two dimensions. The Euler iterative method requires a large number of iterations to converge to the solution. The semismooth Newton's method (SSNM) requires fewer iterations. Even fewer iterations are achieved with the combined method, which alternates between several Euler steps and one SSNM step. The behavior of the three different solution methods is compared extensively, and the combined method is declared as the best.

Abrégé

Les problèmes de l'obstacle peuvent être résolus itérativement. Plusieurs méthodes numériques sont développées et leurs vitesses sont comparées. La vitesse d'une méthode est déterminée en considérant le nombre d'itérations nécessaires pour converger ainsi que le temps moyen requis par le processeur pour une seule itération. La réalisation est faite avec MATLAB en une et deux dimensions. La méthode itérative d'Euler exige un grand nombre d'itérations pour converger vers la solution. La méthode de Newton semismooth (MNSS) exige moins d'itérations. Les deux méthodes sont combinées en alternant quelques pas d'Euler avec un pas de la MNSS. Encore moins d'itérations sont nécessaires pour cette méthode de combinaison. La performance des trois méthodes est comparée et la méthode de combinaison est déclarée gagnante.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Partial differential equations (PDEs) are equations that model real world phenomena. Obstacle problems give rise to a particular PDE [1, 16]. They can be motivated by considering an elastic membrane which is restricted to lie above an obstacle [16]. The optimal stopping problem in control theory can also be formulated as an obstacle problem [14]. Obstacle problems have two principle components: the obstacle, and an elliptic operator which determines how the obstacle is surmounted. The classical obstacle problem has as its operator the negative Laplace operator. The more general obstacle problem allows for nonlinear elliptic operators.

We are interested in solving obstacle problems for linear and nonlinear partial differential equations in one and two dimensions as fast as possible. Solving the problems requires their discretization onto a finite difference grid. Finite difference operators are used to approximate partial derivatives on the grid. We would like the speed of convergence to be independent of the grid size of the discretization. The speed of convergence is measured in terms of the number of iterations needed to converge as well as the average CPU time of one such iteration. We are looking for the most time efficient method for solving the obstacle problem.

The obstacle problem is formulated as a nonlinear complementarity problem (NCP) [11]. An equivalent formulation involving a system of equations with an underlying NCP function is used [9]. This system is solved numerically using an iterative approach. That is, an initial function is chosen, and an iterative update produces each subsequent function. The sequence of functions converges to the solution of the obstacle problem.

Three iterative methods will be implemented and their speeds compared for specific examples. The first method to be considered is the forward Euler iterative method [13]. We will show that the number of iterations required is highly dependent on the chosen grid size. The second method to be used is the semismooth Newton's method (SSNM) [15]. Newton's method cannot be applied directly because the NCP functions considered here are not smooth enough. Nonsmooth analysis will be called upon to prove the NCP functions are semismooth and to apply the SSNM to our problem. Finally, we propose a new solver that combines the existing Euler and semismooth Newton's methods. In the combined method, several iterations of the Euler solver precede one iteration of the SSNM solver. Numerical results will show the combined method to be the fastest method for specific examples in one and two dimensions.

The SSNM has been implemented for the Bratu obstacle problem with the continuously differentiable merit function of an NCP function using the preconditioned conjugate gradient (PCG) method in MATLAB [10]. A range of grid sizes all required only six SSNM iterations for convergence. However, the PCG inner iterations are expensive in terms of computation time, and it is unclear if this result extends to different obstacles.

The thesis is organized as follows. Chapter 2 covers background information on the obstacle problem and the finite difference grid. Chapter 3 introduces the theory behind the Euler method. Chapter 4 gives the numerical results for the Euler method. Chapter 5 contains an

introduction to nonsmooth analysis and the SSNM. Chapter 6 gives the numerical results for the SSNM. Chapters 7 and 8 describe the combined method and its implementation, respectively. Finally, Chapter 9 provides a summary of the work done.

Chapter 2

Background

2.1 The Classical Obstacle Problem

2.1.1 Energy Method for Laplace's Equation

As a motivation for the obstacle problem, we first consider a problem without an obstacle. Let Ω be an open, bounded subset of $\mathbb{R}^d$ with boundary, $\partial\Omega$, C^1 and $h : \partial\Omega \to \mathbb{R}$. The problem of minimizing the functional (called the Dirichlet energy),

$$E_D[u] := \tfrac{1}{2}\int_\Omega |\nabla u|^2 \, dx, \tag{2.1}$$

subject to $u \in \mathcal{A}_D := \{u \in C^2(\bar{\Omega}) \mid u = h \text{ on } \partial\Omega\}$ is equivalent to u solving the PDE

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega, \\ \quad u = h & \text{on } \partial\Omega. \end{cases}$$

This is a well-known fact in harmonic analysis [5].

2.1.2 The Obstacle Problem

We now add an obstacle and an external force to the problem. Let $g, f : \bar{\Omega} \to \mathbb{R}$ be given smooth functions. The obstacle, g, adds pointwise constraints to the function u, while f is a nonnegative external force. Define the new energy functional

$$E[u] := \int_\Omega \tfrac{1}{2}|\nabla u|^2 - fu\,dx, \tag{2.2}$$

and the set $\mathcal{A} := \{u \in H^1(\Omega) \mid u \geq g \text{ a.e. in } \Omega,\ u = h \text{ on } \partial\Omega\}$. Here $H^1(\Omega)$ denotes the space of square integrable functions whose gradient is also square integrable. The obstacle problem with Dirichlet boundary conditions is to minimize $E[\cdot]$ among all functions $u \in \mathcal{A}$. There exists a unique function $u \in \mathcal{A}$ that solves the obstacle problem, i.e. satisfying $E[u] = \min_{w \in \mathcal{A}} E[w]$ [5].

2.1.3 Complementarity Problems

Before reformulating the obstacle problem, we define the nonlinear complementarity problem (NCP) and NCP functions [11, 8]. Let $x \in \mathbb{R}^n$. By $x \geq 0$, we mean $x_i \geq 0$ for all $i = 1, \ldots, n$.

Definition 2.1 (NCP)**.** *Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be a given function. Finding $x \in \mathbb{R}^n$ satisfying*

$$x \geq 0,\ F(x) \geq 0,\ x^T F(x) = 0 \quad \Leftrightarrow \quad x_i \geq 0,\ F_i(x) \geq 0,\ x_i F_i(x) = 0 \quad \forall i = 1, \ldots, n$$

is called a (nonlinear) complementarity problem (NCP)*.*

Definition 2.2 (NCP function)**.** *A function $\varphi : \mathbb{R}^2 \to \mathbb{R}$ is called an* NCP function *if*

$$\varphi(a, b) = 0 \quad \Leftrightarrow \quad a \geq 0,\ b \geq 0,\ ab = 0.$$

We now introduce two NCP functions that will be used in our numerical implementation of the obstacle problem.

Proposition 2.3. *$\varphi_{min}(a,b) := \min(a,b)$ is an NCP function.*

Proof. Suppose $\min(a,b) = 0$. If $a \leq b$, then $a = 0$ and $b \geq 0$. Therefore, $ab = 0$. In particular, $a \geq 0$, $b \geq 0$ and $ab = 0$. By symmetry, the same is obtained if $b \leq a$. On the other hand, suppose $a \geq 0$, $b \geq 0$ and $ab = 0$. Then either $a = 0$ or $b = 0$. If $a = 0$, then $\min(a,b) = 0$ since $b \geq 0$. Again, by symmetry, the same is obtained if $b = 0$.

□

Proposition 2.4. *$\varphi_{FB}(a,b) := \sqrt{a^2+b^2} - a - b$ is an NCP function. It is called the Fischer-Burmeister function [8].*

Proof. To prove φ_{FB} is an NCP function, we need to prove $\varphi_{FB}(a,b) = 0 \;\Leftrightarrow\; a \geq 0$, $b \geq 0$, $ab = 0$.

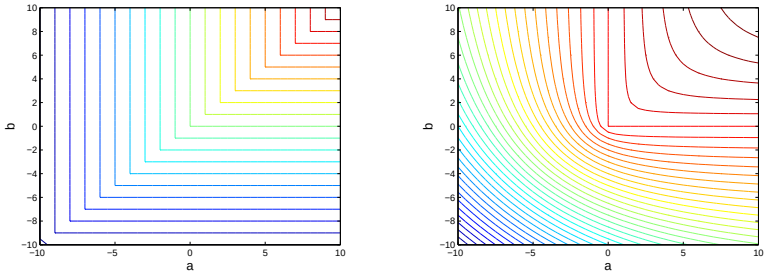
$\Rightarrow$ Assume first $\varphi_{FB}(a,b) = 0$. Then $\sqrt{a^2+b^2} - a - b = 0$ implies $(a+b)^2 = a^2 + b^2$, which is equivalent to $a^2 + 2ab + b^2 = a^2 + b^2$. Therefore we obtain $ab = 0$. Now suppose $a < 0$. Since $ab = 0$, $b = 0$ and therefore $\varphi_{FB}(a,b) = \sqrt{a^2} - a > 0$. This contradicts the assumption. So we must have $a \geq 0$, and by symmetry, $b \geq 0$.

$\Leftarrow$ Next, assume $a \geq 0$, $b \geq 0$, $ab = 0$. Then $a^2 + 2ab + b^2 = a^2 + b^2$, which is equivalent to $(a+b)^2 = a^2 + b^2$. Using the fact that $a \geq 0$ and $b \geq 0$, we can take the square root of both sides of the last equation, obtaining $\sqrt{a^2+b^2} - a - b = 0$, that is, $\varphi_{FB}(a,b) = 0$.

We have proved that φ_{FB} is an NCP function. □

The level sets of φ_{min} and φ_{FB} are shown in Figure 2.1. From these images we can identify areas of nondifferentiability: φ_{min} is not differentiable on the line $a = b$ while φ_{FB} is not differentiable at the point $(a,b) = (0,0)$.

We now introduce a more compact formulation of a complementarity problem. Let



(a) Level sets of φ_{min} *(b) Level sets of φ_{FB}*

Figure 2.1: Comparison of NCP functions

$\Phi : \mathbb{R}^n \to \mathbb{R}^n$ be defined as

$$\Phi(x) := \begin{pmatrix} \varphi(x_1, F_1(x)) \\ \vdots \\ \varphi(x_n, F_n(x)) \end{pmatrix}, \tag{2.3}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ and $\varphi : \mathbb{R}^2 \to \mathbb{R}$ is a given NCP function. The following theorem gives a correspondence between the NCP and the nonlinear system of equations $\Phi(x) = 0$ [11].

Theorem 2.5. *Let Φ be given by (2.3). Then $x \in \mathbb{R}^n$ solves the NCP if and only if $\Phi(x) = 0$.*

Proof. The result follows from the definition of Φ:

$$\begin{aligned} \Phi(x) = 0 \quad &\Leftrightarrow \quad \varphi(x_i, F_i(x)) = 0 \ \ \forall i = 1, \ldots, n, \\ &\Leftrightarrow \quad x_i \geq 0,\ F_i(x) \geq 0,\ x_i F_i(x) = 0 \ \ \forall i = 1, \ldots, n, \end{aligned}$$

which is the NCP. The second equivalence follows from the fact that φ is an NCP function.

□

2.1.4 Variational Inequality

We now consider an alternate formulation of the obstacle problem. The unique minimizer of Equation (2.2) satisfies

$$\int_\Omega \nabla u \cdot \nabla(w - u)\, dx \geq \int_\Omega f(w - u)\, dx \text{ for all } w \in \mathcal{A},$$

which is called a variational inequality [5]. Under a weak regularity condition, the variational inequality is equivalent to

$$u - g \geq 0,\ -\Delta u - f \geq 0,\ (u-g)(-\Delta u - f) = 0,\ \text{ a.e. in } \Omega, \tag{2.4}$$

which is a shifted NCP [5, 16]. To obtain the regular NCP, define $v := u - g$. Then Equation 2.4 becomes

$$v \geq 0,\ -\Delta(v+g) - f \geq 0,\ v(-\Delta(v+g) - f) = 0,\ \text{ a.e. in } \Omega, \tag{2.5}$$

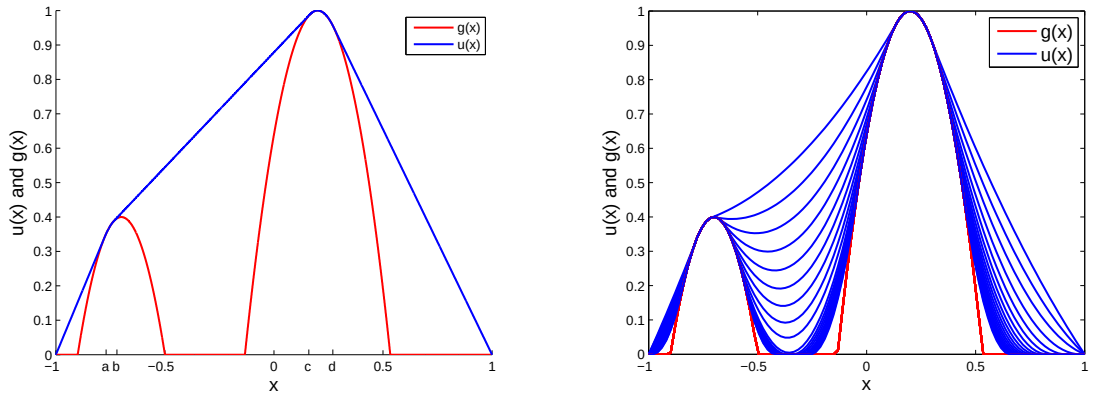
which is an NCP with $F(v) = -\Delta(v+g) - f$. Equation 2.5 is actually what is called a linear complementarity problem. The more general NCP must be considered when dealing with the general obstacle problem.

This NCP formulation implies that at each $x \in \Omega$, the solution u satisfies $-\Delta u(x) - f(x) = 0$ and $u(x) - g(x) \geq 0$ or $u(x) - g(x) = 0$ and $-\Delta u(x) - f(x) \geq 0$. Define the following two regions:

$$C := \{x \in \Omega \mid u(x) - g(x) = 0\},$$
$$D := \{x \in \Omega \mid u(x) - g(x) \geq 0\}.$$

The set $F := C \cap D = \{x \in \Omega \,|\, u(x) - g(x) = 0, -\Delta u(x) - f(x) = 0\}$ is called the *free boundary* [5]. The free boundary plays an important role in the solution methods presented in later chapters.

Figure 2.2 illustrates the one-dimensional obstacle problem with $\Omega = (-1, 1)$ and $h \equiv 0$.



(a) $f \equiv 0$ *(b)* f *constant*

Figure 2.2: Illustration of the obstacle problem in 1d

The problem becomes

$$\begin{cases} u(x) - g(x) \geq 0,\ -u_{xx}(x) - f(x) \geq 0,\ (u(x) - g(x))(-u_{xx}(x) - f(x)) = 0, & x \in (-1, 1), \\ u(x) = 0, & x \in \{-1, 1\}. \end{cases}$$

For the problem shown in Figure 2.2a, $C = [a, b] \cup [c, d]$ and $D = [-1, a] \cup [b, c] \cup [d, 1]$. The solution at points in set C coincides with the obstacle and is concave. The solution at points in set D lies above the obstacle and is a line. The free boundary is the set $F = \{a, b, c, d\}$.

The force f is nonnegative by assumption. The larger f is, the closer the solution to the obstacle problem is to the obstacle, g. Smaller values of f lead to solutions close to the case shown in Figure 2.2a. This is illustrated in Figure 2.2b, with

$$f(x) = \begin{cases} k, & x \in (-1, 1), \\ 0, & x \in \{-1, 1\}, \end{cases} \tag{2.6}$$

where $k \in \{1, 2, \ldots, 20\}$.

By Theorem 2.5, the shifted version of Equation 2.4 is equivalent to solving the system

of equations $\varphi(-\Delta u(x) - f(x), u(x) - g(x)) = 0,\ \ x \in \Omega$, where φ is any NCP function. Adding the given boundary data, the problem becomes

$$\begin{cases} \varphi(-\Delta u(x) - f(x), u(x) - g(x)) = 0, & x \in \Omega, \\ u(x) = h(x), & x \in \partial\Omega. \end{cases} \tag{2.7}$$

Equation 2.7 is the formulation of the classical obstacle problem that will be used in the remainder of this thesis. For example, if the minimum function (Proposition 2.3) is chosen, the obstacle problem becomes

$$\begin{cases} \min(-\Delta u(x) - f(x), u(x) - g(x)) = 0, & x \in \Omega \\ u(x) = h(x), & x \in \partial\Omega. \end{cases} \tag{2.8}$$

It is important to note that the minimum function is not differentiable exactly at the points along the free boundary.

2.2 Applications

Problems in a variety of fields take the form of obstacle problems. Two of these applications are highlighted in the following subsections.

2.2.1 Deformation of an Elastic Membrane

The classical obstacle problem models the equilibrium position of a membrane in a domain which is restricted to lie above an obstacle g in the interior of the domain and whose boundary position is fixed. Consider a two-dimensional membrane. A membrane is a thin plate acting only in tension [16]. Assume the membrane is equally stretched in all directions and subject to an external force f on a domain $\Omega \in \mathbb{R}^2$. Suppose each point $x \in \Omega$ is displaced

perpendicularly to the plane by $u(x)$ and each point $x \in \partial\Omega$ is displaced perpendicularly to the plane by $h(x)$. The surface area after deformation of the membrane is

$$SA[u] = \int_\Omega \sqrt{1 + |\nabla u|^2}\, dx.$$

Minimizing SA subject to $u \in \mathcal{A}_D$ is a minimal surface problem [5]. Using the Taylor approximation,

$$SA[u] \approx \int_\Omega 1 + \tfrac{1}{2}|\nabla u|^2\, dx.$$

The solution of the minimization of the approximation of SA subject to $u \in \mathcal{A}_D$ is therefore an approximation to the solution of the minimal surface problem. The change in surface area is equal to the deformation energy,

$$D[u] = \int_\Omega \tfrac{1}{2}|\nabla u|^2\, dx,$$

up to a positive constant of elasticity. The external force f does work

$$F[u] = \int_\Omega f u\, dx.$$

Therefore, the total potential energy is given by $E = D - F$. Suppose in addition that the membrane is constrained to lie above an obstacle, g. This problem is equivalent to minimizing Equation 2.2 subject to $u \in \mathcal{A}$, which is exactly the obstacle problem defined in Section 2.1.2 [16].

2.2.2 Optimal Stopping

There is also an optimal stopping interpretation of the obstacle problem [14]. Consider a diffusion process of a particle undergoing Brownian motion. The optimal stopping problem is to choose a stopping time to minimize an expected cost. When the process is stopped, a

cost is incurred. The minimum expected cost over all possible stopping times is the solution to an obstacle problem [6].

2.3 The General Obstacle Problem

The need for a more general obstacle problem is motivated by the following example. Define the (negative) convex envelope of a function g as the infimum of all concave functions which are minorized by g: $u(x) = \inf\{v(x) \mid v \text{ is concave, } v(y) \geq g(y) \text{ for all } y \in \mathbb{R}^d\}$. The convex envelope is the solution to the PDE

$$\min(-\lambda_d[u](x), u(x) - g(x)) = 0, \quad x \in \Omega, \tag{2.9}$$

where $\lambda_d[u](x)$ is the largest eigenvalue of the Hessian $D^2u(x)$ [14]. Note that for $d = 1$, the convex envelope is the solution to the classical obstacle problem (2.8) with $f \equiv 0$, as illustrated in Figure 2.2a. For $d = 2$, this is no longer true since $-u_{xx} - u_{yy} \neq -\lambda_2[u]$ in general. The PDE (2.9) is an example of the more general obstacle problem with the minimum function, φ_{min}, as the NCP function,

$$\begin{cases} \min(F[u](x) - f(x), u(x) - g(x)) = 0, & x \in \Omega, \\ u(x) = h(x), & x \in \partial\Omega, \end{cases} \tag{2.10}$$

where $F[u] : \bar{\Omega} \to \mathbb{R}$ is an elliptic operator. However, by Theorem 2.5, (2.10) can be formulated as an NCP and hence any NCP function φ leads to an equivalent problem:

$$\begin{cases} \varphi(F[u](x) - f(x), u(x) - g(x)) = 0, & x \in \Omega, \\ u(x) = h(x), & x \in \partial\Omega. \end{cases} \tag{2.11}$$

The same definitions of the sets C, D and F exist for the general obstacle problem.

Defining $G[u] := \varphi(F[u] - f, u - g)$, the general problem can be rewritten as

$$\begin{cases} G[u] = 0, & \text{on } \Omega, \\ u = h, & \text{on } \partial\Omega. \end{cases}$$

The numerical implementation presented in the following chapters only deals with the classical obstacle problem, however the same concepts could also be applied to a general obstacle problem with a nonlinear underlying operator F.

2.4 Problem Discretization

To solve the obstacle problem numerically, it is first discretized onto the finite difference grid. The domain Ω is discretized in space. There must also be a discretization in time to keep track of the iterations in the iterative process. Let $(x, t) \in \Omega_T := \bar{\Omega} \times [0, T]$, and let $u(x, t) : \Omega_T \to \mathbb{R}$. The domain Ω_T is discretized by a spacing $h > 0$ in all components of x, and a time step $dt > 0$ in t. The discretized grid is given by

$$\Omega_T^{h,dt} = h\mathbb{Z}^d \times [0, dt, 2dt, \ldots, K_T dt].$$

Here we assume $K_T dt = T$, since if this is not the case, it can be made true by rounding. Functions on the grid, $u_J^k : \Omega_T^{h,dt} \to \mathbb{R}$ are defined as

$$u_J^k = u(hJ, kdt),$$

where $k \in \mathbb{N}$ and $J \in \mathbb{Z}^d$ is a d-tuple of indices.

For example, the one-dimensional domain $\Omega = (-L, L)$, $L > 0$, has $N = \frac{2L}{h} + 1$ grid points, where h is the grid spacing. Homogeneous Dirichlet boundary conditions are achieved by

setting $u(L, ndt) = u(-L, ndt) = 0$, for all $n \geq 0$. In general, if w denotes the width of the domain Ω, the number of grid points is $N = \frac{w}{h} + 1$. For a two-dimensional rectangular grid of size $w_x \times w_y$ and grid spacing h in both directions, the number of grid points is $N = \left(\frac{w_x}{h} + 1\right)\left(\frac{w_y}{h} + 1\right)$, and homogeneous Dirichlet boundary conditions are defined in an analogous way to the one-dimensional case.

2.5 Finite Difference Operators

We now introduce the standard finite difference operators which were used to approximate the partial derivatives in the obstacle problem. The Taylor expansion for a smooth function applied at $x + h$ and $x - h$ is

$$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(x) + \mathcal{O}(h^4),$$
$$u(x-h) = u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{6}u'''(x) + \mathcal{O}(h^4).$$

Adding the two equations and isolating the $u''(x)$ term yields the one-dimensional second order centered difference,

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \mathcal{O}(h^2),$$

which is second order accurate. The notation we use for the one-dimensional second order centered difference at grid point j and iterate k is

$$(u_{xx}^C)_j^k = \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2}. \tag{2.12}$$

The two-dimensional Laplacian is

$$\begin{aligned}\Delta u(x,y) &= \frac{u(x+h,y)-2u(x,y)+u(x-h,y)}{h^2} + \frac{u(x,y+h)-2u(x,y)+u(x,y-h)}{h^2} + \mathcal{O}(h^2),\\ &= \frac{u(x+h,y)+u(x-h,y)+u(x,y+h)+u(x,y-h)-4u(x,y)}{h^2} + \mathcal{O}(h^2),\end{aligned}$$

which is second order accurate. Similarly, the notation for the two-dimensional Laplacian on our grid is

$$(\Delta u^C)^k_{i,j} = \frac{u^k_{i-1,j}+u^k_{i+1,j}+u^k_{i,j-1}+u^k_{i,j+1}-4u^k_{i,j}}{h^2}. \tag{2.13}$$

Note that the two-dimensional Laplacian is simply the sum of the one-dimensional Laplacian in the two grid directions: $\Delta u^C = u^C_{xx} + u^C_{yy}$.

Chapter 3

Forward Euler Iterative Method

Consider the PDE with Dirichlet boundary conditions,

$$\begin{cases} G[u](x) = 0 & \text{for } x \text{ in } \Omega, \\ u(x) = h(x) & \text{for } x \text{ on } \partial\Omega. \end{cases} \tag{3.1}$$

The first method to solve (3.1) considered is the forward Euler iterative method. Instead of solving the time independent PDE, a time dependent PDE will be solved. More specifically, the Euler method is used to solve the PDE $u_t + G[u] = 0$ instead of $G[u] = 0$. Then the discretized version of the new time dependent PDE is

$$\frac{u^{k+1} - u^k}{dt} + G[u^k] = 0.$$

Algorithm 1 Forward Euler obstacle solver

Let $G : \mathbb{R}^N \to \mathbb{R}^N$ be given. Select $u^0 \in \mathbb{R}^N$ and set $k := 0$.

1: Unless a stopping rule is satisfied, set:

$$u^{k+1} := u^k - dtG[u^k]. \tag{3.2}$$

2: Set $k := k + 1$, and go to Step 1.

Algorithm 1 is the Euler solver for the obstacle problem with general operator $F[u]$

and arbitrary NCP function φ in pseudocode. For our case with the obstacle problem, $G[u^k] = \varphi\left(F[u^k] - f, u^k - g\right) = 0$. From this point on we consider only problems with homogeneous boundary conditions, $h \equiv 0$. For example, for $F[u] = -u_{xx}$, $f \equiv 0$ and $\varphi = \varphi_{min}$, Equation 3.2 is discretized with Equation 2.12 as

$$u_j^{k+1} = u_j^k - dt \min\left(-\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2}, u_j^k - g_j\right). \tag{3.3}$$

During the implementation, when the exact solution of the particular problem is not known, the stopping rule is chosen to be $\|u^{k+1} - u^k\|_\infty < \varepsilon$ for some $\varepsilon > 0$. Once it is satisfied, the sequence of iterations $\{u^k\}$ is considered to have converged.

We now build the framework which will enable us to argue the convergence of the Euler method for the obstacle problem. The iterative update in Algorithm 1 is $u^{k+1} = u^k - dtG[u^k]$. We define the explicit discretization of the PDE $u_t + G[u] = 0$ as the Euler map [13].

Definition 3.1 (Euler map)**.** *For $dt > 0$, define $S_{dt} : \mathbb{R}^N \to \mathbb{R}^N$ by*

$$S_{dt}(u) = u - dtG[u]. \tag{3.4}$$

In order to comment on the convergence of this method, we need to define several concepts. The following arguments follow the method outlined in [13]. Let ∇u and $\nabla^2 u$ denote the gradient and Hessian of u, respectively. Then let $G(x, r, p, X)$ be a continuous function defined on $\Omega \times \mathbb{R} \times \mathbb{R}^d \times \mathbb{S}^d$, where $\mathbb{S}^d$ is the space of all symmetric $d \times d$ matrices.

Definition 3.2 (Degenerate elliptic)**.** *The PDE $G[u](x) \equiv G(x, u(x), \nabla u(x), \nabla^2 u(x)) = 0$ is called* degenerate elliptic *if*

$$G(x, r, p, X) \leq G(x, s, p, Y),$$

whenever $r \leq s$ and $X \geq Y$, where $x \in \bar{\Omega}$, $r, s \in \mathbb{R}$, $p \in \mathbb{R}^d$, $X, Y \in \mathbb{S}^d$. $X \geq Y$ means that $X - Y$ is a symmetric positive semidefinite matrix.

We now define notation for the finite difference scheme for G. Consider points on the grid, $x_j \in \Omega^h, j = 1, \ldots, N$. Each point has a list of neighboring points, $N(j)$. Define the grid function $u_j = u(x_j)$. We write the finite difference scheme for G as

$$G^h[u] \equiv G^j(u_j, u_j - u_{j'}|_{j'=N(j)}).$$

Definition 3.3 (Degenerate elliptic scheme)**.** *The scheme G^h is* degenerate elliptic *if each component G^j is nondecreasing in each variable.*

That is, the scheme must be a nondecreasing function of u_j and of the difference $u_j - u_{j'}$, for all neighbors $j' \in N(j)$. We next define global Lipschitz continuity and the nonlinear Courant-Friedrichs-Lewy (CFL) condition.

Definition 3.4 (Lipschitz continuity)**.** *The finite difference scheme G^j is Lipschitz continuous if there is a constant K such that for all $j = 1, \ldots, N$, $x, y \in \mathbb{R}^{|N(j)|+1}$,*

$$|G^j(x) - G^j(y)| \leq K\|x - y\|_\infty,$$

where $\|z\|_\infty := \max_j z_j$ is the maximum norm.

Definition 3.5 (Nonlinear CFL condition)**.** *Let G^h be a Lipschitz continuous, degenerate elliptic scheme, with Lipschitz constant K. The nonlinear CFL condition for the Euler map S_{dt} is*

$$dt \leq \frac{1}{K}.$$

The traditional CFL condition is a necessary condition for the convergence of PDEs solved using finite differences [4]. It gives a restriction on the time step dt. The largest possible time step is desired. To prove the convergence of the iterates of the Euler map, the scheme must also be proper.

Definition 3.6 (Proper schemes)**.** *The finite difference scheme G^j is* proper *if there exists $\delta > 0$ such that for $j = 1, \ldots, N$ and for all $y \in \mathbb{R}^{N(j)}$ and $x_1, x_2 \in \mathbb{R}$,*

$$x_1 \leq x_2 \text{ implies that } G^j(x_1, y) - G^j(x_2, y) \leq \delta(x_1 - x_2).$$

If a scheme is not proper, we can instead consider the PDE $G[u] + \varepsilon u = 0$. Choosing ε to be smaller than the discretization error allows us to assume that the scheme is proper. This has no affect on the accuracy [13].

Armed with these definitions, we now state the following two theorems from [13], which will establish convergence of the Euler step method.

Theorem 3.7 (The Euler map is a contraction)**.** *Let G^h be a Lipschitz continuous, degenerate elliptic scheme. Then the Euler map (3.4) is a contraction in $\mathbb{R}^N$ equipped with the maximum norm, provided the nonlinear CFL condition, $dt \leq K^{-1}$ holds. If in addition, G^h is proper, and strict inequality holds in the CFL condition, $dt < K^{-1}$, then the Euler map is a strict contraction.*

Theorem 3.8 (Convergence)**.** *A proper, Lipschitz continuous, degenerate elliptic scheme has a unique solution. The iterates of the Euler map converge to the solution for arbitrary initial data, provided strict inequality holds in the nonlinear CFL condition, $dt < K^{-1}$.*

The proof of this theorem relies on Theorem 3.7 and Banach's fixed point theorem. In practice, convergence is obtained even when the CFL condition inequality is not strict.

We now use these definitions to examine the classical obstacle problem, with the minimum function as the NCP function, $G[u] = \min(-\Delta u - f, u - g)$. We wish to apply Theorem 3.8 to prove the convergence of the Euler method.

Proposition 3.9. *$G[u] = \min(-\Delta u - f, u - g) = 0$ is a degenerate elliptic PDE.*

Proof. $G[u] = \min(-\Delta u - f, u - g)$ can be rewritten as $G(x, r, p, X) = \min(-\operatorname{Tr}(X) - f(x), r - g(x))$. Let $r, s \in \mathbb{R}$ and $X, Y \in \mathbb{S}^d$ be such that $r \leq s$ and $X \geq Y$. This easily gives $r - g(x) \leq s - g(x)$. Since $X - Y \geq 0$, the eigenvalues of $X - Y$ are nonnegative. Since the trace of a matrix is the sum of its eigenvalues, $\operatorname{Tr}(X - Y) \geq 0$. Moreover, $\operatorname{Tr}(X) \geq \operatorname{Tr}(Y)$ since the trace is a linear mapping, which gives us the inequality $-\operatorname{Tr}(X) - f(x) \leq -\operatorname{Tr}(Y) - f(x)$. Then $\min(-\operatorname{Tr}(X) - f(x), r - g(x)) \leq \min(-\operatorname{Tr}(Y) - f(x), s - g(x))$ which is exactly $G(x, r, p, X) \leq G(x, s, p, Y)$. □

We now concentrate on the one-dimensional obstacle problem:

$$G[u] = \min(-u_{xx} - f, u - g) = 0.$$

The finite difference scheme is

$$G^h[u] = \min\left(-\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} - f_j, u_j - g_j\right),$$

which can be rewritten as

$$G^j(x, y_1, y_2) = \min\left(\frac{y_1 + y_2}{h^2} - f_j, x - g_j\right),$$

where $y_1 = u_j - u_{j+1}$, $y_2 = u_j - u_{j-1}$ and $x = u_j$. To prove that G^j is nondecreasing in each variable we can use the fact that if $a_1 \leq a_2$ while b is fixed, $\min(a_1, b) \leq \min(a_2, b)$. As a result, G^j is a degenerate elliptic scheme. The next thing to show is that G^h is Lipschitz continuous. We first need the following Proposition.

Proposition 3.10. *Let G_1^j and G_2^j be Lipschitz continuous finite difference schemes with Lipschitz constants K_1 and K_2, respectively. Then $G := \min(G_1^j, G_2^j)$ is Lipschitz continuous with Lipschitz constant $K := \max(K_1, K_2)$.*

Proof. Let $x, y \in \mathbb{R}^{|N(j)|+1}$. Then, $|G^j(x) - G^j(y)| = |\min(G_1^j(x), G_2^j(x)) - \min(G_1^j(y), G_2^j(y))|$. There are four cases to consider:

Case 1:

$$\begin{aligned} |G^j(x) - G^j(y)| &= |G_1^j(x) - G_1^j(y)| \\ &\leq K_1 \|x - y\|_\infty \end{aligned}$$

Case 2:

$$\begin{aligned} |G^j(x) - G^j(y)| &= |G_2^j(x) - G_2^j(y)| \\ &\leq K_2 \|x - y\|_\infty \end{aligned}$$

Case 3:

$$\begin{aligned} |G^j(x) - G^j(y)| &= |G_1^j(x) - G_2^j(y)| \\ &= G_1^j(x) - G_2^j(y) \\ &\leq G_2^j(x) - G_2^j(y) \\ &\leq |G_2^j(x) - G_2^j(y)| \\ &\leq K_2 \|x - y\|_\infty \end{aligned}$$

The second equality only holds if $G_1^j(x) \geq G_2^j(y)$. If $G_1^j(x) \leq G_2^j(y)$, the chain of inequalities can be modified to lead to $|G^j(x) - G^j(y)| \leq K_1 \|x - y\|_\infty$.

Case 4:

$$|G^j(x) - G^j(y)| = |G_2^j(x) - G_1^j(y)|$$

This case is symmetric to Case 3.

Therefore, for any x, y, $|G^j(x) - G^j(y)| \leq K \|x - y\|_\infty$, where $K := \max(K_1, K_2)$. □

We now return to proving that the finite difference scheme for the one-dimensional classical obstacle problem is Lipschitz continuous. Let $x, y_1, y_2, \hat{x}, \hat{y}_1, \hat{y}_2 \in \mathbb{R}$. Then we consider

two cases. The first case is

$$\begin{aligned}|G^j(x,y_1,y_2) - G^j(\hat{x},\hat{y}_1,\hat{y}_2)| &= \left|\frac{y_1+y_2}{h^2} - f_j - \left(\frac{\hat{y}_1+\hat{y}_2}{h^2} - f_j\right)\right| \\ &= \frac{1}{h^2}\left|y_1 - \hat{y}_1 + y_2 - \hat{y}_2\right| \\ &\leq \frac{1}{h^2}|y_1 - \hat{y}_1| + \frac{1}{h^2}|y_2 - \hat{y}_2| \\ &\leq \frac{2}{h^2}\|(x,y_1,y_2) - (\hat{x},\hat{y}_1,\hat{y}_2)\|_\infty,\end{aligned}$$

where the first inequality is due to the triangle inequality. The Lipschitz constant in this first case is $K_1 := \frac{2}{h^2}$. The second case is

$$\begin{aligned}|G^j(x,y_1,y_2) - G^j(\hat{x},\hat{y}_1,\hat{y}_2)| &= |x - g_j - (\hat{x} - g_j)| \\ &= |x - \hat{x}| \\ &\leq \|(x,y_1,y_2) - (\hat{x},\hat{y}_1,\hat{y}_2)\|_\infty.\end{aligned}$$

The Lipschitz constant in this case is $K_2 := 1$. Using Proposition 3.10, the Lipschitz constant for G^h is $K := \max(K_1, K_2) = K_1$, since we can safely assume that $h^2 \leq 2$. The nonlinear CFL condition for the one-dimensional case is therefore $dt \leq \frac{h^2}{2}$.

We must now prove that the scheme is proper. Let $x_1, x_2 \in \mathbb{R}$ with $x_1 \leq x_2$ and $y := (y_1, y_2) \in \mathbb{R}^{N(i)}$. Then in one case,

$$\begin{aligned}G^j(x_1,y) - G^j(x_2,y) &= \min\left(\frac{y_1+y_2}{h^2} - f_j, x_1 - g_j\right) - \min\left(\frac{y_1+y_2}{h^2} - f_j, x_2 - g_j\right) \\ &= \frac{y_1+y_2}{h^2} - f_j - \left(\frac{y_1+y_2}{h^2} - f_j\right) \\ &= 0.\end{aligned}$$

If $x_1 \neq x_2$, one cannot find a $\delta > 0$ satisfying the proper conditions. However, we can instead

consider the PDE $G[u] + \varepsilon u = 0$, for some $\varepsilon > 0$. The scheme for this PDE is proper. This new PDE is only needed to apply Theorem 3.8. The implementation of the scheme for the new PDE with small enough ε ends up being the same as the implementation for $G[u] = 0$.

In the two-dimensional case, $G[u] = \min(-\Delta u - f, u - g) = 0$, the finite difference scheme is

$$G^h[u] = \min\left(-\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2} - f_{i,j}, u_{i,j} - g_{i,j}\right),$$

which can be rewritten as

$$G^{i,j}(x, y_1, y_2, y_3, y_4) = \min\left(\frac{y_1 + y_2 + y_3 + y_4}{h^2} - f_{i,j}, x - g_{i,j}\right),$$

where $y_1 = u_{i,j} - u_{i-1,j}$, $y_2 = u_{i,j} - u_{i+1,j}$, $y_3 = u_{i,j} - u_{i,j-1}$, $y_4 = u_{i,j} - u_{i,j+1}$ and $x = u_{i,j}$. The same procedure can be followed as for the one-dimensional case. $G^{i,j}$ is a degenerate elliptic scheme since it is nondecreasing in each variable. The scheme $G^{i,j}$ is Lipschitz continuous with Lipschitz constant $K = \frac{4}{h^2}$. $G^{i,j}$ is again not proper, but the same argument can be applied as in the one-dimensional case.

Theorems 3.7 and 3.8 can now be applied to show that the iterates of the Euler map converge to the solution for arbitrary data, provided strict inequality holds in the nonlinear CFL condition. For the one-dimensional case, this means $dt < \frac{h^2}{2}$ and for the two-dimensional case, $dt < \frac{h^2}{4}$. The time step dt is restricted by the grid size of the discretization. In the implementation however, we chose $dt = \frac{h^2}{2}$ and $dt = \frac{h^2}{4}$ which proved to be enough in practice.

The Euler map (3.4) becomes $S_{K^{-1}}(u) = u - K^{-1}G[u]$, where K is the Lipschitz constant for G^h. The iterative update in the one-dimensional case can be written as

$$u_j^{k+1} = u_j^k - \frac{h^2}{2}\min\left(-\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} - f_j, u_j^k - g_j\right). \tag{3.5}$$

In the two-dimensional case, the iterative update is

$$u_{i,j}^{k+1} = u_{i,j}^k - \frac{h^2}{4} \min\left(-\frac{u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k - 4u_{i,j}^k}{h^2} - f_{i,j}, u_{i,j}^k - g_{i,j} \right). \quad (3.6)$$

We now introduce a concept that is useful for a better visualization of the Euler method.

Proposition 3.11. $\min(a,b) = 0 \Leftrightarrow \min(a, \frac{b}{\varepsilon}) = 0$, *for all* $\varepsilon > 0$.

Proof. $\Rightarrow$ Suppose $\min(a,b) = a = 0$. Then $b \geq a = 0$. But we still have $\frac{b}{\varepsilon} \geq 0$. And so $\min(a, \frac{b}{\varepsilon}) = a = 0$, i.e. the first component is still the smaller component. On the other hand, if $\min(a,b) = b = 0$, then $a \geq b = 0$. And we still have $a \geq \frac{b}{\varepsilon} = 0$. Then $\min(a, \frac{b}{\varepsilon}) = \frac{b}{\varepsilon} = 0$, i.e. the second component is still the smaller one.

$\Leftarrow$ Suppose now that $\min(a, \frac{b}{\varepsilon}) = a = 0$. Then $\frac{b}{\varepsilon} \geq a = 0$ and therefore $b \geq 0 = a$ and so $\min(a,b) = a = 0$. On the other hand, if $\min(a, \frac{b}{\varepsilon}) = \frac{b}{\varepsilon} = 0$, then $a \geq \frac{b}{\varepsilon} = 0$. We also have that $b = 0$. Then $\min(a,b) = b = 0$. □

The idea is that the same option in the minimum will be chosen in both cases. This concept is now applied to the one-dimensional PDE $\min(-u_{xx} - f, u - g) = 0$. We consider instead the equation $\min\left(-u_{xx} - f, \frac{u-g}{h^2/2}\right) = 0$. Equation 3.5 is modified to

$$u_j^{k+1} = u_j^k - \frac{h^2}{2} \min\left(-\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} - f_j, \frac{u_j^k - g_j}{h^2/2} \right), \quad (3.7)$$

which further simplifies to

$$\begin{aligned} u_j^{k+1} &= u_j^k - \min\left(-\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{2} - \frac{h^2}{2} f_j, u_j^k - g_j \right) \\ &= u_j^k + \max\left(\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{2} + \frac{h^2}{2} f_j, g_j - u_j^k \right) \\ &= \max\left(\frac{u_{j+1}^k + u_{j-1}^k}{2} + \frac{h^2}{2} f_j, g_j \right). \end{aligned} \quad (3.8)$$

Consider the case when $f \equiv 0$. Intuitively, the iterative update in Equation 3.8 says that

at each grid point, the function either jumps to the obstacle, or is smoothed by taking the average of its neighbors from the previous time step. Figure 3.1 is an illustration of this phenomenon at the first Euler step with the initial guess shown in black, the obstacle in red, and the solution in blue.

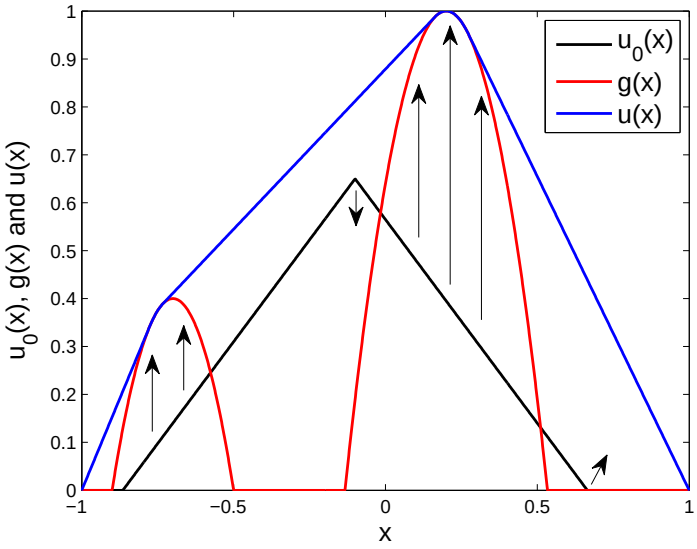


Figure 3.1: The first step of the Euler solver

The Euler iterations given by Equation 3.7 were implemented with

$$u^{k+1} = u^k - \frac{h^2}{2} \min\left(-D_{xx}u^k - f_j, \frac{u^k - g}{h^2/2}\right),$$

where $u^k \in \mathbb{R}^{N-2}$. Recall that N is the number of points in the discretized domain. The homogeneous Dirichlet boundary conditions force two values of u, u_0^k and u_N^k, to be zero for all k. D_{xx} is tridiagonal matrix for the one-dimensional Laplace operator for the problem

with homogeneous Dirichlet boundary conditions:

$$D_{xx} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 \end{bmatrix}.$$

The entries that are not shown are zeros. D_{xx} is an $(N-2)\times(N-2)$ matrix since each iterate u^k is fixed to be zero at the boundary.

The same averaging idea can be applied in the two-dimensional case, but with $u-g$ divided by $\frac{h^2}{4}$. Equation 3.6 becomes:

$$\begin{aligned} u_{i,j}^{k+1} &= u_{i,j}^k - \frac{h^2}{4}\min\left(-\frac{u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k - 4u_{i,j}^k}{h^2} - f_{i,j}, \frac{u_{i,j}^k - g_{i,j}}{h^2/4}\right) \\ &= \max\left(\frac{u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k}{4} + \frac{h^2}{4}f_{i,j}, g_{i,j}\right). \end{aligned} \tag{3.9}$$

The iterations given by Equation 3.9 were implemented with the line of code

$$u_e^{k+1} = u_e^k - \frac{h^2}{4}\min\left(-D_{xx}^{2d}u_e^k - f_e, \frac{u_e^k - g_e}{h^2/4}\right),$$

where the subscript e represents the expanded version of the grid functions, with the indices representing the homogeneous Dirichlet boundary conditions removed. The two-dimensional grid functions u^k, g and f are expanded into one-dimensional vectors. For example, matrix $u_{i,j}$ representing the values of u at position i,j in the grid is transformed into a vector of length $N=n^2$ by mapping index i,j of the matrix to index $(i-1)n+j$ of the vector, where n is the number of points in one dimension. The entries corresponding to the boundary values

of the grid are then removed from the vector since we are considering only homogeneous Dirichlet boundary conditions. D_{xx}^{2d} is the two-dimensional Laplacian and is given by

$$D_{xx}^{2d} = \frac{1}{h^2}\begin{bmatrix}
-4 & 1 & & 1 & & & & & & \\
1 & -4 & 1 & & 1 & & & & & \\
 & 1 & -4 & 1 & & 1 & & & & \\
\ddots & & \ddots & \ddots & \ddots & & \ddots & & & \\
 & 1 & & 1 & -4 & 1 & & 1 & & \\
 & & 1 & & 1 & -4 & 1 & & 1 & \\
 & & & \ddots & & \ddots & \ddots & \ddots & & \ddots \\
 & & & & 1 & & 1 & -4 & 1 & \\
 & & & & & 1 & & 1 & -4 & 1 \\
 & & & & & & 1 & & 1 & -4
\end{bmatrix}.$$

D_{xx}^{2d} is an $(n-2)^2 \times (n-2)^2$ matrix. The blank entries are all zero, and the number of zeros between the 1 entries in each row is $n-4$.

Chapter 4

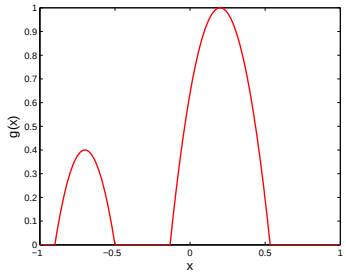
Forward Euler Implementation

4.1 1d Numerical Results

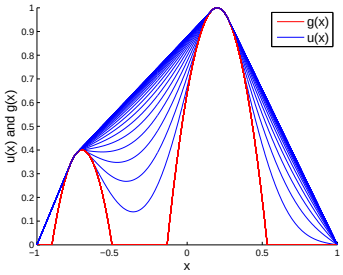
The Euler method was implemented in MATLAB in one dimension for the classical obstacle problem with $\Omega = (-1, 1)$, $u, g = 0$ on $\partial\Omega$ and $f \equiv 0$ or $f \equiv 5$. The obstacle is shown in Figure 4.1a, and was also used as the initial guess, u^0. This choice of u^0 can be justified in part by the Euler method itself. If u^0 is chosen to be a function other than g as in Figure 3.1, the first Euler step will make the second iterate shoot up to the obstacle. Therefore beginning with $u^0 = g$ could save an iteration. This is of course highly dependent on the obstacle itself.

Figure 4.1b shows the iterates of the Euler solver at every 10000 iterations for $N = 1000$ grid points with $f \equiv 0$. The solution is attained after 301892 iterations. Each iterate is closer to the solution than the previous one. The number of iterations needed for convergence for several values of N is shown in Table 4.1. The number of iterations is highly dependent on the grid size. It appears to grow quadratically with the number of grid points. In fact, as was discussed before, at each iteration, every point on the grid either jumps to the obstacle or is the average between its neighbors. That is, every point looks only at its

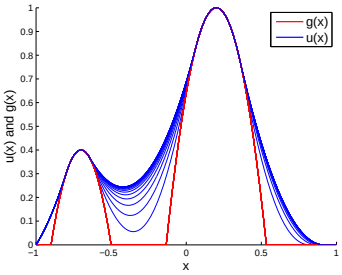
immediate surroundings. The higher the number of points on the grid, the longer it will take for this process to converge. The stopping condition used in all numerical implementations was $\|u^{k+1} - u^k\|_\infty < 10^{-8}$.



(a) Obstacle g and initial u



(b) Euler solver each 10000 iterations for $N = 1000$ with $f \equiv 0$



(c) Euler solver each 10000 iterations for $N = 1000$ with $f \equiv 5$

Figure 4.1: 1d Euler solver

Table 4.1: Number of Euler iterations for the 1d obstacle problem

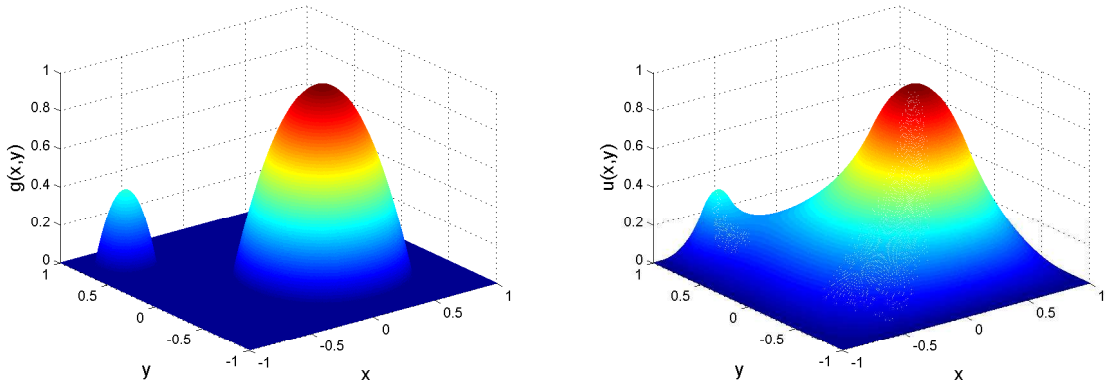
N	50	100	500	1000
Euler iterations, $f \equiv 0$	1234	4996	88209	301892
Euler iterations, $f \equiv 5$	794	3034	54888	187300

The number of iterations is smaller for the case with the nonzero force term. In this particular case, this can be explained by noting that the solution is closer to the obstacle, i.e. the starting iterate, than in the case when the force term is zero. However, the number of iterations still grows quadratically with N. This suggests that the speed of the method is independent of the forcing function, f.

4.2 2d Numerical Results

The Euler method was implemented in two dimensions for the classical obstacle problem with $\Omega = (-1, 1)^2$ and $u, g = 0$ on $\partial\Omega$ as well as $f \equiv 0$. The obstacle is shown in Figure

4.2a, and was also used as the initial iterate, u^0. Figure 4.2b shows the result of the Euler method with $N = 500^2$. The number of iterations needed for convergence for several values of N is shown in Table 4.2. Similarly to the one-dimensional case, the number of iterations grows rapidly with N.



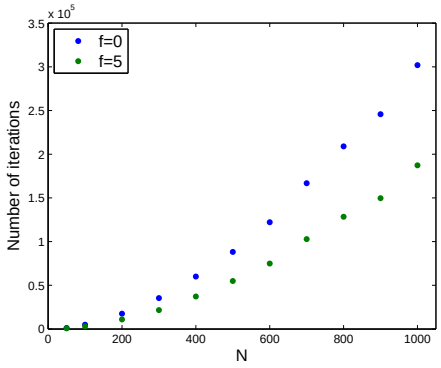
(a) Obstacle g and initial u *(b) Euler method for $N = 500^2$*

Figure 4.2: 2d Euler solver

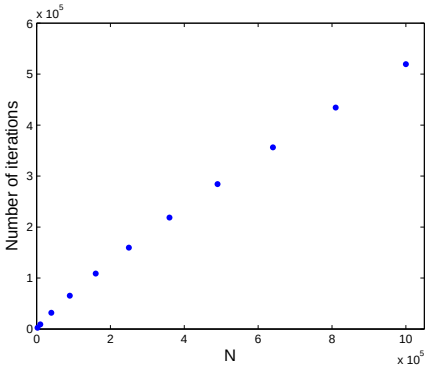
Table 4.2: Number of Euler iterations for the 2d obstacle problem

N	50^2	100^2	500^2	1000^2
Euler iterations	2520	9076	159679	519456

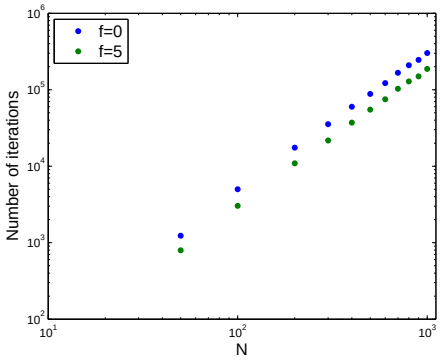
Figures 4.3a and 4.3b show the number of iterations as a function of N for the one-dimensional and the two-dimensional examples. The number of iterations in both cases grows rapidly with N. The same data was plotted with logarithmically scaled axes and is shown in Figures 4.3c and 4.3d. The slopes in the one-dimensional case with $f \equiv 0$ and $f \equiv 5$ were found to be approximately 1.8195 and 1.8112, respectively. That is, the method is near $\mathcal{O}(N^2)$, regardless of f. The slope in the two-dimensional case was found to be approximately 0.8873, which implies an $\mathcal{O}(N)$ method. Such a dependence on the grid size is not desirable. Refining a grid should ideally not cost more in terms of the number of iterations.



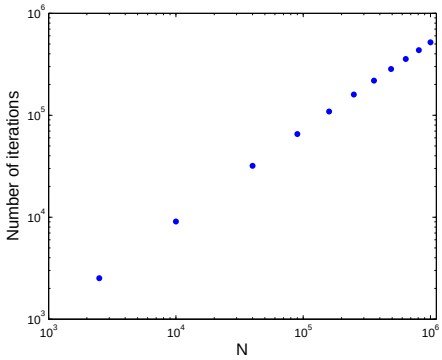
(a) Number of Euler iterations for the 1d obstacle problem



(b) Number of Euler iterations for the 2d obstacle problem



(c) Number of Euler iterations for the 1d obstacle problem, log-log plot.



(d) Number of Euler iterations for the 2d obstacle problem, log-log plot.

Figure 4.3: Number of iterations for the Euler solver

Chapter 5

Semismooth Newton's Method

We are interested in solving the equation $G[u] = 0$. When solved numerically, $u \in \mathbb{R}^N$ is a grid function. A classical root finding method is Newton's method. However, due to G being nonsmooth, Newton's method cannot applied directly. A generalized version of it, the semismooth Newton's method (SSNM), will be used. Before introducing this generalized version, we recall Newton's method and several of its properties.

5.1 Newton's Method

Newton's method is used for finding a root of a system of equations: given $G : \mathbb{R}^N \to \mathbb{R}^N$, find $x_* \in \mathbb{R}^N$ such that $G(x_*) = 0$. The Newton's method algorithm for finding a root is given in Algorithm 2.

Algorithm 2 Newton's method for smooth systems

Let $G : \mathbb{R}^N \to \mathbb{R}^N$ be a given continuously differentiable function. Select $x_0 \in \mathbb{R}^N$ and set $k := 0$.

1: Unless a stopping rule is satisfied, solve (for d_k):

$$\nabla G(x_k) d_k = -G(x_k). \tag{5.1}$$

2: Set $x_{k+1} := x_k + d_k$, $k := k + 1$, and go to Step 1.

In Algorithm 2, $\nabla G : \mathbb{R}^N \to \mathbb{R}^{N\times N}$ denotes the Jacobian matrix of G. Clearly, to solve for d_k in Equation 5.1, $\nabla G(x_k)$ must be nonsingular for all k. Newton's method can be motivated by a linearization about a given estimate $x_k \in \mathbb{R}^N$, $G(x_k + d_k) \approx G(x_k) + \nabla G(x_k)d_k$, for some $d_k \in \mathbb{R}^N$. Setting the left hand side to zero and solving for d_k gives the search direction in Algorithm 2. The one-dimensional interpretation is as follows: the next iterate x_{k+1} is the point of intersection of the line of slope $g'(x_k)$, passing through x_k with the horizontal axis. We now define several convergence rate concepts.

Definition 5.1 (Quotient factor)**.** *Let $\{x_k\} \subset \mathbb{R}^n$ denote a sequence with limit $x_* \in \mathbb{R}^n$, and let $p \in [1, +\infty)$. Then for some $k_0 \in \mathbb{N}$,*

$$Q_p\{x_k\} := \begin{cases} \limsup\limits_{k\to\infty} \dfrac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|^p}, & \text{if } x_k \neq x_* \text{ for all } k \geq k_0, \\ 0, & \text{if } x_k = x_* \text{ for all } k \geq k_0, \\ +\infty, & \text{otherwise,} \end{cases}$$

is called the quotient factor *of $\{x_k\}$.*

Three special quotient factor cases will be of interest to us. The case when $p = 1$ and and $Q_1\{x_k\} = 0$ is called superlinear convergence. The case when $p = 1$ and $0 < Q_1\{x_k\} < 1$ is called linear convergence. Finally, the case when $p = 2$ and $0 < Q_2\{x_k\} < +\infty$ is called quadratic convergence. One can prove that Algorithm 2 converges locally at a quadratic rate [9]. Global convergence of Newton's method can be obtained with additional assumptions.

5.2 Nonsmooth Analysis

Often, a function whose root is to be found is not continuously differentiable, and hence Algorithm 2 cannot be used. In particular the NCP functions used for the obstacle problem here are not continuously differentiable. A different method, using a replacement for the

Jacobian, must be implemented. It is worth noting that there are continuously differentiable NCP functions. However an important disadvantage of such functions is that they often lead to singular Jacobians. For this reason we focus on φ_{min} and φ_{FB}.

The nonsmooth version of Newton's method that we will use instead requires results from nonsmooth analysis that will be introduced here.

5.2.1 Generalized Jacobian

Several preliminary definitions are necessary before defining the generalized Jacobian. The statements in this subsection are due to Clarke [3].

Definition 5.2 (Local Lipschitz continuity)**.** *Let $G : \mathbb{R}^n \to \mathbb{R}^m$. G is* locally Lipschitz continuous *if for all $x \in \mathbb{R}^n$, there exist ε and L depending on x such that*

$$\|G(x_1) - G(x_2)\| \leq L\|x_1 - x_2\| \quad \forall\, x_1, x_2 \in B_\varepsilon(x).$$

Theorem 5.3 (Rademacher)**.** *Let $G : \mathbb{R}^n \to \mathbb{R}^m$ be a locally Lipschitz continuous function. Then G is almost everywhere differentiable.*

Let $D_G := \{x \in \mathbb{R}^n \,|\, G \text{ is differentiable at } x\}$, i.e. D_G is the subset of points in $\mathbb{R}^n$ at which G is differentiable. By Rademacher's theorem, the set $\mathbb{R}^n \setminus D_G$ is a set of measure zero in the Lebesgue sense. Therefore for each $x \in \mathbb{R}^n$, there exist arbitrarily many sequences $\{x_k\} \subseteq D_G$ converging to x ($x_k \to x$). We are now ready to define the generalized Jacobian.

Definition 5.4 (B-subdifferential, generalized Jacobian)**.** *Let $G : \mathbb{R}^n \to \mathbb{R}^m$ be a locally Lipschitz continuous function and $x \in \mathbb{R}^n$.*

a) The set

$$\partial_B G(x) := \{H \in \mathbb{R}^{m\times n} \,|\, \exists \{x_k\} \subseteq D_G \text{ with } x_k \to x,\, \nabla G(x_k) \to H\}$$

is called the B-subdifferential of G at x.

b) The generalized Jacobian at x *is defined by*

$$\partial G(x) := \text{conv}(\partial_B G(x)),$$

where conv *denotes the convex hull.*

Note that when $m = 1$, $\partial G(x)$ is called the *generalized gradient at* x. There are many properties related to the B-subdifferential and the generalized Jacobian which are analogous to properties of Jacobians of differentiable functions [3].

5.2.2 Semismooth Functions

Since a generalized Jacobian has been defined, it would seem possible to replace the Jacobian in Algorithm 2 with an element of the generalized Jacobian, i.e. Equation 5.1 becomes $H(x_k)d_k = -G(x_k)$, where $H(x_k) \in \partial G(x_k)$ is arbitrary. However, it is not quite this simple. If G is only required to be locally Lipschitz, at best one can expect linear convergence of the new algorithm, if convergence at all [9]. Further properties are required of G to ensure convergence of the modified Newton method as well as fast convergence. The class of functions considered must be restricted to semismooth functions. Semismoothness was first introduced for functionals in [12]. It was extended to functions $G : \mathbb{R}^n \to \mathbb{R}^n$ in [15].

Definition 5.5 (Semismooth)**.** *The function* $G : \mathbb{R}^n \to \mathbb{R}^m$ *is* semismooth at $x \in \mathbb{R}^n$*, if it is locally Lipschitz at* x *and if*

$$\lim_{\substack{H \in \partial G(x+td') \\ d' \to d, t \downarrow 0}} Hd' \tag{5.2}$$

exists for all $d \in \mathbb{R}^n$.

Let the directional derivative of a function $G : \mathbb{R}^n \to \mathbb{R}^m$ in direction d be given by

$$G'(x; d) := \lim_{t \downarrow 0} \frac{G(x + td) - G(x)}{t}.$$

An equivalent characterization of semismoothness is given next.

Proposition 5.6 (Characterization of semismooth functions)**.** *The function $G : \mathbb{R}^n \to \mathbb{R}^m$ is* semismooth at $x \in \mathbb{R}^n$ *if and only if it is locally Lipschitz at x and*

$$\lim_{\substack{d \to 0 \\ H \in \partial G(x+d)}} \frac{\|Hd - G'(x; d)\|}{\|d\|} = 0. \tag{5.3}$$

If G is semismooth at all $x \in \mathbb{R}^n$, we call G *semismooth* [15]. In addition, for $G : \mathbb{R}^n \to \mathbb{R}^m$ locally Lispchitz, if each component of G is semismooth at x, then G is semismooth at x [15]. Semismoothness is equivalent to the uniform convergence of directional derivatives in all directions [15]. Examples of semismooth functions are convex functions, smooth functions and piecewise-smooth functions [12]. Sums, scalar products and compositions of semismooth functions are also semismooth functions [15, 9]. There is a stronger characterization of semismoothness:

Definition 5.7 (Strongly semismooth)**.** *The function $G : \mathbb{R}^n \to \mathbb{R}^m$ is* strongly semismooth at $x \in \mathbb{R}^n$ *if it is locally Lipschitz at x and if*

$$\lim_{\substack{d \to 0 \\ H \in \partial G(x+d)}} \frac{\|Hd - G'(x; d)\|}{\|d\|^2} < +\infty. \tag{5.4}$$

If G is strongly semismooth at all $x \in \mathbb{R}^n$, we call G *strongly semismooth.* We now prove that the two NCP functions introduced in Propositions 2.3 and 2.4 are strongly semismooth.

Proposition 5.8. *φ_{min} is strongly semismooth.*

Proof. Recall that a continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ is said to be *piecewise affine* if there exists a finite family of affine functions $\{f^1, \ldots, f^k\}$ for some positive integer k, where

$f^i : \mathbb{R}^n \to \mathbb{R}^m$, such that for all $x \in \mathbb{R}^n$, $f(x) \in \{f^1(x), \ldots, f^k(x)\}$ [7]. Clearly the minimum function is a piecewise affine function. Proposition 7.4.7 in [7] states that every piecewise affine function is strongly semismooth. Therefore, φ_{min} is strongly semismooth.

□

Proposition 5.9. *φ_{FB} is strongly semismooth.*

Proof. When $(a, b) \neq (0, 0)$, $\varphi_{FB}(a, b)$ is continuously differentiable and therefore strongly semismooth. We need only consider the case when $(a, b) = (0, 0)$. We first compute the directional derivative in direction $d := (d_a, d_b) \neq (0, 0)$.

$$\begin{aligned}
\varphi'_{FB}((0,0);d) &= \lim_{t \downarrow 0} \frac{\varphi_{FB}((0,0) + t(d_a, d_b)) - \varphi_{FB}(0,0)}{t} \\
&= \lim_{t \downarrow 0} \frac{\varphi_{FB}(t(d_a, d_b))}{t} \\
&= \lim_{t \downarrow 0} \frac{t\varphi_{FB}(d_a, d_b)}{t} \\
&= \varphi_{FB}(d_a, d_b).
\end{aligned}$$

Then $\partial\varphi_{FB}((0,0) + (d_a, d_b)) = \left\{ \left(\frac{d_a}{\sqrt{d_a^2 + d_b^2}} - 1, \frac{d_b}{\sqrt{d_a^2 + d_b^2}} - 1 \right) \right\}$, and

$$\begin{aligned}
\lim_{\substack{d \to 0 \\ H \in \partial\varphi_{FB}((a,b)+d)}} \frac{Hd - \varphi'_{FB}((0,0);d)}{\|d\|^2} &= \frac{\left(\frac{d_a}{\sqrt{d_a^2 + d_b^2}} - 1, \frac{d_b}{\sqrt{d_a^2 + d_b^2}} - 1 \right) \begin{pmatrix} d_a \\ d_b \end{pmatrix} - \varphi_{FB}(d_a, d_b)}{\|d\|^2} \\
&= \frac{\frac{d_a^2 + d_b^2}{\sqrt{d_a^2 + d_b^2}} - d_a - d_b - \varphi_{FB}(d_a, d_b)}{\|d\|^2} \\
&= \frac{\varphi_{FB}(d_a, d_b)}{\|d\|^2} \\
&= 0 < +\infty.
\end{aligned}$$

The function φ_{FB} is indeed strongly semismooth. □

5.3 Semismooth Newton's Method

Now that we have defined semismooth functions, we can introduce the modified version of Newton's method, the semismooth Newton's method.

Algorithm 3 Semismooth Newton's method

Let $G : \mathbb{R}^N \to \mathbb{R}^N$ be a given semismooth function. Select $x_0 \in \mathbb{R}^N$ and set $k := 0$.

1: Unless a stopping rule is satisfied, solve (for d_k):

$$H(x_k)d_k = -G(x_k), \tag{5.5}$$

where $H(x_k) \in \partial G(x_k)$.

2: Set $x_{k+1} := x_k + d_k$, $k := k + 1$, and go to Step 1.

Note that Algorithm 3 is identical to Algorithm 2 with the exception that the Jacobian is replaced by the generalized Jacobian. We now make several statements on the convergence of the SSNM.

Theorem 5.10 (Qi, Sun [15, 9])**.** *Suppose that x^* is a solution of $G(x) = 0$, G is locally Lipschitz and semismooth at x^*, and all $H \in \partial G(x^*)$ are nonsingular. Then the iteration method (5.5) is well-defined and convergent to x^* in a neighborhood of x^*.*

It can be shown that the semismooth Newton method converges locally at a superlinear rate [15, 9]. If G is strongly semismooth, then we even have local quadratic convergence. That is, the semismooth Newton's method has the same local convergence rate as Newton's method. Again, further assumptions lead to a global convergence result [15, 9].

Chapter 6

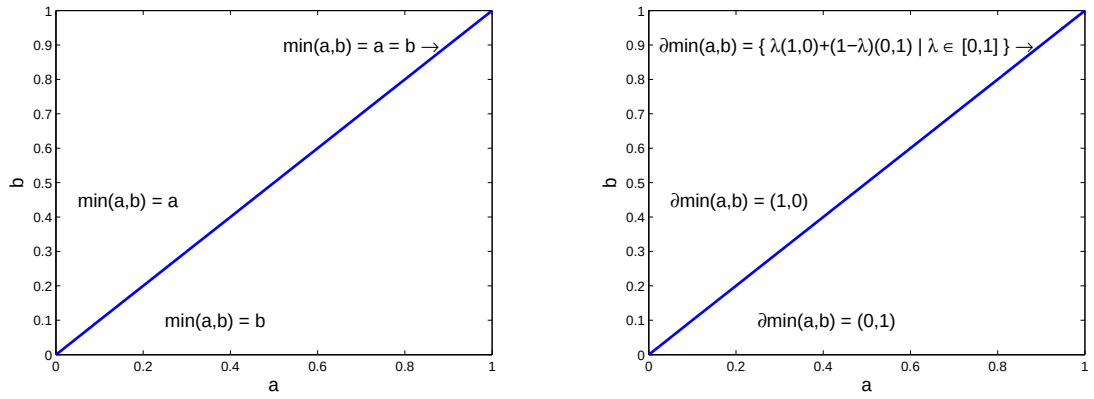
SSNM Implementation

We first consider the one-dimensional obstacle problem using one of the two strongly semismooth NCP functions we are considering, i.e. $G[u] = \varphi(-u_{xx} - f, u - g) = 0$. We assume that both $-u_{xx} - f$ and $u - g$ are continuously differentiable. Since φ is strongly semismooth, the fact that compositions of semismooth functions are semismooth implies that G is semismooth. To apply the SSNM, we need to compute the generalized gradient of G. We begin by considering the generalized gradient of $\varphi_{min}(a, b) := \min(a, b)$. φ_{min} is not differentiable on the line $a = b$. This is illustrated in Figure 6.1a. The generalized gradient of φ_{min} is given by

$$\partial\varphi_{min}(a,b) = \begin{cases} (1,0)\,, & \text{if } a < b, \\ (0,1)\,, & \text{if } a > b, \\ \lambda\,(1,0) + (1-\lambda)\,(0,1)\,, & \text{if } a = b, \end{cases}$$

with $\lambda \in [0, 1]$. When $a = b$, the generalized gradient is exactly the convex hull of the gradient in the other two cases. Figure 6.1b illustrates this generalized gradient.

We now consider the generalized gradient of the Fischer-Burmeister function, $\varphi_{FB}(a, b) := \sqrt{a^2 + b^2} - a - b$. φ_{FB} is not differentiable only when $(a, b) = (0, 0)$. The generalized gradient



(a) $\varphi_{min}(a,b)$ *(b)* $\partial\varphi_{min}(a,b)$

Figure 6.1: The minimum function and its generalized gradient

is given by

$$\partial\varphi_{FB}(a,b) = \begin{cases} \left(\frac{a}{\sqrt{a^2+b^2}} - 1, \frac{b}{\sqrt{a^2+b^2}} - 1\right), & \text{if } (a,b) \neq (0,0), \\ (\rho - 1, \eta - 1), & \text{if } (a,b) = (0,0), \end{cases}$$

where $(\rho, \eta) \in \mathbb{R}^2$ is a vector such that $\|(\rho,\eta)\| \leq 1$ [8].

The problem is then discretized on the grid. Since the boundary conditions are homogeneous Dirichlet boundary conditions, we only need to solve for the solution on the interior of the domain. Therefore each iterate u^k is an element of $\mathbb{R}^{N-2}$. The second derivative u_{xx} is approximated by $(u^C_{xx})_j$ as before, leading to the finite difference operator D_{xx}. Then the discretized version of G becomes $G^h[u] := \varphi(a(u), b(u))$, where $a(u) := -D_{xx}u - \frac{h^2}{2}f$ and $b(u) := u - g$. Here u, f, and g are grid vectors of length $N-2$. The factor of $\frac{h^2}{2}$ in front of the f is necessary to take into account the grid spacing used when computing D_{xx}.

The inputs to φ, $a(u)$ and $b(u)$, are vectors in $\mathbb{R}^{N-2}$. The generalized gradient of $\varphi(a,b)$ is given by $\partial\varphi(a,b) := \left(\frac{\partial\varphi(a,b)}{\partial a}, \frac{\partial\varphi(a,b)}{\partial b}\right)$, where row r is a random element of the generalized gradient of $\varphi(a_r, b_r)$. To compute the generalized Jacobian of G^h, we first transform the columns of $\partial\varphi(a,b)$ into $N-2$ by $N-2$ diagonal matrices, which we call D_a and D_b. By

the chain rule, a random element of the generalized Jacobian of G^h is

$$\begin{aligned}\partial G^h[u] &= D_a \nabla_u (-D_{xx} u - \frac{h^2}{2} f) + D_b \nabla_u (u - g) \\ &= D_a(-D_{xx}) + D_b I.\end{aligned}$$

Then the semismooth Newton iteration is

$$u^{k+1} = u^k - (\partial G^h[u^k])^{-1} G^h[u^k].$$

The two-dimensional case again uses the extended vector version, where D_{xx} and $\frac{h^2}{2}$ are replaced by D_{xx}^{2d} and $\frac{h^2}{4}$, respectively.

6.1 1d Numerical Results

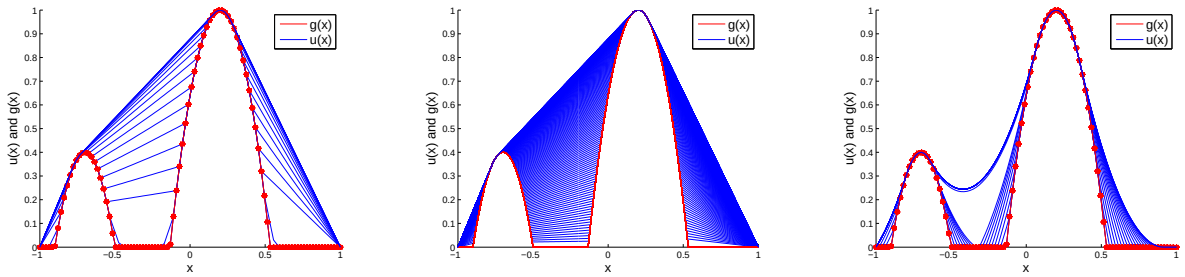
The SSNM solver was implemented in one dimension for the same example as the Euler solver. The number of iterations needed for convergence for several values of N is shown in Table 6.1.

Table 6.1: Number of SSNM iterations for the 1d obstacle problem

N	50	100	500	1000	5000	10000
SSNM iterations, $f \equiv 0$	7	16	74	149	741	1482
SSNM iterations, $f \equiv 5$	10	20	96	192	954	1909

The SSNM solver is much faster than the Euler solver in terms of the number of iterations. This is a significant improvement on the quadratic increase of iterations exhibited by the Euler solver. Figure 6.2 shows the SSNM solver every iteration for $N = 100$ and $N = 1000$ for $f \equiv 0$ and every iteration for $N = 100$ and $f \equiv 5$. Each iteration appears to

move the free boundary by only one grid point. Just like the Euler method, the example with the nonzero force requires more iterations for convergence than the example with zero force.



(a) SSNM solver each 1 iteration for $N = 100$ for $f \equiv 0$

(b) SSNM solver each 1 iteration for $N = 1000$ for $f \equiv 0$

(c) SSNM solver each 1 iteration for $N = 100$ for $f \equiv 5$

Figure 6.2: 1d SSNM solver

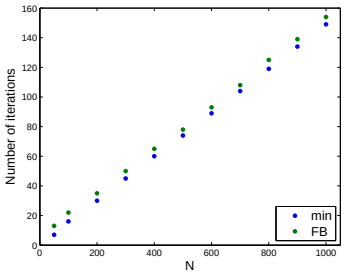
These results were then compared with φ_{FB} as the NCP function for $f \equiv 0$. The comparison of the results is shown in Table 6.2.

Table 6.2: Number of SSNM iterations for the 1d obstacle problem with different NCP functions

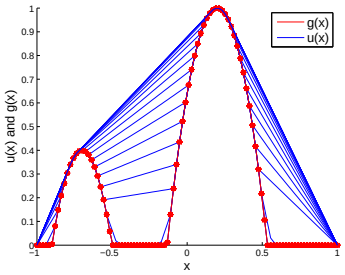
N	50	100	500	1000
φ_{min} iterations	7	16	74	149
φ_{FB} iterations	13	22	78	154

Figure 6.3a shows a comparison of the number of iterations of the two NCP functions considered for various values of N. The number of iterations appears to increase linearly as a function of the number of grid points. Figures 6.3b and 6.3c show a comparison of the evolution of the SSNM solver for the two NCP functions. The iterates move up the free boundary more smoothly than with φ_{min}. Evidently, φ_{min} has the lowest number of iterations for each case considered. However the trend in the number of iterations is very similar for each of the NCP functions. From this point on, only the minimum function was

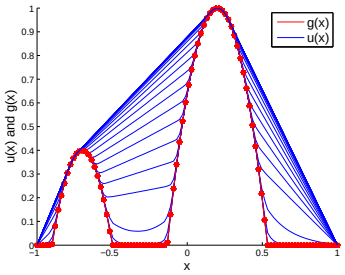
considered.



(a) Number of SSNM iterations



(b) SSNM solver with φ_{min} each 1 iteration for $N = 100$



(c) SSNM solver with φ_{FB} each 1 iteration for $N = 100$

Figure 6.3: Comparison of NCP functions in 1d

6.2 2d Numerical Results

The SSNM solver was implemented in the two-dimensional case for the same example as was considered for the Euler solver. The iterates u^k are now two-dimensional grid functions. The number of iterations for various N values are given in Table 6.3. The solution obtained is shown in Figure 6.4 and is identical to Figure 4.2b.

Table 6.3: Number of SSNM iterations for the 2d obstacle problem

N	50^2	100^2	500^2	1000^2
SSNM iterations	10	18	87	172

Figures 6.5a and 6.5b show that, as in the case of the Euler method, the number of iterations grows with N. The slopes of the lines in the plots with logarithmically scaled axes are found to be approximately 0.9985 and 0.4794, for the one- and two-dimensional cases, respectively. This suggests that the method is $\mathcal{O}(N)$ in the one-dimensional case and $\mathcal{O}(N^{\frac{1}{2}})$ in the two-dimensional case.

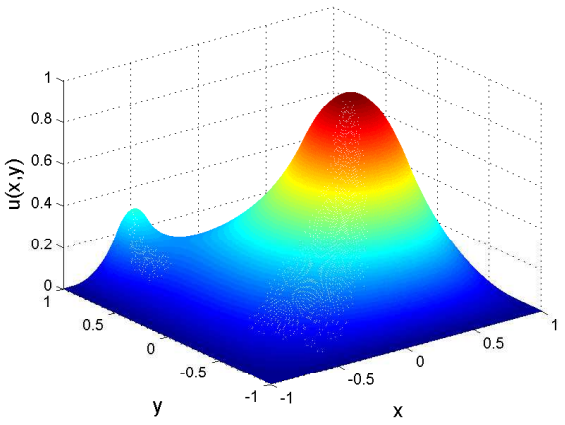
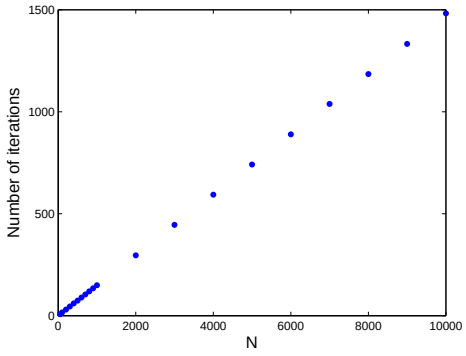
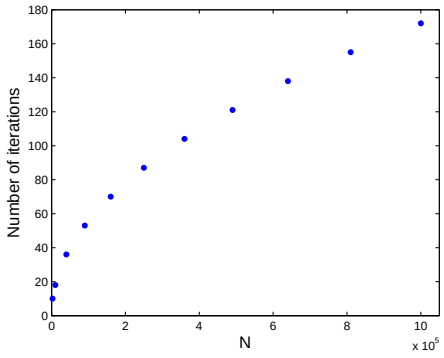


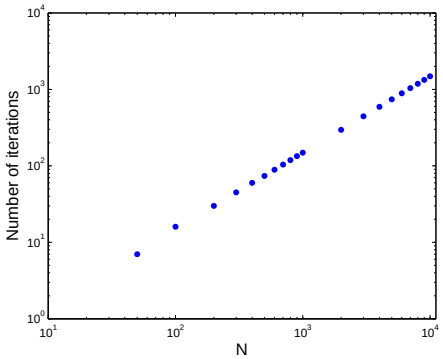
Figure 6.4: SSNM solver for $N = 500^2$



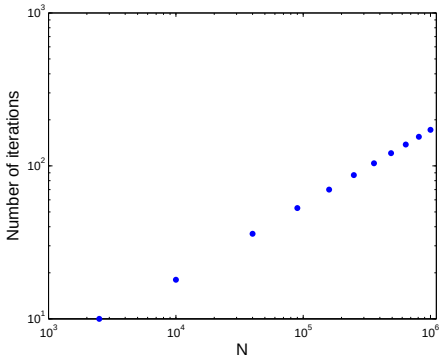
(a) Number of SSNM iterations for the 1d obstacle problem



(b) Number of iterations for the 2d obstacle problem



(c) Number of SSNM iterations for the 1d obstacle problem, log-log plot.



(d) Number of iterations for the 2d obstacle problem, log-log plot.

Figure 6.5: Number of iterations for the SSNM solver

Chapter 7

Combined Method

7.1 Comparison of the Euler and SSNM Solvers

As can be seen when comparing Tables 4.1 and 4.2 with Tables 6.1 and 6.3, the Euler solver takes a much larger number of steps to converge for each value of N tested, than the SSNM solver. However, the two methods should not only be compared in terms of the number of steps. The CPU time needed for the computation of each iteration must also be considered.

The complexity of the computation of each subsequent iterate must also be taken into account. We recall here the iterative updates for both methods in the one-dimensional case with the minimum NCP function. The update for the Euler method is

$$u^{k+1} = u^k - \frac{h^2}{2} \min\left(-D_{xx}u^k - f_j, \frac{u^k - g}{h^2/2}\right),$$

and the SSNM update is

$$u^{k+1} = u^k - (\partial G^h[u^k])^{-1} G^h[u^k],$$

with $G^h[u^k] = \min(-D_{xx}u^k - \frac{h^2}{2}f, u^k - g)$. As in the case with the Euler method, this iteration can use $G^h[u^k] = \min(-D_{xx}u^k - \frac{h^2}{2}f, \frac{u^k-g}{h^2/2})$ instead. Clearly, the SSNM update is

more expensive computationally due to the matrix inversion of the generalized Jacobian at every step. For this reason it is expected that the CPU time taken to compute one SSNM step is much greater than the time taken to compute one Euler step.

The CPU time taken to obtain the results in Tables 4.1 and 6.1 are shown in Table 7.1. Not only does the SSNM solver take fewer iterations until it converges, it also does so much faster than the Euler method. The two-dimensional comparison is shown in Table 7.2, and leads to the same conclusion. The average computation time for one SSNM iteration is much smaller than for one Euler iteration in the two-dimensional case. This suggests that several Euler iterations are as computationally expensive as one SSNM iteration.

Table 7.1: 1d computation time comparison

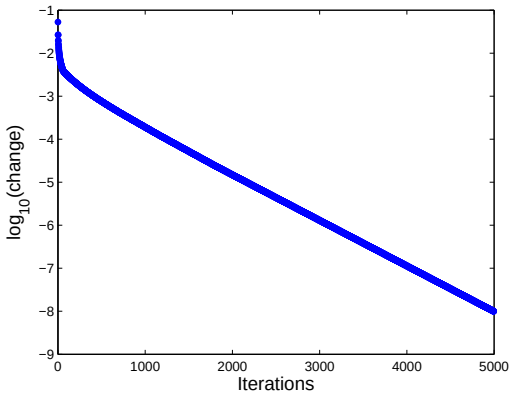
N	50	100	500	1000
Euler iterations	1234	4996	88209	301892
Euler CPU time (s)	0.0359	0.2501	23.6825	416.8883
Euler average time (ms)	0.0291	0.0501	0.2685	1.3809
SSNM iterations	7	16	74	149
SSNM CPU time (s)	0.0193	0.0281	0.0608	0.1352
SSNM average time (ms)	2.7571	1.7563	0.8216	0.9074

Table 7.2: 2d computation time comparison

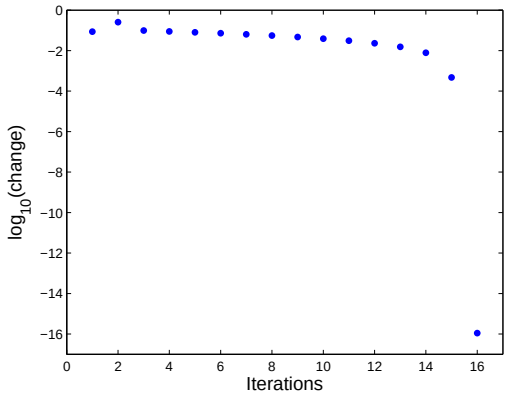
N	50^2	100^2	500^2	1000^2
Euler iterations	2520	9076	159679	519456
Euler CPU time (s)	0.2491	2.4615	1847.2	31727
Euler average time (ms)	0.0988	0.2712	11.5682	61.0774
SSNM iterations	10	18	87	172
SSNM CPU time (s)	0.1377	1.1058	158.4441	2110.9
SSNM average time (s)	0.0138	0.0614	1.8212	12.2727

Each method is said to have converged once $change := \|u^k - u^{k-1}\|_\infty < 10^{-8}$. The evolution of this change as a function of the number of iterations is shown in Figure 7.1 for

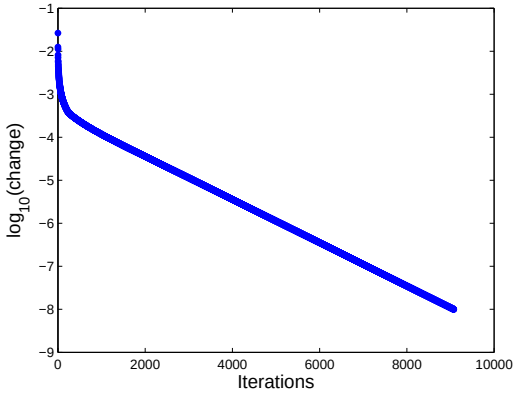
both methods and for the one-dimensional and two-dimensional examples considered. The Euler solver takes smaller and smaller steps toward the solution as the number of iterations increases. The change decreases exponentially as a function of the number of iterations. On the other hand, the SSNM solver takes much larger steps. Once it is sufficiently close to the solution, one large step brings the iterate to the solution. This is expected, since Newton's method is a good local method.



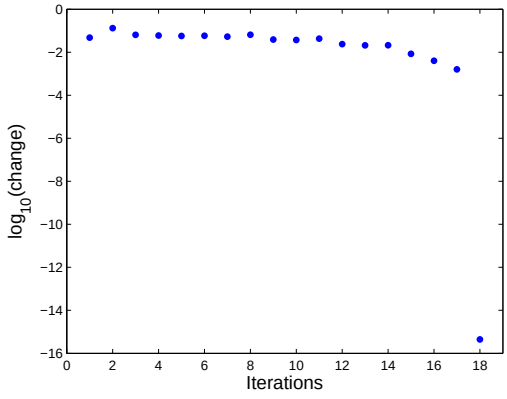
(a) Change at each iteration for the 1d Euler solver with $N = 100$



(b) Change at each iteration for the 1d SSNM solver with $N = 100$



(c) Change at each iteration for the 2d Euler solver with $N = 100^2$



(d) Change at each iteration for the 2d SSNM solver with $N = 100^2$

Figure 7.1: Comparison of methods via the change at each iteration

By comparing Figures 4.1 and 6.2, the following observations can be made. Each iterate in the Euler solver smoothes out the previous iterate as much as the obstacle allows. Each

component of the grid function evolves using only information from its neighbors. Larger values of N naturally lead to slower convergence. The iterates of the SSNM solver move the free boundary up one grid point at a time. Although the SSNM solver provides an important improvement, it would be ideal if the SSNM solver would not cling to the grid points along the obstacle. To combat this problem, we introduce a combined method.

7.2 Combined Algorithm

The idea of the combined method is to alternate between Euler and SSNM iterations. More specifically, since Tables 7.1 and 7.2 suggest one SSNM step corresponds to many Euler steps, in terms of CPU computation time, one combined step will consist of many Euler steps followed by one SSNM step. Let n_E be the number of Euler steps which correspond to one SSNM step in terms of CPU computation time. n_E is dependent on the grid size since the computation time for both solvers varies with N. n_E is found by finding the number of Euler steps which has computation time closest to the computation time of one SSNM step. Let NE be a set of possible n_E values. This set is a range which is chosen through trial and error to be near the final n_E value. Algorithm 4 outlines the method of finding n_E for a given N.

For increased accuracy, this process is repeated several times. Averages are taken to obtain the best value possible. Once n_E has been found, the combined method, Algorithm 5 can be run. Algorithm 5 is essentially Algorithm 4 with $NE = \{\lfloor n_E^* \rfloor\}$. Algorithm 5 in addition allows stopping during the n_E Euler steps, unlike Algorithm 4.

Algorithm 4 Finding n_E

Let $G : \mathbb{R}^N \to \mathbb{R}^N$ be a given semismooth function. Select $u^0 \in \mathbb{R}^N$ and the set NE.

1: Set $k := 0$, $n := 0$, $t^E_{n_E} := 0$, $t^S_{n_E} := 0$ for all n_E.
2: **for** $n_E \in NE$ **do**
3: **for** $n \leq n_E$ **do**
4: Set:

$$u^{k+1} := u^k - dtG[u^k]. \tag{7.1}$$

5: Set $k := k + 1$, $n := n + 1$, $t^E_{n_E} = t^E_{n_E}$+ CPU time for Step 4.
6: **end for**
7: Unless a stopping rule is satisfied, solve (for d^k):

$$H[u^k]d^k = -G[u^k], \tag{7.2}$$

 where $H[u^k] \in \partial G[u^k]$.
8: Set $u^k := u^k + d^k$, $k := k + 1$, $t^S_{n_E} = t^S_{n_E}$+ CPU time for Step 7. Go to Step 3.
9: **end for**
10: Compute the average, $T^S := \text{average}(t^S_{n_E})$.
11: Perform linear regression on $t^E_{n_E}$ as a function of the elements of NE to obtain a linear function $T^E(n_E)$.
12: Find n^*_E such that $T^S = T^E(n^*_E)$.
13: Output $\lfloor n^*_E \rfloor$.

Algorithm 5 Combined obstacle solver

Let $G : \mathbb{R}^N \to \mathbb{R}^N$ be a given semismooth function. Select $u^0 \in \mathbb{R}^N$ and n_E.

1: Set $k := 0$, $n := 0$.
2: **for** $n \leq n_E$ **do**
3: Unless a stopping rule is satisfied, set:

$$u^{k+1} := u^k - dtG[u^k]. \tag{7.3}$$

4: Set $k := k + 1$, $n := n + 1$.
5: **end for**
6: Unless a stopping rule is satisfied, solve (for d^k):

$$H[u^k]d^k = -G[u^k], \tag{7.4}$$

 where $H[u^k] \in \partial G[u^k]$.
7: Set $u^k := u^k + d^k$, $k := k + 1$, $n := 0$.
8: Go to Step 2.

Chapter 8

Combined Implementation

8.1 1d Numerical Results

The combined solver was implemented for the same one-dimensional example problem as in the previous chapters. n_E was first computed for this specific problem using Algorithm 4. Several values of n_E are shown in Table 8.1 and depicted in Figure 8.1. There is an increase in n_E up to around $N = 1000$, at which point it appears to level out.

Table 8.1: Number of Euler iterations corresponding to one SSNM iteration in 1d

N	50	100	500	1000	5000	10000
n_E	18	19	21	23	23	24

The number of iterations needed for convergence of the combined method are shown in Table 8.2 for several values of N. One combined iteration consists of n_E Euler iterations and one SSNM iteration. The Euler steps row includes the total number of Euler steps performed in the combined method. The same is the case for the SSNM steps row. Therefore the number of combined iterations is equal to the number of SSNM steps. Figure 8.2 shows the solution of the combined method for $N = 1000$, plotting every combined iteration. It suggests that the solver does not evolve one grid point at a time along the free boundary, as

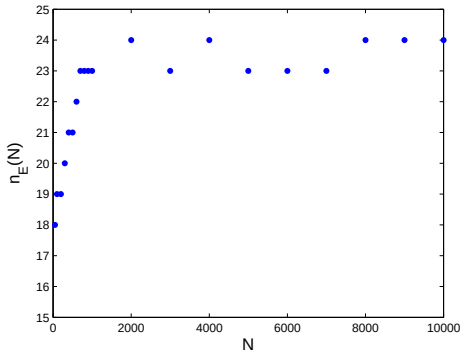


Figure 8.1: Number of Euler iterations corresponding to one SSNM iteration in 1d

was observed in Figure 6.2b. This marks a significant improvement.

Table 8.2: Number of iterations for the 1d obstacle problem using the combined method

N	50	100	500	1000	5000	10000
Euler steps	37	77	232	392	1445	2677
SSNM steps	2	4	11	17	62	111

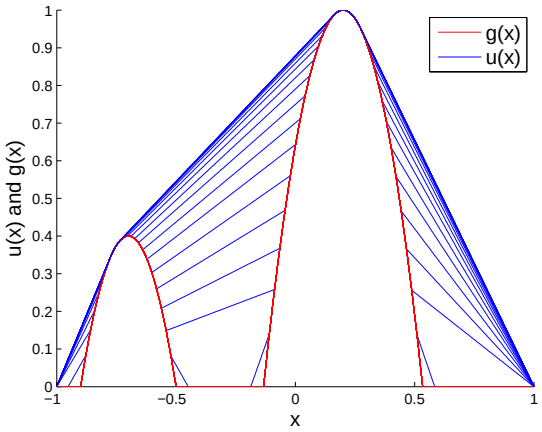
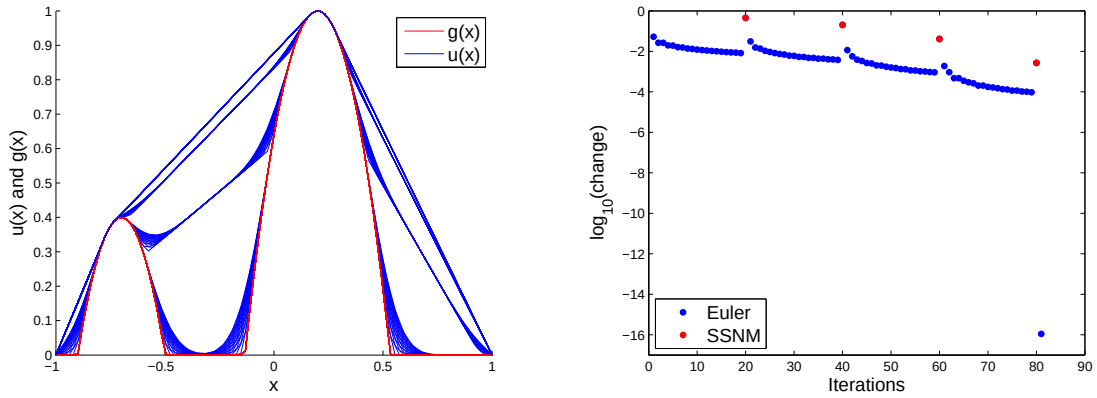


Figure 8.2: Combined solver each combined iteration for $N = 1000$

To examine this improvement more closely, consider Figure 8.3a, which plots the Euler steps involved in each combined step, for $N = 100$. Each set of n_E Euler steps gradually detaches the iterates from the obstacle. When a SSNM step follows, the combined method appears to have taken a much larger step. The n_E Euler steps allow the Newton step to

move the free boundary further up the obstacle. Using a larger value of n_E would allow the iterates to jump further to the solution. However, a larger number n_E of Euler steps would no longer correspond to one SSNM step. The change as a function of the total number of iterations (Euler and SSNM) is shown in Figure 8.3b. Euler steps change the iterates very slowly. Newton steps, on the other hand, advance the iterates faster toward the solution. Once an iterate is close enough to the solution, one Newton step brings the next iterate to the solution. This is again because Newton's method is a good local method.



(a) Combined solver showing Euler and SSNM iterations for $N = 100$

(b) Change at all inner iterations for the combined solver with $N = 100$

Figure 8.3: Analysis of the inner iterations of the combined solver

8.2 2d Numerical Results

The implementation process of the combined method for the two-dimensional case is the same as in the one-dimensional case, with the exception that the two-dimensional grid functions are expanded to be one-dimensional vectors. Again, before running Algorithm 5, n_E was found for several grid sizes with Algorithm 4. The results of this process are shown in Table 8.3 and depicted in Figure 8.4.

Table 8.3: Number of Euler iterations corresponding to one SSNM iteration in 2d

N	50^2	100^2	500^2	1000^2
n_E	180	219	238	412

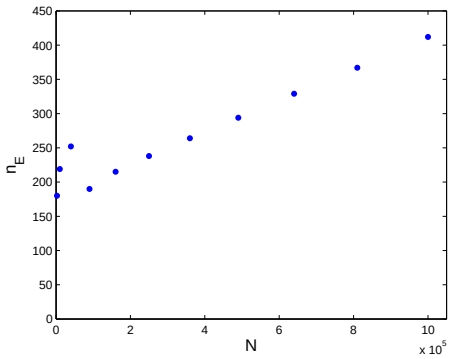


Figure 8.4: Number of Euler iterations corresponding to one SSNM iteration in 2d

The results of Algorithm 5 are shown in Table 8.4. Figure 8.5 shows the solution obtained from the combined method. As in the one-dimensional case, the combined method is significantly faster than the SSNM in terms of the number of SSNM steps.

Table 8.4: Number of iterations for the 2d obstacle problem using the combined method

N	50^2	100^2	500^2	1000^2
Euler steps	361	358	1667	3708
SSNM steps	2	3	7	8

A closer look at the change in the inner iterations of the combined method is shown in Figure 8.6. Similarly to the one-dimensional example, the inner Euler steps provide relatively small changes, while the SSNM steps provide large steps.

Figure 8.7 shows the number of iterations as a function of N for both the one-dimensional and two-dimensional cases. The plots with the logarithmically scaled axes have slopes of 0.7442 and 0.2412, for the one- and two-dimensional cases, respectively. This suggests that the combined method is of order $\mathcal{O}(N^{\frac{3}{4}})$ in one dimension and $\mathcal{O}(N^{\frac{1}{4}})$ in two dimensions. In

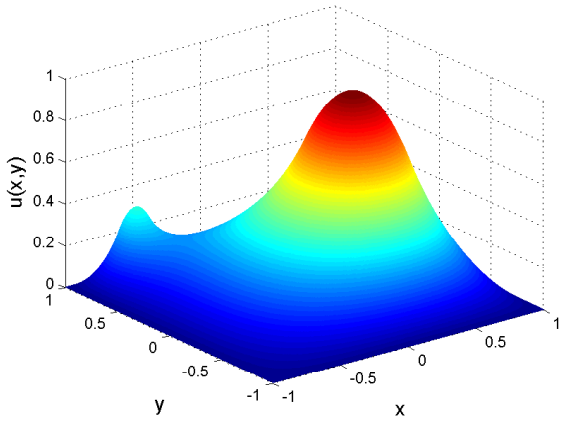


Figure 8.5: Combined solver for $N = 1000^2$

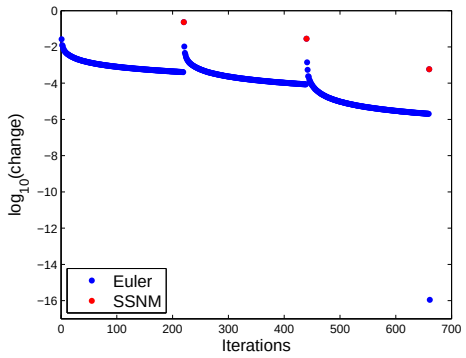
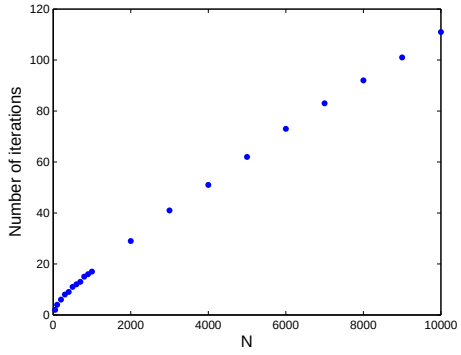


Figure 8.6: Change at all inner iterations for the combined solver with $N = 100^2$

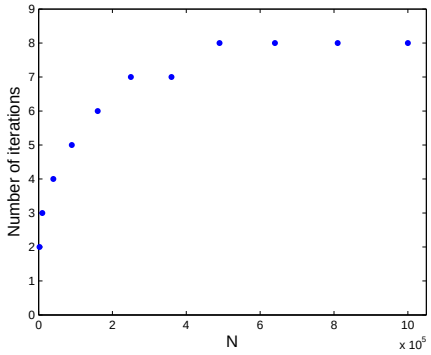
two dimensions, the number of iterations levels out at eight. This is quite impressive. As N becomes large, the computation time for one SSNM step is very large, allowing many Euler steps to be performed in one combined iteration.

8.3 Comparison of the Three Solvers

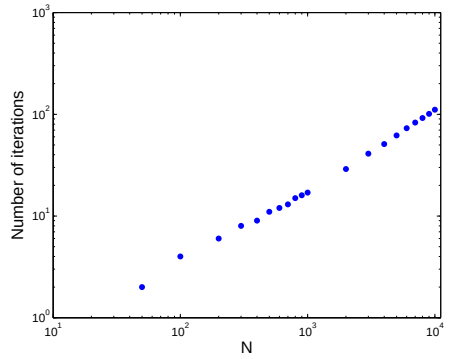
To compare the combined method to the two previously considered methods, all iterations must be given on the same scale. To achieve this, we convert the Euler iterations to be of SSNM "size". The values in Table 4.1 are divided by the values in Table 8.1 and rounded to the nearest integer to obtain SSNM "sized" iterations. For the combined iterations, the



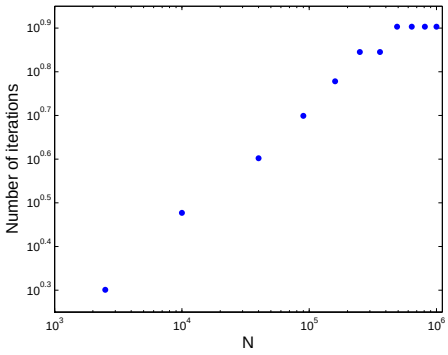
(a) Number of combined iterations for the 1d obstacle problem



(b) Number of combined iterations for the 2d obstacle problem



(c) Number of combined iterations for the 1d obstacle problem, log-log plot.



(d) Number of combined iterations for the 2d obstacle problem, log-log plot.

Figure 8.7: Number of iterations for the combined solver

number of Euler steps and SSNM steps must also be combined in a specific way to reflect SSNM "sized" iterations. The number of weighted combined iterations is computed as

$$\#\text{ weighted combined iterations} = \left\lfloor \frac{\#\text{ Euler iterations}}{n_E} + \#\text{ SSNM iterations} \right\rfloor. \qquad (8.1)$$

Table 8.5 combines the results of Table 6.1 with these new computations. Table 8.6 shows the results from Tables 4.2 and 8.4 weighted to be in terms of SSNM iteration time. In both cases, each subsequent method requires fewer iterations.

Figure 8.8 compares the number of iterations for the three methods, using the iterations in Tables 8.5 and 8.6. Clearly, the combined method is a big improvement on the two pre-

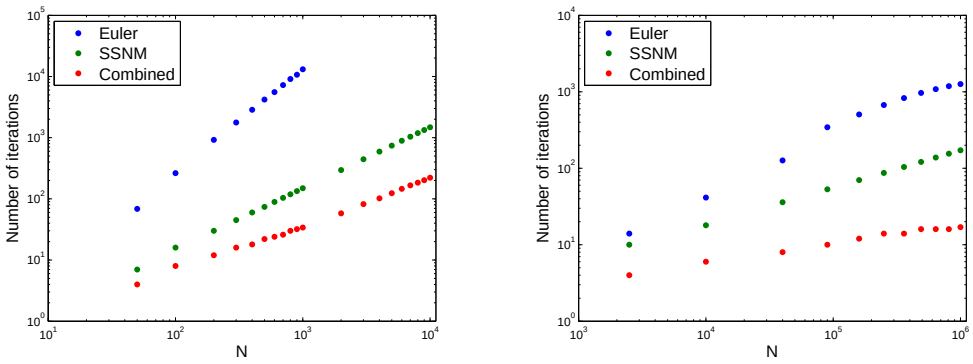
Table 8.5: Number of iterations for the 1d obstacle problem weighted on SSNM iterations

N	50	100	500	1000	5000	10000
Euler iterations	69	263	4200	13126		
SSNM iterations	7	16	74	149	741	1482
Combined iterations	4	8	22	34	124	222

Table 8.6: Number of iterations for the 2d obstacle problem weighted on SSNM iterations

N	50^2	100^2	500^2	1000^2
Euler iterations	14	41	671	1261
SSNM iterations	10	18	87	172
Combined iterations	4	6	14	17

vious methods. The slopes of the lines in Figure 8.8 are shown in Table 8.7. The combined method significantly reduces the dependence on the grid size.



(a) 1d comparison of all three methods *(b) 2d comparison of all three methods*

Figure 8.8: Comparison of the number of iterations for the methods

Table 8.7: Slopes of the log-log plots of Figure 8.8

Method	Euler	SSNM	Combined
1d	1.7295	0.9985	0.7442
2d	0.7773	0.4794	0.2445

Chapter 9

Conclusions

9.1 Summary

We began by introducing the classical obstacle problem in terms of its energy formulation. The variational formulation was then shown to be a nonlinear complementarity problem. The minimum function and the Fischer-Burmeister functions were chosen as the NCP functions for the remainder of the thesis. The more general obstacle problem was then introduced.

After discretizing the problem and defining finite difference operators, three methods were introduced and compared numerically with examples in one and two dimensions. The Euler iterative method was found to be the slowest. The SSNM significantly speeds up convergence, however the combined method is the best. The proposed combined method alternates several (n_E) Euler steps with one SSNM step. The number of Euler steps chosen corresponds to the CPU time of one SSNM step for the particular problem considered. The combined method succeeds in significantly reducing the number of Newton sized steps for convergence. The number of Newton sized iterations is still dependent on the grid size chosen, but the number of iterations is very small.

9.2 Future Work

The number of Euler steps corresponding to one SSNM step is not calculated in a precise way. It is done by averaging CPU times of both methods over several identical tests. This leads to variations in the results for n_E. Algorithm 4 is always performed several times and the results are averaged in an attempt to reduce error. n_E is very dependent on N, the size of the grid, since the CPU time taken to perform one SSNM step is dependent on N. The larger N is, the larger the matrices used in each iterate computation are, and hence the longer it takes to solve for each iterate.

So far it is assumed that n_E must be found for each operator, $F[u]$, considered. In addition, n_E must ideally be found for each specific obstacle considered. Perhaps it may be possible to find a general trend in n_E for similar problems or similar obstacles. It would be ideal for n_E to at least be independent of the obstacle. It is no good finding a fast combined method if one first needs to spend a considerable amount of time finding n_E before implementing the fast method. If estimates for n_E for many values of N were found, the whole process could be sped up. It was noted that more Euler steps per SSNM step results in a smaller number of total SSNM sized steps. It may be possible to find some bounds on the value of n_E for a particular obstacle, operator, or size N. More effort should be focused on finding a theoretical model for the values of n_E.

We only considered φ_{min} and φ_{FB} as NCP functions. It would be worth comparing its performance with other NCP functions such as the penalized Fischer-Burmeister NCP function [2]. The Fischer-Burmeister function has a smooth merit function. A possible next step would be to globalize the SSNM using the merit function as done in [10] and pair it with iterative steps to again obtain a combined method.

The next step would be to implement the combined method on a general obstacle prob-

lem with a nonlinear operator. Ideally, the performance of the combined method would be similar to the linear case.

Bibliography

[1] L. A. Caffarelli. The obstacle problem revisited. *The Journal of Fourier Analysis and Applications*, 4(4-5):383–402, 1998.

[2] B. Chen, X. Chen, and C. Kanzow. A penalized Fischer-Burmeister NCP-function. *Mathematical Programming*, 88:211–216, 2000.

[3] F. H. Clarke. *Optimization and Nonsmooth Analysis*. John Wiley & Sons, New York, NY, 1983.

[4] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100:32–754, 1928.

[5] L. C. Evans. *Partial Differential Equations, 2nd edition*. Graduate Studies in Mathematics, vol. 19. American Mathematical Society, Providence, RI, 2010.

[6] L. C. Evans. *An Introduction to Stochastic Differential Equations*. American Mathematical Society, Providence, RI, 2013.

[7] F. Facchinei and J.-S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems*. Springer New York, New York, NY, 2003.

[8] F. Facchinei and J. Soares. A new merit function for nonlinear complementarity problems and a related algorithm. *SIAM Journal on Optimization*, 7.1:225–247, 1997.

[9] M. Hintermüller. Semismooth Newton methods and applications. *Department of Mathematics, Humboldt-University of Berlin*, 2010.

[10] C. Kanzow. Inexact semismooth Newton methods for large-scale complementarity problems. 19(3–4):309–325, 2004.

[11] T. De Luca, F. Facchinei, and C. Kanzow. A theoretical and numerical comparison of some semismooth algorithms for complementarity problems. *Computational Optimization and Applications, Springer Verlag*, 16:173–205, 2000.

[12] R. Mifflin. Semismooth and semiconvex functions in constrained optimization. *SIAM Journal on Control and Optimization*, 15:957–972, 1977.

[13] A. M. Oberman. Convergent difference schemes for degenerate elliptic and parabolic equations: Hamilton-Jacobi equations and free boundary problems. *SIAM Journal on Numerical Analysis*, 44:879–895, 2006.

[14] A. M. Oberman. The convex envelope is the solution of a nonlinear obstacle problem. *Proceedings of the American Mathematical Society*, 135(6):1689–1694, 2007.

[15] L. Qi and J. Sun. A nonsmooth version of Newton's method. *Mathematical Programming*, 58:353–368, 1993.

[16] J.-F. Rodrigues. *Obstacle Problems in Mathematical Physics*. North-Holland Publishing Company, Amsterdam, Netherlands, 1987.