TOWARD AN OPTIMAL SOLVER FOR THE OBSTACLE PROBLEM

By

Max Heldman, A.B.

A Project Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in

Mathematics

University of Alaska Fairbanks

April 2018

APPROVED:

Ed Bueler, Committee Chair
David Maxwell, Committee Member
John Rhodes, Committee Member
Leah Wrenn Berman, Chair
Department of Mathematics and Statistics
Anupma Prakash, Dean
College of Natural Science and Mathematics
Michael Castellini, *Dean of the Graduate School*

Abstract

An optimal algorithm for solving a problem with m degrees of freedom is one that computes a solution in $O(m)$ time. In this paper, we discuss a class of optimal algorithms for the numerical solution of PDEs called *multigrid methods*. We go on to examine numerical solvers for the obstacle problem, a constrained PDE, with the goal of demonstrating optimality. We discuss two known algorithms, the so-called reduced space method (RSP) [BM03] and the multigrid-based projected full-approximation scheme (PFAS) [BC83]. We compare the performance of PFAS and RSP on a few example problems, finding numerical evidence of optimality or near-optimality for PFAS.

Table of Contents

Page

Title Page i

Abstract iii

Chapter 1: Introduction 1
1.1 Introduction 1

Acknowledgments 0

Chapter 2: Background and theory for obstacle problems 5

Chapter 3: Multigrid 11
3.1 Geometric Multigrid 11
3.1.1 Smoothing and Error Modes 12
3.1.2 Coarse Grid Correction 19
3.1.3 Multigrid Cycles 23
3.2 Algebraic Multigrid 28

Chapter 4: Algorithms 31
4.1 Reduced Space Method 31
4.2 Projected Full-Approximation Scheme 35

Chapter 5: Numerical implementation 37
5.1 A general 2D obstacle problem solver 39
5.2 PFAS implementation 40
5.3 Reduced space method 42

Chapter 6: Results and analysis 43
6.1 PFAS results 45
6.2 Reduced space method 48

References 49

Chapter 1

Introduction

1.1 Introduction

The classical obstacle problem seeks the equilibrium position of a membrane stretched over a convex 3-dimensional obstacle ψ sitting on a precompact domain $\Omega \subseteq \mathbb{R}^2$. The membrane is assumed to exhibit no resistance to bending, so that the only force interior to the system of obstacle and membrane comes from the uniform tension created by fixing the edges of the membrane to the boundary of Ω.

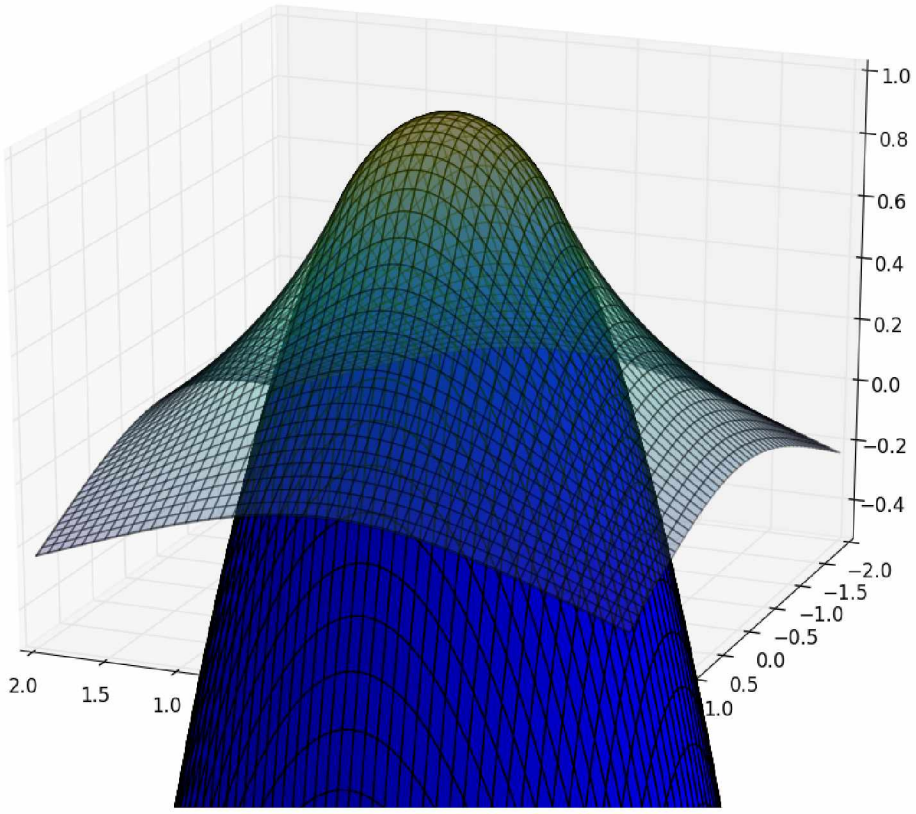


Figure 1.1: Solution to an obstacle problem.

The obstacle problem can be formulated as the constrained minimization of the potential energy functional $J[v]$ and generalized to $\mathbb{R}^d$, given by the formula

$$\min_v J[v] := \frac{1}{2}\int_\Omega |\nabla v|^2 - vf \qquad \text{subject to} \qquad \begin{matrix} v \geq \psi, \\ v|_{\partial\Omega} = g. \end{matrix} \tag{1.1}$$

In the context of the classical obstacle problem, v gives the position of the membrane, f describes a force on the membrane, where a downward force is negative, ψ describes the obstacle, and g is a boundary condition.

Obstacle problems of the type described by (1.1) occur naturally in various fields of applied mathematics, from engineering applications like the modeling of filtration through a porous medium[1] to problems in economics and financial mathematics such as the pricing of American options.[2]

The motivation of this paper is to provide a rigorous analysis of computationally feasible algorithms that solve the obstacle problem numerically. The ultimate goal is to solve the problem in *optimal time complexity*, meaning roughly that the time to solve the problem (i.e. the number of floating point operations) scales linearly with the size of the problem (i.e. the number of unknowns in a discretization). A main idea underlying the most promising algorithms presented here is called *multigrid* ([TSO01], [BHM00]), which was introduced around 1960 specifically as a method for solving the Poisson problem on a unit square ([Fed01]). About 10 years later, Achi Brandt, who would become the champion of multigrid methods and famously claimed that multigrid was the solution to every problem (including making goat cheese),[3] applied multigrid to a more general class of partial differential equations. The study of multigrid methods has since flourished, being applied to an increasingly broad range of problems, in particular as a solver for general linear systems. Moreover, multigrid methods have been applied to nonlinear problems, eigenvalue problems, bifurcation problems, parabolic problems, hyperbolic problems and mixed elliptic/hyperbolic problems, optimization problems, and constrained PDEs (as in this paper) etc. (see, e.g., [Bor03]) (but as far as the author knows, not yet toward making goat cheese).

The organization of this paper is as follows. In the next chapter, we briefly discuss some theory of the obstacle problem, including the proof of a well-known formulation of the obstacle problem as a linear complementarity problem (LCP). We go on in Chapter 3 to give a somewhat detailed but introductory description of multigrid methods. In Chapter 4, we describe the main algorithms considered in this paper, the projected full approximation

[1] E.g. the example in [BC83], used in this paper as well.

[2] [FLMN10], for example

[3] See https://www.youtube.com/watch?v=psRA0pHA_Zo, or youtube search Achi Brandt goat cheese.

scheme (PFAS) due to [BC83] and the reduced space Newton method [BM03]. In Chapter 5, we give a few details of our Python-based implementation of these algorithms. We present numerical results and analysis for PFAS and the reduced space method in Chapter 6.

Chapter 2

Background and theory for obstacle problems

The derivation of the classical obstacle problem from the previous chapter, which we have described mathematically by (1.1), goes as follows. The problem is to minimize the potential energy of a membrane subject to the constraint imposed by the obstacle and boundary conditions. Assuming that the potential energy is proportional to the increase in the area of the membrane surface, this takes the form

$$\min_v I[v] := \int_\Omega \sqrt{1+|\nabla v|^2} \qquad \text{subject to} \qquad \begin{matrix} v \geq \psi, \\ v|_{\partial\Omega} = g, \end{matrix}$$

where the functional $I[v]$ is the surface area integral again ψ represents the obstacle and g the boundary condition. Expanding the function under the integral in Taylor series and dropping higher order terms gives us

$$I[v] \approx \int_\Omega 1 + \frac{1}{2}|\nabla v|^2.$$

Hence, in the case of small first derivatives, minimizing the Dirichlet energy $\int_\Omega \frac{1}{2}|\nabla v|^2$ is equivalent to minimizing potential energy for the membrane in a closed system. If we then consider work done by external forces, given by

$$\int_\Omega vf,$$

we see that the solution to (1.1) minimizes the potential energy of deformation, given by the functional

$$J[v] := \frac{1}{2}\int_\Omega |\nabla v|^2 - vf.$$

Intuitively, the Dirichlet energy provides a measure of the "smoothness" of a function u. We will see that the solution u to the obstacle problem, i.e., the minimizer of (1.1) in the special

case $\psi = -\infty$, also solves the *Poisson equation*

$$\Delta u = -f, \qquad u = g \text{ on } \partial\Omega, \tag{2.1}$$

where $\Delta u = \sum_{i=1}^{n} \frac{\partial v^2}{\partial^2 x_i}$ is the Laplacian. In the case $f = 0$, the solution is harmonic.

The obstacle problem with an arbitrary obstacle is also closely related to the Poisson equation, as we see from the following well-known theorem which characterizes the obstacle problem as an energy minimization problem (2.3), a *variational inequality* (2.4), and infinite-dimensional *linear complementarity problem* (LCP) (2.5) - (2.7). This particular proof follows the strategy outlined in the monograph [Rod87], whose argument specifies $d = 2$ but has been generalized here with little effort to arbitrary d.

Theorem 2.1. *Fix $g, \psi, f \in C^\infty(\Omega)$. Define*

$$\mathcal{K} = \{v \in H^1(\Omega) : v \geq \psi \text{ on } \Omega, v = g \text{ on } \partial\Omega\}. \tag{2.2}$$

The following two statements are equivalent for a function $u \in \mathcal{K}$:

(a) u is a solution to the energy minimization problem

$$\min_{v \in H^1(\Omega)} J[v] = \frac{1}{2}\int_\Omega |\nabla v|^2 - vf \qquad \text{subject to} \qquad \begin{matrix} v \geq \psi, \\ v|_{\partial\Omega} = g; \end{matrix} \tag{2.3}$$

(b) u solves the variational inequality

$$\int_\Omega \nabla u \cdot \nabla(v - u) \geq \int_\Omega f(v - u), \qquad \text{for all } v \in \mathcal{K}. \tag{2.4}$$

If in addition $u \in C(\Omega)$, then (a) and (b) are equivalent to a third condition:

(c) u solves the linear complementarity problem

$$u(x) - \psi(x) \geq 0, \tag{2.5}$$

$$-f(x) - \Delta u(x) \geq 0, \tag{2.6}$$

$$[-f(x) - \Delta u(x)][u(x) - \psi(x)] = 0, \tag{2.7}$$

where the inequalities and the operator Δ are taken in a distributional sense, i.e., $q(x) \geq 0$ on Ω if and only if

$$\int_\Omega q\varphi \geq 0$$

for all compactly supported nonnegative test functions $\varphi \in C^\infty(\Omega)$.

Proof. First, suppose that u solves (2.3), so that u minimizes the energy functional $J[v]$ over $\mathcal{K}$ The set $\mathcal{K}$ is convex, so $u+\epsilon(v-u) \in \mathcal{K}$ for any $\epsilon \in [0,1]$ and any $v \in \mathcal{K}$. Hence, if $v \in \mathcal{K}$, the (continuous) function $h : [0,1] \to \Omega$ defined by

$$h(\epsilon) = J[u+\epsilon(v-u)]$$

has a minimum at $\epsilon = 0$, which implies $h'(0) \geq 0$. Thus, we have

$$\begin{aligned}
&\int_\Omega \nabla u \cdot \nabla(v-u) - \int_\Omega f(v-u) \\
&= \lim_{\epsilon\to 0^+} \frac{1}{2}\left(\epsilon \int_\Omega |\nabla(v-u)|^2 + 2\int_\Omega \nabla u \cdot \nabla(v-u)\right) - \int_\Omega f(v-u) \\
&= \lim_{\epsilon\to 0^+} \frac{\frac{1}{2}\left(\int_\Omega |\nabla u|^2 + \epsilon^2 \int_\Omega |\nabla(v-u)|^2 + 2\epsilon \int_\Omega \nabla u \cdot \nabla(v-u)\right) - \frac{1}{2}\int_\Omega |\nabla u|^2 - \int_\Omega uf + \int_\Omega uf + \epsilon \int_\Omega f(v-u)}{\epsilon} \\
&= \lim_{\epsilon\to 0^+} \frac{\frac{1}{2}\int_\Omega |\nabla(u+\epsilon(v-u))|^2 - \int_\Omega (u+\epsilon(v-u))f - \left(\frac{1}{2}\int_\Omega |\nabla u|^2 - \int_\Omega uf\right)}{\epsilon} \\
&= \lim_{\epsilon\to 0^+} \frac{J[u+\epsilon(v-u)] - J[u]}{\epsilon} \\
&= \lim_{\epsilon\to 0^+} \frac{h(\epsilon)-h(0)}{\epsilon} = h'(0).
\end{aligned}$$

Hence

$$\int_\Omega \nabla u \cdot \nabla(v-u) - \int_\Omega f(v-u) \geq 0,$$

so u solves (2.4).

Conversely, let u be a solution of (2.4). Then we have

$$\begin{aligned}
J[v] &= J[u+(v-u)] \\
&= \frac{1}{2}\int_\Omega |\nabla(u+(v-u))|^2 - (u+(v-u)f) \\
&= \frac{1}{2}\int_\Omega |\nabla u + \nabla(v-u)|^2 - (u+(v-u)f) \\
&= \frac{1}{2}\int_\Omega |\nabla u|^2 - uf + \frac{1}{2}\int_\Omega |\nabla(v-u)|^2 + \left[\int_\Omega \nabla u \cdot \nabla(v-u) - (v-u)f\right] \\
&= J[u] + \frac{1}{2}\int_\Omega |\nabla(v-u)|^2 + \left[\int_\Omega \nabla u \cdot \nabla(v-u) - (v-u)f\right] \geq J[u].
\end{aligned}$$

Hence (2.3) and (2.4) are equivalent.

Now suppose that u is continuous. We will show that u solves (2.5) - (2.7) if and only if u solves (2.4). Assume u solves (2.4). It will be sufficient to show

1. $\Delta u = -f$ outside of the open set

$$E_u = \{x \in \Omega : u(x) = \psi(x)\}; \tag{2.8}$$

2. $f + \Delta u \leq 0$;

3. $[f(x) + \Delta u(x)][u(x) - \psi(x)] = 0$

in the sense of distributions.

For claim 1, choose a function $\varphi \in C^\infty(\Omega)$ with compact support on $(E_u)^c$. By continuity of u there exists $\epsilon > 0$ such that $u + t\varphi \in \mathcal{K}$ whenever $|t| < \epsilon$. Hence, for any $t \in (0, \epsilon)$ we have

$$\begin{aligned} t\langle f + \Delta u, \varphi\rangle &= t\int_{(E_u)^c} f\varphi - \nabla u \cdot \nabla\varphi \\ &= \int_\Omega f((u + t\varphi) - u) - \nabla u \cdot \nabla((u + t\varphi) - u) \leq 0. \end{aligned}$$

Since this is also true for $-t$,

$$\langle f + \Delta u, \varphi\rangle = 0,$$

and claim 1 is proven.

Moreover, if we take $\varphi \in C^\infty(\Omega)$ to be compactly supported on Ω and positive, then $u + \varphi \in \mathcal{K}$, and by the same method we find that claim 2 holds:

$$\begin{aligned} \langle f + \Delta u, \varphi\rangle &= \int_\Omega f\varphi - \nabla u \cdot \nabla\varphi \\ &= \int_\Omega f((u + \varphi) - u) - \nabla u \cdot \nabla((u + \varphi) - u) \leq 0, \end{aligned}$$

which is what we needed to show.

Finally, for claim 3 we have for any compactly supported test function φ

$$\begin{aligned} \langle [f(x) + \Delta u(x)][u(x) - \psi(x)], \varphi\rangle &= \int_\Omega f(u - \psi)\varphi - \nabla u \cdot \nabla((u - \psi)\varphi) \\ &= \int_{(E_u)^c} f[(u - \psi)\varphi] - \nabla u \cdot \nabla[(u - \psi)\varphi] = 0, \end{aligned}$$

where the second equality follows from the fact that $u - \psi = 0$ on E_u and the third follows by the argument from claim 1, noting that $(u - \psi)\varphi \in H_0^1(E_u^c)$.

Conversely, assume (2.5) - (2.7) hold. Note that on $(E_u)^c$, $u > \psi$, and this combined with the conditions (2.6) and (2.7) imply that $\Delta u = -f$ on $(E_u)^c$. We call the property that either $u - \psi = 0$ or $-\Delta u - f = 0$ on all of Ω the complementarity condition.

For any $v \in \mathcal{K}$ we find

$$\begin{aligned}
\int_\Omega \nabla u \cdot \nabla(v-u) &= \int_\Omega \nabla u \cdot \nabla(v - \psi + \psi - u) \\
&= \int_\Omega \nabla u \cdot \nabla(v-\psi) + \int_\Omega \nabla u \cdot \nabla(\psi - u) \\
&\geq \int_\Omega f(v-\psi) + \int_\Omega \nabla u \cdot \nabla(\psi - u) && \text{(by (2.6), recalling } v - \psi \geq 0) \\
&= \int_\Omega f(v-\psi) + \int_{(E_u)^c} \nabla u \cdot \nabla(\psi - u) && \text{(because } u - \psi \equiv 0 \text{ on } E_u) \\
&= \int_\Omega f(v-\psi) + \int_{(E_u)^c} f(\psi - u) && (\Delta u = -f \text{ on } (E_u)^c) \\
&= \int_\Omega f(v-\psi) + \int_\Omega f(\psi - u) && \text{(again, } u - \psi \equiv 0 \text{ on } E_u) \\
&= \int_\Omega f(v-u).
\end{aligned}$$

□

The region $E_u \equiv \{u(x) = \psi(x)\}$ defined in (2.8) is known as the *contact region* or *coincidence set* for the solution u, while the boundary of the contact set $\partial E_u = \Gamma_u$ is known as the *free boundary* or the *dynamic interface*. In general, a *free boundary problem* is a partial differential equation whose solution consists of an unknown region E_u and a function u which are computed simultaneously. Often, if the region E_u is known, the free boundary problem reduces to a nonlinear partial differential equation. The obstacle problem (2.3), for example, reduces to solving

$$\Delta u(x) = -f(x) \qquad \text{subject to} \qquad \begin{aligned} u(x) &= \psi(x), \quad x \in \Gamma_u, \\ u(x) &= g(x), \quad x \in \partial\Omega \end{aligned}$$

on $(E_u)^c$. The LCP formulation of the obstacle problem is particularly useful for numerical solvers, since it does not require explicit computation of the free boundary.

As promised, (2.5) - (2.7) show that the solution u of the obstacle problem satisfies the Poisson equation on the entire domain in the case $\psi \equiv -\infty$, since in that case inequality (2.5) is always strict. Moreover, even in the generic case the obstacle satisfies the Poisson equation everywhere outside the contact set. Thus an understanding of the Poisson equation is necessary to solving the obstacle problem, and is sufficient if we know the location of the

free boundary. One might imagine an iterative method for solving the obstacle problem that first approximates the free boundary, and reverts to a Poisson solver once the contact set has been established.

The constraint imposed by the Poisson equation for the solution of the obstacle problem implies that we cannot possibly expect continuous second derivatives except in the case where the obstacle itself satisfies $-\Delta\psi = f$. However, it is well-known that solutions for obstacle-type problems share regularity properties with the obstacle for lower derivatives; for example, it is known that $u \in C^1(\Omega)$ if and only if $\psi \in C^1(\Omega)$ [Rod87].

As mentioned earlier, the proof of Theorem 2.1 was adapted from the monograph [Rod87], which also gives a nice introduction to general free boundary problems that arise in physics: the bending of a plate over an obstacle and the closely-related minimal surface equation, elastoplastic torsion of a cylindrical bar, the cavitation problem in hydrodynamic lubrication, the dam problem (filtration through a porous medium) are just a few examples from the first chapters of his book. Since our focus is on numerical methods, rather than proving any more results about obstacle problems, we encourage the interested reader to consult [Rod87] for a rigorous introduction to the continuous theory of obstacle-type problems.

Chapter 3

Multigrid

The standard algorithms for solving a dense linear system with n unknowns have $O(n^3)$ complexity. However, for a number of specific cases, there are more efficient methods. *Multigrid* is an algorithmic framework for solving large linear systems $Ax = b$ with optimal or near-optimal complexity. In this context, this means that an effective multigrid algorithm solves a linear system with n unknowns in just $O(n)$ time, i.e., multigrid methods scale linearly with the number of unknowns.

There are two broad types of multigrid methods: geometric multigrid (GMG), which is only applicable to those linear systems which have a geometric interpretation, e.g. those arising from a mesh-based solution of a PDE, and algebraic multigrid (AMG), which generalizes the ideas from geometric multigrid to problems with no geometric structure at all. We deal mostly with the former in this chapter, since the underlying ideas are easier to visualize and the analysis more established, before talking about AMG, which tends to be more abstract and heuristic rather than theory based.

This chapter gives a basic introduction to multigrid ideas with the intent of understanding the ideas underlying certain multigrid-based obstacle problem solvers. For a comprehensive introduction to multigrid algorithms, the reader is encouraged to consult the texts [BHM00] and [TSO01] on which much of the discussion in the following section is heavily based.

3.1 Geometric Multigrid

In this section, we will consider the solution of an invertible linear equation $A^h u^h = f^h$ arising from the discretization of a PDE $\mathcal{L}u = f$ occurring on a domain $\Omega \subseteq \mathbb{R}^d$, with the imposed boundary condition $u|_{\partial\Omega} = g$. Here $\mathcal{L}$ is a linear differential operator, and $f : \Omega \to \mathbb{R}$ is a function independent of the unknown u. The discretization occurs on a grid G^h, by which we mean a finite subset of Ω. The h superscripts indicate the grid spacing, i.e, the distance between adjacent grid points, which we assume for now is uniform.

Suppose v^h is an approximation to the continuum solution u on some grid

$$G^h = \{x_0, x_1, ..., x_N\} \subseteq \Omega.$$

We make some *algebraic error* $e^h = u^h - v^h$, where u^h is the exact solution vector $(A^h)^{-1} f^h$. We emphasize that u^h should not be confused with the exact solution to the continuum problem imposed on the grid,

$$\begin{bmatrix} u(x_1) & u(x_2) & ... & u(x_N) \end{bmatrix}.$$

The *residual* r^h associated with the approximation v^h is defined as $r^h = f^h - A^h v^h$. This quantity describes in some sense how closely v^h approximates the exact solution, since $\left\| f^h - A^h v^h \right\| = 0$ if and only if $v^h = u^h$. In fact, we can say even more about the relationship between the error e^h and the residual r^h. The following derivation is a common one in numerical analysis, and is particularly useful to multigrid ideas:

$$A^h e^h = A^h (u^h - v^h) = A^h u^h - A^h v^h = f^h - A^h v^h = r^h.$$

That is, solving the error equation

$$A^h e^h = r^h \tag{3.1}$$

for e^h is equivalent to computing the exact solution u^h on G^h. However, (3.1) shows that $r^h \approx 0$ does not necessarily imply $e^h \approx 0$ without some assumptions on the size of $\|A^{-1}\|$.

3.1.1 Smoothing and Error Modes

A multigrid solver consists of two important pieces of machinery. The first is a *smoother*, which generally refers to an iterative relaxation scheme for solving linear systems, but really encompasses any method for "smoothing" errors in a linear system. We focus on smoothers in this section. The other is a *coarse grid correction* scheme, which will be considered later.

The two most well-known relaxation schemes are the Gauss-Seidel and the Jacobi iterations. The smoothing property that these methods exhibit, and that multigrid methods take advantage of, will be described algebraically and numerically below. The basic idea is that these methods, when applied to the equation $Au = 0$, effectively reduce 'oscillatory' components in an initial guess u. This makes u smoother in a geometric sense without necessarily making much progress toward a global solution. In this paper, we will only discuss and use the Gauss-Seidel iterative relaxation scheme.

In this section, we will describe the smoothing property in detail, motivated by numerical examples, but supported by rigorous theoretical work. This will come in handy later when we consider the problem of applying multigrid to constrained PDEs.

Model Problem 3.1.1. Our model problem for the remainder of the section on geometric multigrid is the Poisson equation (2.1). In particular, because of the simplicity of representing one-dimensional problems, we will deal here with the one-dimensional problem $u''(x) = 0$ on some closed interval with zero Dirichlet boundary conditions. This problem has a unique exact solution of $u(x) \equiv 0$ on Ω. The standard finite difference discretization for this problem is as follows.

Discretization

Consider a grid of N points in $\mathbb{R}$ with uniform grid spacing $h = \frac{1}{N+1}$. The second order differentiation operator $\frac{d^2}{dx^2}$ can be approximated at each point by the finite difference scheme

$$u''(x) \approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}$$

with truncation error $O(h^2)$. The associated system of linear equations is represented by the (negative-definite) matrix

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

The error analysis is easy in this case, since with a zero right hand side, the current iterate is the error; that is, $u^h = e^h$.

Basic Iterative Methods

The Gauss-Seidel iteration for a general linear system $Mx = b$ is given by

Algorithm 1 Gauss-Seidel Iteration

Given $M \in \mathbb{R}^{n\times n}$; $x^{(0)} \in \mathbb{R}^n$; $b \in \mathbb{R}^n$;
$x^{(k)} \leftarrow x^{(0)}$
for $k = 1, 2, \ldots$ **do**
  for $i = 1, 2, \ldots, n$ **do**
    $x_i^{(k+1)} = \frac{1}{M_{ii}}\left(b_i - \sum_{j=1}^{i-1} x_j^{(k+1)} M_{ij} - \sum_{j=i+1}^{n} x^{(k)} M_{ij}\right)$ (3.2)
  end for
end for

Formula (3.2) approximately solves the linear equation represented by the coefficients in the ith row of the matrix M for the ith unknown $x_i^{(k)}$ by using the current iterate as a guess for the values of the other variables. The Gauss-Seidel iteration can also be expressed

using the matrix splitting $M = U + L + D$, where U is the upper-triangular part of M, L is the lower-triangular part, and D is the diagonal part. The resulting formula is given by $x^{(k+1)} = (D + L)^{-1}\left(b - Ux^{(k)}\right)$.

It can be shown that iterative methods for solving a linear system $Mx = b$ that can be written in the form

$$x^{(k+1)} = Rx^{(k)} + d, \tag{3.3}$$

where x solves $x = Rx + b$ if and only if $Mx = b$, converge for all initial iterates and all data d if and only if the spectrum of R is within the unit circle, i.e., $\rho(R) < 1$.

Local Fourier Analysis

In this section we discuss the smoothing property through numerical examples before taking a more analytical approach to describing the action of Gauss-Seidel on an arbitrary vector. If the Gauss-Seidel iteration has the properties we claim, we expect oscillatory components to be quickly eliminated and smooth components to persist. We apply *local Fourier analysis* to distinguish between oscillatory and smooth components of a vector. To start, we take an initial iterate $u \in \mathbb{R}^n$ and express it as a linear combination of *Fourier modes*:

$$u_j = \sum_{k=0}^{n-1} c_k e^{\frac{2\pi ijk}{n}}, \qquad j = 0, 1, ..., n-1 \tag{3.4}$$

for some scalar coefficients $c_k \in \mathbb{C}$ (where i is the complex unit and the subscripts j represent components of u). This should be compared with infinite-dimensional Fourier analysis, where a function is locally written as a linear combination of countably many complex exponential functions. The $n \times n$ matrix with columns $\{e^{\frac{2\pi ijk}{n}}\}_{k=0}^{n-1}$ for $j = 0, 1, ..., n-1$ has orthogonal columns so the expansion (3.4) exists and is unique for any $u \in \mathbb{C}^n$ [Koh15]. We will refer to the number $k \in \{0, 1, ..., n-1\}$ associated with the Fourier mode $\{e^{\frac{2\pi ijk}{n}}\}_{j=0}^{n-1}$ as its *wave number*.

The advantage of describing a vector using Fourier modes is that we are able to distinguish between oscillatory ($n/4 \leq k \leq 3n/4$) and smooth components ($k < n/4$ or $k > 3n/4$). For example, taking $c_k = 0$ except when $k = 1$ in the previous equation, we have $u_j = e^{\frac{2\pi ij}{n}}$ and one sees that the real and imaginary components of u are smooth, in the sense that their values do not fluctuate wildly from one grid point to the next. On the other hand, if only $c_{n/2}$ is nonzero, then the vector u is extremely oscillatory. In this analysis, the reader should note the that the descriptors "smooth" and "oscillatory" are largely grid dependent. If we think of our vector as a discretization of a function (which we should), then a function that seems relatively smooth on one grid may appear highly oscillatory on another.

Example 3.1 (Gauss-Seidel smoothing)**.** For a concrete example, we apply the Gauss-Seidel iteration to the vectors $v^{(1)}$, $v^{(2)}$, and $v^{(3)}$ defined componentwise by

$$v_j^{(1)} = \sin\left(\frac{j\pi}{12}\right), \qquad v_j^{(2)} = \sin\left(\frac{3j\pi}{12}\right), \qquad v_j^{(3)} = -\sin\left(\frac{11j\pi}{12}\right),$$

and the vector $v^{(4)}$ which is a linear combination of the first three:

$$v^{(4)} = \frac{1}{3}\sum_{i=1}^{3} v^{(i)}.$$

Figures 3.1 and 3.2 plot these vectors on a 13-point grid (black). Note that while $v^{(1)}$, $v^{(2)}$, and $v^{(3)}$ correspond to imaginary parts of our original Fourier modes, we have dropped a factor of two in the numerator inside the sine functions which has the effect of shifting oscillatory modes to the top half of the spectrum. Larger wave numbers then correspond to more oscillatory Fourier modes. For example, the highly oscillatory behavior of the grid function $v^{(3)}$ is apparent (on this grid). Fluctuations in its value from one grid point to the next are proportional to its infinity norm $\left\|v^{(3)}\right\|_\infty$.

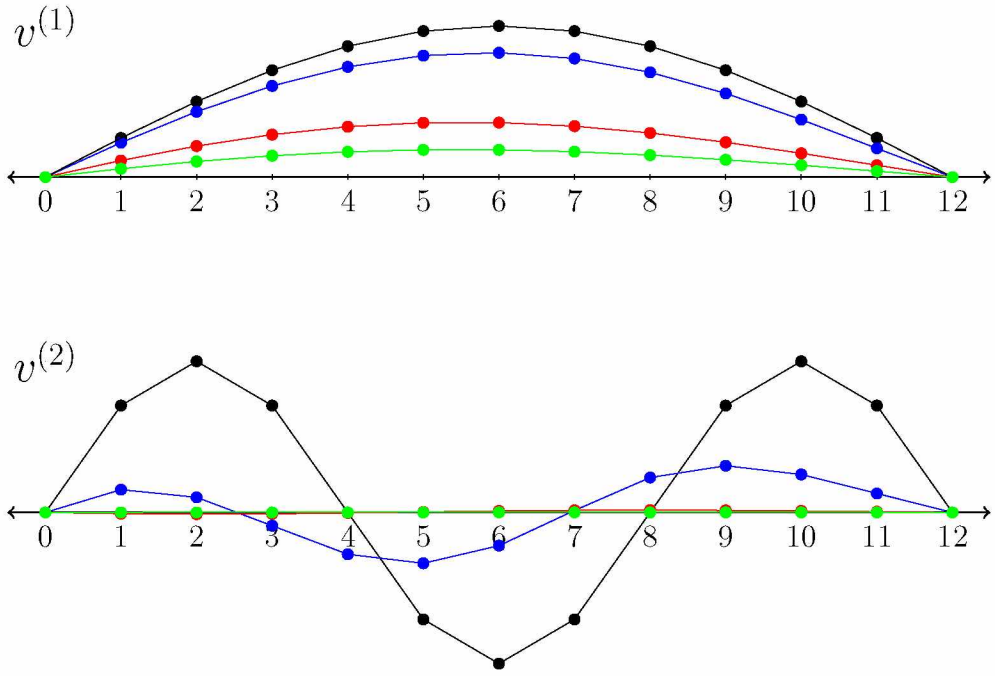


Figure 3.1: Imaginary parts of discrete Fourier modes for the grid $\{0, 1, ..., 11, 12\}$ with wave numbers $k = 1$ (top) and 3 (bottom). Higher wave numbers k correspond to more oscillatory modes. Superimposed are the grid functions after 3 iterations (blue), 15 iterations (red), and 25 iterations (green) of Gauss-Seidel relaxation on the equation $Au = 0$. Relaxation is less effective on the smooth modes pictured here.

Superimposed on the vectors $v^{(i)}$ in Figures 3.1 and 3.2 are the smoothed functions after 3 iterations (blue), 15 iterations (red), and 25 iterations (green). Table 3.1 tracks the ∞-norm of the grid functions shown in Figures 3.1 and 3.2. The most oscillatory vector, $v^{(3)}$, is reduced in ∞-norm by nearly two orders of magnitude after two Gauss-Seidel sweeps while the smooth modes, $v^{(1)}$ and $v^{(2)}$, are eliminated much more slowly. Comparing the linear combination vector $v^{(4)}$ with $v^{(1)}$, we can see that the remaining error after 25 GS sweeps (green on the first and last plots) for $v^{(4)}$ is largely due to the original smooth component $v^{(1)}$.

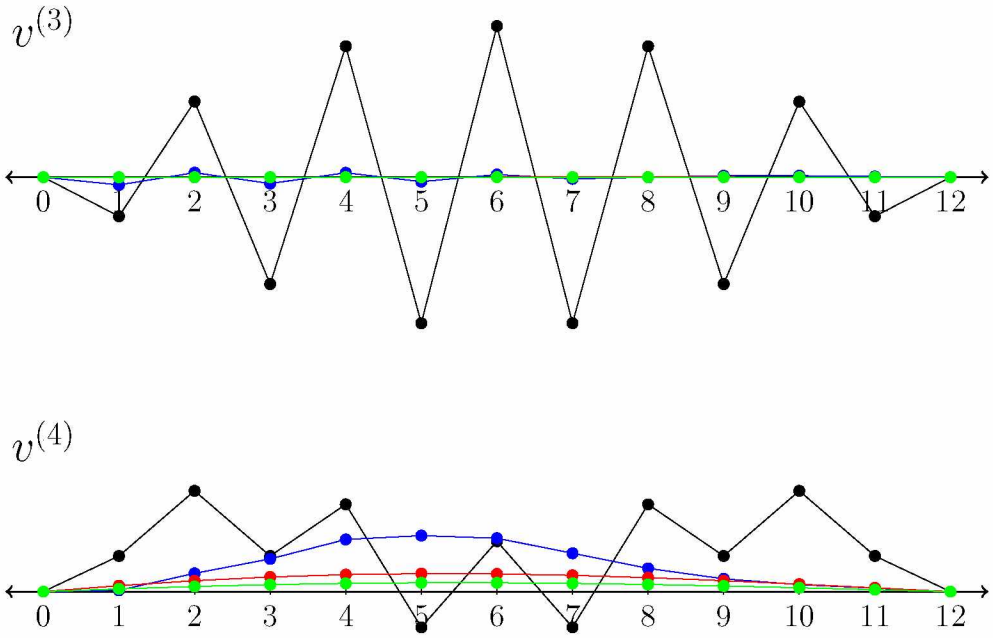


Figure 3.2: Imaginary part of discrete the 11th Fourier mode $v^{(3)}$ (top) and the linear combination of Fourier modes $v^{(4)}$ (bottom) on the grid $\{0, 1, ..., 11, 12\}$. Superimposed are the grid functions after 3 iterations (blue), 15 iterations (red), and 25 iterations (green) of Gauss-Seidel relaxation on the equation $Au = 0$. Relaxation is more effective on the oscillatory mode $v^{(3)}$, while smooth modes are more slowly eliminated from $v^{(4)}$.

Table 3.1: Infinity norm of the vectors plotted in (3.1) and (3.2). Relaxation is less effective on the smooth modes: 25 Gauss-Seidel sweeps reduces $\left\|v^{(1)}\right\|_\infty$ by only one order of magnitude.

	$\|\|v^{(1)}\|\|_\infty$	$\|\|v^{(2)}\|\|_\infty$	$\|\|v^{(3)}\|\|_\infty$	$\|\|v^{(4)}\|\|_\infty$
initial	1.0	1.0	1.0	$8.8 \cdot 10^{-1}$
3 iters	$8.2 \cdot 10^{-1}$	$3.4 \cdot 10^{-1}$	$5.1 \cdot 10^{-2}$	$3.7 \cdot 10^{-1}$
15 iters	$3.6 \cdot 10^{-1}$	$1.5 \cdot 10^{-2}$	$5.4 \cdot 10^{-4}$	$1.2 \cdot 10^{-1}$
25 iters	$1.8 \cdot 10^{-1}$	$5.1 \cdot 10^{-3}$	$2.2 \cdot 10^{-4}$	$5.9 \cdot 10^{-2}$

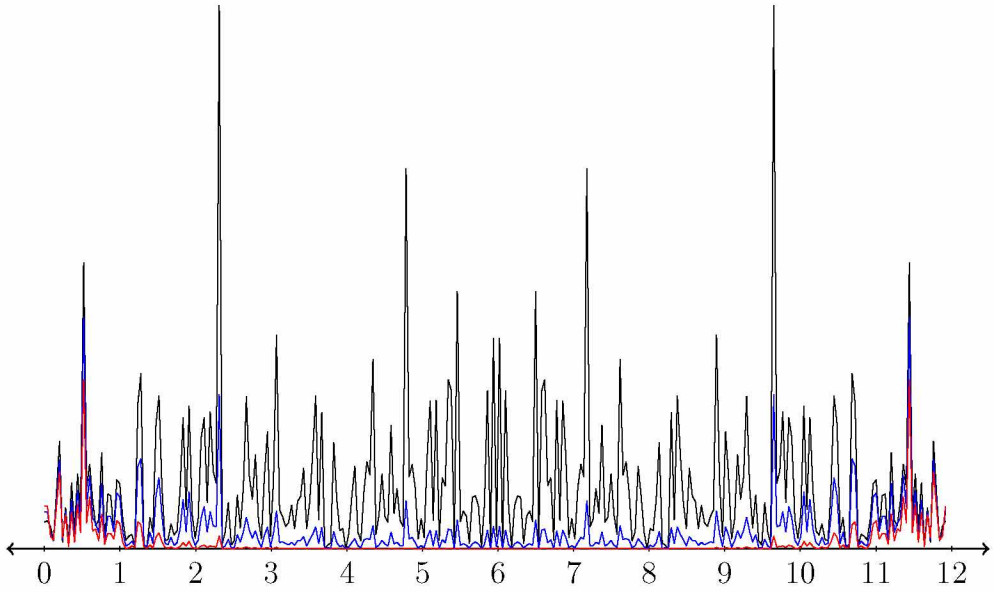


Figure 3.3: The coefficients c_k in the discrete Fourier expansion for a random initial vector u after zero (black), one (blue), and four (red) Gauss-Seidel relaxations. The components with wave numbers toward the middle of the spectrum (oscillatory modes) are quickly eliminated, while the smooth modes near the extremes persist.

We can compute the coefficients c_k in the discrete Fourier expansion for any vector using a matrix inversion, which we have done here for a Gaussian white noise vector u, shown in black in Figure 3.3. Also shown are the Fourier coefficients of u after one (blue) and four (red) iterations of Gauss-Seidel smoothing. The oscillatory modes, which correspond to wave numbers on the middle of the spectrum, are quickly eliminated, while the smooth modes toward the extremes persist, hardly reduced at all from their original magnitude.

Taking a more analytical approach to local Fourier analysis, the Gauss-Seidel iteration applied to a linear equation with zero right-hand side is a power iteration on an initial iterate $v^{(0)}$ using the matrix $R = (D + L)^{-1}U$. We define the *asymptotic convergence rate* of a relaxation scheme with iteration matrix R to be the number of iterations required to reduce the error of the equation $Au = f$ by one digit, given by the formula $-\log(\rho(R))^{-1}$. This formula is derived by observing that the reduction in error norm after m iterations is related to $\rho(R)$ by

$$[\rho(R)]^m \sim C\frac{\left\|e^{(m)}\right\|}{\left\|e^{(0)}\right\|}, \qquad m \to \infty.$$

We might anticipate the damping that occurs if the Fourier modes are eigenvectors of the Gauss-Seidel iteration matrix. In fact, the Fourier modes (3.4) are eigenfunctions of any matrix arising from, say, the discretization of an elliptic PDE with constant coefficients and

periodic boundary conditions. The Gauss-Seidel iteration matrix in our model problem is not itself the discretization of an elliptic operator, but, as we will see shortly, the Fourier modes $\left[e^{\frac{2\pi ijk}{n}}\right]_{j=0}^{n-1}$ are eigenfunctions of this matrix. The corresponding eigenvalues λ_k determine how much the kth component in the Fourier expansion is damped, so we expect smooth modes to have large eigenvalues relative to oscillatory modes. The maximum of the eigenvalues corresponding to oscillatory modes (recall that these occur in the middle of the spectrum), is called the *smoothing factor of the iteration* and is denoted by

$$\mu_{\text{loc}}(R) = \max_{\frac{n}{4}\leq k\leq \frac{3n}{4}} |\lambda_k|$$

where R is the iteration matrix.

For example, the difference equations corresponding to the matrix A from our model problem are given by the formula

$$\frac{1}{h^2}\left(u_{\ell-1}^{(k+1)} - 2u_{\ell}^{(k+1)} + u_{\ell+1}^{(k)}\right) = 0, \tag{3.5}$$

where subscripts represent vector components and superscripts give the iteration number for Gauss-Seidel. The action of the Gauss-Seidel matrix can be described locally by solving this equation for $u_\ell^{(k)}$. One may use also use the splitting $D + L + U$ to write this equation in the intermediate form $(D + L)u^{(k+1)} = Uu^{(k)}$. The eigenvalues of $R = (D + L)^{-1}U$ can be found by substituting an exponential $u_\ell^{(k)} = \mu^\ell$ into (3.5). The result is

$$\mu^{\ell+1} = \lambda\left(2\mu^\ell - \mu^{\ell-1}\right), \qquad \lambda = \frac{\mu}{2-\mu^{-1}}.$$

In particular, for $\mu_k = e^{\frac{2\pi ik}{n}}$, i.e. the kth Fourier mode, we have

$$|\lambda_k| = \left|\frac{e^{\frac{2\pi ik}{n}}}{2 - e^{\frac{-2\pi ik}{n}}}\right| \leq 1.$$

By eliminating ℓ, we have shown that these really are eigenvalues of the iteration. The eigenvalues are important in the sense that applying the iteration to Fourier modes on the interior nodes damps (or magnifies, if $|\lambda_k| > 1$) errors uniformly across components by a factor of λ_k. For $n/4 \leq k \leq 3n/4$, we compute $|2 - e^{\frac{-2\pi ik}{n}}| \geq \sqrt{5}$, and hence $|\lambda_k| \leq \frac{1}{\sqrt{5}}$. The bound is achieved at $\frac{2\pi k}{n} = \frac{\pi}{2}$, for example. Note that the iteration does not converge for all starting vectors since $\lambda_0 = 1$.

In the remainder of this paper, we will be mostly working with the two-dimensional Laplace equation $u_{xx} + u_{yy} = 0$ on a rectangular domain $\Omega \subseteq \mathbb{R}^2$, with the matrix equation

given by the finite-difference discretization

$$\frac{u(x-h_x,y)-2u(x,y)+u(x+h_x,y)}{h_x^2}+\frac{u(x,y-h_y)-2u(x,y)+u(x,y+h_y)}{h_y^2}=0$$

and the lexicographical ordering of unknowns (details in Chapter 5). It can be shown using a similar analysis that the Gauss-Seidel iteration matrix R associated with this discretization has $\mu_{\text{loc}}(R)=\frac{1}{2}$ [TSO01]. It should be noted that in this case and for higher dimensional analogues, the smoothing factor μ_{loc} is bounded independent of the grid spacing h. This does not hold in general [TSO01].

3.1.2 Coarse Grid Correction

The coarse grid correction is the second major component of a multigrid algorithm after the smoother. The coarse grid correction works in complement with the smoother with the following principle in mind:

An iterative method whose convergence toward a solution of a problem stalls is no longer efficiently capturing the essential features of the original problem.

This hints toward Achi Brandt's "golden rule" of computational mathematics [BEL11]:

"The amount of computational work should be proportional to the amount of real physical changes in the computed solution."

Concretely, when we apply an iterative method to solve a linear system, we have observed that while the errors decrease slowly in a global sense, the oscillatory components are rapidly damped. As a result of smoothing, grid points become more similar to their neighbors. Hence the amount of information needed to describe the system is reduced. Defining a coarser grid to be a (strict) subset G^H of G^h (so $H>h$), we see that a smooth error can be transferred to a coarser grid without losing much in translation.

Now the question is, will the error be reduced on a coarse grid? Thinking back, the answer again follows from our discussion of smoothers. These methods stall when the error is too smooth, i.e., grid points are too closely approximated by their neighbors. Transferring a smooth error to a coarse grid causes it to become more oscillatory and hence susceptible to relaxation methods.

In a multigrid framework, the grid G^h is said to be finer than G^H if $G^H\subseteq G^h$. Passing from G^H to G^h is then referred to as a refinement while passing from G^h to G^H is referred to as a coarsening. Although these terms are relative, we sometimes refer to G^h as the fine

grid and G^H as the coarse grid in the context of a particular algorithm. Using this language, the following recursive algorithmic framework emerges for the solution of a linear system $A^h u^h = 0$ (e.g. the model problem):

Algorithmic Framework 2 Coarse grid correction + smoothing (`CGCSm`)

u^h, an initial approximation
while iteration has not stalled **do**
    relax on $A^h u^h = 0$
end while
transfer u^h to coarse grid
if grid is coarsest permissible **then**
    transfer u^h to fine grid
    relax on $A^h u^h = 0$
else
    apply `CGSm` to current approximation u^h on the coarse grid
end if

This model will be fleshed out more concretely before we proceed with a full analysis of a multigrid algorithm. We have a number of practical issues to address. We first discuss the problem of how to translate grid functions between grids.

Grid Transfers

The transfer from a fine grid to a coarse grid is done by an *restriction operator* R. Similarly, a *prolongation* or *interpolation operator* P is used to transfer from a coarse grid to a fine grid. If the grids have uniform spacing, that is, $h = h_x = h_y$, then the prolongation operator is sometimes denoted P_H^h and the restriction operator is written R_h^H, where h is the fine grid spacing and H is the coarse grid spacing. That is, if v^h is a fine grid function, then the operation $R_h^H v^h = v^H$ transfers the vector v^h on the grid G^h to the grid G^H. Applying the prolongation operator $P_H^h v^H = v^h$ transfers a coarse grid function to the fine grid. A standard method of coarsening is by doubling the grid spacing on the coarse grid, so that $H = 2h$.

The coarsening operator depends closely on the choice of the coarse grid. The canonical method of restriction is called injection, which preserves fine grid function values exactly. That is, for any $\mathbf{x}$, we have

$$v^H(\mathbf{x}) = R_h^H v^h(\mathbf{x}) = v^h(\mathbf{x}).$$

On a one-dimensional grid with fine grid spacing $h_x = h_y = h$, coarse grid spacing $H = 2h$, and a coarsening scheme that preserves every other grid point, this means that $v_j^{2h} = v_{2j}^h$ for $j = 0, 1, 2, ..., n$. This scheme is depicted in Figure 3.4 (below).

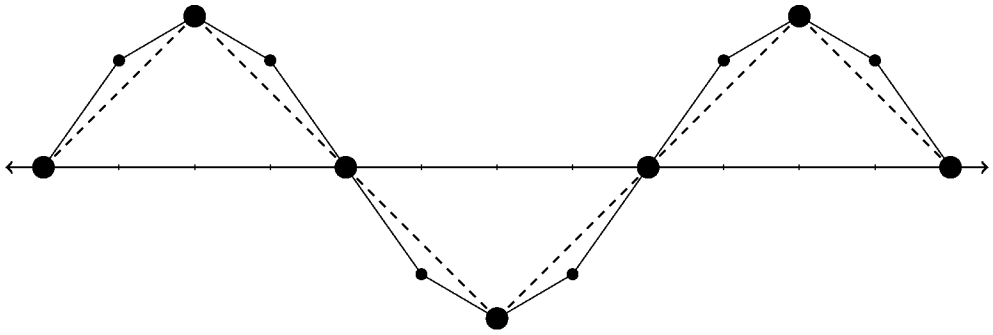


Figure 3.4: v^h, the fine grid representation of v (solid), superimposed on its coarse grid counterpart, v^{2h} (dashed). Here v^h has been transferred to the coarse grid by injection, and hence the function values at every other node have been preserved.

The other commonly used method is called full weighting, which involves writing the node v_j^h as a weighted average of its neighbors. In one dimension, the full-weighting formula is given by

$$v_j^{2h} = \frac{v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}}{4}.$$

In two dimensions, the formula is similar, except that 8 points are used in the linear combination: v_{2j}^{2h}, and its neighbors above, below, and diagonal.

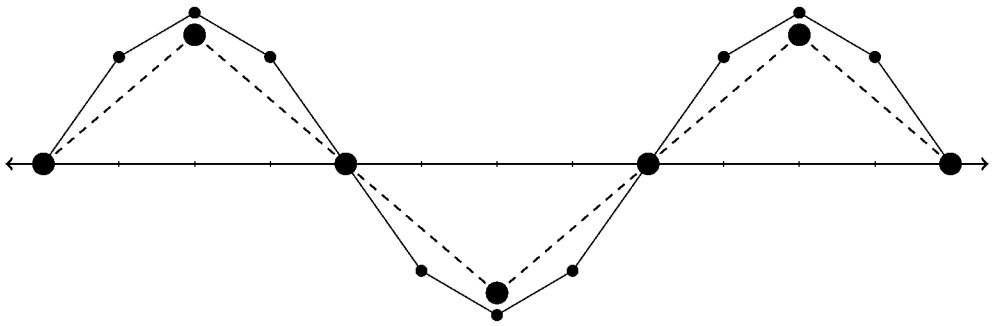


Figure 3.5: v^h, the fine grid representation of v (solid), superimposed on its coarse grid counterpart, v^{2h} (dashed), the coarse grid representation of v induced by full weighting. Function values are not necessarily preserved.

The coarse grid representations produced by full-weighting and injection are nearly identical in this case.

For the interpolation operator, there is no canonical method. In practice, a common choice is linear interpolation in 1D or bilinear interpolation in 2D. Both methods involve an injection for the coarse grid points and a weighted average for others.

Coarse grid correction as an iterative method

We have now seen the basics of how one might transfer a vector between a coarse grid and a fine grid. The goal we have in mind is the implementation of the ideas that led to our formulation of **Algorithmic Framework 2** – that is, combining relaxation methods with grid transfer operators to take full advantage of the smoothing property without running afoul of the poor asymptotic convergence results encountered in our discussion of local Fourier analysis. The key insight that supported that framework was the fact that smooth errors can be accurately represented on a coarse grid. As we did with the relaxation schemes, we now wish to take a step back and examine the coarse grid correction as an iteration independent of other machinery like smoothers.

Suppose we have a discrete linear elliptic boundary value problem of the form

$$A^h u^h = f^h$$

where A^h is invertible. The model problem would be one example, although the discussion that follows is widely applicable. Recall that solving the linear equation listed above is equivalent to solving the error equation (3.1).

The idea of the coarse grid correction is to transfer the residual to the coarse grid, solve the error equation to to approximate the coarse grid error, return our approximation to the fine grid, and correct the previous guess with the old error. Concretely, we start with initial data

$$\begin{aligned} A^h &: \text{fine grid operator} \\ A^H &: \text{coarse grid operator} \\ v^h &: \text{initial guess} \\ R_h^H &: \text{restriction operator (fine to coarse transfer)} \\ P_H^h &: \text{interpolation operator (coarse to fine transfer).} \end{aligned}$$

We compute the residual

$$r^h = f^h - Av^h,$$

and transfer to the coarse grid to find

$$r^H = R_h^H r^h.$$

We then solve the coarse grid equation

$$A^H e^H = r^H$$

and transfer the result back to the fine grid

$$e^h = P_H^h e^H.$$

Our new guess to the solution of the original system $A^h u^h = f^h$ is then given by

$$w^h = e^h + v^h.$$

Repeated iteratively, this method has the form of (3.3), and the iteration matrix is

$$I_h - P_H^h (A^H)^{-1} R_h^H A^h,$$

where I_h is the identity matrix on the grid G^h. Since $R_h^H \in \mathbb{R}^{N(h) \times N(2h)}$ has rank of at most $N(2h) < N(h)$, where $N(h)$ denotes the number of grid points with spacing h, the matrix $P_H^h (A^H)^{-1} R_h^H A^h$ has nontrivial kernel. Hence

$$\rho\left(I_h - P_H^h (A^H)^{-1} R_h^H A^h\right) \geq 1,$$

and there is no hope of the coarse grid correction converging on its own, despite the potentially large computational speed-ups from implementing the scheme recursively. Fortunately, we can return to our original goal of combining the coarse grid correction and relaxation in a single algorithm.

3.1.3 Multigrid Cycles

We return to the original idea of a multigrid cycle. The two grid correction scheme, as we outlined previously in **Algorithmic Framework 2**, is stated formally and completely in **Algorithm 3** on the next page. The algorithm is called a V-cycle because of its V-shaped trajectory between the coarse grid and the fine grid, pictured in Figure 3.6 (below). This is the simplest, but not the only choice for a multigrid cycle; the W-cycle and F-cycle are also common.

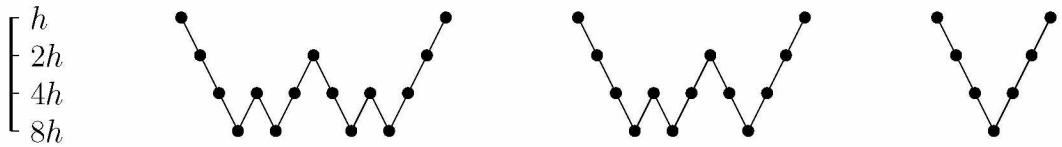


Figure 3.6: From left to right: The W-cycle, F-cycle, and V-cycle for a four grid solver

Algorithm 3 Multigrid V-cycle (`MGV`)

`MGV`$\left(u^h, G, G_A, G_R, G_P, f^{h_1}, \mathcal{S}, \texttt{linear\_solver}, \nu^0, \nu^1\right)$

Parameters: initial guess v_0^h

sequence of grids $G = \left\{G^{h_1}, ..., G^{h_\ell}\right\}$ with uniform spacing $h_1 < ... < h_\ell$

sequence of operators $G_A = \left\{A^{h_1}, ..., A^{h_\ell}\right\}$

Sequences of restriction operators and prolongation operators

$$G_R = \left\{R_{h_i}^{h_{i+1}}\right\}_{i=1}^{\ell-1}, G_P = \left\{P_{h_{i-1}}^{h_i}\right\}_{i=2}^{\ell}$$

Fine grid right-hand side function f^{h_1}

Smoother $\mathcal{S} = \mathcal{S}(x, A, b)$

`linear_solver`(A, b), coarse grid solver

$\nu^0, \nu^1 \in \mathbb{N}_0$: number of pre- and post-smoothing steps

$h = h_1$

for $j = 1, 2, ..., \nu^0$ **do**

  $u^h = \mathcal{S}(u^h, A^h, f^h)$ (ν^0 smoothing steps)

end for

$r^h = f^h - A^h u^h$ (compute the residual)

if $|G| = 1$ **then**

  $e^h =$ `linear_solver`(A^h, r^h) (solve the coarse grid system for error e^h)

  $e^H = P_h^H e^h$, where $G_P = \left\{P_h^H\right\}$ (interpolate the error back to the fine grid)

  $u^H = u^H + e^H$ (coarse grid correction)

  return u^H

else (solve for the error on the coarser grid G^H by recursive call to `MGV` w/ `0` initial guess)

  $H = h_2$

  $r^H = R_h^H r^h$

  $u^H =$ `MGV`$(0, G', G'_A, G'_R, G'_P, r^H, \mathcal{S}, \texttt{linear\_solver}, \nu^0, \nu^1)$

    where $G' = \left\{G^{h_2}, ..., G^{h_\ell}\right\}$, $G'_A = \left\{A^{h_2}, ..., A^{h_\ell}\right\}$, and so on

  for $j = 1, 2, ..., \nu^1$ **do**

    $v^H = \mathcal{S}(u^H, A^H, r^H)$ (ν^1 smoothing steps)

  end for

  return u^H

end if

Usually, the V-cycle is first described in a two-grid framework similar to the one discussed in the previous section on the coarse grid correction, with the addition of smoothing steps before (called *pre-smoothing* steps) and after (called *post-smoothing* steps) grid transfers. The extension to multiple grids from the two grid framework occurs in our choice of linear solver: we solve the error equation on the second grid by a recursive call to the multigrid V-cycle, and do not call a direct solver until we reach the coarsest grid.

In any of the multigrid solvers mentioned, we have choices to make. At a minimum, the number of pre- and post-smoothing steps, the linear solver on the coarse grid, the prolongation and restriction operators, the grid spacing $\{h_1, ..., h_\ell\}$ and number of grids, and the smoother are all chosen by the user. Our choices are also determined in part by our particular discretization scheme. For example, if we are working on a box grid embedded in $\mathbb{R}^d$, we can choose to use nonuniform spacing in our d directions, so that $h_i = \left(h_i^{(1)}, ..., h_i^{(d)}\right)$ for $i = 1, 2, ..., \ell$. In a finite elements discretization, we may have totally nonuniform spacing. We may also make a choice of coarse grid operators – the canonical choice is to choose A^H as the discrete operator on the coarse grid; however, other choices are also effective in practice, including the *Galerkin operator* defined by $A^{h_{i+1}} = R_{h_i}^{h_{i+1}} A^{h_i} P_{h_i}^{h_{i-1}}$, the canonical choice in algebraic multigrid.

It is also possible to vary the number of multigrid sweeps done on each grid. For example, the F-cycle is obtained by doing an extra V-cycle on each grid. Hence, one can also construct cycles of increasing complexity simply by varying the recursive structure.

Algorithm 3 has iteration matrix M_ℓ^V given by the recursion

$$\begin{aligned} M_0^V &= 0 \\ M_k^V &= S^{\nu^1}\left[I_k - P_{h_{k-1}}^{h_k}\left(I_{k-1} - M_{k-1}^V\right)\left(A^{h_{k-1}}\right)^{-1} R_{h_k}^{h_{k-1}} A^{h_k}) S_k^{\nu^0}\right], \qquad k = 1, 2, ..., \ell, \end{aligned} \tag{3.6}$$

where S_k is the iteration matrix corresponding to the relaxation determined by A^{h_k} and $\mathcal{S}$. Similarly, the iteration matrix for the F-cycle M_ℓ^F is given by the recursion

$$\begin{aligned} M_0^F &= 0 \\ M_1^F &= M_1 \text{ (from the recursion above)}, \\ M_k^F &= S^{\nu^1}\left[I_k - P_{h_{k-1}}^{h_k}\left(I_{k-1} - M_{k-1}^V M_{k-1}^F\right)\left(A^{h_{k-1}}\right)^{-1} R_{h_k}^{h_{k-1}} A^{h_k}) S_k^{\nu^0}\right], \qquad k = 2, ..., \ell. \end{aligned} \tag{3.7}$$

Computational Efficiency

As we alluded to earlier, the main impetus behind using a multigrid algorithm is *optimality* – the amount of work the solver does is "proportional to the amount of real physical changes in the computed solution." For one thing, this means that a multigrid solver requires

a fixed number of iterations independent of the grid spacing h on the finest grid. That is, a multigrid solver does work on the order of $O(N)$ as the grid is refined *but* the number of iterations required for convergence is fixed. This property stands in contrast with other iterative matrix methods: for example, the Gauss-Seidel method does $O(N)$ work on each sweep (assuming constant sparsity, i.e., the number of nonzeros in the iteration matrix is $O(1)$, as in the discretization of most PDEs), but typically requires an additional $O(N)$ iterations to converge.

Of course, the number of grids varies between implementations, and hence the method of refinement matters. If we assume uniform spacing and the grids are refined by a factor of two with a coarse grid spacing of h, so that an n-grid implementation of **Algorithm 3** has grids $G_n = \left\{G^h, G^{2h}, ..., G^{2^n h}\right\}$, then we can demonstrate optimality, since the amount of work W_n on each iteration is proportional to

$$W_n \propto \sum_{k=1}^{n} \frac{N}{2^k} + C < N + C,$$

where C is the work done on the coarse grid. Similar results hold for more general multigrid cycles with standard coarsening. We emphasize that the method of coarsening may change these estimates, but under the mild assumption that the coarsening factor on each iteration is bounded by a constant factor $r > 1$ then the amount of work per level is bounded by a geometric series in N and is always $O(N)$.

The essential idea to prove h-independent multigrid convergence factors is to first bound the two grid multigrid operators, and with this bound in hand use the recursive nature of the multigrid algorithm to bound a general multigrid operator like those given by the recursions (3.6) and (3.7). These proofs are simple and are covered in Chapters 2 and 3 of [TSO01].

Nonlinear Equations

For a nonlinear system of equations of the form $A(u) = f$, the coarse grid correction is not effective because the error equation does not hold. That is, we may still define the residual

$$r^h = f^h - A^h(v^h),$$

but the relation

$$A^h(e^h) = r^h$$

is false. One option is to expand $A^h(e^h)$ in Taylor series for a local linearization, solve the resulting linear system for the error using multigrid, and correct the error for a new approximation. This approach is called Newton-multigrid.

Another idea is to restrict the residual to the coarse grid G^H as before, yielding the equation

$$s^H := R_h^H r^h = R_h^H(f^h - A^h(u^h)) = f^H - R_h^H A^H(u^h) = A^H(\bar{u}^H) - R_h^H A^h(u^h),$$

where we set $s^H := R_h^H r^h$ to avoid confusion with the true coarse grid residual, $r^H = f^H - A^H(u^H)$.

The resulting equation

$$A^H(\bar{u}^H) = s^H + R_h^H A^h(u^h) \tag{3.8}$$

can then be iteratively solved for $\bar{u}^H$, the coarse grid error approximated by $e^H = \bar{u}^H - u^H$, the error estimate transferred back to the fine grid and corrected by the formula

$$u_{\text{new}}^h = P_H^h e^H + u^h = P_H^h \bar{u}^H - P_H^h u^H + u^h.$$

Rewriting (3.8) in terms of the restriction operator R_h^H gives us

$$A^H(\bar{u}^H) = R_h^H\left(f^h - A^h(v^h)\right) + A^H(R_h^H u^h) = f^H + A^H(R_h^H u^h) - R_h^H A^h(u^h),$$

where the term

$$\tau_h^H = A^H(R_h^H u^h) - R_h^H A^h(u^h)$$

is sometimes called the τ correction. The goal of solving (3.8), rather than directly solving the coarse grid system $A^H(u^H) = f^H$, is to achieve a truncation error for the coarse grid solutions rivaling that on the fine grid (albeit at a lower resolution). One way to see that this is achieved is through the *fixed point property* of this scheme – the exact solution u^h is a fixed point of the iteration, a desirable attribute of any iterative method. Substituting $\bar{u}^h$ for u^h in (3.8) yields

$$A^H(\bar{u}^H) = f^H + A^H(R_h^H \bar{u}^h) - R_h^H A^h(\bar{u}^h) = A^H(R_h^H \bar{u}^h),$$

with the solution $\bar{u}^H = R_h^H \bar{u}^h$. For the coarse grid error we have $e^H = 0$, and our fine grid estimate for the error is exact.

The algorithm described above reduces to the coarse grid correction in the case where A is linear. Combining this nonlinear version of the coarse grid correction with a (nonlinear) smoother is called the *full approximation scheme (FAS)*. An important thing to note is that

FAS does not correspond to a matrix iteration like (3.3). The behavior of FAS therefore cannot be understood by estimating the eigenvalues of a matrix as is possible for linear iterations like the multigrid V-cycle or F-cycle.

3.2 Algebraic Multigrid

Algebraic multigrid (AMG) can be broadly described as the class of algorithms produced by applying multigrid ideas to linear systems

$$Au = f$$

which are not necessarily defined on a grid. The goal of algebraic multigrid is the same as that for geometric multigrid: apply smoothers until convergence stalls and the error becomes smoother, then reduce the dimensionality of the problem, solve the error equation, and return to the fine grid. The tools are also the same: coarse grid correction and smoothers. The main difference lies in how we distinguish oscillatory errors and smooth errors and how we define a coarse grid.

The canonical methods described in previous sections for coarsening and smoothing proved ineffective for certain classes of problems beyond the type that we have been considering, namely constant coefficient elliptic PDEs. Specifically, PDEs with highly irregular coefficients or similar problems are not treated correctly by the usual relaxation methods or geometric interpolation operators. Algebraic multigrid is an extension of the ideas that arose from the discovery that purely geometric ideas were not sufficient for such problems.

Smoothing, coarsening, and prolongation for AMG

Algebraic multigrid remains most effective when applied to linear problems arising from PDEs, despite using no geometric information whatsoever. Moreover, from a theoretical standpoint algebraic multigrid solvers remain largely heuristic. That is, there are no nontrivial black box algebraic multigrid solvers that are proven to be robust. However, experience shows that AMG black box solvers are effective on a variety of problems, especially when the linear operator is symmetric positive definite (SPD).

In the case of GMG, oscillatory error could be analytically described as having large coefficients on high frequencies in its discrete Fourier expansion or intuitively as error that can be accurately represented on a coarse grid. Both of these descriptions rely on underlying geometric properties. However, we know that the purpose of moving errors to the coarse grid was to counteract the failure of relaxation methods on geometrically smooth errors.

Applying this idea, we define the error of the system of equation $Au = f$ to be *algebraically smooth* if relaxation no longer effectively reduces errors. We might hope that algebraically smooth error is always geometrically smooth, but this is not the case.

Coarsening, refining, and grid transfers also need to be defined for a non-geometric problem. For GMG, we coarsened the grid and hence reduced the dimensionality of the problem by taking a subset of unknowns deemed representative of the errors. In the purely algebraic case, we try to choose a subset of unknowns which are representative of the algebraic relationships between unknowns.

For a PDE, the value of a function and its derivatives at a grid point x_i are most strongly related to their values at grid points that are geometrically close to x_i. Typically, the ith linear equation relates the value of the ith unknown with the unknowns which are geometrically adjacent to it. In a typical finite difference scheme on a box grid, for example, a particular grid point is only directly determined by those directly next to it in the box structure.

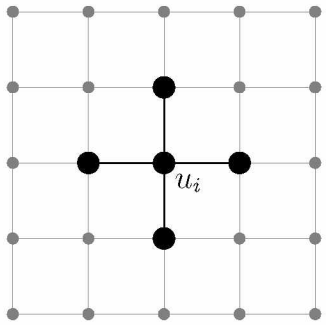


Figure 3.7: The adjacency diagram for a discrete PDE. The node u_i has strong algebraic connections to its nearest geometric neighbors.

In algebraic multigrid, we use the opposite idea; loosely speaking, given two unknowns u_i and u_j, we say u_j is a determinant of u_i ($u_j \sim u_i$) if there is a large nonzero coefficient in the A_{ij} position of the operator matrix A. These relationships can be represented in a graph. When the matrix A is a differential operator, geometric relationships are usually preserved algebraically as depicted in Figure 3.7. Prolongation operators can be derived algebraically as well.

For a concise introduction to AMG, we recommend [Fal06]. For a more substantial overview of the topic, see [TSO01], [BHM00], or [BEL11].

Chapter 4

Algorithms

4.1 Reduced Space Method

The reduced space method (RSP) [BM03] is a heuristic iterative approach to solving nonlinear complementarity problems (NCPs). Recall the definition of a nonlinear complementarity problem:

Definition 1. *Let $F : \mathbb{R}^n \to \mathbb{R}^n$. The nonlinear complementarity problem is to find $x \in \mathbb{R}^n$ satisfying*

$$0 \leq x \perp F(x) \geq 0.$$

The nonlinear complementarity problem associated with F is denoted $NCP(F)$.

Here we will focus on the special case where $F(x) = Mx + q$ is affine, called a linear complementarity problem:

Definition 2. *Let $M \in \mathbb{R}^{n \times n}$, $q \in \mathbb{R}^n$. The linear complementarity problem is to find $x \in \mathbb{R}^n$ satisfying*

$$0 \leq x \perp Mx + q \geq 0. \tag{4.1}$$

The linear complementarity problem associated with M and q is denoted $LCP(M, q)$.

On each iteration of the reduced space method, we compute an active set of indices

$$\mathcal{A}(x) = \{i : x_i = 0 \text{ and } F_i(x) > 0\}$$

from the current iterate x.

The inactive set of indices $\mathcal{I}(x)$ is the complement of $\mathcal{A}(x)$,

$$\mathcal{I}(x) = \{i : x_i > 0 \text{ or } F_i(x) \leq 0\}.$$

The next iterate is found by taking a Newton step in the subspace $\mathbb{R}^{|\mathcal{I}(x)|}$ defined by the inactive indices, called the reduced space. A submatrix $[\nabla F(x)]_{\mathcal{I}(x),\mathcal{I}(x)}$ of $\nabla F(x)$ is computed

by deleting rows and columns corresponding to the indices in the active set, and similarly $F(x)_{\mathcal{I}(x)}$ is formed by deleting elements in the positions of the vector $F(x)$ corresponding to the indices in $\mathcal{A}(x)$. The system

$$\left([\nabla F(x)]_{\mathcal{I}(x),\mathcal{I}(x)}\right)\delta_{\mathcal{I}(x)} = -[F(x)]_{\mathcal{I}(x)}$$

is solved or approximately solved for $\delta_{\mathcal{I}(x)}$, which gives the new search direction in the reduced space. A step size α is determined by a line search using the merit function $\|F_\Omega(x)\|_2$, where F_Ω defined component-wise by

$$[F_\Omega(x)]_i = \begin{cases} F_i(x) & x_i > 0 \\ \min\{F_i(x), 0\} & x_i = 0 \end{cases}. \tag{4.2}$$

If the line search fails in the Newton direction, the line search is repeated in the gradient descent direction $\delta = -F(x)$. In case of a second line search failure, the iteration is terminated.

The new iterate is finally

$$x^{\text{new}} = \pi(x + \alpha\delta),$$

where $\pi : \mathbb{R}^n \to \{x \in \mathbb{R}^n : x_i \geq 0\}$ is the projection onto the boundary of the feasible set, i.e.,

$$\pi(x) = \begin{cases} x_i & x_i \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

and δ represents the extension of the reduced space search direction $\delta_{\mathcal{I}(x)}$ to $\mathbb{R}^n$ by setting $\delta_{\mathcal{A}(x)} = 0$. Note that the projection ensures that every iterate satisfies the constraint $x_i \geq 0$ but not necessarily the constraint on $F(x)$. The merit function $\|F_\Omega(x)\|_2$ then measures the magnitude of the constraint violation by F and the violation of the complementarity condition.

The affine case is where $F(x) = Mx + q$, the only case considered in this project. Then $\nabla F = M$. Letting $\mathcal{I} = \mathcal{I}(x) \subseteq \{1, 2, ..., n\}$ and using the notation $\mathcal{I}_j$ for the jth largest element in $\mathcal{I}$, we have

$$\begin{aligned} [Mx]_{\mathcal{I}} &= P(Mx) \\ &= (PM)(P^\top Px) \\ &= (PMP^\top)(Px) \\ &= (M_{\mathcal{I},\mathcal{I}})\, x_{\mathcal{I}}, \end{aligned}$$

where $P \in \mathbb{R}^{|\mathcal{I}| \times n}$ is the matrix with rows $P_j = e_{\mathcal{I}_j}$, i.e., the jth row of P has a one in the entry corresponding to the jth largest element of $\mathcal{I}$ and zeros everywhere else. The equality $P^\top P x = x$ holds because $P^\top P$ has entries

$$(P^\top P)_{ij} = \begin{cases} 1 & i = j \text{ and } i \in \mathcal{I} \\ 0 & \text{otherwise.} \end{cases}$$

Hence

$$[P^\top P x]_i = \begin{cases} x_i & i \in \mathcal{I} \\ 0 & \text{otherwise.} \end{cases}$$

Since $x_i = 0$ if $i \notin \mathcal{I}$, it follows that $[P^\top P x]_i = x_i$ for all i.

Letting $x^{(k)}$ represent the current iterate, we set

$$\mathcal{I}^k = \mathcal{I}\left(x^{(k)}\right), \qquad x^{(k)} = x, \qquad x^{(k+1)} = x^{\text{new}}, \qquad \text{and} \qquad P^k = P \in \mathbb{R}^{|\mathcal{I}^k| \times n}$$

where P^k is the matrix associated with $\mathcal{I}^k$ described in the previous paragraph. The new iterate x^{new} before projection is found to be a convex combination of the old iterate x and the solution s of the reduced space equation $\left(-M_{\mathcal{I}^k,\mathcal{I}^k}\right) s = q_{\mathcal{I}^k}$:

$$\begin{aligned} \left(-M_{\mathcal{I}^k,\mathcal{I}^k}\right) \delta &= \left(M_{\mathcal{I}^k,\mathcal{I}^k}\right) x_{\mathcal{I}^k} + q_{\mathcal{I}^k} \\ \delta &= -x_{\mathcal{I}^k} - [M_{\mathcal{I}^k,\mathcal{I}^k}]^{-1} q_{\mathcal{I}^k} \\ x^{\text{new}} &= x + \alpha P^\top \delta = (1-\alpha)x + \alpha P^\top s. \end{aligned} \tag{4.3}$$

In particular, for a full Newton step ($\alpha = 1$), (4.3) becomes $x^{\text{new}} = P^\top s$.

The full algorithm for the affine case is presented below (Algorithm 4).

Algorithm 4 Affine reduced space method (`RSP`)

`RSP`$\left(x^{(0)}, M, q, \texttt{linear\_solver}, \texttt{tol}, \texttt{maxiters}, \beta, \sigma, \gamma\right)$
Parameters: initial guess $x^{(0)} \in \mathbb{R}^n$
$M \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$ (define the function $F(x) = Mx + q$)
Approximate linear solver `linear_solver` for calculating the Newton steps
Desired residual reduction `tol`
Maximum number of allowable iterations `maxiters`
Line search parameters (β, σ, γ)
for $j = 0, 1, 2, ..., \texttt{maxiters}$ **do**
$\mathcal{A}(x) = \{i : x_i = 0 \text{ and } [Mx + q]_i > 0\}$
$\mathcal{I}(x) = \{1, 2, ..., n\} \setminus \mathcal{A}(x)$
$\delta_{\mathcal{I}(x)} = \texttt{linear\_solver}(M_{\mathcal{I}(x),\mathcal{I}(x)}, q_{\mathcal{I}(x)})$
`fail = 0`
$\alpha = 1$
while $\|F_\Omega(\pi(x + \alpha\delta))\|_2 \geq (1 - \sigma\alpha) \|F_\Omega(x)\|_2$ **do**
if $\alpha < \gamma$ **then**
`fail` $=$ `fail` $+ 1$
if `fail` > 1 **then**
`Terminate line search`
end if
$\delta = -[Mx + q]$
$\alpha = 1$
else
$\alpha = \alpha \cdot \beta$
end if
$x^{(j+1)} = x^{(j)} + \alpha\delta$
end while
end for

There are a couple of comments to be made on Algorithm 4 specifically regarding the details of the line search:

- We use parameters $\gamma = 10^{-12}$, $\sigma = 10^{-4}$, and $\beta = .5$ as suggested in [BM03].
- If the line search fails, then we switch to the steepest descent direction setting $\delta = -[Mx + q]$. If it fails again, we terminate the line search.

A desirable attribute of the reduced space method is that the submatrices $M_{\mathcal{I}(x),\mathcal{I}(x)}$ remain symmetric and positive definite whenever the original matrix is. The matrix produced by discretizing the Poisson operator is SPD. To have any hope of an optimal `RSP` solver, the linear solver for the Newton steps must be optimal. Given our discussion from the previous chapter, the obvious candidate for computing the Newton steps are AMG solvers which are generally targeted at SPD systems.

4.2 Projected Full-Approximation Scheme

The projected full-approximation scheme (PFAS) extends multigrid ideas to linear complementarity problems arising from PDEs. The full algorithm applies the FAS correction scheme to solve $LCP(A^h, f^h)$ where $A^h u^h = f^h$ is the discretization of an elliptic differential equation $\mathcal{L}u = f$. Smoothing is done using the projected Gauss-Seidel (PGS) method, presented for the general $LCP(M, q)$ in Algorithm 5 (below).

Algorithm 5 Projected Gauss-Seidel Iteration

Given $M \in \mathbb{R}^{n\times n}$; $x^{(0)} \in \mathbb{R}^n$; $q \in \mathbb{R}^n$;
for $k = 1, 2, ...$ **do**
  for $i = 1, 2, ..., n$ **do**
    $x_i^{(k+1)} = \max\left\{\frac{1}{M_{ii}}\left(q_i - \sum_{j=1}^{i-1} x_j^{(k+1)} M_{ij} - \sum_{j=i+1}^{n} x^{(k)} M_{ij}\right), 0\right\}$
  end for
end for

The only modification to the Gauss-Seidel iteration is the projection step. Like relaxation methods for linear problems, projected Gauss-Seidel converges to a solution of the LCP [BC83]. As noted in [BC83], however, it typically settles onto positive values of both the objective function and the iterate rather quickly and behaves thereafter like classical Gauss-Seidel. In particular, we expect the iteration to stall after the error becomes smooth, in which case it can be accurately transferred to the coarse grid. We should note that the constraint makes the problem nonlinear, and hence FAS is used rather than usual linear multigrid solvers.

It was observed in [BC83] that the convergence of PFAS is typically roughly twice as slow as that of FAS. The main difficulty occurs in locating the discrete free boundary Γ^k, which is typically $O(h_k)$ from the true free boundary Γ *and* $O(h_k)$ from Γ^{k-1}.

Algorithm 6 PFAS V-cycle (`PFASV`)

`PFASV`$\left(v_0^h, G, G_A, G_R, G_P, f^{h_1}, \texttt{coarse\_solver}, \nu^0, \nu^1\right)$
Parameters: initial guess v_0^h
sequence of grids $G = \left\{G^{h_1}, ..., G^{h_\ell}\right\}$ with uniform spacing $h_1 < ... < h_\ell$
sequence of operators $G_A = \left\{A^{h_1}, ..., A^{h_\ell}\right\}$
Sequences of restriction operators and prolongation operators

$$G_R = \left\{R_{h_i}^{h_{i+1}}\right\}_{i=1}^{\ell-1}, G_P = \left\{P_{h_{i-1}}^{h_i}\right\}_{i=2}^{\ell}$$

Fine grid right-hand side function f^{h_1}
`coarse_solver`(A, b), coarse grid solver
$\nu^0, \nu^1 \in \mathbb{N}$: number of pre- and post-smoothing steps
$h = h_1$, $v^h = v_0^h$, $H = h_2$
for $j = 1, 2, ..., \nu^0$ **do**
  $v^h =$ `PGS`(v^h, A^h, f^h) (ν^0 smoothing steps)
end for
$r^h = f^h - A^h v^h$ (compute the residual)
$r^H = R_h^H r^h$ (transfer residual to coarse grid)
$v^H = R_h^H v^h$ (transfer current iterate to coarse grid)
$r^H = r^H + A^H v^H$ (correct residual)
if $|G| = 2$ **then**
  $v^H =$ `coarse_solver`(A^H, r^H) (solve the coarse grid LCP)
else (solve LCP on the coarser grid G^H by recursive call to `PFASV`)
  $v^H =$ `PFASV` $\left(v^H, G, G'_A, G'_R, G'_P, f^H, \texttt{coarse\_solver}, \nu^0, \nu^1\right)$
    where $G' = \left\{G^{h_2}, G^{h_3}, ..., G^{h_\ell}\right\}$, $G'_A = \left\{A^{h_2}, ..., A^{h_\ell}\right\}$, and so on
  for $j = 1, 2, ..., \nu^1$ **do**
    $v^H =$ `PGS`(v^H, A^H, f^H) (ν^1 smoothing steps)
  end for
end if
$v^h = v^h + P_H^h(v^H - R_h^H v^h)$ (coarse grid correction)
return v^h

Chapter 5

Numerical implementation

In this section, we present a general obstacle problem solver constructed in Python with `scipy` sparse linear algebra for the numerical analysis of the algorithms discussed in the previous chapter. It is worth, for the sake of having the notation handy, recalling the original statement of the obstacle problem from Chapter 2:

$$\min_{v \in H^1(\Omega)} J[v] = \int_\Omega |\nabla v|^2 - vf \text{ subject to } v \geq \psi,$$

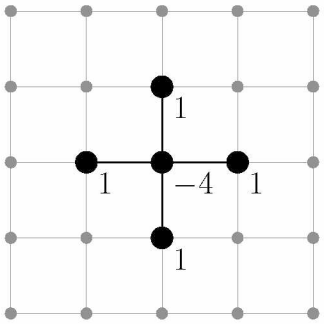
where $\Omega \subseteq \mathbb{R}^d$, and the formulation from Theorem 2.1 as the linear complementarity problem

$$\begin{aligned} u(x) - \psi(x) &\geq 0, \\ -f(x) - \Delta u(x) &\geq 0 \\ [-f(x) - \Delta u(x)][u(x) - \psi(x)] &= 0. \end{aligned}$$

Our implementation makes the following simplifying assumptions about the problem being solved and its discretization, and makes use of the following notation:

- The problem is two-dimensional (i.e. $d = 2$);
- $\Omega = [x_1, y_1] \times [x_2, y_2]$ is rectangular;
- We will label the number of unknowns in the horizontal direction m_x and in the vertical direction m_y. If there are the same number of unknowns in each direction, we will write $m = m_x = m_y$;
- The grid spacing is uniform in the horizontal and vertical directions separately and defined by $h_x = \frac{x_2 - x_1}{m_x + 1}$, $h_y = \frac{y_2 - y_2}{m_y + 1}$. If m_x and m_y are chosen so that $h_x = h_y$, we will write $h = h_x = h_y$;
- We write $N = (m_x + 2)(m_y + 2) = O(m_x m_y)$. This is the total number of grid points including known boundary values. The number of unknowns is $m_x m_y$.

As we have seen, it is usually desirable theoretically to choose m_x and m_y so that $h_x = h_y$. In this case, the 2D discrete Poisson operator has the stencil:



We elect to employ a *lexicographical ordering* of the unknowns (denoted $\leq_{\text{LEX}}$) , where an unknown $u_i = u^h(x_i, y_i) \leq_{\text{LEX}} u^h(x_j, y_j)$ if $y_i \leq y_j$ or $y_i = y_j$ and $x_i \leq x_j$. This corresponds to ordering the unknowns by sweeping through each horizontal row from left to right, starting with the bottom most row (i.e. the row with the lowest y-values).

With these assumptions and without eliminating boundary values, the Poisson equation $\Delta u = f$ uses the discretized Laplacian

$$\Delta u^h(x_i, y_i) = \frac{u_{i+1} - 2u_i + u_{i-1}}{h_x^2} + \frac{u_{(i+m_x+1)+1} - 2u_i + u_{(i+m_x+3)+1}}{h_y^2} + O(h_x^2 + h_y^2).$$

The Poisson operator has the following form:

$$A = \begin{bmatrix} I & & & & \\ I' & T & I' & & \\ & \ddots & \ddots & \ddots & \\ & & I' & T & I' \\ & & & & I \end{bmatrix} \in \mathbb{R}^{N \times N}, \qquad I' = \frac{1}{h_y^2} \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} \in \mathbb{R}^{(m_x+2)\times(m_x+2)},$$

$$T = \begin{bmatrix} 1 & & & & & \\ h_x^{-2} & -2(h_x^{-2} + h_y^{-2}) & h_x^{-2} & & & \\ & \ddots & \ddots & \ddots & & \\ & & h_x^{-2} & -2(h_x^{-1} + h_y^{-2}) & h_x^{-2} & \\ & & & & 1 \end{bmatrix} \in \mathbb{R}^{(m_x+2)\times(m_x+2)}$$

and $I \in \mathbb{R}^{(m_x+2)\times(m_x+2)}$ is the identity matrix.

5.1 A general 2D obstacle problem solver

We briefly go through the details of implementation of a general obstacle problem solver. The obstacle problem itself is stored as an instance of the class `box_obstacle_problem`, whose attributes are the bounds $\{x_1, x_2, y_1, y_2\}$ of the rectangular region Ω, a right-hand side function f, a boundary value function g, an obstacle function ψ, and discrete versions of the Poisson operator Δ and the functions ψ and f. The solver can also store a current guess to the solution (usually used as an initial iterate in an iterative solver) and the number of unknowns in the horizontal direction m_x and in the vertical direction m_y.

Attribute	Description
`bounds`	4-tuple: `(x1, x2, y1, y2)`
`f`	callable: rhs function handle
`g`	callable: boundary function handle
`psi`	callable: obstacle function handle
`mx`	integer: # unknowns in horizontal direction
`my`	integer: # unknowns in vertical direction
`A`	sparse matrix $N \times N$ matrix: discrete Poisson matrix
`U`	N-vector: guess to discrete solution
`P`	N-vector: discrete obstacle
`F`	N-vector: discrete rhs

An instance of `box_obstacle_problem` can be initialized with only the first four attributes listed above. The others can be generated by the methods `initialize` and `discretize`. The `discretize` method takes arguments `mx` and `my` and generates the Poisson matrix A and the grid functions, along with the discrete version of the default initial guess $u^{(0)} = \max\{\psi, 0\}$.

By setting $v(x) := u(x) - \psi(x)$, $b := -f + \Delta\psi$, the continuous version of the LCP can be put in canonical form:

$$\begin{aligned} v(x) &\geq 0, && x \in \Omega \\ b(x) - \Delta v(x) &\geq 0, && x \in \Omega \\ [b(x) - \Delta v(x)]v(x) &= 0, && x \in \Omega. \end{aligned}$$

The `initialize` method makes the change of variables for the discrete LCP. Regardless of the smoothness of the obstacle, the change of variables is feasible mathematically. However, a highly irregular obstacle may cause numerical difficulties.

Finally, the obstacle problem is solved via the `solve` method, which takes an obstacle problem solver (or, more generally, an LCP solver) as an argument and returns a vector u containing the function value at each grid point.

5.2 PFAS implementation

Our implementation of PFAS follows closely the Python library of algebraic multigrid solvers `pyamg`. The class `linear_pfas_solver` stores the multigrid hierarchy, which is further embedded in a list of structs containing the information for each grid level. The information stored at each level is listed below.

Attribute	Description
`bounds`	4-tuple: `(x1, x2, y1, y2)`
`mx`	integer: # unknowns in horizontal direction
`my`	integer: # unknowns in vertical direction
`hx`	float: grid spacing in horizontal direction
`hy`	float: grid spacing in vertical direction
`A`	sparse matrix: discrete Poisson matrix $A \in \mathbb{R}^{(m_x+2)\times(m_y+2)}$
`R`	sparse matrix: restriction matrix
`P`	sparse matrix: prolongation matrix

The coarsening scheme is normally assumed to be standard coarsening, where the grid spacing differs by a factor of two in both the horizontal and vertical directions between consecutive grids. However, this implementation is flexible enough to support arbitrary coarsening schemes.

Other than the grid, attributes that are chosen by the user are the coarse grid solver and the smoother. The default for both is projected Gauss-Seidel. The prolongation and restriction operators are also choices made at each level. These can be added to a level directly as a matrix or as a callable function which generates a grid transfer matrix given a grid spacing. The second option assumes a standard coarsening scheme. In general, a PFAS solver would allow the user to choose the number of pre- and post-smoothing steps. In our implementation we only use one of each.

The recursion is quite straightforward to implement in this framework. The `solve` method takes a right hand side vector `b` and optionally an initial guess. The default for the initial guess is the zero vector. It also asks for a cycle type, which can be an F-cycle, V-cycle, or W-cycle. The solve itself is initiated by a call to the method `lvl_solve`, which

takes as arguments a grid number (0 being the finest), initial guess, right-hand side, and cycle type. The recursion occurs within the `lvl_solve` method, which calls itself on each grid level until the coarsest grid.

```
if lvl < len(self.levels) - 2:

'''
The recursion occurs here, with the restricted iterate as the initial guess for
the coarse grid and the restricted and tau-corrected residual as the right hand side
'''

    if cycle == 'W':
        self.lvl_solve(lvl + 1, coarse_u, coarse_b, cycle)
        self.lvl_solve(lvl + 1, coarse_u, coarse_b, cycle)

    elif cycle == 'V':
        self.lvl_solve(lvl + 1, coarse_u, coarse_b, cycle)

    elif cycle == 'FV':
        self.lvl_solve(lvl + 1, coarse_u, coarse_b, 'FV')
        self.lvl_solve(lvl + 1, coarse_u, coarse_b, 'V')

    else: # The coarse grid problem is solved by smoothing until the iteration stalls.

        uold = np.zeros_like(coarse_u)
        du = 1.0
        while du > 10 ** -12:
            for i in range(0, len(uold)):
                uold[i] = coarse_u[i]
            self.smoother(coarse_A, coarse_u, coarse_b)
            du = (self.coarse_mx + 1) * np.linalg.norm(uold - coarse_u)

    P = self.levels[lvl + 1].P
    u += P.dot(coarse_u - R.dot(u))    # coarse grid correction
    self.level = self.levels[lvl]
    self.smoother(A, u, b)             # one more smoothing step on the way up
```

```
    def solve(self, b, u0=None, cycle='FV'):

        if u0 is None:
            u0 = np.zeros_like(b)

        u = np.array(u0)
        self.lvl_solve(0, u, b, cycle)
        return u


    def lvl_solve(self, lvl, u, b, cycle):

        self.level = self.levels[lvl]
        A = self.level.A
        self.smoother(A, u, b)                    # one smoothing step
        R = self.level.R
        coarse_u = R.dot(u)                       # restrict current iterate
        coarse_b = R.dot(b - A.dot(u))            # restrict residual
        coarse_A = self.levels[lvl + 1].A         # needs at least two levels
        coarse_b = coarse_b + coarse_A.dot(coarse_u) # tau correction
```

5.3 Reduced space method

The reduced space method is implemented as a general LCP solver. The solver class stores the matrix and vector information defining the affine function $F(x) = Mx + q$, along with the number of total iterations. Information about the current iterate (current guess, search direction, reduced space dimension, etc.) is stored in the struct `rsp_lcp_iterate`, which also computes the active set, inactive set and residual for each iterate. The user chooses the solver for linear systems. In this paper, we restrict ourselves to sparse LU factorization using the `spsolve` method from the `scipy.sparse` library. The code for this algorithm follows rather closely to Algorithm 4, so we omit the details and encourage the reader to consult the GitHub page[1] associated with this project.

[1] https://github.com/mheldman/Obstacle-Problem

Chapter 6

Results and analysis

Example 6.1 (radial problem)**.** This obstacle problem was adapted from [Bue04]. The problem is to solve the LCP (2.5) - (2.7)

$$\Omega = \{x^2 + y^2 \leq 4\}, \qquad \psi(x,y) = \sqrt{\max\{1 - x^2 - y^2, 0\}}, \qquad f \equiv 0, \qquad u|_{\partial\Omega} \equiv 0.$$

For compatibility with our structured grid, we extend the problem to the box

$$\Omega = \{\max\{x, y\} \leq 2\}.$$

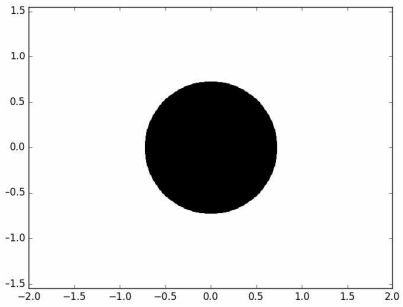


Figure 6.1: The contact region $\{u = \psi\}$ for the solution u to the radial problem. The solution u and obstacle ψ are pictured in Figure 1.1. Both the solution and the contact region were computed with $m = 500$.

The problem has exact solution

$$\overline{u}(r) = -A\log(r) + B, \qquad r = \sqrt{x^2 + y^2},$$

where A and B are constants computed numerically to 5 digits of accuracy. This defines the boundary condition ($g = \overline{u}|_{\partial\Omega}$). The exact solution and the obstacle are plotted in Figure 1.1, and the contact set is pictured in Figure 6.1.

Example 6.2 (dam problem)**.** The dam problem is used as an example in [BC83]. The problem is to describe the flow of water from a reservoir of height y_1 through a porous rectangular dam of width x_1 into a reservoir of height $a < y_1$. Only part of the dam is contacted by the water, so that the dam is separated by an unknown boundary into wet and dry regions. This situation is pictured in Figure 6.2 (below).

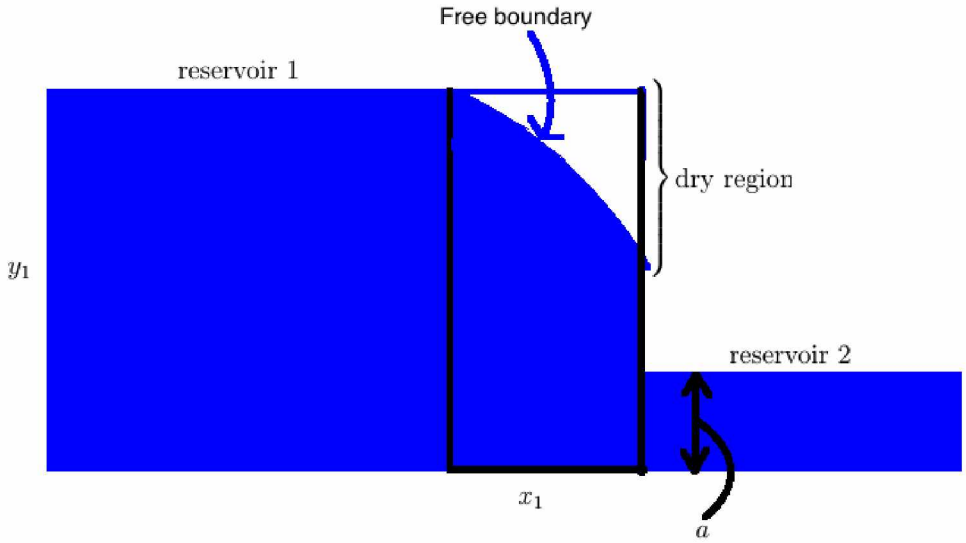


Figure 6.2: The dam problem is to compute the wet and dry regions of a porous dam, which are separated by an unknown free boundary. The dam problem can be formulated as an LCP.

Taking $\Omega = [0, x_1] \times [0, y_1]$, the problem is formally written as an LCP by taking

$$f \equiv 1, \qquad \psi \equiv 0, \qquad g(x,y) = \begin{cases} \frac{(y_1-y)^2}{2} & (x,y) \in \{0\} \times [0, y_1] \\ \frac{(a-y)^2}{2} & x \in \{x_1\} \times [0, a] \\ \frac{y_1^2(x_1-x)^2 + a^2 x}{2x_1} & x \in [0, x_1] \times \{0\} \\ 0 & \text{otherwise.} \end{cases}$$

We consider the case $y_1 = 24$, $a = 4$, and $x_1 = 16$ as in [BC83] so we can compare results. The solution and obstacle are plotted alongside the contact set in Figure 6.3.

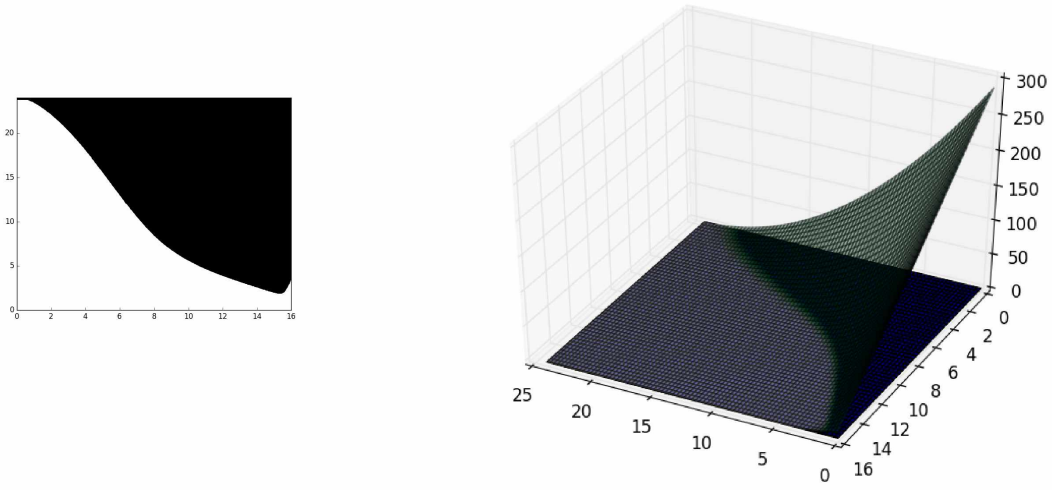


Figure 6.3: The solution to the dam problem and the obstacle (right) and the contact region (left). The solution was computed using the `RSP` solver with $m_x = 511$, $m_y = 767$.

6.1 PFAS results

We begin with a comparison between PFAS and the unconstrained FAS iteration. The residual $F_\Omega\left(u^{(k)}\right)$, defined by (4.2), will henceforth be denoted by $s^{(k)}$ instead of $r^{(k)}$, so as not to confuse with the residual of the unconstrained equation $r^{(k)} = f - Au^{(k)}$. It will be our proxy for the grid error $e^{(k)}$. It is observed in [BC83] that, for example,

$$\left\|e^{(k)}\right\|_2 \leq \alpha^{-1} \left\|s^{(k)}\right\|_2,$$

where α is the coercivity constant for the SPD matrix A (the smallest of the eigenvalues for A). We define the *geometric convergence factor* μ of the iteration as the geometric average norm reduction per iteration, $\mu \approx \left\|\frac{s^{(k)}}{s^{(0)}}\right\|^{\frac{1}{k}}$ for some k. We use this number to approximate the asymptotic convergence factor of the iteration and compare to the standard multigrid iterations. For this purpose, we include an unconstrained example problem:

Example 6.3 (unconstrained)**.** Choose $\psi = -\infty$, $g = 0$, and

$$f(x, y) = 12(y^4 - 1)x^2 + 12(x^4 - 1)y^2$$

on the square $\Omega = [-1, 1] \times [-1, 1]$.

We experimentally observe a grid independent geometric convergence factor for Example 6.3 of $\mu \approx .2$, meaning errors were reduced by roughly a factor of five on each iteration. The

runtime of the FAS iteration seems to scale linearly with the number of unknowns, increasing by a factor of four as the number of points on the fine grid is doubled. The results for FAS are tabled below, where k is the number of iterations to convergence and $N = (m+2)^2$. All calculations were made to a residual tolerance of 10^{-10}.

Table 6.1: Convergence results for the usual FAS iteration applied to the unconstrained example. The convergence factors μ are grid independent and the timing supports the conclusion that FAS is optimal.

N	k	μ	time
33^2	15	.2	7.7s
65^2	16	.21	31s
129^2	16	.21	139s
257^2	16	.21	460s

Because the number of iterations to convergence and the convergence factors are grid independent, the method appears to be optimal on this problem. The times are not expected to be a totally accurate reflection of the complexity of the algorithm, but they give a rough sense of how the algorithm scales. In particular, the ratio between consecutive refinements are 4.02, 4.39, and 3.3, which average to the factor of four we would expect for an optimal method.

In contrast, applying PFAS with V-cycles (denoted `PFASV`) to Examples 6.1 and 6.2 we observe convergence factors that are both worse than those observed for FAS and are grid dependent. On the radial problem the iteration stagnates after initially quick residual reductions (below). In the largest problem tested, the iteration stalls after several iterations and residuals are no longer reduced. `PFASV` does better on the smallest version of the dam problem than it did on the radial problem of comparable size, but its performance on the dam problem performance degrades much more quickly with refinement.

Table 6.2: Convergence statistics for `PFASV` on the radial problem (left) and the dam problem (right). The number of iterations (capped at 50), convergence factor, and final residual norm are shown.

N	k	μ	$\left\|s^{(k)}\right\|_\infty$
33^2	50	.67	$2.4 \cdot 10^{-8}$
65^2	50	.8	$8.6 \cdot 10^{-6}$
129^2	50	.72	$3.4 \cdot 10^{-6}$
257^2	50	.92	2.14

N	k	μ	$\left\|s^{(k)}\right\|_\infty$
$33 \cdot 49$	40	.57	$7.92 \cdot 10^{-9}$
$65 \cdot 97$	50	.9	$9.8 \cdot 10^{-1}$
$129 \cdot 193$	50	.9	1.79
$257 \cdot 385$	50	.91	6.11

The reason for this behavior was not immediately obvious. In [BC83], it is posited that large residuals near the free boundary are to blame. In multigrid theory, it is known

that proper boundary treatment is important in systems which enforce implicit boundary conditions. The residuals tend to be less smooth near the free boundary, and hence see slower convergence than points on the interior. The jump in the second derivative of the function u at the free boundary in obstacle-type problems may also play a role. To test this, we plotted a heat map of the residuals after six iterations of PFAS, which is usually when the iteration begins to stagnate. The lighter colors near the free boundary indicate that residuals are large there. The trend is even more obvious when we look at the same map for the dam problem, where the only nonzero residual appear at the location of the free boundary.

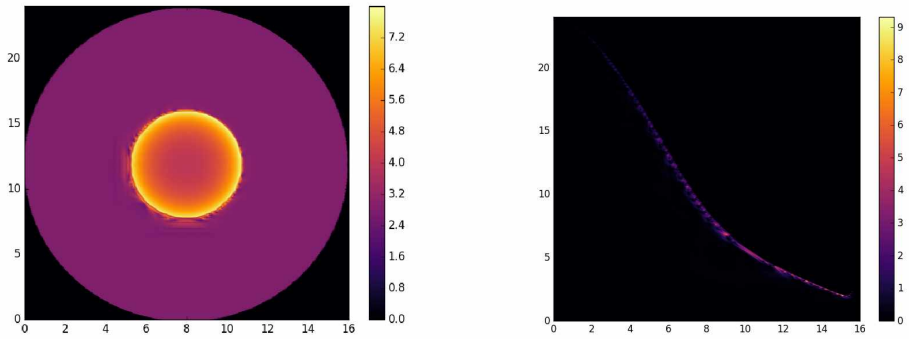


Figure 6.4: Heat map of the residuals $\left|s_i^{(k)}\right|$ for the radial problem (left) and the dam problem (right) after six iterations of `PFASV`, when stagnation typically begins. Lighter colors around the free boundary indicate larger residuals, while darker colors indicate lower residuals. In both cases, residuals are largest in the neighborhood immediately surrounding the free boundary.

As in classic multigrid theory, one solution may be to accelerate convergence near the free boundary by adding extra `PGS` relaxations at nodes in the area. In experiments, adding three pre- and post-smoothing sweeps at the nodes with residuals in 90-100 percentile range improved geometric convergence factors for the smaller dimensional problems considerably but did not bring them close to the benchmark set by FAS. Moreover, the extra smoothing steps were not enough to make the observed convergence factors grid independent. For larger problems, the iteration still stalled as before, albeit with better residual reductions before getting stuck.

We also apply PFAS with an F-cycle instead of a V-cycle, which we will call `PFASF`. Convergence factors are only slightly worse than FAS in this case and appear more robust with respect to the fine grid size. This comes at a cost of extra work per iteration by the F-cycle.

Table 6.3: Convergence statistics for `PFASF` on the radial problem (left) and the dam problem (right). The number of iterations (capped at 50), convergence factor, and time to convergence are shown.

N	k	μ	time
33^2	23	.4	15s
65^2	29	.47	75s
129^2	16	.26	156s
257^2	27	.42	940s

N	k	μ	time
$33 \cdot 49$	18	.28	17s
$65 \cdot 97$	19	.29	68s
$129 \cdot 193$	25	.38	341s
$257 \cdot 385$	21	.31	1163s

The `PFASF` solver seems fast and scalable on these examples. The convergence factors remain roughly similar, although they seem to increase slightly with the number of iterations. As we mentioned earlier, this may have to do with large residuals at the free boundary that the multigrid solver is not equipped to handle.

6.2 Reduced space method

This method is generally considered as the alternative to `PGS` for LCPs, and the results show that it is far outpaces PFAS on small problems, but scales much worse than linearly. The number of iterations to convergence and the convergence factors increase substantially with the size of the problem.

Table 6.4: Convergence statistics for RSP on the radial problem (left) and the dam problem (right). The number of iterations (capped at 100), convergence factor, and time to convergence are shown.

N	k	μ	time
65^2	13	.09	.28s
129^2	24	.28	2.1s
257^2	45	.51	26s
513^2	85	.71	296s

N	k	μ	time
$129 \cdot 193$	41	.47	2.8s
$257 \cdot 385$	79	.68	28s
$513 \cdot 769$	100+	.99	–

The decent convergence factors that we see in the all of the reduced space experiments are mostly driven by the last several iterations where the norm is reduced by roughly 12 orders of magnitude in a single step.

References

[BC83] Achi Brandt and Colin W. Cryer. Multigrid algorithms for the solution of linear complementarity problems arising from free boundary problems. *SIAM Journal on Scientific and Statistical Computing*, 4(4):655–684, 1983.

[BEL11] Achi Brandt and Oren E. Livne. *Multigrid techniques. 1984 guide with applications to fluid dynamics.* revised ed. 2011.

[BHM00] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial (2nd Ed.).* Society for Industrial and Applied Mathematics, Philadelphia, USA, 2000.

[BM03] S. J. Benson and T. S. Munson. Flexible complementarity solvers for large-scale applications. 2003. https://arxiv.org/pdf/math/0307305.pdf.

[Bor03] Alfio Borzì. Introduction to multigrid methods. 2003. https://www.mathematik.uni-wuerzburg.de/~borzi/mgintro.pdf.

[Bue04] Edward Bueler. An easy finite element implementation of the obstacle problem for Poisson's equation. 2004. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.538.7987&rep=rep1&type=pdf.

[Fal06] Robert D. Falgout. An introduction to algebraic multigrid. *Computing in Science and Engg.*, 8(6):24–33, November 2006.

[Fed01] R.P. Fedorenko. On the history of the multigrid method creation. 2001. http://wwwhome.math.utwente.nl/~botchevma/fedorenko/index.php.

[FLMN10] Liming Feng, Vadim Linetsky, José Luis Morales, and Jorge Nocedal. On the solution of complementarity problems arising in american options pricing. *Optimization Methods and Software*, 26(4-5):813–825, 2010.

[Koh15] Ludwig Kohaupt. Introduction to the discrete Fourier series considering both mathematical and engineering aspects - a linear-algebra approach. *Cogent Education*, 2(1), 2015.

[Rod87] José Rodrigues. *Obstacle Problems in Mathematical Physics*, volume 134 of *North-Holland Mathematics Studies*. North-Holland, 1987.

[TSO01] Ulrich Trottenberg, Anton Schüller, and Cornelis Oosterlee. *Multigrid*. Academic Press, Inc., Orlando, USA, 2001.