Multi-Resolution Approximate Inverses

by

Robert Edward Bridson

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1999
©Robert Edward Bridson 1999

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Multi-Resolution Approximate Inverses

This thesis presents a new preconditioner for elliptic PDE problems on unstructured meshes. Using ideas from second generation wavelets, a multi-resolution basis is constructed to effectively compress the inverse of the matrix, resolving the sparsity vs. quality problem of standard approximate inverses. This finally allows the approximate inverse approach to scale well, giving fast convergence for Krylov subspace accelerators on a wide variety of large unstructured problems. Implementation details are discussed, including ordering and construction of factored approximate inverses, discretization and basis construction in one and two dimensions, and possibilities for parallelism. The numerical experiments in one and two dimensions confirm the capabilities of the scheme. Along the way I highlight many new avenues for research, including the connections to multigrid and other multi-resolution schemes.

Acknowledgements

I would like to first thank Wei-Pai Tang for his excellent ideas, support, and guidance. I'm also indebted to Peter Forsyth for advice on discretization and multigrid, and very helpful suggestions for revisions; to Sivabal Sivaloganathan for introducing me to Green's functions and reading the drafts; to Rob Zvan for getting me started on irregular meshes; to Justin Wan for some fruitful conversations and sample meshes; to David Pooley for the barrier option pricing problem; and of course to the Natural Sciences and Engineering Research Council of Canada for their financial support. I especially want to thank my family for their much needed love and encouragement throughout the last year and particularly the final hectic weeks.

Dedication

For Rowena.

Contents

List of Tables

List of Figures

Chapter 1

Preliminaries

1.1 Introduction

One of the biggest challenges facing scientific computing today is accurately solving partial differential equations. Advances in computer performance help fuel this demand for higher quality numerical solutions to problems from science, engineering, finance, etc. However, effectively using the increased power of workstations and supercomputers requires new methods: algorithms that worked in the past often don't scale well to the new architectures and bigger problems.

This thesis proposes a new technique for solving the systems of linear equations which take up so much of the time for elliptic PDE's on unstructured meshes. Current approximate inverse preconditioners do not scale well: there must be a trade-off between sparsity and quality in the approximation, with the disparity quickly growing as the problem size increases. However, by compressing the inverse with techniques from second generation wavelets, a sparse *and* high quality preconditioner can be found, giving fast and scalable convergence for iterative methods. Furthermore, the attractive parallelism of approximate inverses is retained.

The main effort in this work is to present the essential ideas and motivations behind multiresolution approximate inverses, with proofs of concept showing their capabilities in one and two dimensions. This is not a theoretical treatise proving optimality of the method, nor is it

a blueprint for a high-performance implementation. I have instead striven for a practical and intuitive middle route that will ease the way for progress in all directions.

1.2 Elliptic PDE's

This thesis is concerned with the numerical solution of elliptic partial differential equations (PDE's). More specifically, second order linear scalar problems will be considered:

$$\mathcal{L}u = f \quad \text{on} \quad \Omega \tag{1.1}$$

where Ω is the region of interest, f is some forcing function, and the differential operator $\mathcal{L}$ is of the form:

$$\mathcal{L}u = \nabla \cdot (K\nabla u - bu) + cu$$

Here K is a positive definite second order tensor field, b is a vector field, and c is a scalar field. Note that these coefficients may vary over Ω—in some applications, possibly with jump discontinuities of several orders of magnitude.

One can physically interpret 1.1 as describing the concentration u of some quantity—e.g. heat, a chemical dissolved in fluid, neutrons, etc. The expression $(K\nabla u - bu)$ is the "flux", measuring how fast and in what direction the quantity is flowing. The $K\nabla u$ term represents *diffusion*, how the quantity naturally flows from regions of high concentration towards regions of low concentration at a rate proportional to the gradient (though if K is not a scalar multiple of the identity, anisotropy in the underlying medium can distort this flow). The bu term represents the *convection* of the quantity by some underlying flow field—a current in the medium described by b carries the quantity along with it. Thus $\nabla \cdot (K\nabla u - bu)$ at a point gives the total change in u at that point due to flow of the quantity. The final term cu represents *reaction*, where the quantity is created ($c > 0$) or destroyed ($c < 0$) at a rate proportional to its concentration. In the former case, $c > 0$, the problem may become indefinite or even ill-posed.

There are several special examples of elliptic equations which don't exactly fall under this interpretation. For example, irrotational, inviscid, incompressible fluid flow can be determined by solving Laplace's equation

$$\nabla^2 \phi = \nabla \cdot \nabla \phi = 0$$

for a potential function ϕ, giving the velocity field $v = \nabla\phi$. Another example is the Helmholtz equation arising in electromagnetics

$$\nabla^2\psi + k^2\psi = f$$

Of course, 1.1 is under-determined without appropriate boundary conditions. For example, the value or derivatives of u could be specified along the boundary. To be precise, letting the boundary $\partial\Omega$ of Ω be partitioned into $\partial\Omega_D$, $\partial\Omega_N$, and $\partial\Omega_R$, impose the following on 1.1:

$$u = g_D \quad \text{on} \quad \partial\Omega_D$$

$$(K\nabla u)\cdot\hat{n} = g_N \quad \text{on} \quad \partial\Omega_N$$

$$(K\nabla u)\cdot\hat{n} + au = g_R \quad \text{on} \quad \partial\Omega_R$$

Here $\hat{n}$ denotes the normal vector to the boundary. The first condition, where u is specified, is called a Dirichlet condition. The second condition, where the diffusive flux through the boundary is specified, is the physical generalization of the Neumann boundary condition $\nabla u \cdot \hat{n} = g$. The third, a linear combination of the first two, is the generalization of the Robin condition $\nabla u \cdot \hat{n} + au = g$. Note in particular that specifying the combined diffusive and convective flux, $(K\nabla u - bu)\cdot\hat{n} = (K\nabla u)\cdot\hat{n} + (-b\cdot\hat{n})u$, is a special case of this third type.

An important application of solving equations like 1.1 arises in the implicit numerical solution of time-dependent parabolic partial differential equations, of the form:

$$\frac{\partial u}{\partial t} = \mathcal{L}u \quad \text{on} \quad \Omega, \quad t \geq 0$$

Another big application is non-linear elliptic problems, where each step of Newton's method will involve solving a linearized problem of the type 1.1, with the coefficients depending on the solution from the previous step.

Finally, the problem 1.1 is also seen in inverse iteration methods for finding eigenvalues and eigenfunctions of the operator, i.e. scalars λ and functions u such that $\mathcal{L}u = \lambda u$.

1.3 Discretization

To numerically approximate the solution of the PDE, the equation 1.1 must be *discretized*, reducing it from an infinite dimensional linear system to a finite dimensional one. Typically this is done by first determining a set of points (the "nodes") in the region where the approximate values of u are sought. A mesh is generated that connects those nodes, breaking up the region into small and simple subregions—e.g. sub-intervals in 1D, triangles or quadrilaterals in 2D, tetrahedra or prisms in 3D. For each node, a linear equation involving nearby nodes is determined from the mesh, attempting to approximate the true equation 1.1 or the boundary condition at that node. The resulting finite system of equations is then solved for the approximate values of u at the nodes. The system will be written as $\mathbf{A}\mathbf{u} = \mathbf{f}$, where $\mathbf{A}$ is the matrix of coefficients of the equations, $\mathbf{u}$ is the vector of unknown values of u at the nodes, and $\mathbf{f}$ is the known right-hand side vector arising from f and the boundary conditions.

This thesis is concerned with problems on unstructured meshes, that is meshes that are not regularly arranged grids. This is of particular interest for two reasons. First, most real life problems involve regions of such geometric complexity that it is difficult to faithfully represent them with a structured grid. Second, in most real life problems the magnitudes of the derivatives of the solution, which govern the accuracy of the discretization, vary considerably: in regions of rapid changes, more nodes are required for adequate accuracy. This "adaptive meshing" is often difficult to manage with structured grids.

For discretizing a PDE on an unstructured mesh, either the finite volume method or the finite element method is normally used. Particularly for difficult problems with discontinuities in the coefficients, fairly low-order approximations using many nodes are preferred. In this case, the finite element method can often be interpreted as a type of finite volume method or vice versa, and so in fact often the two methods are used together. See sections 4.2 and 5.2 for details.

1.4 Iterative Solvers

The meshes on which PDE's are discretized are often very fine, using many nodes, in order to give more accurate solutions. This gives rise to very large systems of equations to be solved. Fortunately, the coefficient matrices are sparse: almost all the entries are zero. The traditional

approach of using Gaussian elimination to decompose the matrix into triangular factors, then solving triangular systems, can be enhanced to exploit this sparsity. Modern "direct methods" use sophisticated reorderings of the rows and columns to keep the storage requirements for the factors as low as possible, and clever data structure algorithms to reduce the factorization and solution time to a minimum.

However, as problems have continued to grow and computer architecture changed, direct methods have hit serious difficulties. Matrices with millions of rows are becoming common, and at that size just storing the factors can be too expensive, let alone computing them! Furthermore, Gaussian elimination and triangular solves can be difficult to effectively parallelize, resulting in poor efficiency on todays high performance machines with tens, hundreds or even thousands of processors.

Alternatives include "fast solvers," which typically use the Fast Fourier Transform to solve certain PDE problems very efficiently. Unfortunately their use is generally restricted to some special constant coefficient PDE's discretized only on uniform Cartesian grids, which is inadequate for many applications.

The search for scalable algorithms for effectively solving very large problems, especially on parallel computers, turns instead to "iterative methods". The essential idea behind these schemes is that starting with an initial guess $\mathbf{u}^0$ for the solution of $\mathbf{A}\mathbf{u} = \mathbf{f}$, refinements are made to get better guesses $\mathbf{u}^1$, $\mathbf{u}^2$, ... The process is halted when $\mathbf{u}^i$ is deemed accurate enough, say when $||\mathbf{A}\mathbf{u}^i - \mathbf{f}|| < 10^{-6}||\mathbf{A}\mathbf{u}^0 - \mathbf{f}||$.

There are many possibilities for determining better guesses at the solution. The most popular general purpose algorithms are called "Krylov subspace accelerators". There are many different schemes in this framework, but it is widely accepted that the choice of accelerator isn't crucial compared to the choice of preconditioner, explained below. In this thesis I follow a popular choice of using the Conjugate Gradient method for symmetric positive definite matrices, and the Biconjugate Gradient Stabilized (BiCGStab) method for all other problems. See [32] for an exposition of these and other methods.

The advantages of these iterative methods are twofold: firstly, there are no large factors needed—just the matrix and a few auxiliary vectors—and secondly, only easily parallelized matrix-vector multiplies are used. On the other hand, the major disadvantage of iterative methods is robustness. The rate of convergence to the correct solution depends upon the condition

number[1] which is often so large that the accelerator simply won't converge at all: finite precision arithmetic errors build up faster than the theoretical convergence. This is especially a problem for large, highly nonuniform unstructured meshes: the condition number increases not only with the size of the problem, but also with the degree of irregularity in the mesh. For problems with highly variable coefficients, or that show strong anisotropy, or that are indefinite or close to indefinite, the condition number is still worse. It is generally agreed that not much improvement can be made to the standard accelerators; the key is instead "preconditioning".

A preconditioner in general is a pair of non-singular linear operators $\mathbf{M}_L$ and $\mathbf{M}_R$ such that $\mathbf{M}_L\mathbf{A}\mathbf{M}_R \approx \mathbf{I}$. Note that they don't have to be explicitly known in matrix form, instead being implicitly represented by linear algorithms or products of matrices for example. Special cases where one of the operators is just the identity (and so is ignored) are referred to as left or right preconditioning. The key observation is that the system $\mathbf{A}\mathbf{u} = \mathbf{f}$ is equivalent to the system $(\mathbf{M}_L\mathbf{A}\mathbf{M}_R)\mathbf{v} = \mathbf{M}_L\mathbf{f}$, $\mathbf{u} = \mathbf{M}_R\mathbf{v}$, but the second system should be much easier to solve iteratively thanks to its improved condition number. The goal then is to find preconditioners that are effective in improving the condition number, but that are cheap to compute, store, and apply.

1.5 Approximate Inverse Preconditioners

One preconditioning strategy exemplified by ILU and Gauss-Seidel (see [32]) is to determine very sparse approximations to the triangular factors of $\mathbf{A}$. Triangular solves can then be used to approximate the application of $\mathbf{A}^{-1}$, just as a direct method uses the exact factors to exactly apply $\mathbf{A}^{-1}$, modulo rounding errors. Even though this can be much cheaper than full Gaussian elimination, since the full factors need not be computed, the approach inherits the parallelism problems of direct methods, and so much research has been devoted to alternative schemes.

Of particular interest today are approximate inverse preconditioners where $\mathbf{A}^{-1}$ is directly approximated with a sparse matrix, or more generally, a product of sparse matrices. This restores the attractive parallelism of accelerators, since to apply the preconditioner again only easily parallelized matrix-vector multiplies are needed.

[1] The condition number of $\mathbf{A}$, denoted by $\kappa(\mathbf{A})$, is a measure of how far the matrix is from the identity $\mathbf{I}$. It is generally defined as $\kappa(\mathbf{A}) = ||\mathbf{A}|| \cdot ||\mathbf{A}^{-1}||$ for some appropriate matrix norm.

Several algorithms have been proposed for constructing approximate inverses. These can be loosely categorized first by their result: some algorithms produce a single sparse matrix approximating $\mathbf{A}^{-1}$—e.g. SPAI[22], Chow and Saad's MR method[15, 16], Tang and Wan's local inverse[35]—and others produce factored approximations (approximate inverses of the triangular factors)—e.g. FSAI[25], AINV[4]. The factored form has the advantages of guaranteed non-singularity, extra sparsity from good orderings, and apparently more effect per nonzero thanks to its more implicit nature. However, the non-factored form has the advantages of robustness with respect to orderings—bad pivots are not an issue—and more parallel application.

The algorithms can secondly be classified according to their general approach: minimization of the Frobenius norm of the error between the preconditioned matrix and the identity under sparsity constraints (e.g. SPAI, FSAI), limited optimization of that error (e.g. Chow and Saad's MR method, Tang and Wan's local inverse), or incomplete inversion algorithms (e.g. AINV).

There is much work to be done in improving these algorithms—implementation details, parallelism in construction, finding good sparsity patterns, etc.—but it appears at the moment that the factored, incomplete inversion algorithms are the most practical. The algorithm used in this thesis is AINV.

1.6 Discrete Green's Functions

One general method for analytically solving 1.1 is to find the Green's function, a function $G : \Omega \times \Omega \to \mathbb{R}$ which satisfies:

$$\mathcal{L}_x G(x, y) = \delta(x - y), \quad \text{for} \quad x, y \in \Omega, \qquad (y \text{ held fixed}) \tag{1.2}$$

and suitable boundary conditions on $\partial\Omega$. Here the derivatives are with respect to x, and δ is the Dirac delta distribution. Neglecting the boundary conditions for simplicity, the solution to 1.1 is:

$$u(x) = \int_\Omega G(x, y) f(y)\, dy \tag{1.3}$$

since then

$$\mathcal{L}(u(x)) \quad = \quad \mathcal{L} \int_\Omega G(x, y) f(y)\, dy$$

$$
\begin{aligned}
&= \int_\Omega \mathcal{L}_x G(x,y) f(y)\, dy \\
&= \int_\Omega \delta(x-y) f(y)\, dy \\
&= f(x)
\end{aligned}
$$

Suppose 1.1 has been discretized as $\mathbf{A}\mathbf{u} = \mathbf{f}$, where $\mathbf{A}$ approximates $\mathcal{L}$, $\mathbf{u}$ approximates u with $u_i \approx u(x_i)$, and $\mathbf{f}$ approximates f with $f_i \approx f(x_i)$, again ignoring boundary conditions for simplicity. With the matrix $\mathbf{A}^{-1}$ satisfying

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I} \tag{1.4}$$

write the discrete solution as:

$$\mathbf{u} = \mathbf{A}^{-1}\mathbf{f}, \qquad \text{i.e.} \quad u_i = \sum_j A^{-1}_{ij}\, f_j \tag{1.5}$$

Note that the identity matrix $\mathbf{I}$ is the discrete Dirac delta: $I_{ij} = \delta_{ij}$. The analogy between 1.2 and 1.4, and between 1.3 and 1.5 is then clear. The matrix $\mathbf{A}^{-1}$ is a discrete version of the Green's function G.

For most elliptic problems the Green's function G is known to be nonzero on all of $\Omega \times \Omega$, and possibly significantly large on a lot of that domain. If the discretization $\mathbf{A}$ of the operator $\mathcal{L}$ is of any value, $\mathbf{A}^{-1}$ must similarly be mostly nonzero with possibly many large entries. Unfortunately this means a sparse yet high quality approximation is impossible in general—there is no hope for scalable approximate inverses. That is, in the standard basis: the aim of this thesis is to find a better basis where sparsity and quality aren't mutually exclusive.

1.7 Multi-Resolution Bases

Before going further, I shall introduce some notation. Take Ω to be the (bounded) domain of interest. Suppose there are n points $x_1, \ldots, x_n \in \Omega$ identified at which the value of some function $f : \Omega \to \mathbb{R}$ is known; call the vector $\mathbf{f} = (f_1, \ldots, f_n) \in \mathbb{R}^n$, where $f_i = f(x_i)$, the discretized version of f. Often the word "signal" is used instead of function.

The standard basis vectors for $\mathbb{R}^n$ are $\mathbf{e}^1, \ldots, \mathbf{e}^n$, with $e^i_j = \delta_{ij}$; for example, $\mathbf{e}^2 = (0, 1, 0, 0, \ldots, 0)$. Thus expressing the discretized function as $\mathbf{f} = (f_1, \ldots, f_n) = f_1\mathbf{e}^1 + \cdots + f_n\mathbf{e}^n$ uses the standard basis. The coefficients of $\mathbf{f}$ in a different basis $\mathbf{v}^1, \ldots, \mathbf{v}^n$ are the real numbers $a_1, \ldots, a_n$ such that $\mathbf{f} = a_1\mathbf{v}^1 + \cdots + a_n\mathbf{v}^n$. The dual basis consists of the vectors $\mathbf{w}^1, \ldots, \mathbf{w}^n$ such that $a_i = \mathbf{f} \cdot \mathbf{w}^i$. If the basis is orthogonal ($\mathbf{v}^i \cdot \mathbf{v}^j = \delta_{ij}$) then the dual basis is the basis itself; in general, the two are distinct but biorthogonal ($\mathbf{v}^i \cdot \mathbf{w}^j = \delta_{ij}$).

A multi-resolution basis is a choice of basis vectors that seeks to represent smooth functions efficiently—if f is smooth, most of the coefficients in its multi-resolution representation will be very close to zero. (This opens the way for data compression, as a very good approximation to $\mathbf{f}$ is retained when only the few large coefficients are stored and the rest are assumed zero). Smoothness essentially means that large changes in the function value only happen over large length scales. In other words, most of the information in the signal is at a low resolution. Then what is needed is a basis that includes a few vectors with variation over long length scales, spanning a low resolution subspace, and is filled up with other vectors that vary on shorter length scales, giving the high resolution components. The coefficients for the handful of low resolution vectors will be large, but the coefficients for the high resolution vectors, which form the greater part of the basis, should be small. A natural way of adapting to different degrees of smoothness in functions is to in fact have a whole spectrum or hierarchy of different resolution vectors—hence the name multi-resolution.

The simplest multi-resolution basis is the Fourier basis, constructed from the functions $1, \sin(x), \cos(x), \sin(2x), \cos(2x), \sin(3x), \ldots$ on the interval $[0, 2\pi]$, with a uniform spacing of the points $x_1, \ldots, x_n$ usually assumed. This gives a spectrum of resolutions, with $\sin(kx)$ or $\cos(kx)$ varying on a length scale of $O(1/k)$. Many theorems have been proved showing the link between smoothness (precisely characterized by bounded derivatives of a certain order, for example) and small coefficients for the high resolution components.

Unfortunately, the Fourier basis has a significant flaw: global smoothness is required. For example, a single jump discontinuity in an otherwise extremely smooth function will give relatively large coefficients even in the high resolutions. The problems that these singularities can cause for the Fourier basis have again been precisely characterized in many theorems. In our application, compression of discrete Green's functions with multi-resolution bases, this is devastating—there is a guaranteed singularity along the diagonal produced by the Dirac delta, not to mention the possibility of singularities from discontinuous coefficients.

The remedy is "compact support", or more generally, fast decay of the dual basis vectors. If $\mathbf{w}^i$ is zero or very small far away from x_i, the coefficient a_i will be completely or mostly independent of the values of f far away—and thus unaffected by distant singularities. Where the function is smooth, the high resolution coefficients will be small.

Wavelets are an attractive class of multi-resolution bases with this compact support property. They also are constructed to allow very fast transformation algorithms, converting a signal from the standard basis to wavelet coefficients or vice versa in $O(n)$ time, and to handle noisy signals. The next chapter will deal with them more thoroughly.

1.8 Related Methods

Probably the first multi-resolution method for solving linear systems was multigrid[23]. The most basic idea behind multigrid is instead of directly solving the original problem, "restrict" the initial guess to a coarser grid, correct all the lower resolution errors there at lower cost, then project the corrected solution back to the original grid and cheaply correct the remaining high resolution errors—but do this recursively with a hierarchy of grids for greater efficiency. This works remarkably well for many problems, and is backed up with considerable theory showing optimal $O(n)$ complexity is achieved for some equations on regular grids. Extending multigrid to more challenging PDE's and to irregular meshes is a subject of current research.

This research began by considering how to improve the Wavelet Sparse Approximate Inverse proposed in [14]. As will be elaborated in the next chapter, the weakness of this method is its restriction to uniformly spaced regular grids that scale strictly by powers of two, due to using classical wavelets (such as the Daubechies D4 wavelet[18]) for the multi-resolution basis.

The hierarchical basis technique[28] is similar to the technique proposed here—in fact, it can be viewed as a special case where the mesh is well structured, the interpolation (see later) is simple linear, and the approximate inverse is trivial.

There are also classes of algebraic multi-resolution methods, where the actual mesh and PDE are forgotten and only the matrix $\mathbf{A}$ is available. Examples of this include algebraic multigrid[31], BILUM[33], and repeated red-black ILU[8].

1.9 Roadmap

Chapter two begins with a brief review of classical wavelets. The main work is an exposition of a popular construction of second generation wavelets, going into the details of the transform algorithms and presenting new ideas about construction on unstructured meshes.

The general multi-resolution approximate inverse algorithm is laid out in chapter three, complete with sections on interpolation, factored approximate inverse construction, and ordering.

Chapters four and five contain implementation details for one dimension and two dimensions respectively. After going through the discretization and the multi-resolution basis construction, test results are presented showing some of the capabilities of the algorithm.

Chapter six summarizes the main results of the thesis, and finishes with some open problems.

Chapter 2

Constructing a Wavelet Basis

2.1 Overview

From the mathematical side, wavelets and their name came about in an attempt to fix the singularity problem of the Fourier basis. Taking the full sine and cosine waves and modifying them to get compact support produces smaller abbreviated waves, or "wave-lets". From the signal processing side, quadrature mirror filters, which apply a recursive sequence of low and high-pass filters, gave a fast $O(n)$ linear transform that could handle noisy signals, implicitly defining a multi-resolution basis.

As the initial motivation for wavelets came from improving the Fourier basis or Fast Fourier Transform, classical wavelets were developed in the same context: uniform sampling (with a small multiple of a power of two sample points) in one dimension along an interval with periodic boundaries[1]. Higher dimensional wavelets on similarly structured Cartesian grids are formed as tensor products of one dimensional wavelets.

However, the periodic boundary conditions necessary for the algorithms and theory caused problems for many applications—for example, wavelet transforming a typical non-periodic photograph would implicitly find a jump discontinuity at the boundary, degrading analysis and

[1]Just as with Fourier series, uniform sampling along the infinite real line is also studied but of course is of less practical interest, and will not be considered here.

compression nearby. The restriction to powers of two and uniform sampling, and the simplistic tensor product approach to higher dimensions, similarly grew inconvenient. While developments in classical wavelet theory could fix some of these problems, it became clear that a new approach was required.

The general term for these new methods is second generation wavelets: bases that preserve the multi-resolution, compact support, fast transform, and noise tolerance properties but that can be applied on irregular multi-dimensional domains with all kinds of boundary conditions. The popular approach used in this research is the lifting scheme[34]; other possibilities include Harten's work[1].

2.2 Classical Wavelets

Classical wavelets, described in [18] for example, are constructed from two functions on the real line, the scaling function $\phi(x)$ and the wavelet function $\psi(x)$. For this simple review, it is assumed they are zero outside of the interval $[0,\ 2N-1)$, where N is some positive integer. Define their periodic translates and dyadic (power of two) dilates:

$$\begin{aligned} \phi_j^i(x) &= \phi(2^i x - j \bmod 2^i q) \\ \psi_j^i(x) &= \psi(2^i x - j \bmod 2^i q) \end{aligned}$$

for some integer $q \geq 2N-1$. The modulo operation reduces the argument to a number in the interval $[0,\ 2^i q)$ by subtracting multiples of $2^i q$, making the functions q-periodic.

Biorthogonal wavelets also implicitly have dual functions $\widetilde{\phi}(x)$ and $\widetilde{\psi}(x)$, along with their translates and dilates, for the dual basis. To keep things simple, I shall assume that an orthogonal basis is constructed, so $\widetilde{\phi}(x) = \phi(x)$ and $\widetilde{\psi}(x) = \psi(x)$.

The lowest possible resolution, scaling level 0, is provided by the functions

$$\phi_0^0(x), \quad \phi_1^0(x), \quad \ldots, \quad \phi_{q-1}^0(x)$$

The next resolution, wavelet level 0, is given by

$$\psi_0^0(x), \quad \psi_1^0(x), \quad \ldots, \quad \psi_{q-1}^0(x)$$

Higher resolutions consist of dilates of the wavelet function—wavelet level i contains the $2^i q$ functions

$$\psi_0^i(x), \quad \ldots, \quad \psi_{2^i q-1}^i(x)$$

For $n = 2^k q$ sample points, the basis stops at wavelet level $k-1$ for a total of $q + q + 2q + \cdots + 2^{k-1}q = 2^k q = n$ discretized basis functions. For orthogonal wavelets these functions should all be orthogonal, both in the continuous and discrete settings.

The multi-resolution property comes from the dilations: level i functions handle features with variations on a length scale of $O(2^{-i})$. Compact support comes from restricting $\psi(x) = 0$ outside $[0,\ 2N-1)$ so that $\psi_j^i(x)$ is nonzero only in the length $2^{-i}(2N-1)$ interval starting at $2^{-i}j$.

The fast transforms are derived from the dilation equations:

$$\begin{aligned} \phi_0^0(x) &= \sum_{j=0}^{2N-1} a_j \phi_j^1(x) \\ \psi_0^0(x) &= \sum_{j=0}^{2N-1} b_j \phi_j^1(x) \end{aligned}$$

for some constant coefficients $a_0, \ldots, a_{2N-1}$ and $b_0, \ldots, b_{2N-1}$ to be determined. Assume the discrete input signal f has the continuous form

$$f(x) = \sum_{i=0}^{2^k q-1} f_i\, \phi_i^k(x)$$

The level $k-1$ wavelet coefficients, δ_j^{k-1} for basis function $\psi_j^{k-1}(x)$, can be easily computed by:

$$\begin{aligned} \delta_j^{k-1} &= \int_0^q f(x)\psi_j^{k-1}(x)\,dx \\ &= \int_0^q \left(\sum_i f_i \phi_i^k(x)\right) \psi_j^{k-1}(x)\,dx \\ &= \sum_i f_i \int_0^q \phi_i^k(x)\psi_j^{k-1}(x)\,dx \\ &= \sum_i f_i \int_0^q \phi_i^k(x) \left(\sum_{r=0}^{2N-1} b_r \phi_{2j+r}^k\right) dx \end{aligned}$$

$$
\begin{aligned}
&= \sum_i f_i \sum_{r=0}^{2N-1} b_r \int_0^q \phi_i^k(x)\phi_{2j+r}^k \, dx \\
&= \sum_{r=0}^{2N-1} b_r f_{2j+r}
\end{aligned}
$$

using orthogonality. Each coefficient then takes $2N$ flops[2] to compute. Temporary scaling level $k-1$ coefficients, μ_j^{k-1} for $\phi_j^{k-1}(x)$, can similarly be computed as:

$$
\mu_j^{k-1} = \sum_{r=0}^{2N-1} a_r f_{2j+r}
$$

with $2N$ flops. From these scaling level $k-1$ coefficients both sets of level $k-2$ coefficients can be computed:

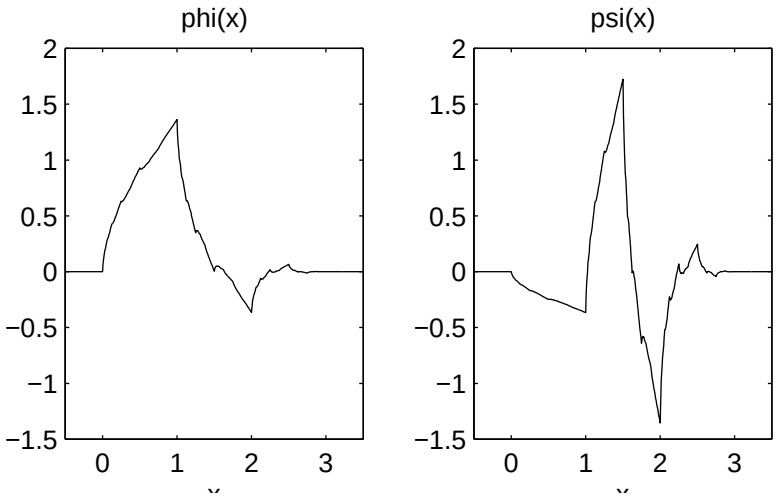
$$
\begin{aligned}
\mu_j^{k-2} &= \sum_{r=0}^{2N-1} a_r \mu_{2j+r}^{k-1} \\
\delta_j^{k-2} &= \sum_{r=0}^{2N-1} b_r \mu_{2j+r}^{k-1}
\end{aligned}
$$

and the process recursively continues until the level 0 coefficients are found. Including the temporary scaling levels, there are a little less than $2n$ coefficients computed at $2N$ flops a-piece, giving a total run-time of about $4Nn$ for the forward transform. Typically N is very small, so this is effectively $O(n)$. Note also that at each level, all coefficients can be computed independently in parallel, allowing an optimum parallel complexity of $O(\log n)$. Using orthogonality, a similar inverse transform algorithm can be derived with the same complexity.

From the signal processing viewpoint, the a coefficients define a low-pass filter, blurring out the high resolution components of the signal and keeping just the smoother low resolution part. This is what gives the noise tolerance property: if a smooth signal is contaminated by high-resolution noise, the low-pass filter cuts it out so lower levels only see the underlying smooth signal without random artifacts, and can then properly process it. This concept is made precise by the idea of moment preservation. The j'th moment of a function f is the value $\int x^j f(x)\, dx$, a generalization of the average value (when $j = 0$). A faithful lower resolution representation

[2]Flop stands for floating point operation. A multiply and add are traditionally counted together as one flop.

Figure 2.1: The D4 scaling and wavelet functions



of the original signal is ensured by requiring that the first s moments are preserved:

$$\int x^j \left(\sum_i \mu_j^{k-2} \phi_j^{k-2}(x) \right) dx = \int x^j f(x)\, dx \quad \text{for} \quad j = 0, \ldots, s-1$$

Applying the dilation equation reduces this to s conditions on the b coefficients.

The b coefficients define a complementary high-pass filter, differencing out the lower resolution components and isolating the high resolution part. This can be made precise by the idea of vanishing moments, requiring that the wavelet coefficients of the ideally smooth functions 1, x, x^2, …, x^s be zero. From orthogonality, this is equivalent to the moment preservation condition on the a's.

Other conditions to be imposed on the a and b coefficients result from requiring orthogonality in this case, and possibly other desirable features. Of course, more conditions require more coefficients, i.e. larger N, which both slows down the transform algorithms and makes the support larger and hence the basis more susceptible to damage from singularities. One common choice, used in [14] for example, is the D4 wavelet of Daubechies[18], an orthogonal basis with $N = 2$ and two preserved moments. See figure 2.1 for a picture of its fundamental functions.

At the core of classical wavelet theory are the dilation equations, which unfortunately are

also the root cause of all the restrictions: powers of two, periodic boundaries, uniform sampling, etc. The key to second generation wavelets is to focus instead on the transform algorithm.

2.3 The Lifting Scheme

The lifting scheme[34] proposed by Sweldens is a way of constructing biorthogonal second generation wavelets—ones that retain the essential properties of multi-resolution, compact support, fast transforms, and noise tolerance, but not the limitation to such regular domains.

The core of the lifting scheme is its transform algorithms, rather than what the basis functions actually are. Of course, any invertible linear transformation can be viewed as a change-of-basis, and it is possible to recover the j'th basis function simply by inverting $\mathbf{e}^j$ (and the j'th dual basis function by transforming $\mathbf{e}^j$).

Figures 2.2 and 2.3 give the general forms of the forward transform algorithm and inverse transform algorithm respectively. Note that the resolution levels are in reverse from the following section on classical wavelets, so level 0 is the highest resolution.

2.4 The Multi-Resolution Property

The lifting scheme is a natural way of arranging for small wavelet coefficients where the function is smooth—presumably at those points, the prediction will be very accurate, so the prediction error γ^j will be close to zero. This can also be naturally interpreted as the multi-resolution property: the coarsest level λ^j gives the lowest resolution view of the function, with γ^j giving the next higher resolution details that were missed, then γ^{j-1} the further level of resolution details that were missed, etc., finishing with γ^1 giving the highest resolution details. Smooth functions will naturally have near negligible high resolution details, hence small γ coefficients.

In order for this to be successful, the $\mathbf{P}$ at each level must be accurate of course. One of the fundamental requirements for the partition into fine and coarse nodes is then that each fine node should be easily predicted from the coarse nodes. Normally this will mean that there should be coarse nodes in close proximity to every fine node—the fine and coarse nodes must

Figure 2.2: The forward transform for the lifting scheme.

- Start with the function values $f_1, \ldots, f_n$ at sample points $x_1, \ldots, x_n$.
- Let $\lambda_i^0 = f_i$ for all i, $\mathcal{C}^0 = \{x_1, \ldots, x_n\}$, and $j = 0$.
- Begin loop:
  - Split up the sample points $\mathcal{C}^j$ into two disjoint subsets, the fine nodes $\mathcal{F}^{j+1}$ and the coarse nodes $\mathcal{C}^{j+1}$.
  - Predict λ_F^j, the values at the fine nodes, from λ_C^j, the values at the coarse nodes, with some linear prediction operator $\mathbf{P}$: $\lambda_F^j \approx \mathbf{P}\lambda_C^j$.
  - Store the wavelet coefficient $\gamma_i^{j+1} = \lambda_i^j - (\mathbf{P}\lambda_C^j)_i$ for each fine node $x_i \in \mathcal{F}^{j+1}$.
  - Update the value at each coarse node by $\lambda_i^{j+1} = \lambda_i^j + (\mathbf{U}\gamma^{j+1})_i$ for each $x_i \in \mathcal{C}^{j+1}$ so that the required moments will be preserved. This update operator $\mathbf{U}$ must also be linear.
  - If $|\mathcal{C}^{j+1}|$ is small enough, below some constant, break out of the loop. Otherwise, set $j \leftarrow j + 1$ and continue.
- Return λ^j from the coarsest level along with the wavelet coefficients $\gamma^1, \ldots, \gamma^j$ from each level.

Figure 2.3: The inverse transform for the lifting scheme.

- Start with λ^j and the wavelet coefficients $\gamma^1, \ldots, \gamma^j$.
- Begin loop:
  - Reconstruct λ_C^{j-1} at the coarse nodes by $\lambda_i^{j-1} = \lambda_i^j - (\mathbf{U}\gamma^j)_i$ for each $x_i \in \mathcal{C}^j$.
  - Reconstruct λ_F^{j-1} at the fine nodes by $\lambda_i^{j-1} = \gamma_i^j + (\mathbf{P}\lambda_C^{j-1})_i$ for each $x_i \in \mathcal{F}^j$.
  - Continue with $j \leftarrow j - 1$ until $j = 1$.
- Return $f_i = \lambda_i^0$ for all i.

be fully intermingled. This might be achieved in one dimension, for example, by selecting every even node to be coarse and every odd node to be fine.[3] Continuing in one dimension, $\mathbf{P}$ could then be defined to do linear interpolation between the two coarse nodes surrounding each fine node. Usually $\mathbf{P}$ is defined to do polynomial interpolation through the surrounding coarse nodes, which corresponds precisely to vanishing moments as discussed in the previous section.

2.5 Compact Support

Compact support is arranged by making each $\mathbf{P}$ and $\mathbf{U}$ a sparse matrix, so that the prediction at each fine node and the update at each coarse node depend upon only a small number of nearby nodes. For example, linear-interpolating $\mathbf{P}$ in one dimension satisfies this by only using the two surrounding coarse nodes to predict at the fine node between. The basic goal of compact support, containing the damage done by singularities, is naturally achieved in this way: although the prediction will likely be inaccurate near the singularity, the wavelet coefficients γ further away are completely independent of the function values at the singularity and thus cannot be adversely affected.

2.6 Fast Transforms

Choosing the $\mathbf{P}$ and $\mathbf{U}$ operators very sparse also makes the transform algorithms fast. For example, if each fine node is predicted from at most q nearby coarse nodes, taking at most q flops, then the operation $\mathbf{P}\lambda_C^j$ will take at most $|\mathcal{F}^{j+1}|q$ flops. Adding up the operations for the entire transform, noting that the fine node sets are disjoint, we have a strict upper bound of nq flops for the predict operations. Similarly if each coarse node is updated from at most q nearby fine nodes, the operation $\mathbf{U}\gamma^{j+1}$ will take at most $|\mathcal{C}^{j+1}|q$ flops. The coarse node sets are nested, not disjoint, but if we assume that the number of coarse nodes is at least halved at each level ($|\mathcal{C}^{j+1}| \leq |\mathcal{C}^j|/2$), then from the geometric sum we still have an upper bound of nq flops for all update operations. Therefore under these assumptions both transform algorithms run in $O(n)$ time, in fact bounded by $2nq$ flops, twice as fast as classical wavelets.

[3] Based on this, Swelden's original presentation of the scheme actually uses the terms even and odd instead of coarse and fine throughout

2.7 Noise Tolerance

Just as with classical wavelets, the key to noise tolerance is moment preservation. The update operator $\mathbf{U}$ is chosen so that approximating the input function by zeroing out wavelet coefficients preserves its moments. Particularly on a multi-dimensional irregularly sampled domain, the classical definition of moment isn't necessarily meaningful, thus I forward the notion of generalized moments $m_j = \int_\Omega \sigma_j(x) f(x)\, dx$ for some smooth moment kernel functions $\sigma_j(x)$. (The classical 1D choice is $\sigma_j(x) = x^j$.) The discrete form is then:

$$m_j = \sum_{i=1}^{n} S_{ji} f_i$$

for some appropriate discretization $S_{ji} \approx \int_{C_i} \sigma_j(x)\, dx$ with C_i a small cell around x_i. Letting $\mathbf{m}$ be the vector of moments, this can be written as

$$\mathbf{m} = \mathbf{S}\mathbf{f}$$

At each transform step, the original function is λ^j and the coarsened function is:

$$\begin{aligned} \lambda^{j+1} &= \lambda_C^j + \mathbf{U}\gamma^{j+1} \\ &= \lambda_C^j + \mathbf{U}(\lambda_F^j - \mathbf{P}\lambda_C^j) \\ &= (\mathbf{I} - \mathbf{U}\mathbf{P})\lambda_C^j + \mathbf{U}\lambda_F^j \end{aligned}$$

The approximate function reconstructed without wavelet coefficients γ^{j+1} is given by:

$$\begin{aligned} \widetilde{\lambda}_C^j &= \lambda^{j+1} - \mathbf{U}\mathbf{0} \\ &= (\mathbf{I} - \mathbf{U}\mathbf{P})\lambda_C^j + \mathbf{U}\lambda_F^j \\ \widetilde{\lambda}_F^j &= \mathbf{0} + \mathbf{P}\widetilde{\lambda}_C^j \\ &= \mathbf{P}(\mathbf{I} - \mathbf{U}\mathbf{P})\lambda_C^j + \mathbf{P}\mathbf{U}\lambda_F^j \end{aligned}$$

Splitting up the moment kernel matrix $\mathbf{S}$ into $\mathbf{S}_F$ for the fine node columns and $\mathbf{S}_C$ for the coarse node columns, we then require:

$$\begin{aligned} \mathbf{S}\lambda^j &= \mathbf{S}\widetilde{\lambda}^j \\ \mathbf{S}_F\lambda_F^j + \mathbf{S}_C\lambda_C^j &= \mathbf{S}_F\widetilde{\lambda}_F^j + \mathbf{S}_C\widetilde{\lambda}_C^j \end{aligned}$$

$$
\begin{aligned}
\mathbf{S}_F\lambda_F^j + \mathbf{S}_C\lambda_C^j &= \mathbf{S}_F\left(\mathbf{P}(\mathbf{I}-\mathbf{UP})\lambda_C^j + \mathbf{PU}\lambda_F^j\right) \\
&\quad + \mathbf{S}_C\left((\mathbf{I}-\mathbf{UP})\lambda_C^j + \mathbf{U}\lambda_F^j\right) \\
(\mathbf{S}_F - \mathbf{S}_F\mathbf{PU} - \mathbf{S}_C\mathbf{U})\lambda_F^j &= (\mathbf{S}_F\mathbf{P} - \mathbf{S}_F\mathbf{PUP} - \mathbf{S}_C\mathbf{UP})\lambda_C^j \\
(\mathbf{S}_F - \mathbf{S}_F\mathbf{PU} - \mathbf{S}_C\mathbf{U})\lambda_F^j &= (\mathbf{S}_F - \mathbf{S}_F\mathbf{PU} - \mathbf{S}_C\mathbf{U})\mathbf{P}\lambda_C^j
\end{aligned}
$$

This is true automatically if the prediction is perfect, i.e. the function perfectly fits our notion of smoothness. However, this should be true for *any* function, irregardless of the independent values of λ_F^j and λ_C^j, so we must have:

$$
\begin{aligned}
\mathbf{S}_F - \mathbf{S}_F\mathbf{PU} - \mathbf{S}_C\mathbf{U} &= \mathbf{0} \\
(\mathbf{S}_F\mathbf{P} + \mathbf{S}_C)\mathbf{U} &= \mathbf{S}_F
\end{aligned}
$$

This equation coupled with the sparsity constraints should determine the entries of $\mathbf{U}$.

Each column of $\mathbf{U}$ can be computed independently, and actually only involves solving a small submatrix of $\mathbf{S}_F\mathbf{P} + \mathbf{S}_C$ thanks to the sparsity constraint. Note that for the submatrix to be invertible it must be square, so the number of γ coefficients used to update each coarse node must equal the number of preserved moments.

Generalizing the notion of moment preservation even further, I here propose constructing the small system independently for each coarse node, allowing different sets of moments to be *locally* preserved at different places. This might be desirable, for example, if it is inconvenient to arrange for all coarse nodes to use the same number of γ coefficients in the update step.

2.8 Additional Algorithm Features

Another nice feature of these algorithms is that they can work in place: γ^{j+1} can overwrite λ_F^j and λ^{j+1} can overwrite λ_C^j in the forward transform; vice versa for the inverse transform. In particular, this is irregardless of the order in which the fine nodes are predicted and the order in which the coarse nodes are updated.

These algorithms are also naturally parallel. Not only can each fine node be predicted simultaneously and each coarse node updated simultaneously, but assuming that only nearby

neighbours are used for both operations only a small amount of local communication between nodes is needed. Again assuming that the number of coarse nodes is halved at each level, there are only $O(\log n)$ steps in the algorithms, so with $O(n)$ processors it is theoretically possible to do the transforms in $O(\log n)$ time.

2.9 The Matrix Formulation

The transform algorithms can also be described with matrix notation. Assume that the nodes are ordered from finest to coarsest, namely with $\mathcal{F}^1$ first, then $\mathcal{F}^2$, ..., then $\mathcal{F}^j$, and finally $\mathcal{C}^j$ last. Let $\mathbf{P}^i$ and $\mathbf{U}^i$ be the prediction and update operators at step i. Then the forward transform can be written with the following product:

$$\begin{pmatrix} \gamma^1 \\ \vdots \\ \gamma^j \\ \lambda^j \end{pmatrix} = \mathbf{M}\mathbf{f} = \mathbf{M}_j \cdots \mathbf{M}_2 \mathbf{M}_1 \begin{pmatrix} \mathbf{f}_{F1} \\ \vdots \\ \mathbf{f}_{Fj} \\ \mathbf{f}_{Cj} \end{pmatrix}$$

Step i of the transform is given by $\mathbf{M}_i$:

$$\left(\begin{array}{c} \gamma^1 \\ \vdots \\ \hline \gamma^i \\ \lambda^i \end{array}\right) = \underbrace{\left(\begin{array}{c|cc} \mathbf{I} & & \\ \hline & \mathbf{I} & \\ & \mathbf{U}^i & \mathbf{I} \end{array}\right)\left(\begin{array}{c|cc} \mathbf{I} & & \\ \hline & \mathbf{I} & -\mathbf{P}^i \\ & & \mathbf{I} \end{array}\right)}_{\mathbf{M}_i} \left(\begin{array}{c} \gamma^1 \\ \vdots \\ \hline \lambda_F^{i-1} \\ \lambda_C^{i-1} \end{array}\right)$$

The inverse transform can similarly be written:

$$\mathbf{f} = \mathbf{M}^{-1} \begin{pmatrix} \gamma^1 \\ \vdots \\ \gamma^j \\ \lambda^j \end{pmatrix} = \mathbf{M}_1^{-1} \mathbf{M}_2^{-1} \cdots \mathbf{M}_j^{-1} \begin{pmatrix} \gamma^1 \\ \vdots \\ \gamma^j \\ \lambda^j \end{pmatrix}$$

where the inverse transform at step i is $\mathbf{M}_i^{-1}$:

$$\left(\begin{array}{c} \gamma^1 \\ \vdots \\ \hline \lambda_F^{i-1} \\ \lambda_C^{i-1} \end{array}\right) = \underbrace{\left(\begin{array}{c|cc} \mathbf{I} & & \\ \hline & \mathbf{I} & \mathbf{P}^i \\ & & \mathbf{I} \end{array}\right)\left(\begin{array}{c|cc} \mathbf{I} & & \\ \hline & \mathbf{I} & \\ & -\mathbf{U}^i & \mathbf{I} \end{array}\right)}_{\mathbf{M}_i^{-1}} \left(\begin{array}{c} \gamma^1 \\ \vdots \\ \hline \gamma^i \\ \lambda^i \end{array}\right)$$

Of course, these matrices should be treated as sparse matrices, i.e. only the nonzeros and their locations should be saved. Standard sparse matrix multiplication routines can then be used to do the wavelet transform efficiently.

As will be the case in section 3.6, these algorithms can furthermore operate in sparse-sparse mode, where the vector to be transformed is sparse too. The matrix formulation is then simplest to use, since again standard sparse-sparse multiply routines can be used.

Chapter 3

The General Algorithm

3.1 Overview

As the preceding chapters have suggested, the basic idea behind the multi-resolution approximate inverse is to construct, via the lifting scheme, a multi-resolution basis for compressing the discrete Green's function. Later chapters will deal with the details of basis construction for one or two dimensional problems; this chapter will cover the details that are independent of dimension.

3.2 Using the Multi-Resolution Basis

The goal is to compress $\mathbf{A}^{-1}$, obtaining an accurate but highly sparse approximate inverse from just the large coefficients in its multi-resolution representation. However, $\mathbf{A}^{-1}$ is unknown of course, so this is not just a simple matter of applying the transform algorithm.

Recall that $\mathbf{A}^{-1}$ is the discrete version of the Green's function $G(x, y)$, which is defined on $\Omega \times \Omega$. It is then natural to look for a basis Π on (the discretized form of) $\Omega \times \Omega$ that is a tensor product of two bases α and β on Ω: $\Pi = \alpha \otimes \beta$. Each element $\mathbf{p} \in \Pi$ is then a separable function $p_{ij} = a_i b_j$ with $\mathbf{a} \in \alpha$ and $\mathbf{b} \in \beta$. (In the continuous form, $p(x, y) = a(x)b(y)$.)

Let the bases be $\alpha = \{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^n\}$, $\beta = \{\mathbf{b}^1, \mathbf{b}^2, \ldots, \mathbf{b}^n\}$, and their tensor product $\Pi = \{\mathbf{p}^{11}, \mathbf{p}^{21}, \ldots, \mathbf{p}^{n1}, \ldots, \mathbf{p}^{nn}\}$, where $p_{ij}^{kl} = a_i^k b_j^l$. To express the discrete Green's function in the basis Π, find coefficients Q_{kl} so that:

$$\begin{aligned} A_{ij}^{-1} &= \sum_{k=1}^{n}\sum_{l=1}^{n} Q_{kl} p_{ij}^{kl} \\ &= \sum_{k=1}^{n}\sum_{l=1}^{n} Q_{kl} a_i^k b_j^l \\ &= \sum_{k=1}^{n} a_i^k \sum_{l=1}^{n} Q_{kl} b_j^l \end{aligned}$$

In this last expression, viewing j as fixed, observe $\sum_{l=1}^{n} Q_{kl} b_j^l$ is the k'th coefficient of the α basis representation of column j of $\mathbf{A}^{-1}$. Then letting j vary again, Q_{kl} is the l'th coefficient of the β basis representation of those k'th coefficients. If $\mathbf{M}_\alpha$ is the forward transform operator from the standard basis to the α basis, $\mathbf{M}_\beta$ is the forward transform to β, and the Q_{kl} are arranged as an $n \times n$ matrix, this can be written more clearly as

$$Q_{kl} = (\mathbf{M}_\beta(\mathbf{M}_\alpha \mathbf{A}^{-1})^T)_{kl}$$

or simply

$$\mathbf{Q} = \mathbf{M}_\alpha \mathbf{A}^{-1} \mathbf{M}_\beta^T$$

which is equivalent to

$$\mathbf{A}^{-1} = \mathbf{M}_\alpha^{-1} \mathbf{Q} \mathbf{M}_\beta^{-T}$$

Of course, the same result can be obtained by first viewing i as fixed and the rows of $\mathbf{A}^{-1}$ being compressed with β.

The preconditioner is going to be a compressed form of $\mathbf{A}^{-1}$, where small Π coefficients have been dropped, i.e. $\mathbf{M}_\alpha^{-1} \widetilde{\mathbf{Q}} \mathbf{M}_\beta^{-T}$ for $\widetilde{\mathbf{Q}}$ a sparse approximation to $\mathbf{Q}$. If the Π basis does a good job, a very sparse yet high quality approximation will be possible. Notice that

$$\mathbf{Q} = \mathbf{M}_\alpha \mathbf{A}^{-1} \mathbf{M}_\beta^T = (\mathbf{M}_\beta^{-T} \mathbf{A} \mathbf{M}_\alpha^{-1})^{-1}$$

so $\widetilde{\mathbf{Q}}$ is in fact a sparse approximate inverse for $\mathbf{M}_\beta^{-T} \mathbf{A} \mathbf{M}_\alpha^{-1}$. All of these matrices are known, so we now have a tractable proposition. The general outline of the algorithm is given in figure 3.1.

Figure 3.1: The multi-resolution approximate inverse algorithm.

- Compute the transform coefficients for α and β through the lifting scheme.
- Compute a sparse approximate inverse $\widetilde{\mathbf{Q}} \approx (\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1})^{-1}$.
- The preconditioner is then $\mathbf{M}_\alpha^{-1}\widetilde{\mathbf{Q}}\mathbf{M}_\beta^{-T} \approx \mathbf{A}^{-1}$.

There is some flexibility in choosing the preconditioned system. In exact arithmetic with $\widetilde{\mathbf{Q}} = \mathbf{Q}$, all of $(\mathbf{M}_\alpha^{-1}\widetilde{\mathbf{Q}}\mathbf{M}_\beta^{-T})\mathbf{A}$, $(\widetilde{\mathbf{Q}}\mathbf{M}_\beta^{-T})\mathbf{A}(\mathbf{M}_\alpha^{-1})$, $(\mathbf{M}_\beta^{-T})\mathbf{A}(\mathbf{M}_\alpha^{-1}\widetilde{\mathbf{Q}})$, and $\mathbf{A}(\mathbf{M}_\alpha^{-1}\widetilde{\mathbf{Q}}\mathbf{M}_\beta^{-T})$ are equal to the identity—and if $\widetilde{\mathbf{Q}}$ is in factored form, even more possibilities exist. The choice of which is best when $\widetilde{\mathbf{Q}} \neq \mathbf{Q}$ should generally be made according to how $\widetilde{\mathbf{Q}}$ is constructed; see section 3.6 for details on the choice made for this thesis.

The preconditioned system must be non-singular, thus in particular $\widetilde{\mathbf{Q}}$ must be non-singular. At the same time, we want $\widetilde{\mathbf{Q}}$ to be very sparse; an obvious goal is then to make $\mathbf{Q}$ as close as possible to a diagonal matrix, with diagonal entries much larger than off-diagonal entries. (This is especially the case for factored approximate inverse algorithms without pivoting.) Intuitively speaking this should naturally be the case, since the Green's function should be smooth off the diagonal—allowing very small off-diagonal $\mathbf{Q}$ coefficients—but should have a singularity along the diagonal caused by the Dirac delta—giving large diagonal $\mathbf{Q}$ coefficients. The next few sections will outline how to best achieve this.

Notice that the bases α and β can be constructed completely independently; not only can the choices of prediction and update operators be different, but the hierarchy of fine/coarse nodes can be completely different too. Later this flexibility in choosing different predictions will be exploited, but throughout the rest of the thesis it will be assumed that the hierarchies are the same. The first advantage of this restriction is that it is possible to speak about a coarse node unambiguously; this much simplifies analysis of the algorithm. Also important for ordering the nodes prior to computing a factored approximate inverse, some degree of symmetry is preserved—if $\mathbf{A}$ is structurally symmetric, and the prediction and update operators for α and β have the same structure, then $\mathbf{M}_\beta^{-T}A\mathbf{M}_\alpha^{-1}$ is structurally symmetric too.

3.3 PDE-Interpolation

Examine more closely what the α basis transform does in compressing the columns of $\mathbf{A}^{-1}$. Since $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$, the j'th column of $\mathbf{A}^{-1}$ is the solution of $\mathbf{A}\mathbf{u} = \mathbf{e}^j$, which is the discretized form of $\mathcal{L}u(x) = \delta(x - x_j)$ (cf. section 1.6). Thus each column being transformed satisfies $\mathcal{L}u = 0$ everywhere except at the diagonal.

A better choice than the usual polynomial interpolation for P_α now presents itself, what I call "PDE-interpolation". When predicting the value at fine node x_i from nearby coarse nodes x_{j_1}, ..., x_{j_k}, treat it as a small PDE problem $\mathcal{L}u = 0$ with an unknown at x_i and specified "boundary" values at $x_{j_1}, \ldots, x_{j_k}$. After defining a small mesh on these nodes, the discretization routine can be called to give the linear equation approximating $\mathcal{L}u = 0$ at the fine node, and this can be immediately solved since we know the values at all the other points. In particular, if the discretization at x_i is

$$0 = \mathcal{L}u \approx a_{ii}u_i + a_{ij_1}u_{j_1} + \cdots + a_{ij_k}u_{j_k}$$

then the prediction should be

$$u_i \approx -\frac{a_{ij_1}}{a_{ii}}u_{j_1} - \cdots - \frac{a_{ij_k}}{a_{ii}}u_{j_k}$$

Of course, at boundary nodes the boundary condition should be discretized rather than the PDE.

Similar arguments can be made for β, only since $\mathbf{A}^T\mathbf{A}^{-T} = \mathbf{I}$ the rows of $\mathbf{A}^{-1}$ are discrete solutions of the *adjoint* problem. If $\mathcal{L}$ is not self-adjoint, this makes a crucial difference. $\mathbf{P}_\beta$ should be generated by discretizing $\mathcal{L}u$ at the fine node and neighbouring coarse nodes, transposing the resulting small matrix to get the discrete adjoint operator, and *then* solving for the fine node value.

For nearly self-adjoint problems, i.e. those with relatively weak convection, the extra storage spent on distinct $\mathbf{P}_\alpha$'s and $\mathbf{P}_\beta$'s might not be worth it, and the symmetrized equation should be used instead. However, testing results in later chapters show the benefit of choosing $\mathbf{P}_\alpha \neq \mathbf{P}_\beta$ for strong convection PDE's.

Note that constructing the PDE-interpolation is only a constant factor more expensive than linear interpolation—once the neighbouring nodes have been found, which can be done in $O(1)$

amortized time[12], there is only $O(1)$ work left to do. Further note that the local mesh and discretization need only be computed once; the coefficients $-a_{ij_1}/a_{ii}$, ..., $-a_{ij_k}/a_{ii}$ can then be stored in $\mathbf{P}_\alpha$ for future use. The construction costs can be further amortized if there are many solves to be done, and even if the PDE coefficients change between solves (as in a non-linear problem or some time-dependent problems) at least the local mesh construction costs may be amortized.

Reassuringly, PDE-interpolation often reduces to polynomial interpolation when $\mathcal{L}$ is the Laplacian operator ∇^2. In 1D for example, suppose the fine node is at point x_i with coarse neighbours x_{i-1} to the left and x_{i+1} to the right. The normal second order discretization of $\mathcal{L}u = u'' = 0$ at x_i is:

$$\frac{2u_{i+1}}{(x_{i+1}-x_i)(x_{i+1}-x_{i-1})} - \frac{2u_i}{(x_{i+1}-x_i)(x_i-x_{i-1})} + \frac{2u_{i-1}}{(x_i-x_{i-1})(x_{i+1}-x_{i-1})} = 0$$

Solving for u_i gives:

$$u_i = \left(\frac{x_i-x_{i-1}}{x_{i+1}-x_{i-1}}\right) u_{i+1} + \left(\frac{x_{i+1}-x_i}{x_{i+1}-x_{i-1}}\right) u_{i-1}$$

which is just linear interpolation.

The same thing happens in 2D for piecewise linear finite elements on triangles. If the fine node is inside a triangle of three coarse nodes, linear interpolation in this triangle is equivalent to splitting the triangle into three subtriangles, constructing the linear finite element discretization, and solving for the fine node.

Notice that it is important that the discretization not be limited to fine meshes for stability and accuracy; the interpolation will need to be carried out at the coarser levels where the distance between nodes is much larger than in the original mesh. For example, a scheme like upwinding should be used for the convection term, and discontinuous coefficients must be handled physically correctly (e.g. with harmonic means in one dimension). Ideally the PDE coefficients themselves should be coarsened along with the mesh in some homogenization procedure—perhaps taking appropriate averages of the coefficients at nearby fine nodes. I leave this coefficient homogenization problem for future research.

The expense of setting up these local meshes, even when amortized, might not be worthwhile for simple problems, but the benefit should be clear for tough PDE's. For example, if

there is strong convection, the naturally centrally weighted polynomial interpolants will give equal weight to downstream values—a clear mistake—when the PDE-interpolant correctly emphasizes the upstream values. If the diffusion coefficient is discontinuous then solutions won't be smooth at the discontinuities, violating the assumption underlying polynomial interpolants, but appropriate PDE-interpolation should still work.

As an addendum, other methods for improving on polynomial interpolation, algebraic in nature, have arisen in multigrid, such as the energy minimization approach from [13] or "Blackbox Multigrid" in [2]. It would be interesting to compare the performance and robustness of these interpolations—whether the discretization approximations or the algebraic approximations are better.

3.4 Forgetting Moments

It turns out that taking $\mathbf{U} = \mathbf{0}$, so moments are *not* preserved, appears to be the best choice. The next chapter illustrates this with numerical experiments; this section provides a theoretical justification.

In signal processing, the noise tolerance provided by preserving moments is crucial. The functions being transformed often have random fluctuations due to background noise or errors in the sampling process, so added to the underlying smooth signal is a high resolution error. Without an update step in the transform the function values at the coarse nodes are unchanged at lower resolutions, and so the high resolution error is carried down into a low resolution error. Then at all levels the error would cause problems for the prediction, so despite the fact that the signal really is smooth at lower resolutions, the lower resolution wavelet coefficients won't be small. The introduction of a sparse update step means moments are preserved locally, maintaining local average values and thus smoothing out the function for lower resolutions. This damps out the high resolution error so it can only harm the high resolution wavelet coefficients.

However, in this application there should be no high resolution fluctuations. The accuracy of a solution generally is related to the size of its derivatives; small-scale oscillations would make those large, indicating that the discretization is of little value and probably suffers from instability. Therefore the real need for an update step is gone. It is true that some indefinite problems or problems with rapidly fluctuating coefficients will inherently give rise to solutions with

small-scale oscillations, in which case a multi-resolution method is bound to meet difficulties at low resolutions, though in the latter case a coefficient homogenization procedure might help. Perhaps here multi-resolution methods are simply not suitable; I leave this for future research.

Thus in the cases of interest, though it appears not to be crucially important, can the update step still be of some use?

Write out the first step of the transformation with fine nodes ordered before coarse nodes and $\mathbf{A}^{-1}$ decomposed as $\left(\begin{smallmatrix}\mathbf{B}&\mathbf{C}\\ \mathbf{D}&\mathbf{E}\end{smallmatrix}\right)$:

$$\begin{pmatrix} \mathbf{I} & \\ \mathbf{U}_\alpha & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & -\mathbf{P}_\alpha \\ & \mathbf{I} \end{pmatrix} \overbrace{\begin{pmatrix} \mathbf{B} & \mathbf{C} \\ \mathbf{D} & \mathbf{E} \end{pmatrix}}^{\mathbf{A}^{-1}} \begin{pmatrix} \mathbf{I} & \\ -\mathbf{P}_\beta^T & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{U}_\beta^T \\ & \mathbf{I} \end{pmatrix}$$

Carrying out the prediction step gives:

$$\begin{pmatrix} \mathbf{I} & \\ \mathbf{U}_\alpha & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{B} - \mathbf{P}_\alpha\mathbf{D} - \mathbf{C}\mathbf{P}_\beta^T + \mathbf{P}_\alpha\mathbf{E}\mathbf{P}_\beta^T & \mathbf{C} - \mathbf{P}_\alpha\mathbf{E} \\ \mathbf{D} - \mathbf{E}\mathbf{P}_\beta^T & \mathbf{E} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{U}_\beta^T \\ & \mathbf{I} \end{pmatrix}$$

Now, if the prediction operators are accurate for $\mathbf{C}$ and $\mathbf{D}$ from $\mathbf{E}$ (i.e. $\mathbf{C} - \mathbf{P}_\alpha\mathbf{E} \approx \mathbf{0}$ and $\mathbf{D} - \mathbf{E}\mathbf{P}_\beta^T \approx \mathbf{0}$), they necessarily are close to ideal PDE-interpolation, since in these off-diagonal portions of $\mathbf{A}^{-1}$ we have $\mathcal{L}u = 0$ everywhere. Then the prediction $\mathbf{P}_\alpha\mathbf{D}$ or $\mathbf{C}\mathbf{P}_\beta^T$ for $\mathbf{B}$ will be accurate except at the diagonal, where $\mathcal{L}u = 1$ instead of 0. So the prediction error roughly will be 0 away from the diagonal, and $1/a_{ii}$ (the coefficient in the rediscretization) on the diagonal. If Δ is the diagonal matrix with these coefficients on the diagonal, then $\mathbf{B} - \mathbf{P}_\alpha\mathbf{D} \approx \Delta$ and $\mathbf{B} - \mathbf{C}\mathbf{P}_\beta^T \approx \Delta$. So the scheme will approximately give:

$$\begin{pmatrix} \mathbf{I} & \\ \mathbf{U}_\alpha & \mathbf{I} \end{pmatrix} \begin{pmatrix} \Delta & \mathbf{0} \\ \mathbf{0} & \mathbf{E} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{U}_\beta^T \\ & \mathbf{I} \end{pmatrix}$$

Thus the predict step achieves exactly what is needed: a near diagonal matrix. However, this is the result if the update step is then applied:

$$\begin{pmatrix} \Delta & \Delta\mathbf{U}_\beta^T \\ \mathbf{U}_\alpha\Delta & \mathbf{E} + \mathbf{U}_\alpha\Delta\mathbf{U}_\beta^T \end{pmatrix}$$

The attractive near zero blocks that were created by the prediction have been filled in with scaled versions of the update matrices. Furthermore, the coarsened system $\mathbf{E}$ has been perturbed in a

way that won't necessarily improve later prediction, and will probably mean that it is no longer a discrete Green's function of the PDE—making PDE-interpolation useless. The problem is that there is an essential singularity on the diagonal that we *want* to keep sharp—the error in prediction at the diagonal is beneficial; the algorithm should maintain it at lower resolutions, rather than trying to blur it out with an update step.

Finally, it's clear that including an update step adds expense in storage space and transformation time, not to mention complicating analysis of the algorithm. Therefore I shall assume $\mathbf{U} = \mathbf{0}$ from now on. Since moment preservation is an essential feature of wavelets, I have adopted the name multi-resolution approximate inverse rather than wavelet approximate inverse.

3.5 Multiplying out the Transforms

A second look at the forward transform algorithm (figure 2.2) shows that without the update step, the coarsened signals λ^i are just sub-samplings of the original signal f, values unchanged. In particular then, all the λ^i's are immediately available, so the predictions are independent and may be done simultaneously.

This fact may be seen by multiplying out the matrix product form of the forward transform:

$$\mathbf{M} = \mathbf{M}_j \cdots \mathbf{M}_2 \mathbf{M}_1$$

For example, multiplying the first two steps together gives:

$$\begin{aligned}
\mathbf{M}_2\mathbf{M}_1 &= \left(\begin{array}{c|cc} \mathbf{I} & & \\ \hline & \mathbf{I} & -\mathbf{P}^2 \\ & & \mathbf{I} \end{array}\right) \left(\begin{array}{c|cc} \mathbf{I} & -\mathbf{P}^1 & \\ \hline & \mathbf{I} & \\ & & \end{array}\right) \\
&= \left(\begin{array}{c|cc} \mathbf{I} & -\mathbf{P}^1 & \\ \hline & \mathbf{I} & -\mathbf{P}^2 \\ & & \mathbf{I} \end{array}\right)
\end{aligned}$$

The off-diagonal $-\mathbf{P}$'s simply add, thanks to the diagonal identity blocks and the order of multiplication. It's simple to see how this continues, giving

$$\mathbf{M} = \left(\begin{array}{c|c|c|c|c} \mathbf{I} & -\mathbf{P}^1 & & & \\ \hline & \mathbf{I} & -\mathbf{P}^2 & & \\ \hline & & \ddots & & \\ \hline & & & \mathbf{I} & -\mathbf{P}^j \\ & & & & \mathbf{I} \end{array}\right)$$

The forward transform is now reduced to a single sparse matrix multiply.

On the other hand, the inverse transform cannot be similarly reduced. Reconstructing λ^i depends on γ^{i+1} in the inverse transform algorithm (figure 2.3) even when $\mathbf{U} = \mathbf{0}$; the steps must be done one after the other. From a matrix viewpoint, this can be seen in the fill-in that results when the inverse transform is multiplied out, caused by the reversed order of the factors. For example, multiplying $\mathbf{M}_1^{-1}\mathbf{M}_2^{-1}$ gives:

$$\underbrace{\left(\begin{array}{c|c} \mathbf{I} & \mathbf{P}^1 \\ \hline & \mathbf{I} \end{array}\right)}_{\mathbf{M}_1^{-1}} \underbrace{\left(\begin{array}{c|cc} \mathbf{I} & & \\ \hline & \mathbf{I} & \mathbf{P}^2 \\ & & \mathbf{I} \end{array}\right)}_{\mathbf{M}_2^{-1}} = \left(\begin{array}{c|c} \mathbf{I} & \mathbf{P}^1 \cdot \begin{pmatrix} \mathbf{I} & \mathbf{P}^2 \\ & \mathbf{I} \end{pmatrix} \\ \hline & \begin{array}{cc} \mathbf{I} & \mathbf{P}^2 \\ & \mathbf{I} \end{array} \end{array}\right)$$

Not only is the storage requirement increased when the matrices are multiplied out, but the time required for the inverse transform similarly increases.

However, observe that the forward transform matrix is upper triangular, so the inverse transform can be applied by the backwards substitution of a triangular solve. In fact, the inverse transform algorithm can be interpreted as doing exactly this, but with the potential for parallelism made explicit: nodes from the same level can be solved independently.

As an aside, recall from the previous section that the ideal action of the prediction steps is to reduce $\mathbf{A}^{-1}$ to near diagonal form: $\mathbf{M}_\alpha \mathbf{A}^{-1} \mathbf{M}_\beta^T$ is almost a diagonal matrix $\widetilde{\Delta}$, neglecting the

small sub-matrix of coarsest nodes. Then similarly the inverse is close to diagonal:

$$\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1} \approx \widetilde{\Delta}^{-1}$$

Since $\mathbf{M}_\beta^T$ is lower triangular and $\mathbf{M}_\alpha$ is upper triangular, this can now be interpreted as an incomplete LDU factorization:

$$\mathbf{A} \approx \mathbf{M}_\beta^T \widetilde{\Delta}^{-1}\mathbf{M}_\alpha$$

The transformation to the multi-resolution basis is now seen as an incomplete factorization preconditioner, using triangular solves with approximate factors. This is analogous to BILUM[33] or repeated red-black ILU[8], where the triangular factors are found with a multi-level algebraic algorithm rather than the interpolation approach here. Inspired from this analogy, an interesting extension to this thesis would be an algebraic version of the multi-resolution approximate inverse preconditioner, where the prediction operators are determined algebraically from the original matrix $\mathbf{A}$.

However, return now to the problem of computing $\widetilde{\mathbf{Q}}$, realizing that the inverse transform matrices are only available in factored form.

3.6 Computing the Approximate Inverse

Although $\mathbf{M}_\beta^{-T}$, $\mathbf{A}$, and $\mathbf{M}_\alpha^{-1}$ are known, their product $\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1}$ is not explicitly known—as discussed in the previous section, even just multiplying out the inverse transforms will incur a penalty. Thus $\widetilde{\mathbf{Q}}$ must be found with an approximate inverse algorithm that works when the matrix is known only as a linear operator. Actually, a little more is known: the adjoint of the operator

$$(\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1})^T = \mathbf{M}_\alpha^{-T}\mathbf{A}^T\mathbf{M}_\beta^{-1}$$

may be used in the algorithm as well.

This rules out the Frobenius norm minimization algorithms such as SPAI[22] and FSAI[25], as well as Tang and Wan's local inverse method[35], since they all require the ability to access submatrices of $\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1}$. Chow and Saad's MR method[15, 16] is a possibility as it only uses the matrix as an operator. However, the impressive performance[5] of the incomplete inverse

factorization algorithms makes them the most attractive choice. I chose to adapt the AINV[4] algorithm.

The original form of AINV is a column-oriented, left-looking, dot-product based algorithm that constructs a factored approximate inverse via biconjugation, shown in figure 3.2. Given a matrix $\mathbf{B}$ it returns upper triangular matrices $\mathbf{W}$ and $\mathbf{Z}$ along with a diagonal matrix $\mathbf{D}$, where the columns of $\mathbf{W}$ and $\mathbf{Z}$ are approximately $\mathbf{B}$-biconjugate: $\mathbf{W}^T\mathbf{B}\mathbf{Z} \approx \mathbf{D}$. It can be interpreted as a generalization of the classical Gram-Schmidt orthogonalization algorithm, beginning with the standard basis vectors and making them $\mathbf{B}$-biconjugate.

AINV gives an approximation to the UDL factorization of $\mathbf{B}^{-1}$, since the biconjugation condition is equivalent to

$$\mathbf{B}^{-1} \approx \mathbf{Z}\mathbf{D}^{-1}\mathbf{W}^T$$

However, the choice of preconditioned system should naturally follow the construction of the preconditioner: either $\mathbf{D}^{-1}(\mathbf{W}^T\mathbf{B}\mathbf{Z})$ or $(\mathbf{W}^T\mathbf{B}\mathbf{Z})\mathbf{D}^{-1}$. These choices guarantee a unit diagonal in the preconditioned system, which is often a good property.

Observe that the storage and work can be cut in half when $\mathbf{B}$ is symmetric; then $\mathbf{W} = \mathbf{Z}$, so only $\mathbf{Z}$ need be computed. In addition, if $\mathbf{B}$ is symmetric positive definite and the algorithm is accurate enough, $\mathbf{D}$ should only have non-negative entries, so $\mathbf{D}^{-1/2}$ can be used. Then an approximate inverse of the upper Cholesky factor is $\mathbf{Z}\mathbf{D}^{-1/2}$, and the preconditioned system $\mathbf{D}^{-1/2}\mathbf{Z}^T\mathbf{B}\mathbf{Z}\mathbf{D}^{-1/2}$ not only has a unit diagonal but is also symmetric positive definite, a definite advantage in iterative methods.

The algorithm above works fine even if $\mathbf{B}$ and $\mathbf{B}^T$ are only available as operators; though the rows and columns of $\mathbf{B}$ are actually found explicitly by multiplying with the standard basis vectors, only one row or column needs to be stored at a time, and each is required only once. Of course, it is imperative to do these multiplies in sparse-sparse mode or else the algorithm will run very slowly.

3.7 Improving AINV

The problem with using this algorithm, elaborated in [9], is that the biconjugation j loops are often doing too much work. As it stands the algorithm runs in at least $O(n^2)$ time, even if much

Figure 3.2: The original dot-product form of AINV.

- Take $\mathbf{B}$, an $n \times n$ matrix, and some drop tolerance $\delta \geq 0$ as input.
- For $i = 1, \ldots, n$
  - ▷ *Initialize columns i of $\mathbf{W}$ and $\mathbf{Z}$ to the i'th standard basis vector*
  - Set $\mathbf{W}_i = \mathbf{e}^i$ and $\mathbf{Z}_i = \mathbf{e}^i$.
  - ▷ *Make column i of $\mathbf{W}$ biconjugate with previous columns*
  - Get row i of $\mathbf{B}$: $\mathbf{r} = (\mathbf{e}^i)^T\mathbf{B} = (\mathbf{B}^T\mathbf{e}^i)^T$.
  - For $j = 1, \ldots, i-1$
    - $\mathbf{W}_i \leftarrow \mathbf{W}_i - \frac{\mathbf{r}\mathbf{Z}_j}{D_{jj}}\mathbf{W}_j$
  - ▷ *Make column i of $\mathbf{Z}$ biconjugate with previous columns*
  - Get column i of $\mathbf{B}$: $\mathbf{c} = \mathbf{B}_i = \mathbf{B}\mathbf{e}^i$.
  - For $j = 1, \ldots, i-1$
    - $\mathbf{Z}_i \leftarrow \mathbf{Z}_i - \frac{\mathbf{W}_j^T\mathbf{c}}{D_{jj}}\mathbf{Z}_j$
  - ▷ *Drop small entries to keep $\mathbf{W}$ and $\mathbf{Z}$ sparse*
  - Zero any above-diagonal entry of $\mathbf{W}_i$ or $\mathbf{Z}_i$ with magnitude $\leq \delta$.
  - ▷ *Find the "pivot" D_{ii}*
  - Set $D_{ii} = \mathbf{W}_i^T\mathbf{B}\mathbf{Z}_i$.
- Return $\mathbf{W}$, $\mathbf{Z}$, and $\mathbf{D}$.

less than $O(n^2)$ nonzeros are input and output. Typically, the majority of the time spent in the algorithm is wasted computing the sparse dot-products $\mathbf{r}\mathbf{Z}_j$ and $\mathbf{W}_j^T\mathbf{c}$ when they turn out to be identically zero, due to the vectors having no nonzero entries in common.

There are inexpensive symbolic methods to cut down the j loops from (1, ..., $i-1$) to much smaller lists (especially if the nodes are ordered in a good manner—see the next section). Unfortunately, these symbolic methods make crucial use of the nonzero structure of the matrix $\mathbf{B}$ and its elimination tree[26] or related structures, which aren't directly available here. The structures of the factors in $\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1}$ are available, so it may be possible to recover these symbolic methods and use the dot-product algorithm efficiently. However, I leave this problem for future research and instead turn to a different form of AINV.

Reversing the nesting of the loops, the algorithm can be rearranged into a right-looking outer-product based method, shown in figure 3.3. The same comments about unit diagonals and symmetry apply here. Note that the dropping strategy is slightly different: instead of zeroing out small entries of $\mathbf{W}_i$ and $\mathbf{Z}_i$ after they have been fully computed, small updates simply are not added.

The benefit of this formulation is that the inner i loops can be easily trimmed to just what is needed: a loop over the non-zero values of $\mathbf{l}$ or $\mathbf{u}$. Normally $\mathbf{l}$ and $\mathbf{u}$ will be quite sparse so this means big savings (especially for good orderings of the nodes—see the next section).

Another potential slow-down is the calculation of $\mathbf{l}$ and $\mathbf{u}$; if computed as dense vectors, this takes $O(n)$ time via the lifting scheme, making the whole algorithm at least $O(n^2)$. This can be avoided by doing them in sparse-sparse mode. Potentially even faster is a hybrid mode described in [9] that uses efficient sparse-dense multiplies but keeps track of where nonzeros are created for a fast "gather" operation back to a sparse result.

The down side of this formulation is that whereas the original form constructed the columns of $\mathbf{W}$ and $\mathbf{Z}$ one at a time, here all of columns $j+1, \ldots, n$ are being updated as the algorithm proceeds. Dynamic linked list data structures are required to store the unfinished columns, inevitably bringing up worries about efficiency—e.g. in [4], where a vector processor was used, this outer-product form was dismissed as inappropriate. However, tests comparing this version to the original with symbolic enhancements (for explicitly known $\mathbf{B}$), running on a modern superscalar workstation, show that it is competitive. In fact, since the symbolic algorithms

Figure 3.3: The outer-product form of AINV.

- Take as input $\mathbf{B}$ and δ.
- Set $\mathbf{W} = \mathbf{I}$ and $\mathbf{Z} = \mathbf{I}$.
- For $j = 1, \ldots, n$
  - Set $\mathbf{l} = \mathbf{B}\mathbf{Z}_j$
  - Set $\mathbf{u} = \mathbf{B}^T\mathbf{W}_j$
  - Set $D_{jj} = \mathbf{u}^T\mathbf{Z}_j$
  - For $i = j + 1, \ldots, n$
    - Update $\mathbf{W}_i \leftarrow \mathbf{W}_i - \text{drop}\left(\frac{l_i}{D_{jj}}\mathbf{W}_j, \delta\right)$, where entries of the update vector with magnitude $\leq \delta$ are dropped.
  - For $i = j + 1, \ldots, n$
    - Update $\mathbf{Z}_i \leftarrow \mathbf{Z}_i - \text{drop}\left(\frac{u_i}{D_{jj}}\mathbf{Z}_j, \delta\right)$.
- Return $\mathbf{W}$, $\mathbf{Z}$, and $\mathbf{D}$.

cannot account for sparsity due to the dropping of small elements, but the outer-product form automatically does, this version often is more efficient![9]

3.8 Ordering

Before using AINV, one more thing must be considered: the ordering of the nodes. In [6, 10] it was made clear that ordering has a significant effect on the construction time of the approximate inverse, and on the convergence of the preconditioned system. For fairly isotropic problems, the heuristic of inverse factor fill reduction has proven to be very effective; ordering algorithms like Nested Dissection, Minimum Degree, and Minimum Inverse Penalty[10] do a good job. These often handle more difficult problems, but [10] showed that anisotropic matrices can be better handled by algorithms sensitive to the numerical entries in the matrix. The question of how best to deal with anisotropy still requires more research, so in this thesis I have ignored the issue.

I have chosen to work with Nested Dissection. Despite indications in [10] that there may be slightly superior orderings for convergence, this is not well understood at all, whereas it is clear that Nested Dissection is the best fill reduction and execution speed—particularly on parallel machines—with good implementations like Metis[24].

Unfortunately there is a major difficulty to overcome before running the ordering algorithm: $\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1}$ is known only in factored form, so the nonzero structure required is not explicitly available.

Before going further, recall the graph theory notation often used in sparse matrix ordering. With a given $n \times n$ matrix $\mathbf{B}$, associate the graph $G_{\mathbf{B}}$, or simply G if the context makes it clear, defined on nodes $\{1, \ldots, n\}$ with a directed edge $i \to j$ if and only if $B_{ij} \neq 0$. Thus the nonzero structure of $\mathbf{B}$ and the graph $G_{\mathbf{B}}$ may be identified. As an abbreviation, write $i \to j$ to mean the statement that the directed edge $i \to j$ exists in G. The neighbourhood of a node i is the set of j such that $i \to j$. A path is a sequence of distinct nodes $i_1, \ldots, i_k$ such that $i_1 \to i_2$, $i_2 \to i_3$, …, and $i_{k-1} \to i_k$, often written $i_1 \to \cdots \to i_k$, or simply $i_1 \rightsquigarrow i_k$. The transitive closure G^* of a graph G is one constructed on the same nodes but having $i \to j$ whenever $i \rightsquigarrow j$ in G. For a fuller treatment, see [20, 21].

As is shown in [21], assuming here and for the rest of this section that there is no felicitous

cancellation, the structure of $\mathbf{B}^{-1}$ is given by the transitive closure of the graph of $G_{\mathbf{B}}$. As was mentioned before, when the forward transform $\mathbf{M}_\alpha$ is multiplied out (with no update steps), the off-diagonal $-\mathbf{P}$'s are just added—no fill-in occurs. Then the graph of $\mathbf{M}_\alpha$ satisfies $i \to j$ iff at some level i is a fine node whose prediction uses coarse node j. Therefore the graph of $\mathbf{M}_\alpha^{-1}$ has $i \to j$ iff there is a chain of prediction dependencies $i \rightsquigarrow j$.

Define the support of a node j to be the set *supp*(j) of nodes i such that $(\mathbf{M}_\alpha^{-1})_{ij} \neq 0$—this is actually the support of the j'th multi-resolution basis function. From the transitive closure characterization of inverses, observe that the supports have a nested structure: if $i \in$ *supp*(j) then *supp*$(i) \subset$ *supp*(j). Notice that if j is a fine node at the highest resolution level, *supp*$(j) = \{j\}$, but that if j is at the lowest resolution level its support may be very dense—more justification never to multiply out the inverse transform!

Now examine the structure of $\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1}$. Assume that $\mathbf{A}$ has symmetric structure ($A_{ij} \neq 0$ iff $A_{ji} \neq 0$) and $\mathbf{M}_\beta$ and $\mathbf{M}_\alpha$ have the same structure. Then the product has symmetric structure, and one can speak unambiguously about coarse/fine nodes and the support of a node. Observe

$$\begin{aligned}(\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1})_{ij} &= \sum_{k=1}^{n}\sum_{l=1}^{n}(\mathbf{M}_\beta^{-T})_{ik}A_{kl}(\mathbf{M}_\alpha^{-1})_{lj} \\ &= \sum_{k=1}^{n}\sum_{l=1}^{n}(\mathbf{M}_\beta^{-1})_{ki}A_{kl}(\mathbf{M}_\alpha^{-1})_{lj}\end{aligned}$$

Then $(\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1})_{ij} \neq 0$ iff there exist nodes k and l with $k \in$ *supp*(i), $l \in$ *supp*(j), and $k \to l$ in $\mathbf{A}$. In other words, $i \to j$ in the product iff their supports are adjacent in $\mathbf{A}$. Using the nested structure of the supports, it is then clear that the neighbourhood of any node j contains the neighbourhoods of all nodes in *supp*(j).

Now, the location of nonzeros in column i of the upper inverse triangular factor $\mathbf{Z}$ of a symmetric structure matrix $\mathbf{B}$ can be characterized as follows. $\mathbf{Z}_i$ has an entry for each node before i and reachable from i, via paths in $\mathbf{B}$ using nodes before i. (One easy proof uses induction and the dot-product form of AINV.)

Consider the effect of swapping the positions of $i \neq j$ in some ordering, when $i \in$ *supp*(j). Clearly the number of nonzeros in columns in $\mathbf{Z}$ ordered before both i and j or after both will not be changed. However, the columns in between may be altered. Since the neighbourhood of

j contains the neighbourhood of i, any nodes reachable on paths through i are reachable through j, but not necessarily the other way around. Therefore ordering i before j can't result in more nonzeros in $\mathbf{Z}$, but putting j before i might.

Thus any ordering of the nodes should respect j ordered after all other nodes in *supp*(j). Since *supp*(j) is the set of i such that $(\mathbf{M}_\alpha^{-1})_{ij} \neq 0$, this is equivalent to requiring that i be ordered before j whenever $i \rightsquigarrow j$ in $\mathbf{M}_\alpha$. This is clearly equivalent to ordering i before j whenever $i \to j$ in $\mathbf{M}_\alpha$, which can be enforced by the algorithm in figure 3.4.

Essentially the algorithm outputs the nodes in the existing order except when a coarse node comes before any of its fine dependents. Then the coarse node is made to wait until all the fine dependents have been ordered, at which point it's put on a queue to be ordered as soon as possible. The value *numdep*(i) serves as a counter of how many fine nodes dependent on i have yet to be ordered—since i is only put into p when this reaches zero, the ordering must be consistent.

The initialization loop, assuming sparse storage of the matrix, takes time on the order of the number of nonzeros in the matrix, which should be $O(n)$ as mentioned in section 2.3. The complexity of the main loop is a little more difficult to prove:

> First note that both i and j begin at 1 and never are decremented. Let $d = \sum_{i=1}^{n}$ *numdep*(i), so before the main loop begins $d =$ *nnz*$(\mathbf{M}_\alpha) - n$, the number of off-diagonal nonzeros in $\mathbf{M}_\alpha$. Values in *numdep* are never incremented so d never increases.
>
> A node can only be marked as waiting in the final else clause, and since i is incremented there it can never be marked as waiting again. The only way an entry in *numdep* can be decremented to zero is if it had been marked as waiting, and when it hits 0 its marked as not waiting, so it can never be decremented past 0. Therefore d is always non-negative.
>
> Suppose i is incremented past $n+1$—this can only happen if $i = n+1$ at the start of an iteration with the queue empty. There must be some unordered nodes left, as otherwise j would have been incremented past n and the loop would have stopped. If any of the unordered nodes had *numdep* equal to zero, they either would have started at zero, in which case the first else clause would have been executed for that

Figure 3.4: Modifying an ordering to respect the multi-resolution basis.

- Take as input the structure of $\mathbf{M}_\alpha$ or $\mathbf{M}_\beta$ (multiplied out).
- For $i = 1, \ldots, n$
  - Set *numdep*(i) = number of nodes j with $j \to i$, not including i itself.
  - Set *waiting*(i) to false.
- Initialize a queue with room for n entries, empty at first.
- Set $i = 1$, the first node to attempt to order.
- Set $j = 1$, the first index into the modified ordering p.
- While $j \leq n$
  - If the queue is not empty then
    - Remove the first node k from the front of the queue.
    - Set $p_j = k$ and $j \leftarrow j + 1$.
    - Consider, in order, each $l \neq k$ with $k \to l$ and *waiting*(l) true; decrement *numdep*(l), and if this is 0 set *waiting*(l) to false and append l to the queue.
  - Else if *numdep*$(i) = 0$ then
    - Set $p_j = i$, $j \leftarrow j + 1$, and $i \leftarrow i + 1$.
    - Consider, in order, each $l \neq i$ with $i \to l$ and *waiting*(l) true; decrement *numdep*(l), and if this is 0 set *waiting*(l) to false and append l to the queue.
  - Else (*numdep*$(i) > 0$)
    - Set *waiting*(i) to true, and $i \leftarrow i + 1$.
- Return the modified ordering p.

value of i, or they would have been decremented to zero and added to the queue—in either case implying that they must now be ordered, a contradiction. Thus all the unordered nodes have positive *numdep* counters. However, some unordered node v must be from the finest resolution level of all unordered nodes, and so cannot have any unordered dependent fine nodes—and so must have *numdep*$(v) = 0$, a contradiction. Therefore i never is incremented past $n + 1$.

Clearly j can never be incremented past $n + 1$ thanks to the loop condition. Therefore, since in each iteration either j is incremented, i is incremented, or at least one of the values in *numdep* is decremented, there can be at most $n +$ *nnz*$(\mathbf{M}_\alpha)$ iterations. In fact, assuming constant time queue operations (e.g. as in a simple array implementation) the time spent in the main loop is $O(n) + O($*nnz*$(\mathbf{M}_\alpha))$, which again should be $O(n)$ (see section 2.3). Thus the entire algorithm is $O(n)$.

I now propose the following simple scheme: order $\mathbf{A}$ with Nested Dissection, and then run the above algorithm to make the ordering consistent with the multi-resolution basis. The only worry is that the modification will destroy the good fill-reducing qualities of the original ordering. However, the bulk of the nodes should be at the finest level and thus have trivial supports, so the modification can't change their relative order. The only nodes that can be greatly affected by the ordering modification are the very coarse nodes, which are in a very small minority. Thus the potential damage is very limited.

3.9 Parallel Ordering and Construction

The only unresolved issue is parallelism in the construction and ordering. Although many opportunities exist for limited fine-grain parallelism, probably the most practical approach is coarse-grain, based on the successful parallel AINV described in [7].

Begin by partitioning the graph of $\mathbf{A}$ into disconnected subgraphs (distributed to different processors) and a separator set of the nodes separating the subgraphs. Packages such as Metis[24] provide good parallel routines to do this so that the subgraphs are roughly balanced in size and the separator is small. Conceptually the global ordering will put the subgraphs first and the separator set last, thus restricting fill in the inverse factors and making the subgraph computations independent.

In each subgraph, the nodes can be ordered with Nested Dissection or some other good method, and the modification algorithm from the previous section run to make it consistent with the multi-resolution basis. In this case, some coarse nodes may be discovered with fine dependents in other subgraphs; these nodes must be moved to the separator set. This modification now ensures that the partition is also good for the transformed matrix $\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1}$. The modification can then continue in the separator set to make it consistent with the multi-resolution basis. While each subgraph ordering can be done independently on different processors, doing the separator set in parallel probably will be very challenging, so provided it's not too large doing it serially on one processor should be acceptable.

As soon as the ordering of a subgraph is determined, serial outer-product AINV can be run for those columns of $\mathbf{W}$ and $\mathbf{Z}$: no information from other nodes is required. The bottleneck is again the separator set, which must receive and combine information from all the subgraphs. Possibly the best approach is to use the block dot-product form of AINV from [9] to get the contributions from the subgraphs in parallel—each subgraph providing one block column, with sparse blocks—and then continue with serial outer-product AINV on one processor. The exact details of the implementation are left for future work.

3.10 The Relationship with Multigrid

Although the multi-resolution approximate inverse technique was motivated quite differently from multigrid—using wavelets to compress the discrete Green's function rather than using a hierarchy of grids to damp all the different resolution components of the error efficiently—it appears they are fundamentally very similar. In fact, the software developed here for finding the hierarchy of coarse nodes and the prediction operators could be used with only cosmetic changes in an unstructured mesh node-nested[1] multigrid package, and vice versa. The multi-resolution basis part of the thesis can then be seen as more or less independent of the approximate inverse part, though of course some details of the basis are decided with consideration for implementing the approximate inverse.

[1] A multigrid method is node-nested if the nodes in each coarse mesh are also nodes of the next finer mesh, so the coarsening procedure consists of selecting a subset of the fine mesh nodes to be coarse rather than introducing new nodes.

The relationship can be made more precise by interpreting the multi-resolution approximate inverse as an additive node-nested multigrid algorithm. For simplicity I only consider the "two grid" case, where there are only two levels in the hierarchy: the original problem and one coarse problem. As usual for analysis, I assume that all the fine nodes are ordered before all the coarse nodes, with matrices partitioned accordingly.

The approximation to $\mathbf{A}^{-1}$ is $\mathbf{M}_\alpha^{-1}\widetilde{\mathbf{Q}}\mathbf{M}_\beta^{-T}$. Of course $\widetilde{\mathbf{Q}}$ might be available only as a product of matrices, but for this analysis assume it is explicitly known. Writing this out in matrix form gives:

$$\begin{aligned}
\mathbf{A}^{-1} &\approx \mathbf{M}_\alpha^{-1}\widetilde{\mathbf{Q}}\mathbf{M}_\beta^{-T} \\
&= \begin{pmatrix} \mathbf{I} & \mathbf{P}_\alpha \\ & \mathbf{I} \end{pmatrix} \underbrace{\begin{pmatrix} \widetilde{\mathbf{Q}}_{11} & \widetilde{\mathbf{Q}}_{12} \\ \widetilde{\mathbf{Q}}_{21} & \widetilde{\mathbf{Q}}_{22} \end{pmatrix}}_{\widetilde{\mathbf{Q}}} \begin{pmatrix} \mathbf{I} & \\ \mathbf{P}_\beta^T & \mathbf{I} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix} \widetilde{\mathbf{Q}}_{11} \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{P}_\alpha \\ \mathbf{I} \end{pmatrix} \widetilde{\mathbf{Q}}_{21} \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} \\
&\quad + \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix} \widetilde{\mathbf{Q}}_{12} \begin{pmatrix} \mathbf{P}_\beta^T & \mathbf{I} \end{pmatrix} + \begin{pmatrix} \mathbf{P}_\alpha \\ \mathbf{I} \end{pmatrix} \widetilde{\mathbf{Q}}_{22} \begin{pmatrix} \mathbf{P}_\beta^T & \mathbf{I} \end{pmatrix}
\end{aligned}$$

Now, define the mesh transfer operators: the prolongation $\mathcal{P} = \binom{\mathbf{P}_\alpha}{\mathbf{I}}$ and the restriction $\mathcal{R} = \binom{\mathbf{P}_\beta}{\mathbf{I}}^T$. The prolongation takes a coarse mesh version of a function and returns the interpolated (predicted) fine mesh version. The restriction takes a fine mesh version of a function and returns a coarse mesh version—notice that this process is not simple injection (sub-sampling of just the coarse node values) but instead assigns to each coarse node a linear combination of the coarse node and its fine neighbours. The standard multigrid choice of taking the restriction equal to the transpose of the prolongation corresponds to taking $\mathbf{P}_\alpha = \mathbf{P}_\beta$.

While general multigrid is not constrained to this form for the prolongation and restriction, the only real assumptions underlying this form are:

- The restriction of a function at a particular coarse node should only depend on the function values at that coarse node and possibly some fine nodes.
- The prolongation at a particular coarse node only depends on the value in the coarse mesh version.

Equivalently stated for the Galerkin (or Petrov-Galerkin) viewpoint of multigrid, the support of a coarse mesh basis function should include only one coarse node. Under these assumptions, the coarse part of the two operators becomes diagonal and can be trivially rescaled to the identity. These seem quite reasonable assumptions to make; if the need arises, however, there is the possibility of simply generalizing the lifting scheme transform algorithm, replacing the appropriate identity block with an invertible matrix.

Rewriting the approximation gives:

$$\mathbf{A}^{-1} \approx \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix} \widetilde{\mathbf{Q}}_{11} \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} + \mathcal{P}\widetilde{\mathbf{Q}}_{21} \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix} \widetilde{\mathbf{Q}}_{12}\mathcal{R} + \mathcal{P}\widetilde{\mathbf{Q}}_{22}\mathcal{R}$$

which can be viewed as additive multigrid. The coarse mesh correction corresponds to the $\mathcal{P}\widetilde{\mathbf{Q}}_{22}\mathcal{R}$ term (with $\widetilde{\mathbf{Q}}_{22}$ playing the part of the coarse mesh solver). The $\mathcal{P}\widetilde{\mathbf{Q}}_{21}\begin{pmatrix}\mathbf{I} & \mathbf{0}\end{pmatrix}$ and $\begin{pmatrix}\mathbf{I} \\ \mathbf{0}\end{pmatrix}\widetilde{\mathbf{Q}}_{12}\mathcal{R}$ terms correspond to pre- and post-smoothing respectively, and the $\begin{pmatrix}\mathbf{I} \\ \mathbf{0}\end{pmatrix}\widetilde{\mathbf{Q}}_{11}\begin{pmatrix}\mathbf{I} & \mathbf{0}\end{pmatrix}$ term smooths just the fine nodes independently of the coarse node operations (see [35] for an example of approximate inverses used as smoothers in standard multigrid).

Chapter 4

Implementation in One Dimension

4.1 Overview

One dimensional problems serve as a useful test of the method. Issues such as the coarse/fine splitting and the prediction are easier to deal with, and testing very fine meshes takes less computer resources. Of course, approximate inverse methods would never be used for one-dimensional problems in the real world, since other direct or special methods achieve optimally efficient, robust solutions (at least if the problem isn't too ill-conditioned). However, some of the lessons gained in 1D can be brought to higher dimensional problems where there is real interest in using approximate inverses.

4.2 Discretization

In one dimension the operator $\mathcal{L}$ is of the form:

$$\mathcal{L}u = \frac{d}{dx}\left(K\frac{d}{dx}u - bu\right) + cu$$

where all coefficients are scalars (possibly functions of x). Without loss of generality the equation $\mathcal{L}u = f$ can be taken on the unit interval $[0, 1]$.

For a finite volume discretization, choose points $0 = x_1 < x_2 < \cdots < x_n = 1$ on the interval, at which the solution will be approximated. Let $x_{i+1/2}$ be the midpoint between x_i and x_{i+1}. Define vertex-centred cells (known as finite volumes or control volumes) from these points, with half-cells at the endpoints:

$$\begin{aligned} C_1 &= [x_1, x_{1+1/2}] = [0, x_{1+1/2}] \\ C_i &= [x_{i-1/2}, x_{i+1/2}] \qquad \text{for} \quad i = 2, \ldots, n-1 \\ C_n &= [x_{n-1/2}, x_n] = [x_{n-1/2}, 1] \end{aligned}$$

Integrating the equation over an interior cell gives:

$$\begin{aligned} \int_{C_i} \frac{d}{dx}\left(K\frac{d}{dx}u - bu\right) + cu\,dx &= \int_{C_i} f\,dx \\ \left[K\frac{d}{dx}u - bu\right]_{x=x_{i-1/2}}^{x_{i+1/2}} + \int_{C_i} cu\,dx &= \int_{C_i} f\,dx \end{aligned}$$

A mid-point approximation for the integrals and a second order finite difference approximation for du/dx gives:

$$\left\{ \begin{array}{r} K_{i+1/2}\left(\dfrac{u_{i+1}-u_i}{x_{i+1}-x_i}\right) - b_{i+1/2}u_{i+1/2} \\ -K_{i-1/2}\left(\dfrac{u_i-u_{i-1}}{x_i-x_{i-1}}\right) + b_{i-1/2}u_{i-1/2} \\ + (x_{i+1/2}-x_{i-1/2})c_i u_i \end{array} \right\} \approx (x_{i+1/2}-x_{i-1/2})f_i$$

The value $K_{i+1/2}$ could plausibly be taken as $K(x_{i+1/2})$, but it turns out that a better choice is some kind of mean value of K on the interval between x_i and x_{i+1}. Continuity of the flux or homogenization theory arguments show that the harmonic mean of K in this interval is the correct value. A different intuitive reason for this can be found in the physical interpretation of u as the concentration of some quantity U that diffuses at rate K; then K is the average speed of the tiny particles of U at a given point as they randomly move about. The average speed on a path from x_i to x_{i+1} is the harmonic mean of the speeds along the way: moving distance dx takes time $1/K(x)\,dx$, so the total time is $\int_{x_i}^{x_{i+1}} 1/K(x)\,dx$, giving average speed $(x_{i+1} -$

$x_i)/\int_{x_i}^{x_{i+1}} 1/K(x)\,dx$. Assuming K is only known at the vertices, the natural approximation is:

$$K_{i+1/2} = \frac{1}{\frac{1}{2}(\frac{1}{K_i} + \frac{1}{K_{i+1}})}$$

The value $b_{i+1/2}$ is handled differently—in the physical model, unlike the K term representing diffusion of randomly moving particles, b represents the deterministic underlying current which convects the quantity. Again assume that b is only known at the vertices. There are two cases to consider: where b is the same sign at the vertices, and where b changes sign. In the first case, it is reasonable to appeal to smoothness in b for lack of a better idea, and estimate $b_{i+1/2} = (b_i + b_{i+1})/2$. In the second case, at some point between x_i and x_{i+1} either $b = 0$ or b has a discontinuity spanning 0; this stagnation/source/sink point means there is no convective connection between u_i and u_{i+1}, so $b_{i+1/2}$ should be 0.

The term $u_{i+1/2}$ appearing in the convection term also requires thought. First order upstreaming simply selects $u_{i+1/2}$ to be the upstream value u_i when $b_{i+1/2} > 0$ and u_{i+1} when $b_{i+1/2} < 0$. This is motivated by the physical reasoning that values of u downstream should not effect (via convection) any values that are upstream. It can be more mathematically justified as a sufficient condition for stability of the discretization, guaranteeing amongst other things that the linear system will be an M-matrix (at least if $c \leq 0$). Upstreaming is used throughout this thesis with no exceptions.

Diffusive flux (generalized Neumann) boundary conditions are easy to handle. For example, if $K\,du/dx \cdot \hat{n} = h$ at the left boundary, then there is just an extra source term when integrating the PDE over C_1 with the convective flux $bu(0)$ set to 0 and the reaction term $c(0)$ set to 0:

$$\begin{aligned} \int_{C_1} \frac{d}{dx}\left(K\frac{d}{dx}u - bu\right) &= \int_{C_1} f\,dx + h \\ \left[K\frac{d}{dx}u - bu\right]_{x=0}^{x_{1+1/2}} &= \int_{C_1} f\,dx + h \end{aligned}$$

The above approximations can then be made. In fact, if the discretization code constructs the equations interval by interval (not cell by cell) as is usually done, the only differences between a diffusive flux boundary and an interior point is the slightly different cell-width, the condition $c = 0$, and the additional h term on the right-hand side.

Generalized Robin boundary conditions then simply require the addition of the au term at the boundary node (or taking $c = a$). In the special case of full flux specification, $a \propto (-b) \cdot \hat{n}$.

Dirichlet boundary conditions, e.g. $u(x_n) = g$, might be discretized "as is": simply take the equation $u_n = g$. However, there is a major problem with this approach. For example, consider $u'' = 0$ with $u'(0) = 0$ and $u(1) = 1$ discretized on a uniform mesh of three points $\{0,1/2,1\}$. Putting the resulting linear equations in matrix form gives:

$$\begin{pmatrix} -2 & 2 & 0 \\ 2 & -4 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Notice the PDE problem was self-adjoint, yet the matrix is not. In fact, the transpose of the matrix doesn't represent a discretization of any related PDE, and the multi-resolution method is bound to fail in compressing the rows of the inverse.

The solution is to only approximately enforce the Dirichlet condition, the so-called "big number" approach, by thinking of the Dirichlet condition as the limit as $a \to \infty$ of the Robin condition $(K\nabla u)\cdot\hat{n} + au = ag$ on the boundary or the PDE with extra reaction term $\mathcal{L}u + au = ag$ in the interior. These conditions naturally give correct discretizations for the adjoint.

Begin with the normal flux conditions, or if the Dirichlet point is in the interior of the domain, the normal discretization—which as mentioned is usually handled by the same code. However, then increase the diagonal by a very large number (e.g. 10^{10}) and change the corresponding entry in the right-hand side appropriately:

$$\begin{pmatrix} -2 & 2 & 0 \\ 2 & -4 & 2 \\ 0 & 2 & -10^{10} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -10^{10} \end{pmatrix}$$

It's true that u_n now will only be approximately equal to g, but that should be so much more accurate than the other approximations made that this is no cause for worry. The matrix on the other hand is now symmetric; in general the transpose of the matrix will be a discretization of the adjoint problem, exactly as desired.

At first sight this might seem to pose the danger of making the system badly scaled and ill-conditioned; however, even the simplest of preconditioners will correct this essentially artificial scaling problem. The real issue with this technique is evaluating convergence in an iterative method. When looking at the residual $\mathbf{Au} - \mathbf{f}$, the Dirichlet entries are disproportionately weighted by the big number. Then reducing the norm of the residual by some factor like

10^{-6} can often be accomplished simply by correcting the Dirichlet entries—even if the rest of the approximate solution is completely wrong! Thus it is important to *unweight* the Dirichlet entries of the residual—divide by the big number—before taking the usual norm in evaluating convergence.

4.3 Basis construction

Although it's not clear that this is necessarily the best idea, a natural scheme for splitting the nodes into coarse and fine subsets in 1D is to simply take every second node coarse and the rest fine. As mentioned before, this is the original even/odd splitting proposed in [34]. One slight modification for this application is to always choose Dirichlet nodes as coarse: their value isn't naturally predictable from nearby nodes, plus as coarse nodes they are perfectly handled by the simplest approximate inverse. The same thing applies to Robin condition nodes with dominant Dirichlet part.

The simplest choices for prediction are linear interpolation between the two neighbouring coarse nodes, or cubic interpolation if another two nodes (one on each side) are used. Off-centered interpolation or extrapolation must be used for fine nodes on or near the boundary. A more sophisticated approach is to use PDE-interpolation (see section 3.3), which naturally handles fine flux condition boundary nodes in addition to the interior nodes.

The choices for the update step are nothing ($\mathbf{U} = \mathbf{0}$), first two moments (up to linear) preserved using the two neighbouring fine nodes, or first four moments (up to cubic) preserved using an extra node on each side. Near boundaries the nodes used must also be off-center, as with prediction.

4.4 Test Problems

The following five problems were selected to test a variety of the difficulties that are sometimes encountered. Uniform meshes of various sizes were tested along with some nonuniform meshes (where the nodes were moved to increase accuracy). Besides the multi-resolution approximate inverses, standard basis AINV was tested for comparison.

The following subsections give the details of the testing; section 4.5 summarizes the results.

4.4.1 Testing Protocol

The methods listed in the tables are:

- AINV(δ): the standard basis inner-product AINV with drop tolerance δ.
- Mr.Lin(δ): a multi-resolution basis with linear interpolation but no update, then outer-product AINV with drop tolerance δ.
- Mr.LinUpd(δ): a multi-resolution basis with linear interpolation and moments up to linear preserved with an update step, then outer-product AINV with drop tolerance δ.
- Mr.Cub(δ): cubic interpolation, no update, drop tolerance δ.
- Mr.CubUpd(δ): cubic interpolation and moments up to cubic preserved, drop tolerance δ.
- Mr.PDE(δ): PDE-interpolation, no update, drop tolerance δ.

For the multi-resolution bases, enough levels were allowed so that the coarsest level had about 100 nodes.

The ordering was nested dissection for standard AINV, with the modification algorithm applied for multi-resolution bases with no update step. For the bases with an update step, the basis-transformed matrix was actually multiplied out before nested dissection ordering and AINV.

The drop tolerances were chosen to give approximately the same total number of nonzeros (including prediction and update operators where applicable) for each preconditioner, about 7000 for a problem on 1000 nodes (or 9000 for problem 5).

The symmetric definite problems were solved with CG and the preconditioned system $\mathbf{D}^{-1/2}\mathbf{Z}^T(\mathbf{M}^{-T}\mathbf{A}\mathbf{M}^{-1})\mathbf{Z}\mathbf{D}^{-1/2}$. BiCGStab with $\mathbf{D}^{-1}\mathbf{W}^T(\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1})\mathbf{Z}$ was used for the others. Convergence was flagged when the 2-norm of the residual (with Dirichlet nodes rescaled appropriately, as mentioned before) was decreased by a factor of 10^{-6} beginning from an initial

guess of all zeros; if convergence wasn't reached after 500 iterations, the problem was marked unsolved with an asterisk (*).

After each iteration count in the tables, the "work per unknown" is included in parentheses: the number of iterations times the number of nonzeros in the preconditioner (prediction and update operators included), divided by the number of unknowns. This allows a somewhat fairer comparison between different preconditioners and problems.

Timing counts are not included, as parts of the code run interpreted under MATLAB and other parts in C, some tuned for performance and others not, thus any timing could be misrepresentative. This means in particular that the efficient use of cache memory, superscalar or superpipelined architecture, etc. is not measured at all. However, as all the preconditioner operations essentially boil down to sparse matrix multiplication, which can be coded very effectively, no major problems are anticipated for a real implementation.

4.4.2 Problem 1: Simple Heat Problem

This is the simplest problem, a sample solve from a fully implicit method for the heat equation on a uniform bar with heat applied in one spot:

$$u'' - 0.1u = f$$

where

$$f(x) = \begin{cases} -1 & : \quad 0.4 \leq x \leq 0.5 \\ 0 & : \quad \text{otherwise} \end{cases}$$

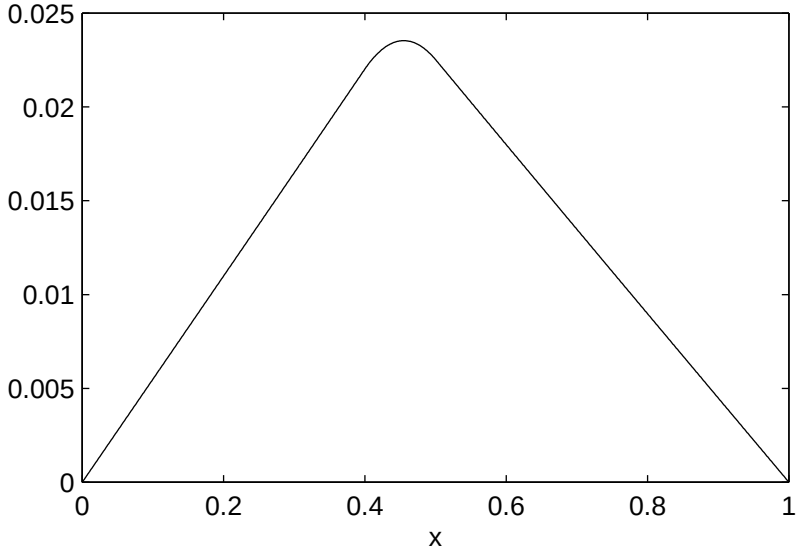
and the boundary conditions are Dirichlet:

$$u(0) = u(1) = 0$$

See 4.1 for a plot of the solution.

Figure 4.2 shows in 3D the negative of the discrete Green's function (the inverse of the matrix) and figure 4.3 shows it in 2D in different bases, symmetrically scaled to have unit diagonal (darker shading indicates larger magnitude). Note how in the standard basis many off-diagonal entries are significantly large, suggesting difficulties for an approximate inverse. In the multi-resolution bases most of the off-diagonal entries are nearly zero (except at the coarsest

Figure 4.1: Solution of 1D problem 1 (simple heat problem).



level in the bottom right corner)—with some exceptions for the cubic interpolation or those with update steps. The iteration results in table 4.1 confirm the suspicion that the linear and PDE-interpolation bases do the best jobs.

As justified earlier, preserving moments is not a good thing; the convergence is slowed enormously or lost altogether for larger problems. From now on, results for bases with updates will not be included.

AINV in the standard basis is reasonably effective for small n, but the work per unknown grows linearly—giving an $O(n^2)$ solution on a serial machine.

The cubic interpolation is a little disappointing. Though providing more efficient solutions than the standard basis, with the work per unknown a very slow growing function of n, it is nowhere near as good as the linear and PDE-interpolation bases. Despite giving a higher order prediction, it takes much more work (and a much higher drop tolerance in the approximate inverse, indicating poorer compression). The essential problem here is that the solutions are not smooth enough to warrant the high order accuracy. I suspect but haven't proven that just as linear interpolation corresponds to PDE-interpolation for the Laplacian (u'' in 1D), cubic

Figure 4.2: Negative of the discrete Green's function for 1D problem 1.

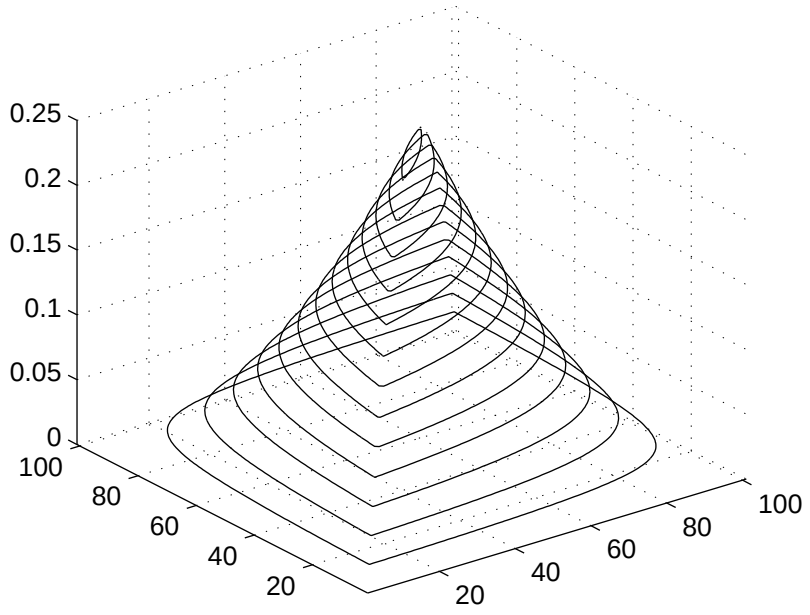


Table 4.1: CG iterations for 1D problem 1 (simple heat problem) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. See page 51 for details.

Method(δ)	$n = 1000$		$n = 2000$		$n = 4000$		$n = 8000$	
AINV(0.03)	23	(157)	39	(269)	73	(506)	141	(980)
Mr.Lin($5 \cdot 10^{-8}$)	3	(18)	3	(14)	3	(11)	3	(10)
Mr.LinUpd(0.07)	44	(323)	74	(466)	102	(606)	*	
Mr.Cub(0.063)	24	(165)	26	(171)	27	(170)	29	(179)
Mr.CubUpd(0.3)	231	(1937)	*		*		*	
Mr.PDE(10^{-10})	2	(7)	2	(7)	2	(6)	2	(6)

Figure 4.3: Inverse of problem 1 matrix in different bases.

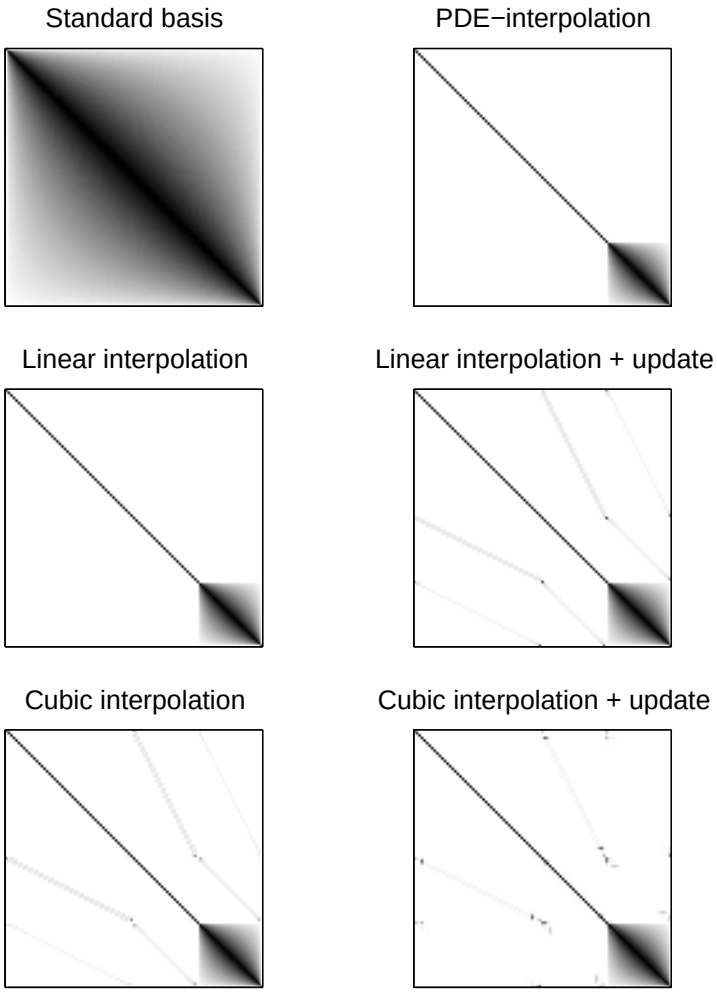


Table 4.2: CG iterations for 1D problem 2 (discontinuous heat problem) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. See page 51 for details.

Method(δ)	$n = 1000$		$n = 2000$		$n = 4000$		$n = 8000$	
AINV(0.01)	12	(82)	16	(118)	25	(197)	*	
Mr.Lin($5 \cdot 10^{-8}$)	3	(21)	3	(22)	3	(23)	3	(24)
Mr.Cub(0.063)	71	(486)	226	(1493)	*		*	
Mr.PDE(10^{-10})	2	(10)	3	(15)	3	(15)	3	(15)

interpolation corresponds to PDE-interpolation for the biharmonic operator (u'''' in 1D), and thus is clearly inappropriate for second order problems.

The highly desirable phenomenon of "grid-independent convergence" is clear in the linear and PDE-interpolation bases. Here the work per unknown stays constant, giving an optimal $O(n)$ solution on a serial machine and potentially optimal scalability on parallel machines.

Linear interpolation does a remarkably good job, almost giving a direct solution. However, PDE interpolation does even better, accounting as it does for the reaction term—not only is less work required, but the smaller drop tolerance indicates better compression.

4.4.3 Problem 2: Discontinuous Heat Problem

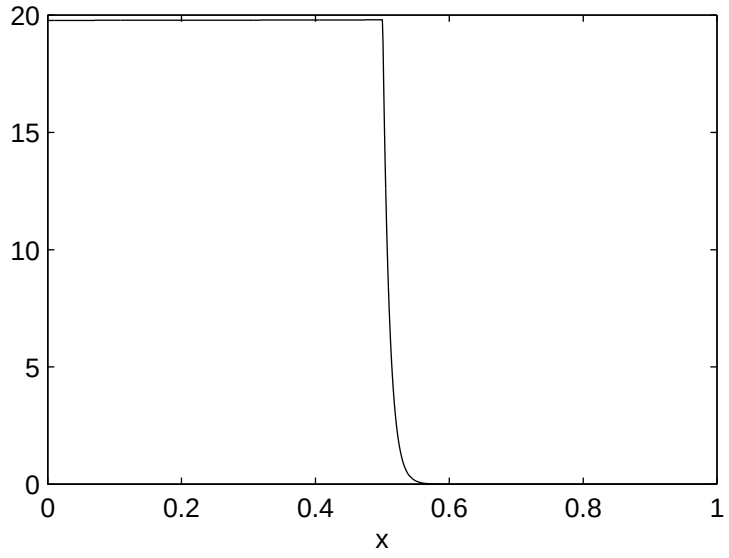
Problem 2 is identical to problem 1 except that the boundaries are insulated (so the condition is Neumann at steady state), the time step is larger (so the reaction term is $-10^{-2}u$), and there is a jump discontinuity in the diffusion coefficient:

$$K(x) = \begin{cases} 1 & : \quad x \leq 0.5 \\ 10^{-6} & : \quad x > 0.5 \end{cases}$$

The solution is shown in figure 4.4, and the inverse of the matrix in different bases in figure 4.5. Table 4.2 gives the iteration results.

Now that the problem really isn't so smooth, the cubic interpolation method fails. In fact, the standard basis is better, although still not robust and still not scaling well. The linear and

Figure 4.4: Solution of problem 2 (discontinuous heat problem).



PDE-interpolation methods perform very well again, with grid-independent convergence, the PDE method just a bit better.

4.4.4 Problem 3: Convection with a Boundary Layer

The next problem is not self-adjoint, dominated by strong convection:

$$\frac{d}{dx}(10^{-6}u' - (x+1)u) = f$$

where

$$f(x) = \begin{cases} -1 & : \quad x < 0.2 \\ 0 & : \quad x \geq 0.2 \end{cases}$$

and the boundary conditions are Dirichlet:

$$u(0) = u(1) = 0$$

A very sharp boundary layer is present at the right boundary—upstream weighting is essential for stability here in particular. See figure 4.6 for a plot of the solution, and figure 4.7 for a picture of the matrix inverse in different bases. The iteration results are shown in table 4.3.

Figure 4.5: Inverse of problem 2 matrix in different bases.

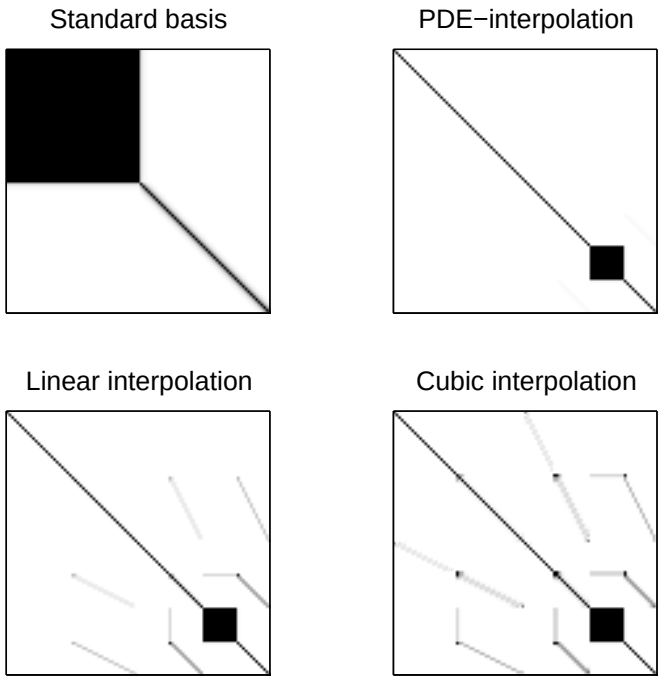
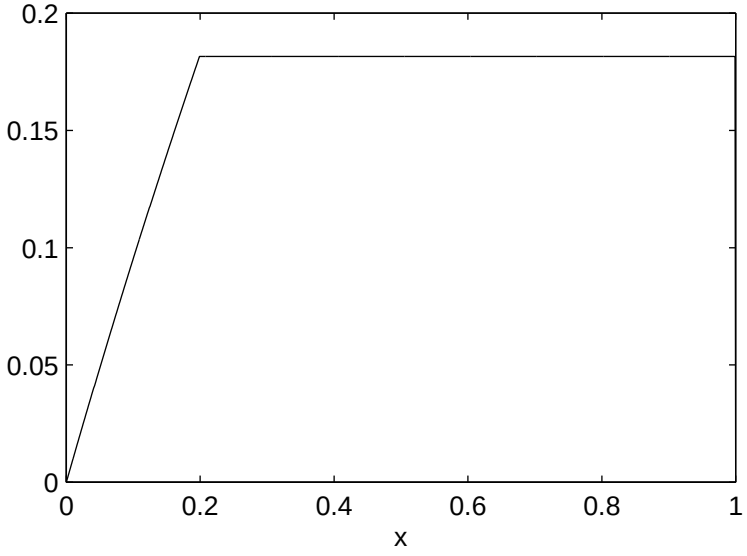


Table 4.3: Bi-CGstab iterations for 1D problem 3 (convection problem) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. See page 51 for details.

Method(δ)	$n = 1000$		$n = 2000$		$n = 4000$		$n = 8000$	
AINV(0.3)	5	(45)	7	(70)	7	(77)	9	(109)
Mr.Lin(0.25)	47	(335)	33	(205)	39	(237)	49	(288)
Mr.Cub(0.4)	71	(492)	99	(687)	103	(718)	123	(859)
Mr.PDE(0.002)	5	(33)	5	(29)	5	(28)	7	(44)

Figure 4.6: Solution of problem 3 (convection with a boundary layer).



As mentioned before, the linear and cubic prediction are centred, equally weighting downstream information as upstream information, and this really shows in their poor convergence. The standard basis is much superior.

However, despite having the additional overhead of the different adjoint prediction, the PDE-interpolation method again works beautifully and shows grid-independent convergence. The drop tolerance is still fairly low showing the superior compression.

To better resolve the boundary layer, I tried a nonuniform mesh such that the spacing Δx decreased cubically near the right boundary. This was much too difficult for convergence with either the standard basis or the cubic interpolation basis, so I have only included the linear and PDE-interpolation results in table 4.4.

Despite the increased difficulty, PDE-interpolation still works fine.

Figure 4.7: Inverse of problem 3 matrix in different bases.

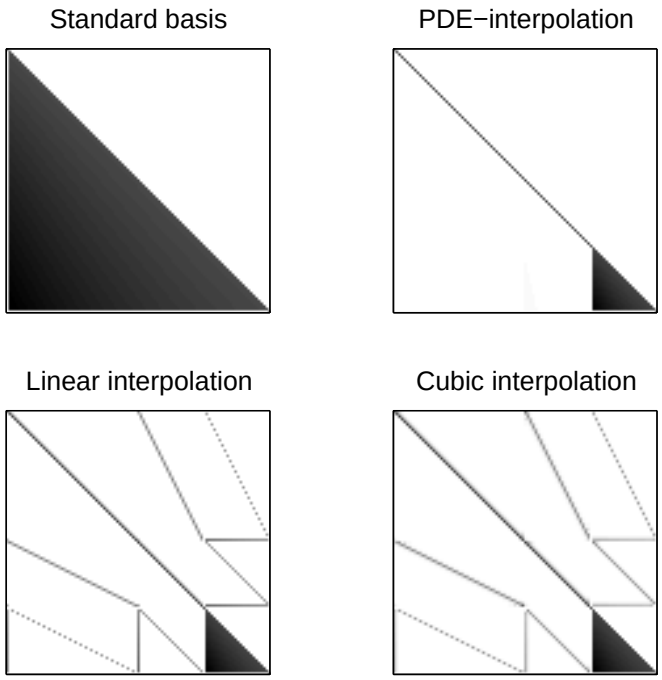
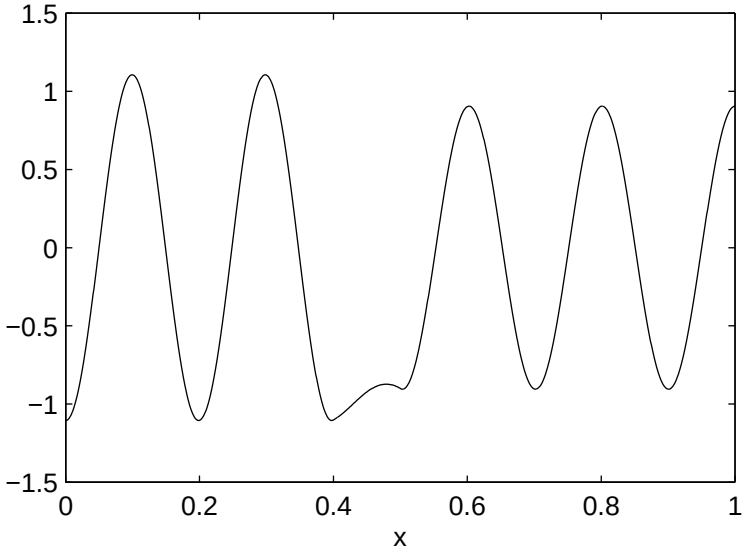


Table 4.4: Iterations for 1D problem 3 on a stretched mesh. Standard basis and cubic interpolation basis methods didn't converge at all.

Method(δ)	$n = 1000$		$n = 2000$		$n = 4000$		$n = 8000$	
Mr.Lin(0.245)	137	(1070)	99	(743)	289	(2081)	321	(2224)
Mr.PDE(0.003)	5	(35)	7	(45)	7	(43)	7	(43)

Figure 4.8: Solution of problem 4 (Indefinite Diffusion-Reaction).



4.4.5 Problem 4: Indefinite Diffusion-Reaction

Problem 4 is self-adjoint without any discontinuities, but is indefinite by virtue of the reaction term:

$$10^{-3}u'' + u = f$$

where

$$f(x) = \begin{cases} -1 & : \quad 0.4 \leq x \leq 0.5 \\ 0 & : \quad \text{otherwise} \end{cases}$$

and the boundary conditions are the natural Robin conditions. Figure 4.8 shows the solution and figure 4.9 the matrix inverse in different bases (the inverse is not scaled so that the oscillations are clearly apparent). Table 4.5 contains the iteration results.

Again the PDE-interpolation is a clear winner. The linear interpolation doesn't do so badly because the problem is self-adjoint, but is still much less effective.

For higher accuracy, I tried an adaptive mesh, where the uniform mesh was modified to improve the error based on the second derivative of the computed solution, and then smoothed

Figure 4.9: Inverse of problem 4 matrix in different bases.

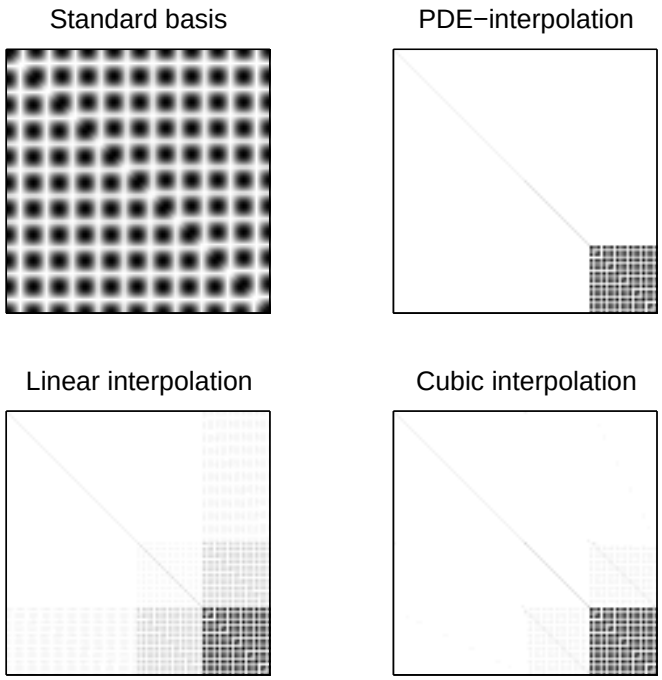


Table 4.5: CG iterations for 1D problem 4 (indefinite diffusion-reaction) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. See page 51 for details.

Method(δ)	$n = 1000$		$n = 2000$		$n = 4000$		$n = 8000$	
AINV(0.035)	73	(531)	315	(1940)	*		*	
Mr.Lin($4 \cdot 10^{-4}$)	29	(209)	25	(205)	27	(110)	25	(88)
Mr.Cub(0.07)	355	(2172)	*		*		*	
Mr.PDE($1 \cdot 10^{-10}$)	5	(35)	5	(38)	5	(41)	5	(29)

Table 4.6: Iterations for 1D problem 4, adaptive mesh. The iteration count is followed by the flops per unknown in parentheses.

Method(δ)	$n = 1000$		$n = 2000$		$n = 4000$		$n = 8000$	
AINV(0.035)	251	(1618)	*		*		*	
Mr.Lin($4 \cdot 10^{-4}$)	17	(119)	19	(99)	19	(78)	19	(68)
Mr.Cub(0.07)	*		*		*		*	
Mr.PDE($1 \cdot 10^{-8}$)	5	(33)	5	(30)	7	(35)	9	(38)

a little. The results are shown in table 4.6. The standard basis and cubic interpolation perform abysmally as with the previous nonuniform mesh, but probably thanks to the symmetry of the PDE the linear interpolation doesn't do badly at all. In fact, the performance of linear interpolation is improved dramatically, presumably because in the new mesh not only is the solution error diminished, but also the prediction error which also relies on the second derivative. The performance of PDE interpolation is decreased slightly (but still consistently beats linear interpolation), probably because not much improvement is made in prediction error while the problem is now worse conditioned.

4.4.6 Problem 5: Combined Difficulties

The final problem has a discontinuous diffusion coefficient, strong convection that changes direction, and an oscillating reaction term:

$$\frac{d}{dx}(Ku' - (|x - 0.5| - 0.05)u) - sin(5\pi x) = -1$$

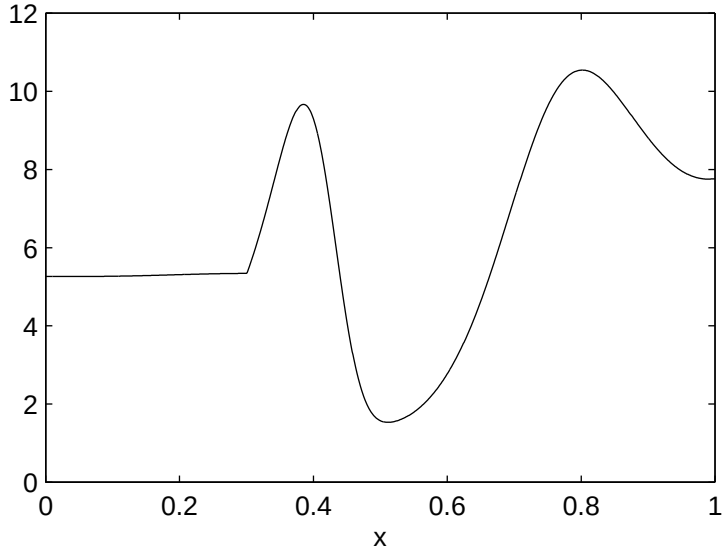
where

$$K(x) = \begin{cases} 1 & : \quad x \leq 0.3 \\ 10^{-3} & : \quad x > 0.3 \end{cases}$$

with Neumann boundary conditions:

$$u'(0) = u'(1) = 0$$

Figure 4.10: Solution of problem 5 (combined difficulties).



Figures 4.10 and 4.11 show the solution and the matrix inverse as before. Because of the difficulty of this problem, I aimed for 9000 nonzeros in the preconditioners when $n = 1000$; the results are given in table 4.7.

4.5 Summary

In one dimension, the multi-resolution basis with linear or PDE-interpolation is much superior to the standard basis, enabling fast grid-independent convergence for tough problems where AINV wouldn't otherwise converge at all.

It is clear that the update step and cubic interpolation are inappropriate. While linear interpolation sometimes works very well, it can have problems with really tough problems. PDE-interpolation gives by far the fastest convergence for all problems tested and should normally be the first choice. In situations where the same mesh is re-used for many problems with different (but not too challenging) coefficients, it might be worthwhile to stick with linear interpolation

Figure 4.11: Inverse of problem 5 matrix in different bases.

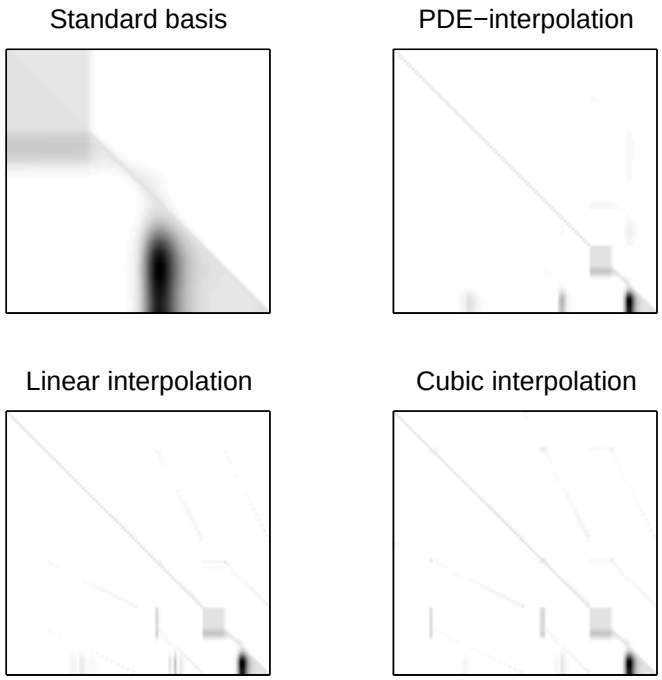


Table 4.7: Bi-CGstab iterations for 1D problem 5 (combined difficulties) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. See page 51 for details.

Method(δ)	$n = 1000$		$n = 2000$		$n = 4000$		$n = 8000$	
AINV(0.11)	127	(1170)	487	(4296)	*		*	
Mr.Lin(0.003)	15	(136)	11	(91)	15	(103)	13	(74)
Mr.Cub(0.063)	103	(898)	*		*		*	
Mr.PDE($4 \cdot 10^{-4}$)	9	(83)	7	(57)	7	(49)	9	(55)

which need only be set up once, rather than recompute the PDE-interpolation for each problem, but for robustness the PDE-interpolation is definitely best.

Chapter 5

Implementation in Two Dimensions

5.1 Overview

In two dimensions, on unstructured meshes, the matrices to solve are no longer trivial with any method. This is an application of interest to the real world. The lessons from one dimension (stick with linear or PDE-interpolation, no update step) carry over, but interpolation is trickier, to say nothing of new challenges in partitioning the nodes into coarse and fine sets and in deciding the nonzero structure of the prediction operators.

5.2 Discretization

There are still many questions left as to the best way of discretizing elliptic operators on two dimensional unstructured meshes, beginning with the choice of mesh itself. For some applications there are packages which use meshes of quadrilaterals, which give nice properties (e.g. near orthogonality) when close to rectangular. However, it is trivial to convert a quadrilateral mesh to a triangle mesh by splitting each quadrilateral along a diagonal, and good discretizations on these triangles can maintain the qualities which make quadrilaterals attractive in the first place. (It's generally impossible to go the other way, convert a triangle mesh into a quadrilateral mesh without adding or moving points.) Triangles are very flexible, able to connect up any collection

of points in the plane even if the prescribed boundary is non-convex. Finally, triangles have the advantage of simpler discretization. Therefore I have chosen to only consider triangle meshes.

There are then several choices for which triangulation should be used. Although the discretization scheme will work on any non-degenerate mesh, the accuracy of the result is highly dependent on the quality of the mesh.

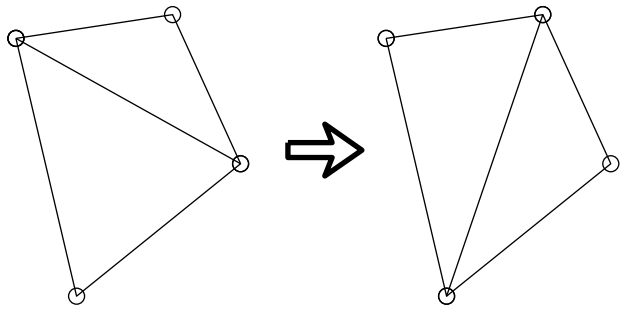
For elliptic problems, the Delaunay triangulation is the usual choice since it features, among other things:

- fast algorithms,
- good geometry (e.g. maximizing the minimum interior angle of any triangle in the mesh), and
- good discretization properties (e.g. linear finite element discretizations of Laplace's equation produce M-matrices, thus strictly maintaining the maximum/minimum properties of the original PDE).

Of course, Delaunay triangulation has its faults. For example, it doesn't directly control the maximum angle and thus may still produce nearly degenerate triangles—this is a particular problem for highly stretched meshes used in aerodynamics, and has prompted the use of a "MinMax" triangulation[3]. This issue hasn't come up during the testing for this thesis, so I have not followed this possibility.

Another problem with Delaunay triangulation is that an M-matrix (and accompanying maximum/minimum properties) is not guaranteed for elliptic PDE's other than Laplace's equation, particularly if the diffusion tensor is highly anisotropic. This suggests modifying the edge-swapping optimization procedure (see figure 5.1) of some incremental Delaunay triangulation algorithms to attempt to produce the desired M-matrices in the discretization, rather than optimize geometry features: edges should be swapped to make off-diagonal entries in the matrix more positive and the diagonal more negative. Delaunay triangulation is then just the special case for Laplace's equation. So-called "coefficient-adaptive triangulation" shows promise for simple problems, but lacks robustness for highly variable coefficients. However, as demonstrated later the results can be improved dramatically if the domain is first split up into separate

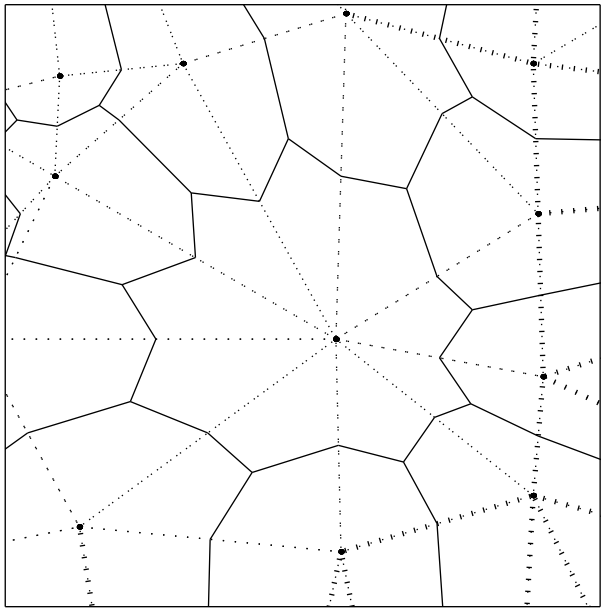
Figure 5.1: Given two adjacent triangles forming a convex quadrilateral, edge swapping may reconfigure the triangles as shown (swapping the diagonal) to locally optimize some property of the triangulation.



regions with roughly constant coefficients, the regions coefficient-adaptively triangulated, and then the full mesh stitched back together. Further research is definitely required.

Once the mesh has been determined, there are two popular approaches to discretizing $\mathcal{L}u = f$. The finite volume discretization used in 1D can be extended to 2D with the appropriate definition of a cell around each vertex—an integral of the PDE over a cell can be reduced to a boundary integral which is straightforward to approximate. Usually a polygonal region is made using the midpoints of the triangle edges along with some point in the interior of each triangle, such as the centroid. (Of course around boundary nodes the cell is chopped in half, just like the half-cells used at the boundaries in 1D.) Although the centroid is a reasonable choice and is quite popular, it may cause badly shaped cells inappropriate for convection problems: see figure 5.2 for an example. A much better choice is to use the circumcentre, where the perpendicular bisectors of the triangle meet—then the problem is that the circumcentre is outside the triangle for obtuse triangles. It is in general impossible to get a triangulation with no obtuse triangles without adding extra points, so one possible remedy is to use the circumcentre for acute triangles, but the midpoint of the side closest to the circumcentre for obtuse triangles. See figure 5.3 for an example. Another possibility not explored here is to use the circumcentres nevertheless—despite giving the non-intuitive property that the interface between

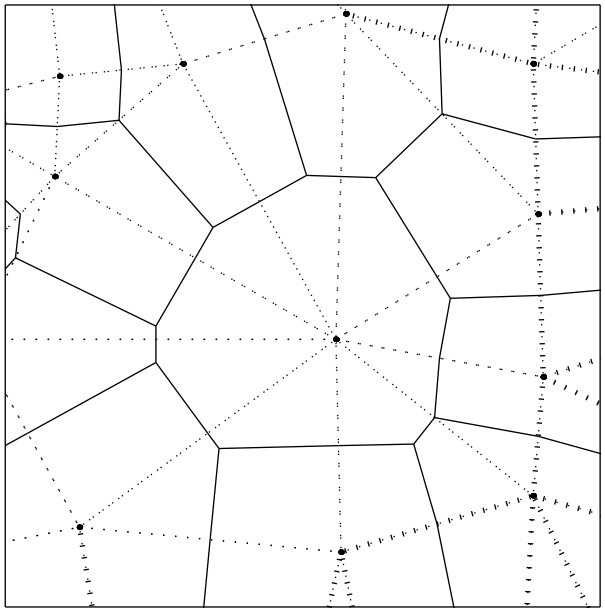
Figure 5.2: An example of problems using centroids for finite volumes.



two cells sometimes doesn't intersect the line joining them, this has the advantage of naturally corresponding to a finite element method, and if a Delaunay triangulation is used, making the finite volumes the Voronoi cells.

The method of finite elements is the second popular choice. It is not as easy to interpret physically, but its simple mathematical structure makes proofs of convergence and generalizations to higher order approximations simpler. The Galerkin formulation of the method essentially approximates u as a linear combination $\sum_{j=1}^{n} u_j \phi_j$ of a finite set of basis functions ϕ_j, then seeks to solve the PDE in a weak sense by requiring $\int_\Omega \mathcal{L}u\phi_i = \int_\Omega f\phi_i$ for all i. This is just a finite linear system for the coefficients u_i. Typically the basis functions are piecewise Lagrange polynomials, with ϕ_i equal to 1 at node i and 0 at other nodes so u_i represents the value of u at node i. If the diffusion term is integrated by parts, the differentiability requirement on the basis functions is reduced, and so actually the most popular choice for second order elliptic equations is piecewise Lagrange polynomials which are linear in each triangle.

Figure 5.3: An example of cells using circumcentres or midpoints.



It can be shown that often the two methods give exactly the same (or almost the same) discretization. In fact, proofs of convergence for finite volumes often go the route of interpreting the method as a finite elements with a particular choice of numerical quadrature for the integrals. It's similarly possible to interpret the linear/triangle finite element method as a finite volume technique. Since finite volumes are particularly good at convection (allowing upstream weighting to be easily implemented) whereas finite elements handle diffusion in a simpler and more elegant way, many people exploit their compatibility in combining the techniques. This is how I have chosen to discretize the PDE.

As noted in 1D, discontinuities in the diffusion coefficient must be handled carefully. Unfortunately, it is not yet clear how to properly treat discontinuous anisotropic diffusion tensors in 2D. For example, the simple-minded approach of taking component-wise harmonic means is clearly wrong since it is not rotationally invariant. Return instead to the physical intuition of K measuring the average speed of randomly moving particles. The negative gradient of the quantity $-\nabla u$ represents the natural diffusive "force" (I use the term with hesitation, as the physics of this argument haven't been clearly worked out) propelling particles. The possibly anisotropic resistance of the medium then results in an average velocity of $-K\nabla u$.

Now, let ϕ_i be the piecewise linear basis function for node i, and approximate the solution as $u = \sum_{j=1}^{n} u_j \phi_j$. The Galerkin condition for the diffusion term (ignoring the other terms for now) gives the following for every ϕ_i:

$$\int_\Omega (\nabla \cdot K\nabla u)\phi_i \, dx \, dy = \int_\Omega f\phi_i \, dx \, dy$$

Integrating by parts:

$$\int_{\partial\Omega} (K\nabla u)\phi_i \cdot \hat{n} \, ds - \int_\Omega (K\nabla u) \cdot \nabla\phi_i \, dx \, dy = \int_\Omega f\phi_i \, dx \, dy$$

As in 1D, the natural boundary conditions are based on the diffusive flux $(K\nabla u) \cdot \hat{n}$, and so the boundary integral can be treated as a known quantity to subtract from the right-hand side (or can simply be assumed 0 for Dirichlet nodes i). This reduces the problem to evaluating:

$$\begin{aligned} -\int_\Omega (K\nabla u) \cdot \nabla\phi_i \, dx \, dy &= -\int_\Omega \left(K\nabla \sum_{j=1}^{n} u_j\phi_j \right) \cdot \nabla\phi_i \, dx \, dy \\ &= \sum_{j=1}^{n} u_j \left(-\int_\Omega (K\nabla\phi_j) \cdot \nabla\phi_i \, dx \, dy \right) \end{aligned}$$

Then the diffusion contribution to the matrix is:

$$A_{ij} = -\int_{\Omega} (K\nabla\phi_j) \cdot \nabla\phi_i \, dx \, dy$$

Since each ϕ_i is linear on each triangle, it is natural to break up this integral into integrals over each triangle where both ϕ_i and ϕ_j are nonzero. Normally the full matrix is "assembled" in this fashion, computing the submatrix of nonzero components from each triangle separately.

The problem is then reduced to evaluating

$$-\int_{\triangle} (K\nabla\phi_j) \cdot \nabla\phi_i \, dx \, dy$$

for some triangle $\triangle$ over which ϕ_i and ϕ_j are nonzero. Note that that means i and j must be vertices of $\triangle$; for the moment, assume that $i \neq j$. As mentioned above the vector $-\nabla\phi_j$ represents the diffusive "force" propelling particles away from node j, and since $\nabla\phi_i$ is a vector pointing in the direction of node i, the term can be interpreted as the average speed of particles diffusing from node j towards node i. Then what the term *should* be is not an arithmetic mean of speeds over the triangle, but a harmonic mean over paths from j to i in the triangle. Noting that $\nabla\phi_i$ and $\nabla\phi_j$ are constant from linearity and assuming that K is only known at the nodes, the approximation is:

$$\text{HM}(-|\triangle|(K_j\nabla\phi_j) \cdot \nabla\phi_i, -|\triangle|(K_i\nabla\phi_i) \cdot \nabla\phi_j)$$

where HM is the harmonic mean and $|\triangle|$ is the area of triangle $\triangle$. It is possible for non-constant K that the K_i and the K_j speeds will have different signs, in which case the harmonic mean is inappropriate and the arithmetic mean is used instead. Finally, just as the diffusion part of the operator is zero for constant functions, the matrix should be zero for constant vectors (or equivalently, since the PDE conserves mass, the discretization should as well). This means $A_{ii} = -\sum_{j\neq i} A_{ij}$.

Notice that reassuringly this scheme gives a self-adjoint matrix when K is self-adjoint, and for K constant it reverts to the standard, well-studied finite element approximation. For non-constant K this discretization scheme is debatable of course, but can be viewed as a particular first-order correct quadrature rule for the standard method's integrals, thus guaranteeing reasonable behaviour—in any case, this is not a central issue for this thesis.

For the convection term, define the cells with midpoints of edges and circumcentres or nearest midpoints, as discussed before. Then integrating the convective term over a cell gives:

$$\begin{aligned}\int_{C_i} -\nabla\cdot(bu)\,dx\,dy &= -\int_{\partial C_i} ub\cdot\hat{n}\,ds \\ &= -\sum_{\sigma\in\,\partial C_i}\int_{\sigma} ub\cdot\hat{n}\,ds\end{aligned}$$

where the summation is over the segments σ making up the boundary of cell C_i. The usual approximations are made:

$$\int_{C_i} -\nabla\cdot(bu)\,dx\,dy \approx -\sum_{i\to j}|\sigma_{ij}|(b\cdot\hat{n})_{ij+1/2}u_{ij+1/2}$$

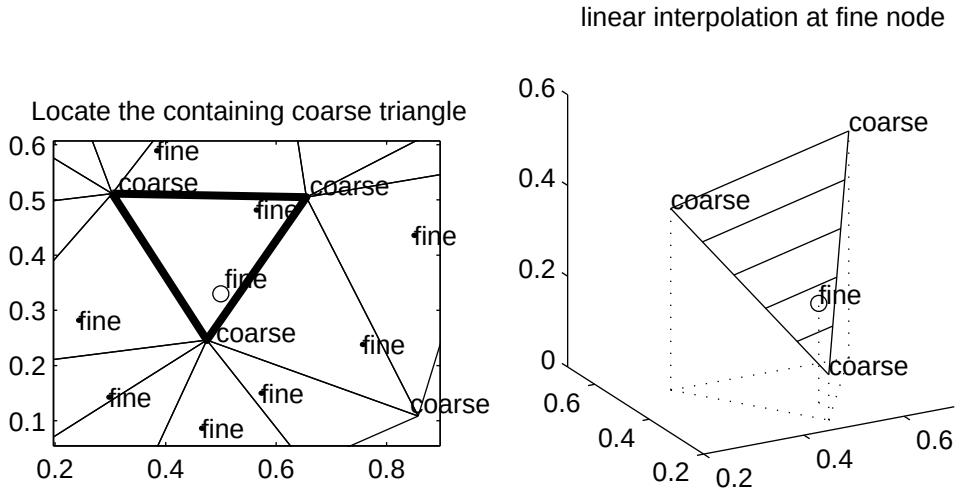
Here the summation is over nodes j connected to node i, σ_{ij} is the segment of the interface between cells around i and j, $|\sigma_{ij}|$ its length, $(b\cdot\hat{n})_{ij+1/2}$ an approximate value for $b\cdot\hat{n}$ along σ_{ij} (with normal pointing from i towards j), and $u_{ij+1/2}$ an approximate value for u along the σ_{ij}. As in 1D, $(b\cdot\hat{n})_{ij+1/2}$ can be taken to be either the average of $b_i\cdot\hat{n}$ and $b_j\cdot\hat{n}$ if they have the same sign, or zero if the sign changes. Similarly, the upstream choice for $u_{ij+1/2}$ is u_i if $(b\cdot\hat{n})_{ij+1/2} > 0$ and u_j otherwise. Upstream weighting is used without exception in this thesis.

Some confusion surrounds the reaction term cu. From the pure finite element approach, the contribution to A_{ij} should be $\int_\Omega c\phi_i\phi_j$. However, since the support of the ϕ_i's overlap, this will be nonzero for $j \neq i$, i.e. off the diagonal; besides spreading out the term in a somewhat non-intuitive way, this has the undesirable effect of automatically losing the M-matrix property and accompanying stability guarantees. The accepted remedy is called mass-lumping, essentially moving the off-diagonal contributions to the diagonal, which is now $\int_\Omega c\phi_i$. Approximating this with $c_i\int_\Omega\phi_i$ allows a nice interpretation as a finite volume method, with $\int_\Omega\phi_i$ being the area of the cell formed (for example) in the midpoint and centroid construction. However, the better shaped cells used in the convection term don't necessarily have this same area; there is a somewhat disconcerting inconsistency in this approach that demands further investigation.

5.3 Basis Construction

The only requirements for the prediction operator are that it be reasonably accurate yet sparse; in principle, unstructured interpolation methods such as distance weighted averages of nearby

Figure 5.4: For simple linear interpolation, a triangular mesh of just the coarse nodes is first constructed. For each fine node, the containing triangle is found, and the fine value is predicted from its coarse corners by the plane passing through them.

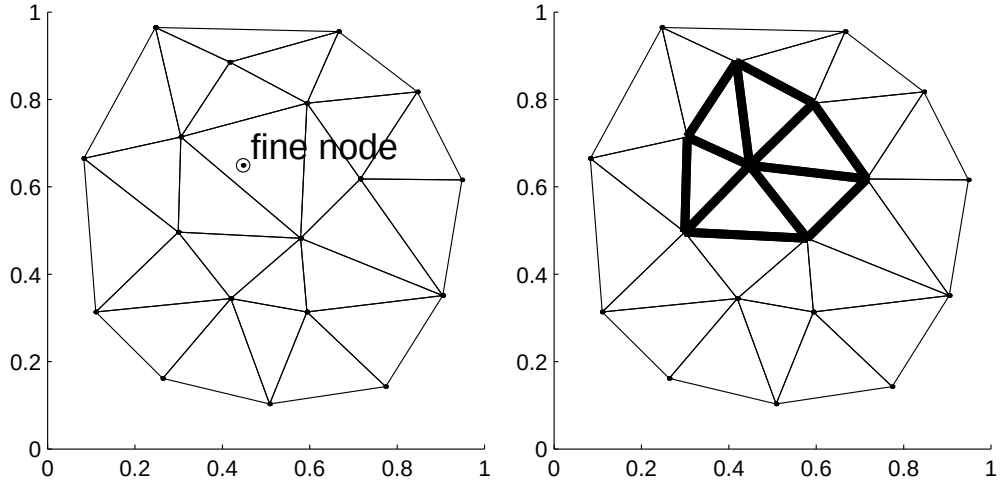


points might be used. However, the effectiveness of linear and PDE-interpolation in 1D suggest that a structured approach is the best route to follow.

Linear interpolation works by triangulating the set of coarse nodes and then constructing the piecewise linear interpolant through those nodes. The predicted value at a fine node is the appropriate linear combination of the values at the three coarse corners of the coarse triangle containing it: see figure 5.4. One of the benefits here is that the prediction operator has guaranteed sparsity: at most 3 nonzeros per fine node.

PDE-interpolation requires that a small mesh be constructed joining the fine node to nearby coarse nodes, upon which the PDE is re-discretized. This local mesh doesn't in principle require any global mesh triangulating the coarse nodes. However, a natural approach to constructing the local meshes would be to begin with a global coarse mesh and, with an incremental triangulation algorithm, insert the fine node and take the new triangles (see figure 5.5). On the other hand, particularly with stretched meshes, such an approach might connect the fine node to too many coarse nodes, allowing problems not only for the sparsity of the prediction operator but also for its accuracy if those connections are inappropriate: see figure 5.6 for an example with Delaunay

Figure 5.5: Remeshing for PDE-interpolation: add the fine node to the coarse triangle mesh, and take the newly created triangles as the local mesh on which the PDE is discretized.

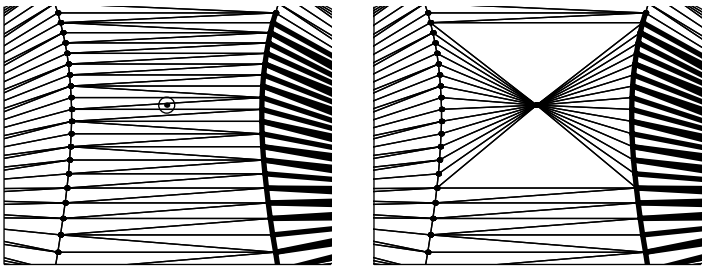


retriangulation.

The solution I have implemented is limited retriangulation. Begin with the three coarse nodes of the coarse triangle containing the fine node, just as with linear interpolation. One could then simply connect the fine node to these three and discretize on the resulting triangles, but there are difficulties with degenerate triangles if the fine node happened to be on or very close to an edge. (Although as mentioned before, it turns out that for Laplace's equation, this simplifies to linear interpolation.) Instead consider using the three additional coarse nodes from the edge-neighbouring triangles; apply the edge-swapping test of the Delaunay or coefficient-adaptive optimization routine on each of the original coarse triangle's edges, excepting boundary edges of course. Then the prediction operator again has guaranteed sparsity, at most 6 nonzeros per fine node.

Just as in 1D, Dirichlet nodes should be carried through without prediction. However, Neumann boundary nodes pose somewhat of a problem. For linear prediction, some form of extrapolation might be used; of course, extrapolation doesn't really fit a homogeneous Neumann boundary condition, where the solution should be flat in the normal direction. This wasn't a big issue in 1D, where there are at most two Neumann nodes, but for 2D a considerable proportion

Figure 5.6: Remeshing around a fine node for PDE-interpolation gone bad: before and after.



of the unknowns could be on Neumann boundaries. A slightly more reasonable choice is to do 1D linear interpolation along the boundary curve, using the neighbouring coarse boundary nodes.

For PDE-interpolation at Neumann boundary nodes, the only difficulty is figuring out the local retriangulation now that the node might not be contained in any coarse triangle, and that even if it is, it shouldn't be treated that way since in reality it is on the boundary. The obvious solution is to begin with the two neighbouring coarse boundary nodes and possibly the third node of their coarse triangle if the swapping test succeeds.

Since linear interpolation can be viewed as a special case of PDE-interpolation with the Laplacian operator and very restricted retriangulation, but might be preferable to PDE-interpolation for some applications, a third possibility suggests itself: do the PDE-interpolation described above with the Laplacian instead of the actual PDE. This simplifies the code somewhat (no convection or reaction terms need to be discretized) and gives predictions that are applicable to many different problems. With more coarse nodes involved, hopefully the accuracy will be improved over simple linear interpolation.

5.4 Automatic Mesh Coarsening

The remaining problem is to select the coarse nodes at each level. This is a challenging open issue shared with unstructured multigrid; as will be seen in the test results, the methods pre-

sented here show promise but fall short of robustness. The simplest solution that sometimes is adopted is to expect the user to provide the hierarchy of coarse meshes—possibly the result of an adaptive refinement process, where the coarse mesh is created first and nodes are added where higher resolution is needed. Unfortunately, even if such a hierarchy exists, there may be problems such as geometric smoothing operations changing the location of nodes in finer meshes or inadequacies in the quality of the hierarchy for a multi-resolution basis.

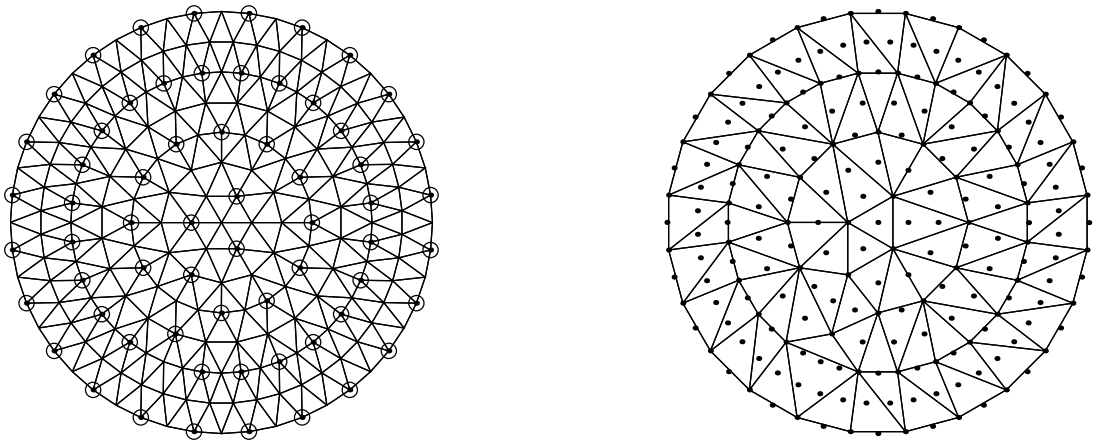
A more attractive approach is automatic mesh coarsening, where just given the finest mesh, and possibly the matrix or PDE coefficients, the computer automatically constructs the hierarchy. The difficulty of course is making such a method robust over irregular meshes and varied coefficients.

One class of methods, which I call top-down approaches, begin with the finest mesh, select a minimal set of nodes to be coarse that still allow good prediction for the remaining nodes, retriangulate the coarse nodes, and continue recursively. The simplest example is that proposed for multigrid in [12], where only the graph structure of the mesh is considered. Every second boundary node is chosen to be coarse, and then a maximal independent set is chosen from the interior nodes via a greedy algorithm on a breadth-first search from a randomly chosen root node (see figure 5.7 for an example). The maximality guarantees that each fine node is adjacent to at least one coarse node, hopefully allowing good prediction. On the other hand, the independence of the coarse nodes guarantees that there won't be too many of them, roughly a third or a quarter of the nodes for mostly regular meshes.

Some simple but important refinements to this independent set algorithm are to make sure that the fine nodes are contained inside the coarse boundary—otherwise the interpolation will be technically difficult and probably inaccurate no matter what—and to allow the user to specify a few important points, such as corners of the domain, that should be kept in all meshes. Note that Dirichlet nodes may and should be eliminated from coarser meshes to reduce the size of coarser meshes; they just shouldn't be treated as fine nodes which can be predicted.

One difficulty caused by ignoring the geometry and the PDE is that stretched meshes or anisotropic problems may be handled incorrectly. If there is strong coupling in one direction the coarsening should only take place along that direction, since predicting a fine node from weakly coupled nodes will inevitably fail. This technique is known as "semi-coarsening", and has proven to be invaluable for multigrid. To correct this deficiency in the independent set al-

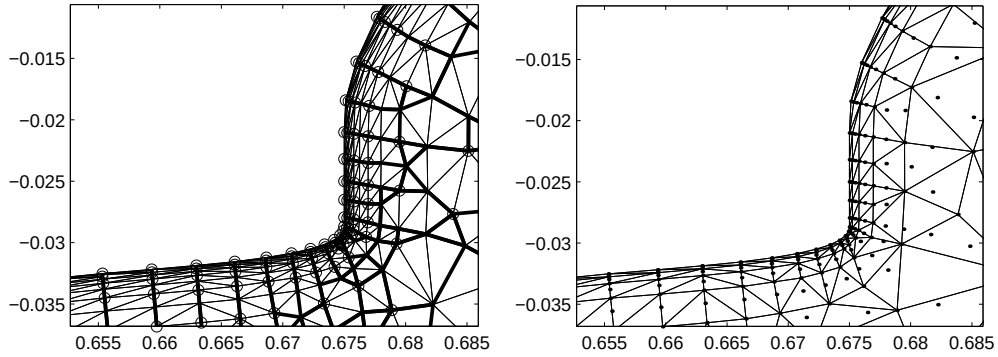
Figure 5.7: In top-down unweighted coarsening, every second boundary node along with a maximal independent set of interior nodes form the coarse nodes. They are retriangulated, and the process may continue recursively.



gorithm, I propose a simple modification: pre-process the graph of the mesh, deleting weak couplings. Discretize the PDE on the mesh to get a matrix A, and to each directed edge $i \to j$ of the graph, attach the weight $|A_{ij}| + |A_{ji}|$. This models the size of the PDE or adjoint interpolation coefficient when predicting j from i (recall that I use the same coarse node hierarchy for the PDE and its adjoint). Then delete any directed edges $i \to j$ with magnitude less than half the maximum of any edge to j. Now, the independent set algorithm will only mark a node as fine if there is a neighbouring coarse node giving a large PDE or adjoint interpolation coefficient, i.e. if there is a coarse neighbour that can be used for effective prediction. To ensure this happens at the boundary as well, any boundary nodes with a strong connection to an interior node should be kept coarse. Note that the discretization and subsequent dropping of small entries is only used for constructing the hierarchy of meshes, *not* for interpolation. See figure 5.8 for an example.

For problems with anisotropic coefficients, the retriangulation of the coarse nodes (in order to generate the next level) may run into difficulties if simple Delaunay triangulation is used. Coefficient-adaptive triangulation potentially can do a better job, as will be seen in the testing,

Figure 5.8: In top-down *weighted* coarsening, the maximal independent set of interior nodes is chosen from a graph of the fine mesh with weak connections removed. Here the strong connections are in bold; the second image shows the retriangulation of coarse nodes. Notice how coarsening is done only in the direction orthogonal to the stretching: semi-coarsening.



serving to undo the anisotropies in the coefficients via cancelling anisotropies in the mesh.

Another class of coarsening algorithms, which I call the decremental approach, begin with the finest mesh and select nodes one by one, deleting and retriangulating at each step. The node picked at each step should be the one easiest to predict; after enough nodes have been deleted, the mesh is saved as the next coarsest level, and the process continues. An example is of this approach is given in [27]. The serial nature of these algorithms discouraged me however. If a decremental algorithm were parallelized, say by eliminating many non-interacting nodes at each step, the result would probably be a somewhat obfuscated top-down approach anyhow.

A third class of coarsening algorithms I call the bottom-up approach. A potential problem with the top-down approach is that the quality of the meshes may be degraded as they get coarser. Seemingly inconsequential details like specifying that the greedy independent set algorithm should work on a breadth-first search actually can have a big effect. A kind of instability may be apparent: mistakes made at one level (e.g. marking an essential node at some kind of junction as fine) can be propagated down to lower levels. Another difficulty is when to stop: automatically identifying when the coarsest possible mesh (that will allow useful accuracy) has been reached. A more robust alternative would be to choose the coarsest mesh first, designed to approximate the solution of the PDE as well as possible (and presumably in this construc-

tion, there would be a way to identify how useful the mesh is). The intermediate meshes in the hierarchy can then be filled in by adding nodes that will allow good prediction at finer levels.

One bottom-up method I have begun to investigate is to choose the coarsest nodes as a set of p-centres in the graph of the fine mesh (possibly weighted in a way similar to that discussed above), i.e. a set of p nodes so that the graph distance between any other node and a coarsest node is minimized. Unfortunately this is an NP-complete problem[19], but a heuristic algorithm might prove effective. A plausible approach is to take an initial guess (e.g. the coarsest level from a top-down algorithm) and iteratively improve it with small, greedy adjustments. Coupling this with a multi-level acceleration, as is done with spectacular success for the NP-complete problem of graph partitioning[24], might prove to be ideal.

5.5 Test Problems

The following two-dimensional test problems were chosen to be representative of the actual problems faced in several different applications. They include irregular meshes, discontinuous coefficients, anisotropy, and convection.

The following subsections give the details of the testing; section 5.6 summarizes the results.

5.5.1 Testing Protocol

The methods listed in the tables are:

- ILUT(δ): the drop-tolerance form of ILU, a popular and generally high quality preconditioner for PDE problems. See [32] for details. The ordering is Reverse Cuthill-McKee[20, 11].
- AINV(δ): the standard basis inner-product AINV with drop tolerance δ, after nested dissection ordering.
- Mr.Lin(δ): a multi-resolution basis with linear interpolation, based on unweighted coarsening. Outer-product AINV with drop tolerance δ is then used, with nested dissection ordering modified for the multi-resolution basis.

- Mr.Lap(δ): a multi-resolution basis with Laplacian interpolation, based on unweighted coarsening. Outer-product AINV with drop tolerance δ is then used, with modified nested dissection ordering.
- Mr.PDE(δ): PDE-interpolation with unweighted coarsening; AINV with drop tolerance δ after modified nested dissection ordering.
- +Mr.Lin(δ), +Mr.Lap(δ), +Mr.PDE(δ): the same as above, only with weighted coarsening using Delaunay retriangulation.
- ++Mr.Lin(δ), ++Mr.Lap(δ), ++Mr.PDE(δ): the same as above, with weighted coarsening using coefficient-adaptive retriangulation.
- $\star$Mr.Lin(δ), $\star$Mr.Lap(δ), $\star$Mr.PDE(δ): the same as above, with weighted coarsening using coefficient-adaptive retriangulation applied separately to each region with near constant coefficients, then stitched up into a global triangulation.

For the multi-resolution bases, enough levels were allowed so that the coarsest level had about 100 nodes, except as noted in the problem commentary.

The drop tolerances were chosen to give roughly the same number of nonzeros in each preconditioner on the coarsest mesh tested for a particular problem.

The symmetric definite problems were solved with CG and the preconditioned system $\mathbf{D}^{-1/2}\mathbf{Z}^T(\mathbf{M}^{-T}\mathbf{A}\mathbf{M}^{-1})\mathbf{Z}\mathbf{D}^{-1/2}$. BiCGStab with $\mathbf{D}^{-1}\mathbf{W}^T(\mathbf{M}_\beta^{-T}\mathbf{A}\mathbf{M}_\alpha^{-1})\mathbf{Z}$ was used for the others. Convergence was flagged when the 2-norm of the residual (with Dirichlet nodes rescaled appropriately, as mentioned before) was decreased by a factor of 10^{-6} beginning from an initial guess of all zeros; if convergence wasn't reached after 1000 iterations, the problem was marked unsolved with an asterisk (*).

After each iteration count in the tables, the "work per unknown" is included in parentheses: the number of iterations times the number of nonzeros in the preconditioner (prediction and update operators included), divided by the number of unknowns. This allows a somewhat fairer comparison between different preconditioners and problems. However, it should be noted that this may be somewhat misrepresentative, particularly as the matrix multiplies of approximate inverses often can be better implemented than the triangular solves of ILU on high performance

hardware; as shown in [4] for example, even when the number of flops required by an approximate inverse is the same as ILU, the approximate inverse can still run significantly faster. Unfortunately the code is still in the prototype stage, with some parts running interpreted in MATLAB and others compiled in C or FORTRAN, so timing counts are not included here.

The triangulation routines for coarsening were adapted from TRIPACK[30].

5.5.2 Problem 1: Poisson equation on a uniform disc

This is Poisson's equation on an unstructured but fairly uniform mesh of a disc (see figure 5.9. To be precise, the PDE is:

$$\nabla^2 u = f$$

where

$$f(x,y) = \begin{cases} 0 & : \quad x \leq 0 \\ -1 & : \quad x > 0 \end{cases}$$

and all boundaries are homogeneous Dirichlet. The solution is plotted in figure 5.10.

For the iterations both the simple independent set coarsening and the weighted independent set coarsening (figure 5.11) were used, stopping at around 100 nodes in the coarsest mesh. Table 5.1 gives the iteration results, with a plus sign before the bases with weighted independent set coarsening. For this problem, PDE-interpolation and Laplacian interpolation are the same thing, so only one is listed.

It is interesting to note that in 2D, the difference between the standard basis and the multi-resolution basis isn't nearly as dramatic. The basic reason for this is that the Green's function decays linearly in 1D but logarithmically in 2D, making a sparse approximate inverse in the standard basis more feasible (since more entries are close to zero), while at the same time the prediction operators become denser and less attractive. Furthermore, while in 1D the optimal interpolation actually becomes exact giving the cyclic reduction direct method, optimal interpolation in 2D still falls short of exact—fine nodes are no longer independent of each other.

As the problem size increases, ILUT and standard basis AINV both slow down roughly like $O(n^{3/2})$. The multi-resolution bases don't quite achieve grid-independent convergence, but come very close. The weighted coarsening is superior to the unweighted, but for this problem

Figure 5.9: Unstructured but uniform triangulation of the disc.

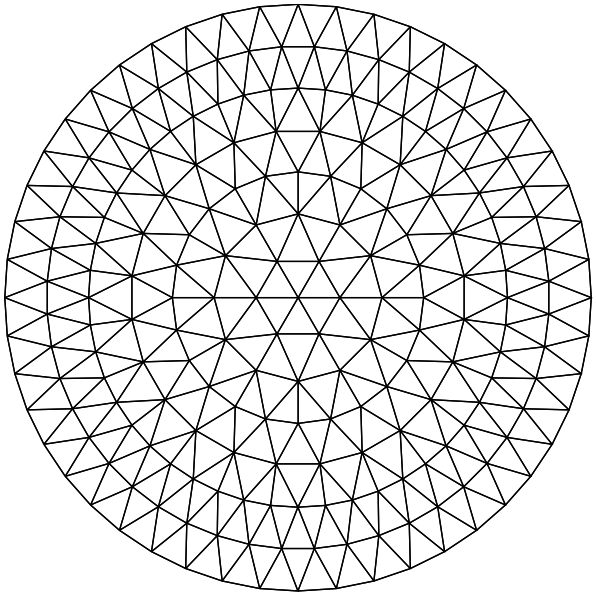
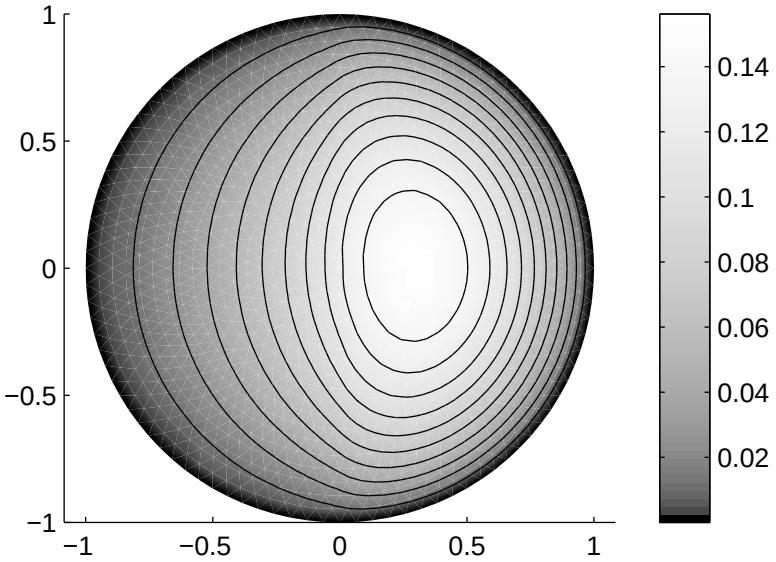


Table 5.1: CG iterations for 2D problem 1 (Poisson equation) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 1195$	$n = 4939$	$n = 20011$	$n = 79531$
ILUT(0.009)	13 (84)	24 (164)	47 (327)	92 (643)
AINV(0.08)	32 (200)	63 (417)	126 (851)	251 (1724)
Mr.Lin(0.12)	23 (147)	26 (177)	32 (222)	35 (240)
Mr.Lap(0.1)	18 (120)	21 (149)	23 (167)	26 (191)
+Mr.Lin(0.12)	22 (142)	26 (176)	29 (201)	37 (258)
+Mr.Lap(0.1)	19 (122)	20 (136)	21 (145)	25 (172)

Figure 5.10: Solution of 2D problem 1 (Poisson equation).



the difference isn't terribly significant—the finest mesh and problem are basically isotropic, so there isn't much opportunity for semi-coarsening. Clearly the denser prediction operators in Laplacian interpolation are more effective than extra nonzeros in the approximate inverse with sparser linear interpolation, but both provide fast solutions.

5.5.3 Problem 2: heat equation on a uniform disc

This is just a step in an implicit solve of the heat equation on an unstructured but fairly uniform mesh of a disc (see figure 5.9. To be precise, the PDE is:

$$\nabla^2 u - 0.1u = f$$

where

$$f(x,y) = \begin{cases} 0 & : \quad x \leq 0 \\ -1 & : \quad x > 0 \end{cases}$$

and all boundaries are Robin (steady state Neumann plus the reaction term from the time derivative). The solution is plotted in figure 5.12, and iteration results are given in table 5.2.

Figure 5.11: Coarsening of uniform disc (unweighted above, weighted below).

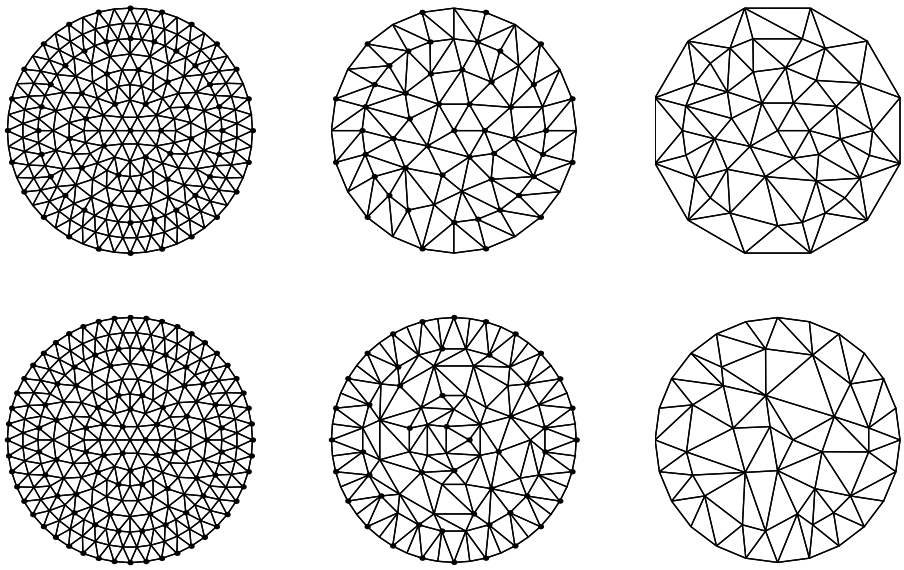
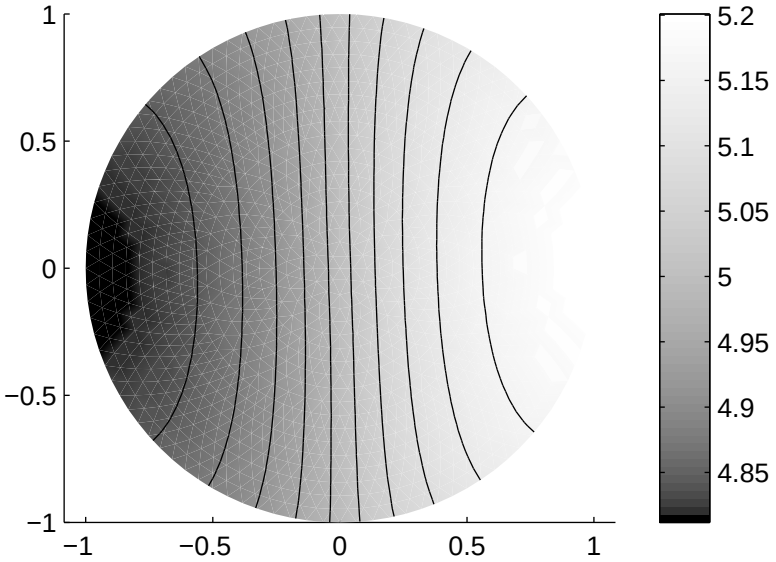


Table 5.2: CG iterations for 2D problem 2 (heat equation) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 1195$		$n = 4939$		$n = 20011$		$n = 79531$	
ILUT(0.01)	22	(151)	41	(284)	79	(549)	154	(1069)
AINV(0.085)	54	(375)	108	(733)	218	(1462)	434	(2898)
Mr.Lin(0.12)	36	(244)	40	(279)	46	(318)	53	(373)
Mr.Lap(0.1)	28	(195)	30	(216)	34	(250)	38	(278)
Mr.PDE(0.1)	28	(194)	30	(214)	34	(246)	38	(274)
+Mr.Lin(0.11)	33	(239)	38	(277)	42	(308)	49	(359)
+Mr.Lap(0.1)	27	(183)	29	(202)	31	(216)	35	(242)
+Mr.PDE(0.09)	26	(180)	28	(198)	30	(212)	34	(239)

Figure 5.12: Solution of 2D problem 2 (heat equation).



There are no surprises here. PDE-interpolation is slightly better than Laplacian interpolation, but the PDE is so close to the Laplacian the difference isn't remarkable.

5.5.4 Problem 3: heat equation on a stretched mesh

This problem is also an implicit step of solving the same heat equation with non-homogeneous boundary conditions:

$$\nabla u \cdot \hat{n} = \text{sign}(\cos(20\theta))$$

where θ is the angle from the origin and the x-axis. The mesh is exponentially stretched towards the boundary—beginning with the same uniform mesh as before, the new distance $\bar{r}$ from the origin is $1 - 25^{-r}$. See figures 5.13 and 5.14 for the mesh and the solution, and table 5.3 for the iteration results.

Now the advantage of semi-coarsening begins to become apparent: see figure 5.15 for a comparison of the coarser meshes in the hierarchies. The unweighted approach preserves the

Figure 5.13: Stretched mesh on disc.

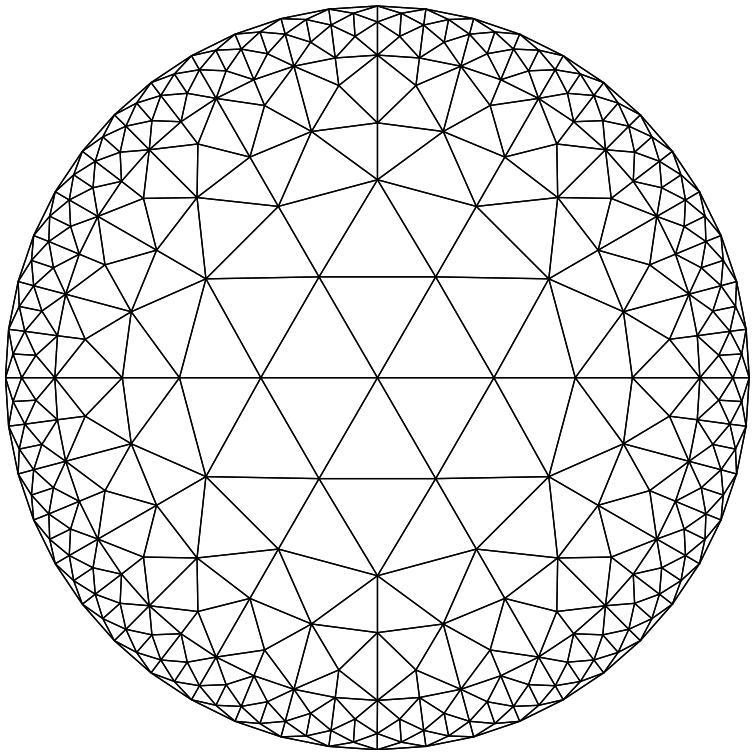


Figure 5.14: Solution of 2D problem 3 (stretched mesh heat equation).

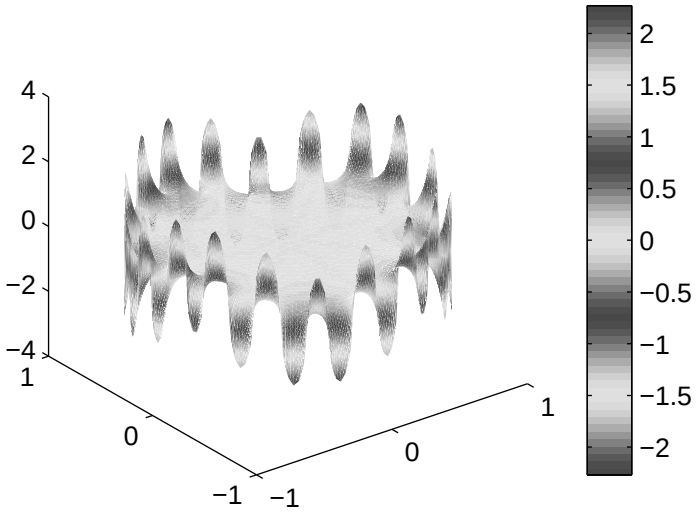
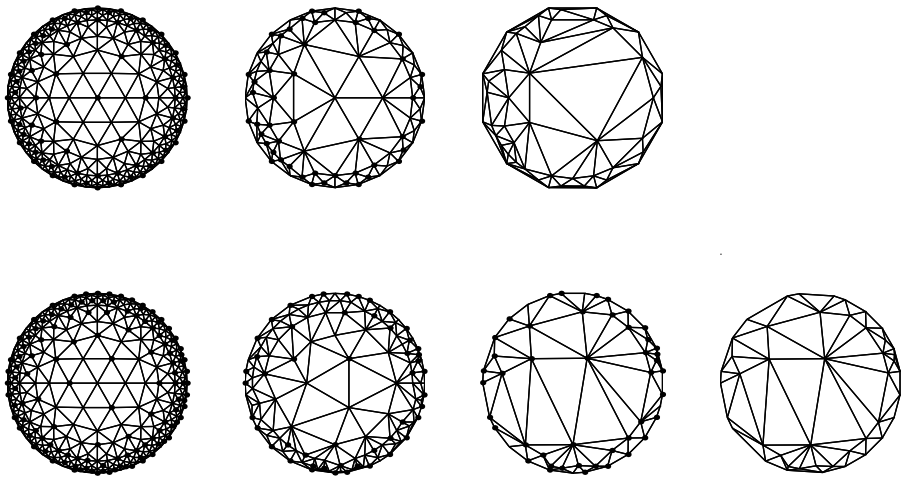


Table 5.3: CG iterations for 2D problem 3 (stretched mesh heat equation) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 1195$		$n = 4939$		$n = 20011$		$n = 79531$	
ILUT(0.009)	22	(149)	42	(293)	73	(513)	153	(1071)
AINV(0.085)	61	(411)	125	(865)	215	(1415)	432	(2806)
Mr.Lin(0.13)	51	(339)	65	(446)	78	(532)	84	(566)
Mr.Lap(0.12)	35	(232)	36	(250)	43	(298)	46	(319)
Mr.PDE(0.11)	35	(237)	34	(239)	40	(281)	44	(309)
+Mr.Lin(0.095)	29	(200)	32	(226)	38	(276)	42	(305)
+Mr.Lap(0.075)	21	(141)	23	(168)	25	(178)	28	(194)
+Mr.PDE(0.075)	21	(141)	23	(166)	25	(174)	28	(189)

Figure 5.15: Sample coarsening hierarchies for stretched disc mesh (unweighted method above, weighted method below)



stretching in the coarser levels, causing near degenerate triangles by the boundary. On the other hand, the weighted approach works to undo the anisotropy, leaving considerably better conditioned meshes from which more accurate interpolation is possible.

5.5.5 Problem 4: Laplace's equation around a simple airfoil

The next problem is Laplace's equation $\nabla^2 u = 0$ around a simple airfoil. There are homogeneous Neumann boundary conditions around each section of the airfoil, and a farfield wind coming slightly from below is approximated by imposing the Dirichlet condition $u = x + 0.3y$ on the exterior boundary. See figure 5.16 for a plot of the mesh, which is highly nonuniform but not stretched, and figure 5.17 for the solution. The mesh hierarchies were constructed with the trailing tip of the foil specified as a key point to keep coarse. Iteration results are given in table 5.4.

Figure 5.16: Mesh for 2D problem 4 (simple airfoil).

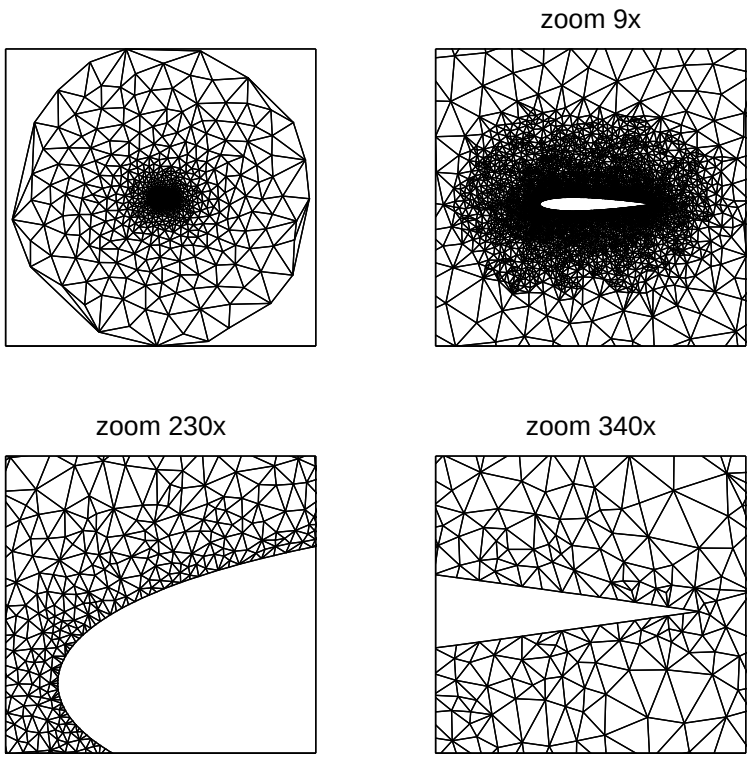


Figure 5.17: Solution of 2D problem 4.

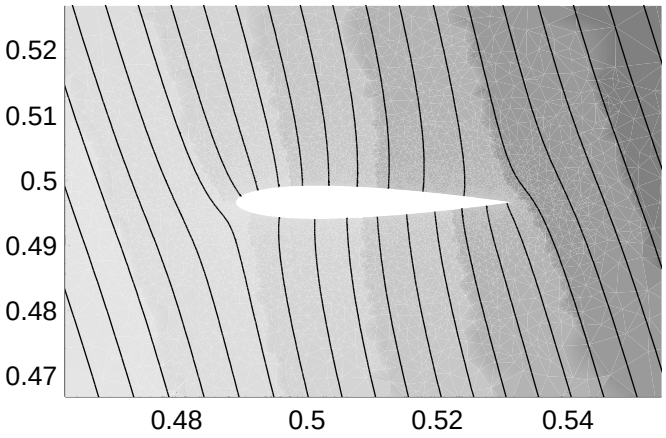


Table 5.4: CG iterations for 2D problem 4 (simple airfoil) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 6691$	
ILUT(0.01)	49	(314)
AINV(0.09)	144	(939)
Mr.Lin(0.12)	62	(402)
Mr.Lap(0.12)	35	(242)
+Mr.Lin(0.11)	36	(234)
+Mr.Lap(0.11)	30	(203)

Table 5.5: CG iterations for 2D problem 5 (multi-segment airfoil) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 8607$	
ILUT(0.004)	43	(229)
AINV(0.08)	170	(908)
Mr.Lin(0.2)	189	(1023)
Mr.Lap(0.4)	291	(1666)
+Mr.Lin(0.12)	54	(299)
+Mr.Lap(0.25)	44	(247)

5.5.6 Problem 5: Laplace's equation around a multi-segment airfoil

This time a multi-segment airfoil with stretched mesh is used, making the problem considerably more difficult. Figure 5.18 shows the new mesh and figure 5.19 the solution. All trailing tips were kept coarse, and to allow for the complex geometry, the coarsening was stopped at about 200 nodes: adequately representing the geometry with fewer nodes appears too difficult. Iteration results are given in table 5.5.

This is probably the best example of the importance of semi-coarsening. The unweighted independent set algorithm gives such a bad hierarchy that the standard basis is better, and the normally more accurate Laplacian interpolation is actually worse than linear interpolation. However, the weighted independent set method gives reasonable convergence—perhaps not as good as one might hope, but probably there is still considerable room for improvement in the coarsening.

Figure 5.18: Mesh for 2D problem 5 (multi-segment airfoil).

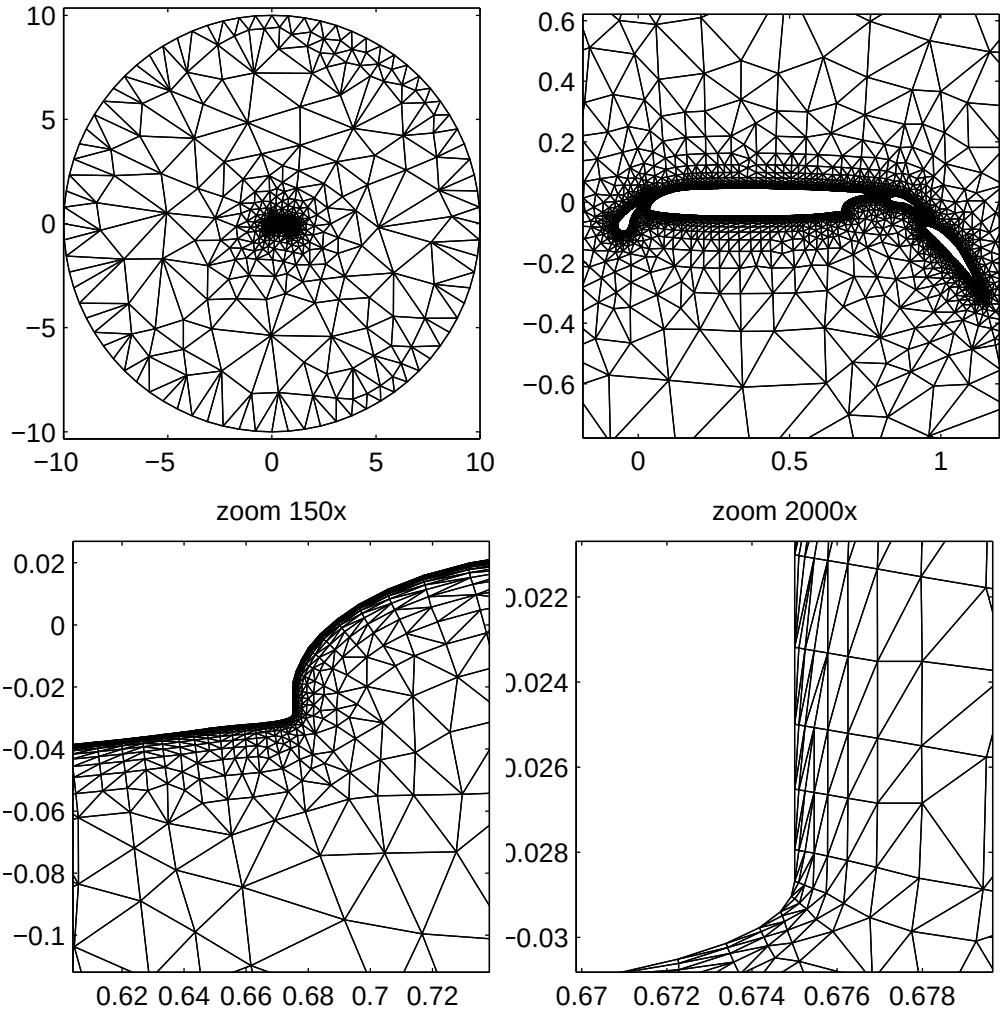
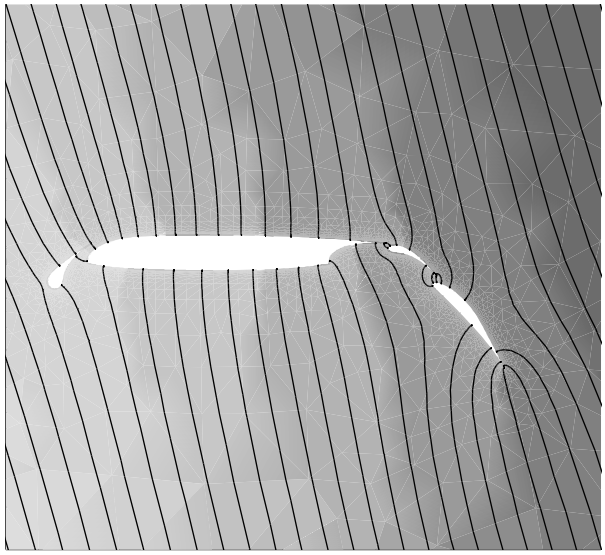


Figure 5.19: Solution of 2D problem 5.



5.5.7 Problem 6: discontinuous coefficient heat equation

We now try introducing discontinuous coefficients in the heat problem on the irregular mesh in figure 5.20, generated with MATLAB's PDETOOL. The PDE is:

$$\nabla \cdot K \nabla u - 10^{-3} u = f$$

where

$$K(x,y) \quad = \quad \begin{cases} 1 & : \quad x \leq 0 \\ 10^{-6} & : \quad x > 0 \end{cases}$$

and $f = -1$ on the left disc but 0 elsewhere, with Neumann boundary conditions. The sharp corners of the mesh, apart from the tiny step in the bottom straight section, are kept coarse.

For this problem, clearly linear interpolation is doing the wrong thing, and Laplacian interpolation is even worse. PDE-interpolation works very nicely still. The mesh is fairly isotropic, so semi-coarsening only helps a little.

Figure 5.20: Mesh for problem 6 (discontinuous heat equation).

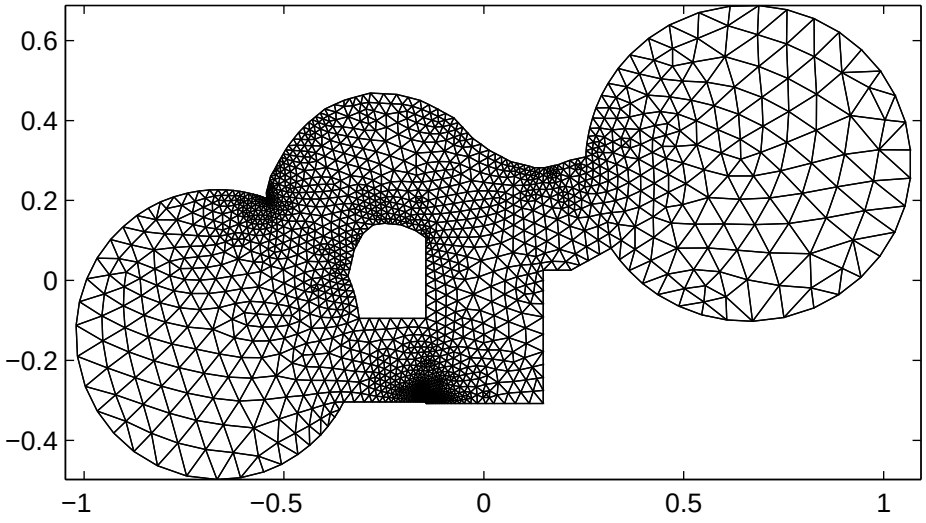


Figure 5.21: Solution of 2D problem 6.

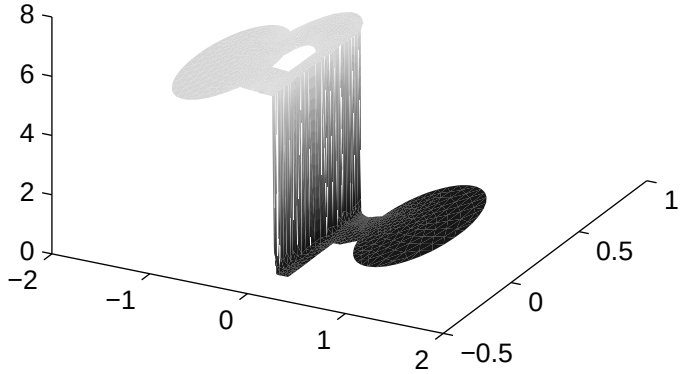


Table 5.6: CG iterations for 2D problem 6 (discontinuous heat equation) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 1918$		$n = 7420$	
ILUT(0.009)	25	(145)	50	(310)
AINV(0.08)	66	(404)	140	(814)
Mr.Lin(0.15)	287	(1964)	*	
Mr.Lap(0.3)	522	(3247)	*	
Mr.PDE(0.11)	31	(202)	34	(225)
+Mr.Lin(0.2)	712	(4024)	*	
+Mr.Lap(0.4)	*		*	
+Mr.PDE(0.11)	27	(178)	31	(206)

5.5.8 Problem 7: simple anisotropy

This is a rather simple constant coefficient problem on a uniform square mesh, but the diffusion tensor coefficient is highly anisotropic. The PDE is:

$$1000u_x + u_y = \frac{1}{20}sin(10\pi y)$$

with homogeneous Neumann boundaries for $y > 0.25$ and the Dirichlet boundary condition $u = x$ for $y \leq 0.25$. See figure 5.22 for the solution.

For the multi-resolution bases, the corners of the mesh are kept coarse. As shown in figures 5.23 unweighted coarsening, the semi-coarsening with Delaunay retriangulation, and semi-coarsening with coefficient-adaptive retriangulation are tested—this last is marked with two plusses in front of the method in table 5.22. Note that while semi-coarsening with Delaunay retriangulation begins with the correct choices of coarse nodes, on the boundary the out-of-phase placement of coarse nodes causes the Delaunay algorithm to generate inappropriate triangles. The mistake is amplified in coarser levels, a fundamental problem with the top-down approach.

The unweighted coarsening does a terrible job, while the semi-coarsening is effective. However, coefficient-adaptive retriangulation is much more effective—it appears not just by a con-

Figure 5.22: Solution of 2D problem 7 (simple anisotropy).

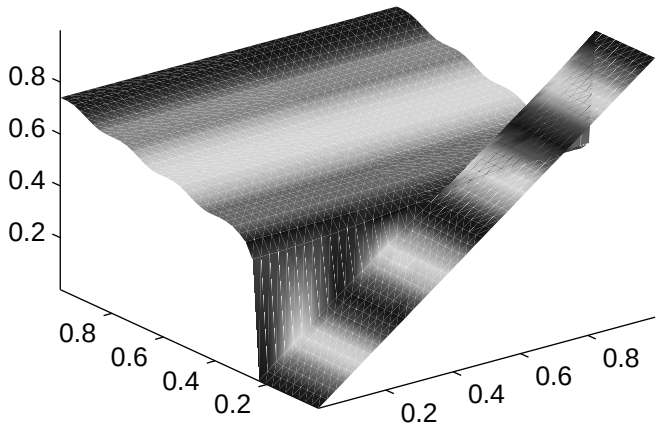


Table 5.7: CG iterations for 2D problem 7 (simple anisotropy) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 900$		$n = 3600$		$n = 14400$	
ILUT($3.2 \cdot 10^{-4}$)	21	(128)	38	(224)	65	(380)
AINV(0.01)	31	(167)	59	(396)	114	(828)
Mr.Lin(0.3)	488	(2623)	605	(3399)	592	(3422)
Mr.Lap(0.45)	433	(2387)	541	(3230)	525	(3243)
Mr.PDE(0.4)	314	(1695)	340	(1993)	354	(2146)
+Mr.Lin(0.01)	23	(144)	47	(366)	81	(656)
+Mr.Lap(0.55)	488	(2553)	610	(3505)	752	(4451)
+Mr.PDE(0.008)	19	(118)	35	(269)	59	(460)
++Mr.Lin(0.01)	15	(79)	21	(122)	32	(195)
++Mr.Lap(0.55)	458	(2400)	770	(4408)	*	
++Mr.PDE(0.008)	13	(65)	18	(100)	24	(137)

Figure 5.23: Coarsening for simple anisotropy (in order from top: unweighted, weighted with Delaunay retriangulation, weighted with coefficient-adaptive retriangulation)

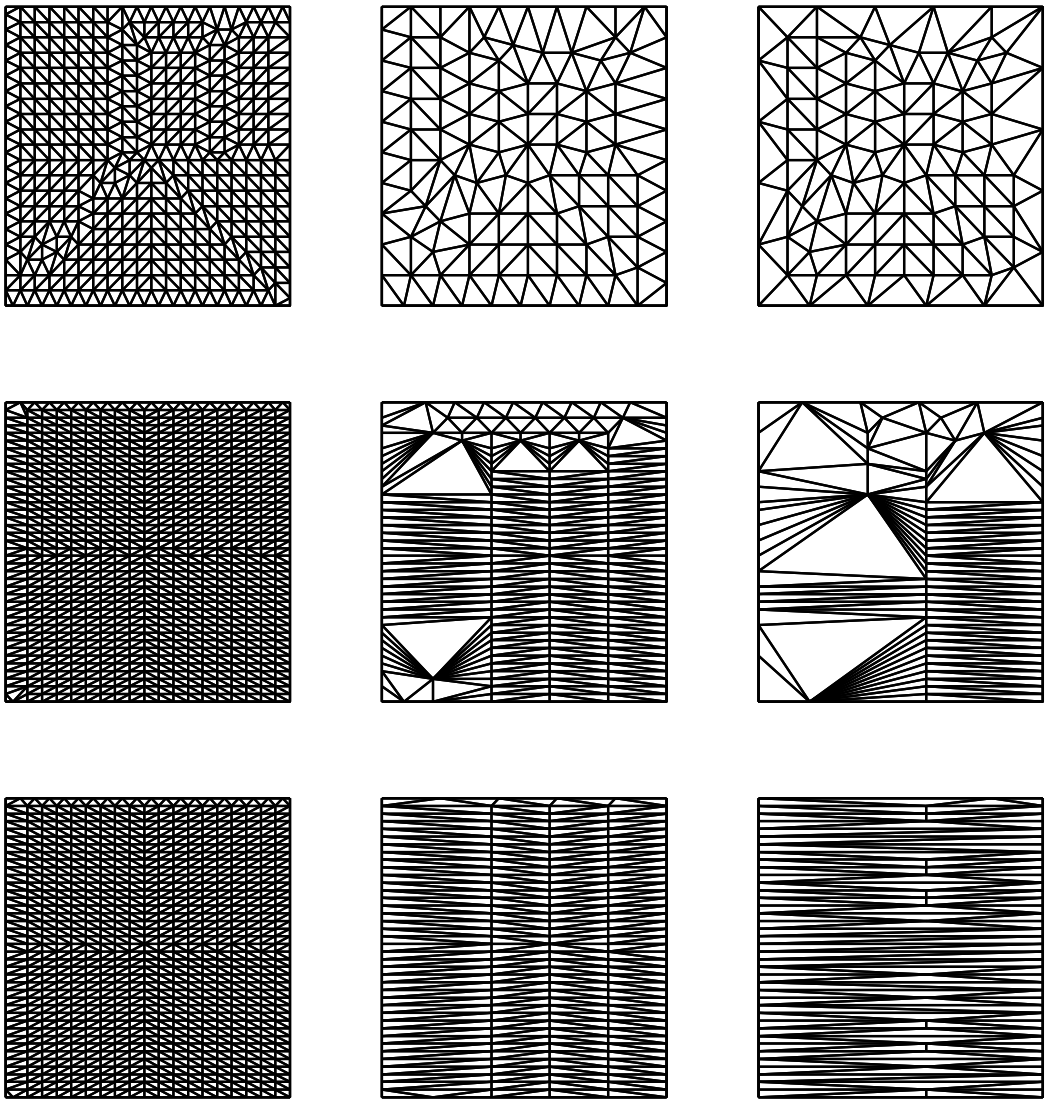
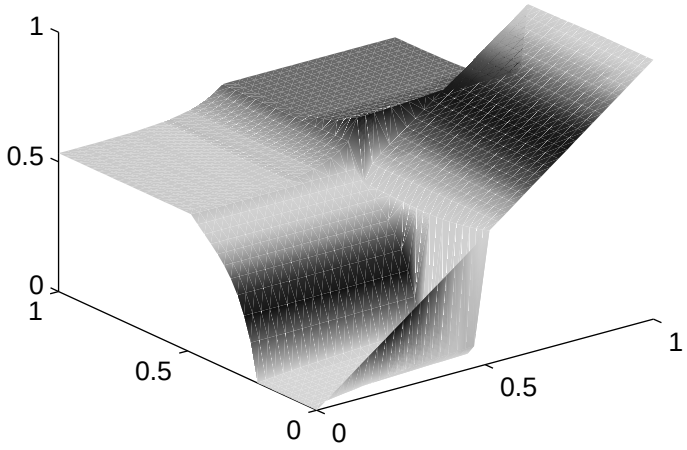


Figure 5.24: Solution of 2D problem 8 (ANISO).



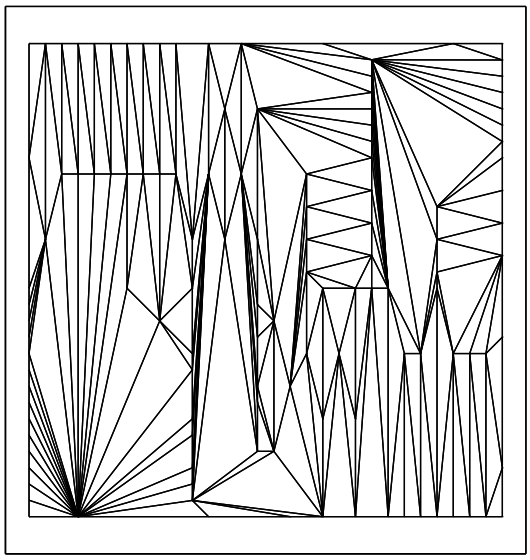
stant factor, but actually improving the scalability. However, grid-independent convergence is still not achieved. As always, PDE-interpolation is the best method (though fails with an inappropriate hierarchy). It is interesting to note that for semi-coarsening, linear interpolation is better than Laplacian interpolation here: though both make the mistake of giving equal weight to weakly coupled nodes in the y direction, the Laplacian prediction confounds the mistake by including more weakly couples nodes.

5.5.9 Problem 8: ANISO

The ANISO problem[17] is a highly anisotropic discontinuous coefficient problem. It splits the unit square into quarters, the south-west and north-east quarters satisfying $1000u_x + u_y = f$ and the other two satisfying $u_x + 1000u_y = f$. The right-hand side and boundary conditions are the same as in problem 7. See figure 5.24 for the solution.

Unweighted coarsening is useless here too. However, the discontinuities confuse the edge-swapping routine so much that coefficient-adaptive retriangulation is even worse—see figure 5.25 for an example of what goes wrong. However, adaptively retriangulating each quarter

Figure 5.25: Coefficient-adaptive triangulation gone wrong.



separately works well, as can be seen in table 5.8 where this method is labelled with stars.

As expected for an anisotropic problem, the Laplacian interpolation is terrible; PDE-interpolation is best, with linear close behind. However, even these work very poorly on the Delaunay retriangulated coarsened hierarchy; the advantage over the standard basis is only realized with the more sophisticated hierarchy. This underscores the over-riding important of good coarsening: it is the most sensitive and difficult part of the multi-resolution scheme.

5.5.10 Problem 9: a model reactor

The final self-adjoint problem is an indefinite problem that loosely models neutron diffusion and reaction. There are 21 circular rods of radius 0.2 arranged neatly in a disc of radius 0.9, with an outer shield going out to radius 1. The PDE is:

$$\nabla \cdot K \nabla u + cu = f$$

Table 5.8: CG iterations for 2D problem 8 (ANISO) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 900$		$n = 3600$		$n = 14400$	
ILUT($3.7 \cdot 10^{-4}$)	21	(110)	47	(256)	82	(429)
AINV(0.006)	39	(207)	66	(459)	113	(950)
+Mr.Lin(0.15)	133	(706)	336	(2098)	410	(2158)
+Mr.Lap(0.45)	399	(2096)	415	(2381)	486	(2894)
+Mr.PDE(0.15)	126	(656)	306	(1716)	341	(1784)
$\star$Mr.Lin(0.01)	14	(77)	16	(89)	20	(118)
$\star$Mr.Lap(0.4)	333	(1800)	382	(2217)	428	(2583)
$\star$Mr.PDE(0.01)	13	(69)	14	(76)	17	(94)

where $K = 1$ and $c = 0.3$ in the rods, $K = 0.005$ and $c = -0.2$ in the disc, and $K = 10^{-6}$ and $c = 0$ in the outer shield. The right-hand side f is -1 inside the reactor and 0 on the shield. All boundary nodes are homogeneous Neumann. See figure 5.26 for the solution.

The multi-resolution basis convergence is disappointing. PDE-interpolation still provides a better solution than the standard basis, but it's still rather slow. The discontinuities and locally indefinite regions in the rods cause catastrophic difficulties for the linear and Laplacian interpolation.

Surprisingly, the semi-coarsening is less effective than the unweighted coarsening, a further indication that this might be the real issue in unstructured multi-resolution methods, and that either the top-down approach needs to be made much more sophisticated or a different approach should be adopted.

5.5.11 Problem 10: simple convection

This is a convection-diffusion equation on a 100×100 square grid:

$$0.01\nabla^2 u - \nabla \cdot (bu) = f$$

Figure 5.26: Solution of 2D problem 9 (model reactor).

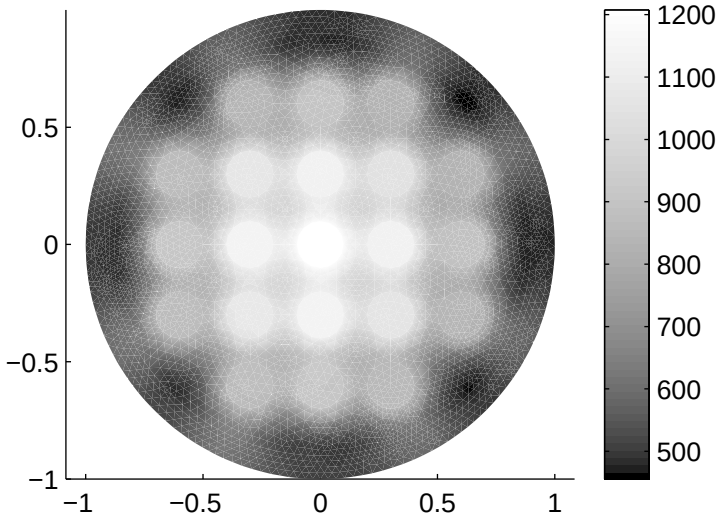
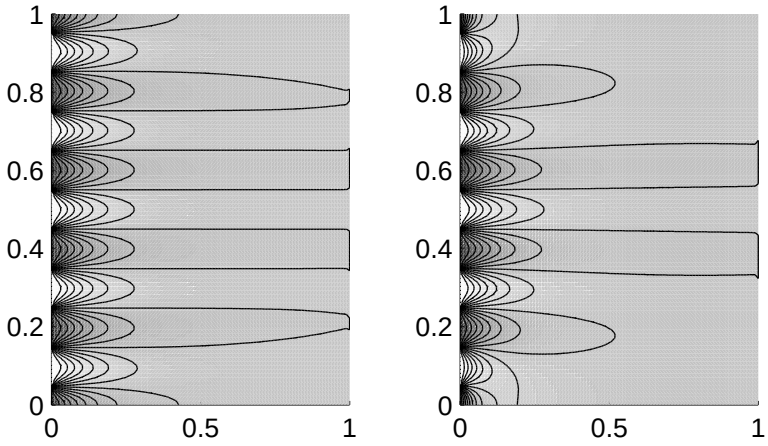


Table 5.9: CG iterations for 2D problem 9 (model reactor) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 4195$		$n = 16613$		$n = 66121$	
ILUT(0.009)	78	(546)	132	(963)	256	(1840)
AINV(0.08)	181	(1260)	355	(2473)	744	(5136)
Mr.Lin(0.15)	*		*		*	
Mr.Lap(0.3)	*		*		*	
Mr.PDE(0.11)	89	(666)	141	(1031)	132	(945)
+Mr.Lin(0.2)	*		*		*	
+Mr.Lap(0.4)	*		*		*	
+Mr.PDE(0.11)	112	(860)	154	(1143)	227	(1621)

Figure 5.27: Solutions to 2D problem 10 (simple convection)



where

$$b(x, y) = (e^x, 0)$$

with Dirichlet conditions at the sides of the square and Neumann conditions on the top and bottom:

$$\begin{aligned} u(0, y) &= \text{sign}(cos(10\pi y)) \\ u(1, y) &= 0 \\ u_y(x, 0) = u_y(x, 1) &= 0 \end{aligned}$$

A slightly more difficult problem arises when b is taken to vary with y:

$$b = \big(e^x(1 - (2y - 1)^2), 0\big)$$

Figure 5.27 shows the solutions, and table 5.10 has the results for the two problems.

The somewhat mixed results are further evidence that though PDE-interpolation works well, coarsening needs further research for robustness; the linear and Laplacian interpolation behave inconsistently, probably indicating some subtle troubles.

Table 5.10: Bi-CGstab iterations for 2D problem 10 (simple convection) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$b_1 = e^x$		$b_1 = e^x(1-(2y-1)^2)$	
ILUT(0.015)	17	(154)	29	(260)
AINV(0.1)	75	(777)	*	
Mr.Lin(0.2)	*		69	(584)
Mr.Lap(0.2)	91	(740)	191	(1504)
Mr.PDE(0.25)	71	(796)	31	(340)
++Mr.Lin(0.13)	93	(1020)	31	(342)
++Mr.Lap(0.12)	*		65	(627)
++Mr.PDE(0.12)	17	(216)	23	(273)

5.5.12 Problem 11: circular convection

This is a rather more difficult problem, as the streamlines are not straight lines but rather closed circles. The PDE is on the unit disc:

$$\nabla^2 u - \nabla \cdot (bu) - 10^{-2} u = f$$

where

$$b(x, y) = (-1000y, 1000x)$$

and

$$f(x, y) = \begin{cases} -1 & : \quad x \leq 0 \\ 0 & : \quad x > 0 \end{cases}$$

with the natural Robin boundary conditions. This is essentially one time-step in a solid-body rotation. The discretization is on the uniform triangulation of the disc from earlier problems, with solution shown in figure 5.28

Like many anisotropic problems, this should be easier since it essentially consists of a set of very weakly coupled one dimensional problems. The difficulty is that automatic methods have

Figure 5.28: Solution to 2D problem 11 (circular convection)

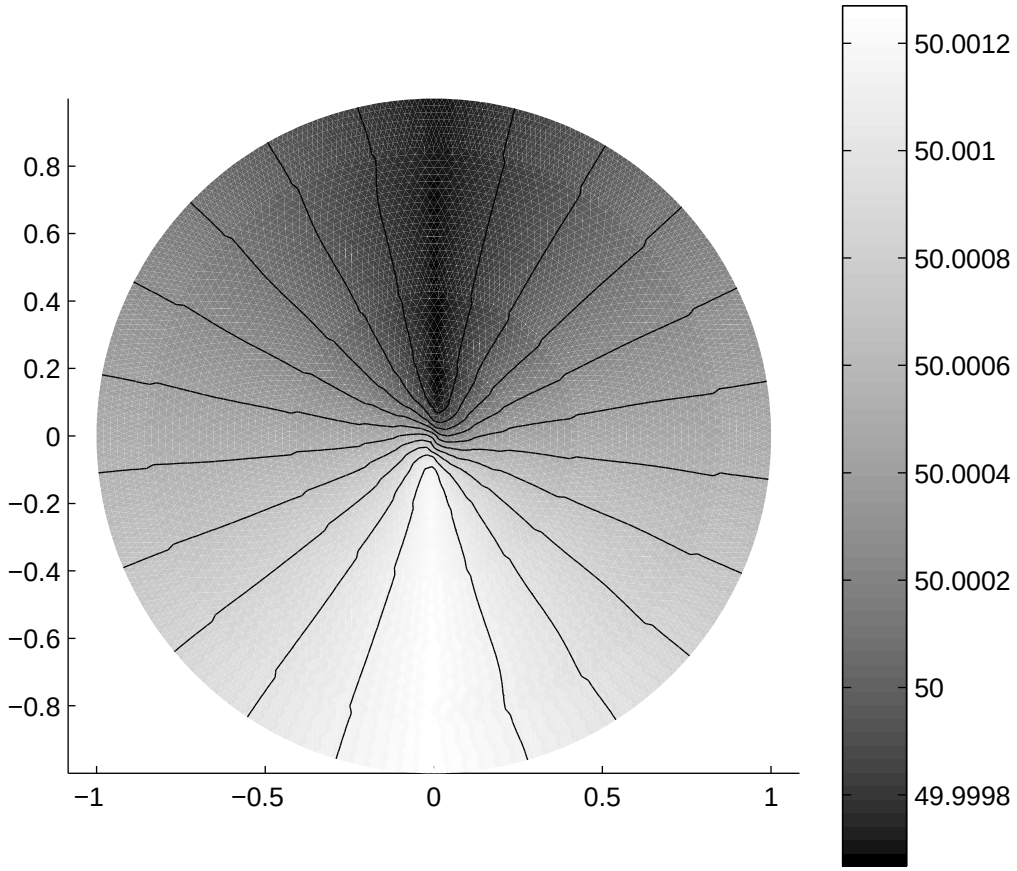


Table 5.11: Bi-CGstab iterations for 2D problem 11 (circular convection) to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 1195$		$n = 4939$		$n = 19627$		$n = 78763$	
ILUT(0.01)	33	(360)	53	(604)	111	(1229)	*	
AINV(0.12)	77	(833)	275	(2694)	755	(6532)	*	
Mr.Lin(0.23)	103	(1103)	933	(9620)	255	(2150)	691	(4896)
Mr.PDE(0.32)	73	(750)	135	(1411)	297	(2970)	879	(8554)
+Mr.Lin(0.18)	69	(709)	157	(1402)	389	(3216)	*	
+Mr.PDE(0.23)	71	(739)	177	(1774)	497	(4900)	*	

to detect this; if they treat the problem incorrectly very bad things can happen. The additional twist in this problem is that the one-dimensional problems are periodic, since the streamlines are closed; this means for example that A is far from triangular, making life more difficult for factored preconditioners.

The results for the multi-resolution approximate inverse are disappointing. I cut back the number of coarse levels to a maximum of two to improve convergence, and while this is still better than the standard basis, it loses scalability. Allowing more levels slows convergence. The problem is that the coarsening and interpolation should happen only along the stream-lines; at low resolutions the stream-lines are very curved so retriangulation is bound to do the wrong thing. A convection-aware coefficient-adaptive triangulation might do the job, but I have left this for future work.

Table 5.11 gives the iteration results, for unweighted and weighted coarsening (with Delaunay retriangulation). The Laplacian interpolation basis is completely unsuited to this problem, and thus is not included. It is clear that the coarsening is a major difficulty—counter-intuitively, the PDE-interpolation works better with the unweighted coarsening than with the weighted scheme.

5.5.13 Problem 12: barrier option pricing

The final problem comes from computational finance, a two-asset barrier option pricing problem in [29]. The PDE, in conservative form[1], is:

$$\frac{\partial u}{\partial \tau} = \nabla \cdot (K \nabla u - bu) + cu$$

with coefficients given by:

$$\begin{aligned} K(x,y) &= \frac{1}{2}\begin{pmatrix} x^2\sigma_1^2 & xy\rho\sigma_1\sigma_2 \\ xy\rho\sigma_1\sigma_2 & y^2\sigma_2^2 \end{pmatrix} \\ b(x,y) &= \begin{pmatrix} -x(r - \sigma_1^2 - \rho\sigma_1\sigma_2/2) \\ -y(r - \sigma_2^2 - \rho\sigma_1\sigma_2/2) \end{pmatrix} \\ c = -3r + \sigma_1^2 + \sigma_2^2 + \rho\sigma_1\sigma_2 \end{aligned}$$

Here $\tau = -t$ is backwards time, x and y are the prices of the underlying assets, and σ_1, σ_2, ρ, and r are constants describing the stochastic evolution of prices. In this example, $\sigma_1 = 0.4$, $\sigma_2 = 0.2$, $r = 0.05$, and $\rho = -0.5$. The payoff function (initial condition) is a basket call, $u(x, y, \tau = 0) = \max(\frac{1}{2}(x + y) - 100, 0)$, except for this example I assume the barrier is applied immediately before, setting $u = 0$ outside of a small ellipse. The boundary conditions are Dirichlet, $u \to x/2$ as $x \to \infty$ and $u \to y/2$ as $y \to \infty$.

The domain is the square $[0, 200] \times [0, 200]$, with an unstructured mesh that is refined around the boundary of the barrier—see figure 5.29. Iteration counts for an initial fully implicit timestep of size $\Delta\tau = 0.01$ years (a fairly long step of about half a week) are given in table 5.12, and for a timestep of size $\Delta\tau = 0.0001$ years (a more typical step of roughly 50 minutes) in table 5.13, both with the unweighted coarsening and the weighted coarsening with Delaunay retriangulation.

For the long timestep, there is considerable correlation between distant nodes. This makes the multi-resolution method more effective than the standard basis, though clearly only with PDE-interpolation—linear or Laplacian interpolation fail. For the largest problem, the superior scaling of PDE interpolation beats even ILUT in flop count.

[1] Although the original equation is non-conservative, and perhaps should be treated as such, it is simpler for the current discretization code to deal with the conservative form

Figure 5.29: Mesh for 2D problem 12 (option pricing)

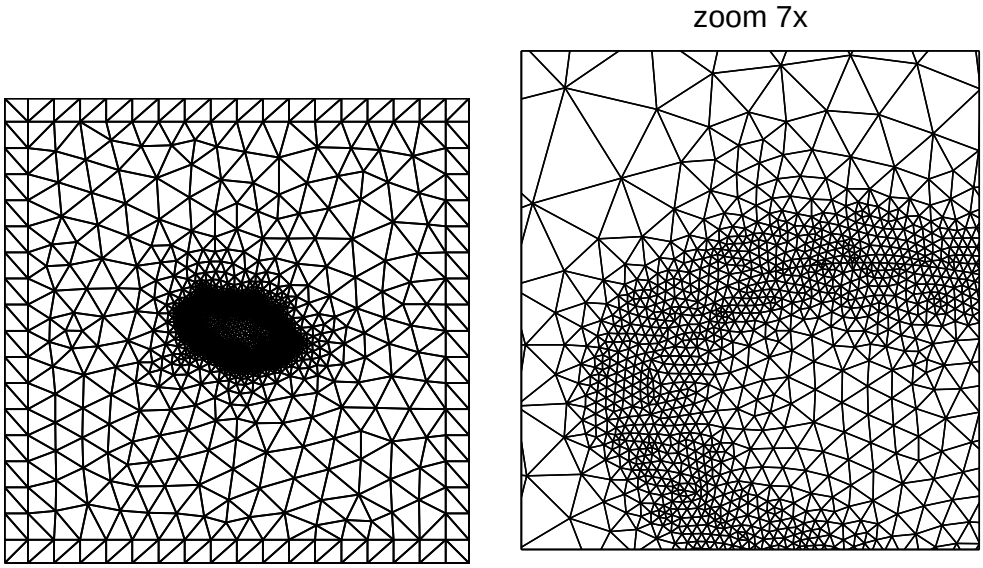


Table 5.12: Bi-CGstab iterations for 2D problem 12 (option pricing) with long timestep to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 3495$		$n = 13905$		$n = 55473$	
ILUT(0.01)	11	(112)	23	(247)	37	(398)
AINV(0.08)	25	(256)	55	(625)	85	(1013)
Mr.Lin(0.14)	89	(877)	157	(1563)	*	
Mr.Lap(0.15)	129	(1276)	331	(3287)	*	
Mr.PDE(0.35)	31	(314)	33	(344)	37	(390)
+Mr.Lin(0.13)	165	(1644)	217	(2232)	*	
+Mr.Lap(0.2)	141	(1408)	481	(3795)	*	
+Mr.PDE(0.24)	21	(210)	29	(296)	29	(296)

Table 5.13: Bi-CGstab iterations for 2D problem 12 (option pricing) with short timestep to reduce the residual norm by 10^{-6}, with flops per unknown in parentheses; no convergence is marked by *. Each preconditioner's drop tolerance δ is chosen to give roughly the same number of nonzeros. Note that ILUT is generally slower than the flop count suggests. See page 81 for details.

Method(δ)	$n = 3495$		$n = 13905$		$n = 55473$	
ILUT(0.0002)	5	(50)	5	(71)	5	(101)
AINV(0.003)	5	(50)	5	(93)	5	(177)
Mr.Lin(0.25)	583	(6002)	*		*	
Mr.Lap(0.3)	585	(5992)	*		*	
Mr.PDE(0.28)	15	(145)	15	(153)	19	(201)
+Mr.Lin(0.28)	557	(5606)	*		*	
+Mr.Lap(0.33)	*		*		*	
+Mr.PDE(0.2)	11	(106)	13	(130)	15	(153)

However, for the short timestep the matrix is very diagonally dominant; there is little correlation for the multi-resolution basis to exploit, yet fast decay in the Green's function to the benefit of the standard basis. Even with PDE-interpolation, the basis transforms are essentially a waste of storage that could be better spent on the approximate inverse, although the flop counts show that the multi-resolution method is scaling better and might be more effective for larger problems.

5.6 Summary

Every test problem was successfully solved with PDE-interpolation and appropriate coarsening in the multi-resolution basis: no other method showed this level of robustness. Furthermore, the multi-resolution method almost always outperformed the standard methods, at least on the largest meshes: their greater scalability is apparent, often running an order of magnitude or more faster. (The exceptions are the strongly diagonally dominant matrices arising from the short timesteps in problem 12.)

It's true the standard methods were more competitive in 2D than in 1D, with ILUT somtimes giving the best flop counts for smaller meshes. However, as mentioned before this performance measure should be taken with a grain of salt since applying the ILUT preconditioner is often more expensive than the approximate inverses, particularly on parallel machines. The fairer comparison with standard AINV always came out in favour of the multi-resolution methods, again apart from problem 12.

The downside of the multi-resolution methods is their sensitivity with respect to the automatic mesh coarsening. Particularly for the anisotropic problems, appropriate semi-coarsening with coefficient-adaptive triangulation (which often required the domain to be first partitioned into regions of roughly constant coefficients for robustness) is crucial. However, these problems were solved with ease once a good hierarchy was found. The only really troubling issue was with the nontrivial convection in problem 11, which featured closed and curved streamlines. The current automatic mesh coarsening algorithms could not find a good hierarchy, so even though standard methods did worse, the multi-resolution results were still far from optimal.

Chapter 6

Conclusions and Future Work

I have presented a new preconditioner for elliptic PDE's, based on the idea of using second generation wavelets to compress the inverse for approximation with sparse matrices. This resolves the inherent scalability problem of existing approximate inverses: in the standard basis, sparsity and quality become increasingly compatible as the problem size grows. Along the way I have pointed out where algorithms are naturally parallel. The test results show that for many fairly difficult problems the method scales well, much better than the standard basis approximate inverse, and even for small problems often gives significantly better convergence.

The key points brought home are:

- Wavelets are a natural choice for approximate inverses, but only when moments are *not* preserved with an update step.
- Interpolation should be chosen carefully with knowledge of the problem; in general, PDE-interpolation is essential for robust convergence. Methods that are higher order than the PDE are useless.
- Good automatic coarsening is crucial, perhaps more important than the choice of interpolation. Simple approaches are bound to fail for tough problems with anisotropy or discontinuities; finding a robust algorithm, especially for convection problems, is still an open problem.

Along the way, several interesting questions have been raised. I will briefly summarize them here.

The parallels to multigrid and other methods would make a comparative study very useful. In particular, the theoretical machinery used for analyzing the convergence of multigrid probably can be put to good use here, and similarly the new perspective of compressing the discrete Green's function might lead to new results for other multi-resolution techniques. On a practical level, the multi-resolution components of the software are compatible with node-nested multigrid, so code may be re-used (and the two techniques could be compared directly).

Developing an algebraic multi-resolution approximate inverse where the prediction operators are derived directly from the matrix might make for simpler and more robust code—the issue of retriangulation in coarsening might be avoided in particular.

Adapting approximate inverse algorithms other than AINV (e.g. Chow and Saad's MR method) may be very useful. In particular, other approaches have more natural parallelism in the construction phase, though perhaps not showing as good convergence rates.

The particle model that served as an intuition for harmonic weighting of the diffusion term deserves more thought—perhaps a discretization based directly on the underlying statistical mechanics rather than via the continuum approximation of the PDE will give a rigourous and powerful solution to the inconsistencies and *ad hoc* nature of the current schemes. Coefficient homogenization for coarsening might also fall naturally out of this research.

Coefficient-adaptive triangulation was used to great effect in a controlled setting, but the present implementation's instability for variable coefficients is clear, as well as the difficulties of convection problems.

The bottom-up approach in automatic unstructured mesh coarsening needs to be considered along with improvements to the weighted top-down approach suggested here. Theoretical work on the best possible node placement for the coarsest mesh could cross-fertilize with adaptive meshing research. This also leads the way to the question of what is the coarsest possible *useful* representation of a given problem.

Probably the most telling weakness with the method as it stands is the restriction to scalar problems. Most real-life problems involve systems of PDE's, often with some variables following an elliptic or parabolic nature and others with a hyperbolic character (so called mixed

systems). From an abstract viewpoint, the general scheme of the multi-resolution approximate inverse appears to apply here, but the problem of good interpolation may be challenging, to say nothing of coefficient dependent coarsening.

Another obvious direction is in implementing the method for higher order PDE's or discretizations. (This arises in structural analysis, flux limiter methods for convection problems, etc.) There doesn't appear to be any great difficulty in doing this, but interpolation again could pose problems, especially determining the coarse nodes used to predict a fine node.

A third and perhaps most challenging direction is the implementation for 3D problems. Here Green's functions decay even faster (reducing the need for multi-resolution compression), interpolation operators are necessarily denser and more expensive, and unstructured meshing is frought with technical difficulties to name only a few problems. Although similar multi-resolution methods can be proven to give "optimal" $O(n)$ convergence even for 3D problems, the constant factor obscured by the O notation is often so large that other preconditioners are more effective for the problem sizes of interest today. It may be that though multi-resolution bases have a role to play, they will only be useful for really big problems—e.g. perhaps coarsening should be stopped at hundreds of thousands of nodes. Of course, some 3D problems have anisotropy or strong convection that essentially reduce them to sets of weakly coupled lower dimensional problems, albeit potentially with very complicated geometry; multi-resolution methods appear to have more potential here.

Finally, although theoretically the algorithms presented in this thesis should be able to run effectively in parallel, this is a far cry from a working parallel implementation. The creation of a truly scalable, parallel high performance multi-resolution approximate inverse will be a real test of the method.

Bibliography

[1] R. Abgrall and A. Harten, *Multiresolution representation in unstructured meshes*, SIAM J. Numer. Anal., 35 (1998), no. 3, pp. 2128–2146.

[2] R. Alcouffe, A. Brandt, J. Dendy, and J. Painter, *The multigrid method for the diffusion equation with strongly discontinuous coefficients*, SIAM J. Sci. Statist. Comput., 2 (1981), no. 4, pp. 430–454.

[3] T. Barth, *Numerical aspects of computing viscous high Reynolds number flows on unstructured meshes*, technical report AIAA 91-0721, Reno, 1991.

[4] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. of Sci. Comput., 19 (1998), no. 3, pp. 968-994.

[5] M. Benzi and M. Tůma, *A comparative study of sparse approximate inverse preconditioners*, to appear in Appl. Numer. Math., 30 (1999).

[6] M. Benzi and M. Tůma, *Orderings for factorized sparse approximate inverse preconditioners*. To appear in SIAM J. of Sci. Comput. (Revised version of Los Alamos National Laboratory Technical Report LA-UR-98-2175, May 1998)

[7] M. Benzi, J. Marin, and M. Tůma, *A two-level parallel preconditioner based on sparse approximate inverses*, in D. Kincaid et. al., eds., Iterative Methods in Scientific Computation II, IMACS Series in Computational and Applied Mathematics, IMACS, NJ, 1999 (in press).

[8] C. Brand, *An incomplete-factorization preconditioning using repeated red-black ordering*, Numer. Math., 61 (1992), no. 4, pp. 433–454.

[9] R. Bridson, D. Pierce, and W.-P. Tang, *Refined algorithms for a factored approximate inverse*. In preparation.

[10] R. Bridson and W.-P. Tang, *Ordering, Anisotropy and Factored Sparse Approximate Inverses*, to be published in SIAM J. Sci. Comput. (1999)

[11] R. Bridson and W.-P. Tang, *A diagnosis of some ILU orderings*, submitted to SIAM J. Sci. Comput.

[12] T. Chan and B. Smith, *Domain decomposition and multigrid algorithms for elliptic problems on unstructured meshes*, Contemporary Math., 180 (1994), pp. 175–189.

[13] T. Chan, B. Smith, and W. L. Wan, *An energy-minimizing interpolation for robust multigrid methods*, technical report CAM98-6, Dept. of Math., University of California at Los Angeles, 1998.

[14] T. Chan, W.-P. Tang, and W. L. Wan, *Wavelet sparse approximate inverse preconditioners*, BIT, 37 (3), 1997.

[15] E. Chow and Y. Saad, *Approximate inverse techniques for general sparse matrices*, Colorado Conference on Iterative Methods, April 5–9, 1994.

[16] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (3), May 1998.

[17] S. Clift, H. Simon, and W.-P. Tang, *Spectral ordering techniques for incomplete LU preconditioners for CG methods*, manuscript.

[18] I. Daubechies, *Ten lectures on wavelets*, CBMS-NSF Series Appl. Math., SIAM: Philadelphia, 1991.

[19] M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman: New York, 1979.

[20] A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[21] J. Gilbert, *Predicting structure in sparse matrix computations*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 62–79.

[22] M. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), no. 3, pp. 838–853.

[23] W. Hackbusch, *Multi-grid methods and applications*, Springer-Verlag: Berlin, 1985.

[24] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1999), no. 1, pp. 359–392 (electronic).

[25] L. Kolotilina and A. Yeremin, *Factorized sparse approximate inverse preconditionings I. theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.

[26] J. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.

[27] E. Morano, D. Mavriplis and V. Venkatakrishnan, *Coarsening strategies for unstructured multigrid techniques with applications to anisotropic problems*, SIAM J. Sci. Comput., 20 (1998), no. 2, pp. 393–415.

[28] E. Ong, *Hierarchical basis preconditioners for second order elliptic problems in three dimensions*, PhD. thesis, University of Washington, Seattle, 1989.

[29] P. Forsyth, D. Pooley, B. Simpson, and K. Vetzal, *Pricing two asset barrier options with unstructured meshes*, preprint, University of Waterloo, Waterloo, 1999.

[30] R. Renka, *TRIPACK*, Algorithm 751, ACM Collected Algorithms, published in Transactions on Mathematical Software, 22 (1996), no. 1, pp. 1–8.

[31] J. Ruge and K. Stuben, *Algebraic multigrid*, in *Multigrid methods*, ed. S. McCormick, *Frontiers in Applied Mathematics*, 3 SIAM: Philadelphia, 1987.

[32] Y. Saad, *Iterative methods for sparse linear systems*, PWS Publishing Co.: Boston, 1996.

[33] Y. Saad and J. Zhang, *BILUM: block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems*, to appear in SIAM J. Sci. Comput.

[34] W. Sweldens, *The lifting scheme: a construction of second generation wavelets*, SIAM J. Math. Anal., 29 (1997), no. 2, pp. 511-546.

[35] W.-P. Tang and W. L. Wan, *Sparse approximate inverse smoother for multigrid*, technical report CAM 98-18, Dept. of Math., University of California at Los Angeles, 1998.