Multilevel Iterative Methods

Eric Yin

Advisor: Professor Randy Bank, UCSD

Contents

1 Introduction 3

2 Basic Multigrid Algorithm

2.1 Multigrid foundations 3

2.2 Multigrid algorithm 4

3 Experimentation 15

4 Algebraic Multigrid (AMG)

4.1 Introduction to AMG 19

4.2 Implementing AMG 23

4.3 Experimentation 24

5 Acknowledgements 27

6 References 27

1 Introduction

This paper involves the analysis and study of multilevel iterative methods, otherwise known as Multigrid. Multigrid is generally used to approximate the solution of elliptic partial differential equations. The definition of a partial differential equation (PDE) is the "relations involving an unknown function of several independent variables and its partial derivatives with respect to those variables." [6] An elliptic PDE is a specific branch of PDEs which will be described in the following section. While there are many methods that can directly solve PDEs (i.e. integral transform, separation of variables, and Green's function), real life problems are often too complex to be solved directly. Thus approximation methods such as SSOR, ILU, Conjugate Gradients, and Multigrid are used. In this paper, we discuss various aspects of Multigrid. We will look at the advantages of Multigrid (as opposed to the other approximation methods), break down the various components that make up Multigrid, describe the Multigrid algorithm as a whole, do some experimentation, and then examine Algebraic Multigrid, a more complex variation of the basic Multigrid.

2 Background

2.1 Multigrid foundations

Multigrid is an iterative method that can be used to solve

$$Au = f \tag{2.1}$$

A is a square matrix, f is a vector termed the right hand side, and u is a vector for which we are trying to solve.

An iterative method is defined as "a method that attempts to solve a problem by finding successive approximations to the solution starting from an initial guess." [6] Multigrid is primarily used to solve elliptic partial differential equations. This branch of problems can be represented in general terms as:

$$a(x,y)u_{xx} + 2b(x,y)u_{xy} + c(x,y)u_{yy} + d(x,y)u_x + e(x,y)u_y + g(x,y)u = f(x,y)$$

where $b^2 - ac < 0$, and Ω (the domain) and $\partial\Omega$ (the boundaries) are provided.

There are other iterative methods used to solve elliptic PDEs but the rate of convergence and cost of Multigrid is what makes the algorithm significant. Below is a chart that depicts the various approximation methods and their respective spectral radius (which measures the rate of convergence) and cost.

Method	Cost	Spectral radius
Jacobi	$N \cdot N^{2/d} \log N$	$1 - ch^2$
SOR/SSOR/ILU	$N \cdot N^{1/d} \log N$	$1 - ch$
Jacobi-CG	$N \cdot N^{1/d} \log N$	-----
SSOR-CG/ILU-CG	$N \cdot N^{1/2d} \log N$	-----
Multigrid	N	$c < 1$

Fig. 2.1

where d is the number of dimensions of the problem, N represents the size of the problem, c is a constant which varies by method, and h=1/N (length of each step size).

2.2 Multigrid Algorithm

There exist several key components of Multigrid. These are: smoothing, restriction, and prolongation. Prior to implementing the Multigrid algorithm, it is necessary to discretize the problem first. The discretization of elliptic partial differential equations leads to solving a problem of the form (2.1). This matrix problem is what Multigrid will be implemented on. So the very first thing we will discuss is how to discretize a problem.

Discretization:

Discretization by definition is "the process of transferring continuous models and equations into discrete counterparts." [6] While discretization takes away from the continuity of the problem, the idea behind it is that the initial problem is well-posed:

Conditions to being well-posed:

1. A solution exists
2. The solution is unique
3. If the data are changed only slightly, then the resulting solution changes only slightly (continuous dependence of solution on data).

When using a numerical algorithm to approximate a solution for a problem it is important to determine if a problem is well-posed. First, if no solution exists then there is no point in applying the algorithm to the problem. Second, if multiple solutions exists, one must determine which computed solution is of interest. Third, if the first two conditions are satisfied but the solution does not depend continuously on the data, then the approximation to the exact solution will most likely be of no value. This is because in approximation techniques, the computations involved do not involve directly solving the problem, but instead a problem that has been slightly altered. The modified problem may have one solution but that solution may be significantly different than the true solution of the original problem if it does not depend continuously on the data.

There are various methods of discretizing a PDE. Some of the more common techniques include:

1. Finite differences
2. Finite volumes
3. Finite elements

It would be too much to go into every single method of discretization, so instead, below is an example of how to discretize a problem using finite differences. Also included are visualizations of a problem using finite discretization.

Example: Finite differences discretization

We first introduce the following notation:

Ω is the domain in which we are working on.

$P_{ij} = (x_i, y_j)$ is a collection of grid points for integer values of i and j with

$$x_i = x_o + ih\ ,\ y_j = y_o + jk$$

where (x_o, y_o) is a chosen specific point and the stepsizes $h = \Delta x$ and $k = \Delta y$ are fixed mesh widths. (note: Δ does not denote the Laplacian but instead a chosen stepsize for both the x and y direction respectively) The discrete point set

$$\Omega_{h,k} = \left\{ P_{ij} = (x_i, y_j) \middle| x_i = x_o + ih, y_j = y_o + jk \text{ for all integers i,j with } P_{ij} \in \Omega \right\}$$

provides a discrete representation of Ω.

Let $u_{ij} = u(x_i, y_j)$ for $(x_i, y_j) \in \Omega_{h,k}$

For a smooth function $u = u(x, y)$ and for any small number Δx, take the Taylor expansion with respect to x to get:

$$u(x + \Delta x, y) = u(x, y) + u_x(x, y)\Delta x + \frac{1}{2}(\Delta x)^2 u_{xx}(x, y) + \frac{1}{3!}(\Delta x)^3 u_{xxx}(x, y) + \vartheta((\Delta x)^4)$$

where $\vartheta((\Delta x)^4)$ refers to a bounded quantity:

$$|\vartheta(h)| \le c|h|$$

where c is a constant and in this case, $h = (\Delta x)$. $\vartheta(h)$ is termed the truncation error. By moving term's of the Taylor expansion around to get the first derivative of u with respect to x isolated, we get the following:

$$u_x(x, y) = \frac{1}{\Delta x}[u(x + \Delta x, y) - u(x, y)] - \frac{1}{2}u_{xx}(x, y)\Delta x + \vartheta(\Delta x)^2$$

or in grid notation with $h = \Delta x$

$$u_x(x_i, y_j) = \frac{1}{h}[u_{i+1,j} - u_{i,j}] - \frac{h}{2}u_{xx} + \vartheta(h)^2 \tag{2.2}$$

and replacing Δx by $-\Delta x$ we can get

$$u_x(x_i, y_j) = \frac{1}{h}[u_{i,j} - u_{i-1,j}] + \frac{h}{2}u_{xx} + \vartheta(h)^2 \tag{2.3}$$

Subtracting formulas (2.2) from (2.3) and isolating u_{xx} :

$$u_{xx}(x_i, y_j) = \frac{1}{h^2}[u_{i+1,j} - 2u_{i,j} + u_{i-1,j}] + \vartheta(h^2) \tag{2.4}$$

and using a common stepsize h=k, a corresponding discretization with respect to y :

$$u_{yy}(x_i, y_j) = \frac{1}{h^2}[u_{i,j+1} - 2u_{i,j} + u_{i,j-1}] + \vartheta(h^2) \tag{2.5}$$

Visual example: Finite differences discretization of $u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j)$

Using the resulting equations (2.4) and (2.5) and adding them together we can get the following:

$$u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j) = \frac{1}{h^2}[u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}] + \vartheta(h^2)$$

In the following figure,

$a = u_{i+1,j}$
$b = u_{i-1,j}$
$c = u_{i,j+1}$
$d = u_{i,j-1}$
$e = u_{i,j}$

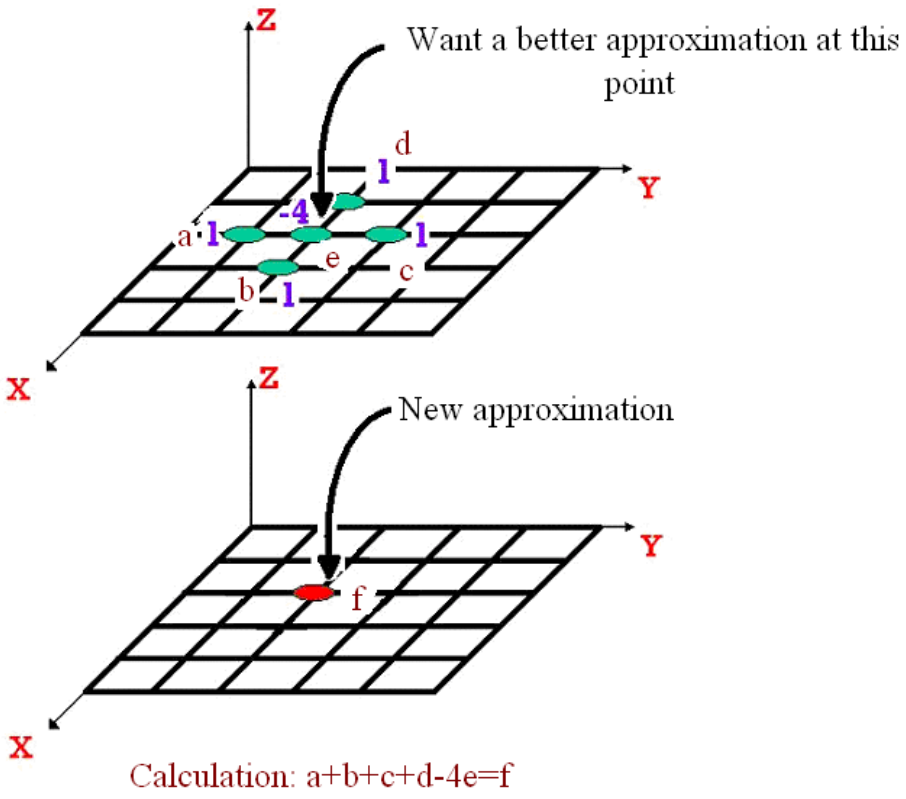


Fig. 2.2

As shown in Figure 2.2, utilizing finite discretization, in order to get a better approximation of u at point e for the problem $u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j)$ using any initial guess, it is necessary to multiply that point by -4 and add it to the four surrounding values around it.

Smoothing/Relaxation Process:

Upon discretizing the initial problem, the problem then becomes one of solving a question of the form (2.1). This can be shown by the following:

Let's examine

$$u_{xx}(x_i, y_j) = f$$

From 2.4,

$$u_{xx}(x_i, y_j) = \frac{1}{h^2}[u_{i+1,j} - 2u_{i,j} + u_{i-1,j}] + \vartheta(h^2) = f(x_j)$$

For convenience, we will drop off the $\frac{1}{h^2}$ and $\vartheta(h^2)$ terms, so the problem now becomes:

$$u_{xx}(x_i, y_j) = [u_{i+1,j} - 2u_{i,j} + u_{i-1,j}] = f(x_j)$$

Furthermore, in this problem, we are only working in one-dimension (x) so for further simplicity, we can drop out the y terms and its corresponding j values:

$$u_{xx}(x_i) = [u_{i+1} - 2u_i + u_{i-1}] = f(x_j) \tag{2.6}$$

(2.6) only refers to one isolated value, x_i, on the grid. So to represent

$$u_{xx}(x_i) = [u_{i+1} - 2u_i + u_{i-1}] = f(x_j) \quad \text{where } 1 \le i \le n$$

meaning there are n grid points, we get the following system of linear equations (in matrix form):

$$\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Notice that for the first and last equations, the first term and last term respectively are dropped off. This is because data points on those corresponding values are out of the bounds of the problems. In this case, in (2.1) the matrix A is the tridiagonal matrix shown above, and the vectors u and f correspond to the u and f vectors.

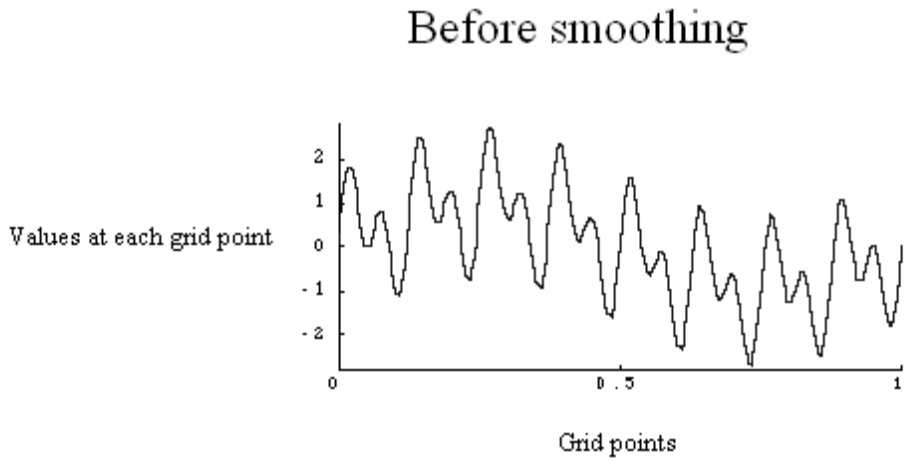
We begin by talking about the first part of the Multigrid algorithm termed *relaxation*.

Let us term are current approximation as u_0. Let's also pretend that we have the real solution u. These two values lead to the following relationship:

$$error = u - u_0$$

(Note: we will only be using u_0 when discussing the Multigrid algorithm, u (the exact solution) and the error are only assumed to be known in order to be utilized as a foundation for comparison)

As a visual example to help explain the smoothing process, let's say our error takes on the following visual form:



The big thing that should be noticed is that there are two types of oscillations in this error, high frequency error and low frequency error. The smoothing process helps get rid of the high frequency error using the following technique.

With u_0 (which we are hoping will eventually approximate u in (2.1)), smoothing involves the following iterative process:

$$r_{k-1} = b - Au_{k-1}$$
$$B\delta_{k-1} = r_{k-1}$$
$$u_k = u_{k-1} + \delta_{k-1}$$

where A and f are from (2.1) and B is a simplified factorization of A termed the *smoother* (note: the terms for the *smoothing* and *relaxation* are synonymous).

The number of iterations used in the smoothing process is up to the user, but generally 1-4 smoothing processes are adequate (this will be discussed later).

Examples of various smoothers:

- $B = (L + D)D^{-1}(D + U) \approx P^t AP$
- $B = D$

L,D,U represent the lower triangular matrix, diagonal, and upper triangular matrix of A respectively. P represents a permutation matrix.

Visual example of the smoothing/relaxation process – The following represents the error before and after smoothing. Figure 2.3 is the initial error, while Figure 2.4 represents the error after the initial guess has been updated within the smoothing process.

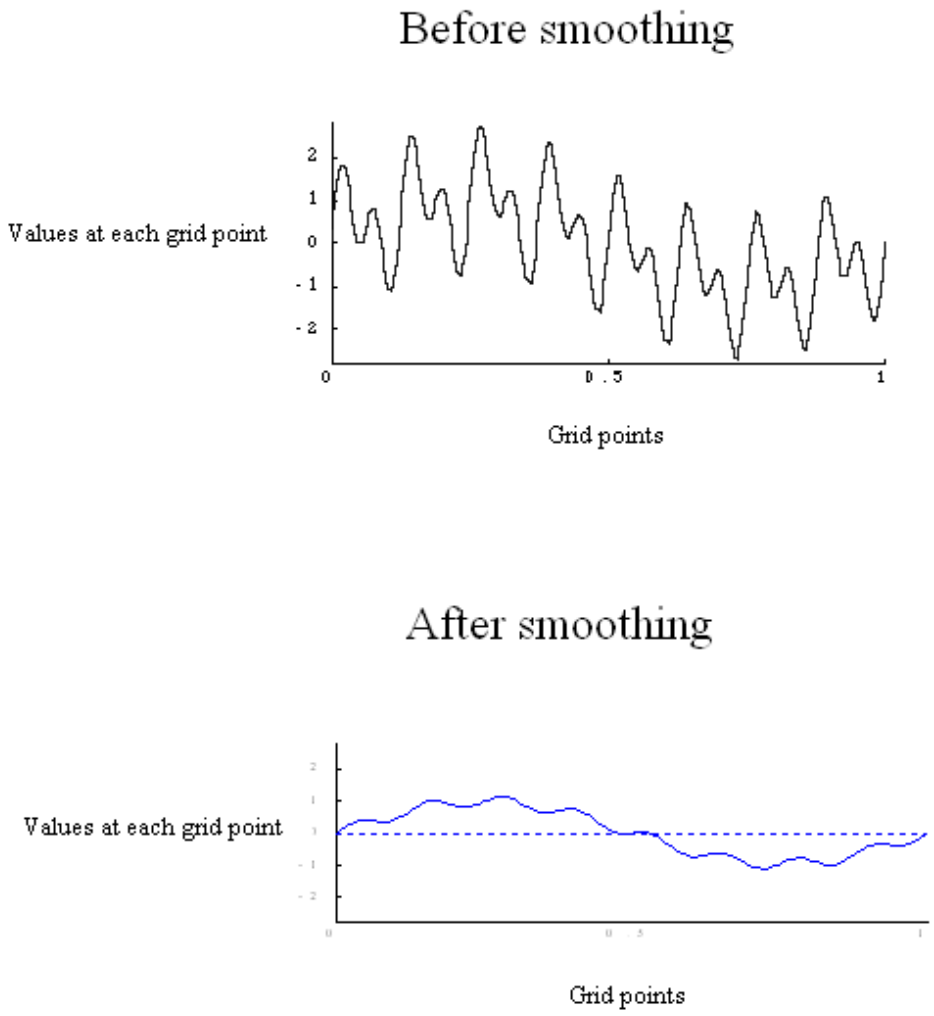


Fig. 2.3

Notice that most of the high frequency oscillations in the top figure are gone in bottom figure (after smoothing). The bottom figure is essentially "smoother". But now, the question arises as to how to get rid of the low frequency oscillations.

Restriction:

Multigrid, much like its name, deals with multiple grids. How are these grids related? The idea behind Multigrid is that through each successive grid, the error is magnified so you are able to identify and remove it at a faster rate. Note: This will only work when

dealing with low frequency, but keep in mind, the smoothing process is extremely effective in dealing with the high frequency error.

Three main questions arise in dealing with the "multiple grid" aspect of Multigrid:

1. What are the different types of grids?
2. How do you get from one grid to another (i.e. what are their relationships)?
3. How does this get rid of the low frequency error?

We will first deal with questions 1 & 2 and then use visual examples to explain question 3.

Let's begin by defining certain terminology used in dealing with the different "classes" of grids. In Multigrid, the discretization grids either have more grid points or less grid points. No two successive grids will have the same number of grid points (note: the key word is successive). A grid is termed to be more *fine* if it has more grid points. Likewise, the *coarser grids* have less grid points. The interpolation matrix called the *restriction matrix* helps turn a fine grid into a coarser grid. The interpolating matrix that brings the coarse grid back into the fine grid again is called the *prolongation matrix*.

The restriction process (from fine to coarse) takes on the following generalized method (restriction/prolongation can be very difficult to explain so the use of examples, both visually and computationally, will be demonstrated following its introduction).

Let's say the current approximation to u you have is termed r, where in this case, r is a vector. First it must be determined what points will be coarse points and what points will be fine points. Coarse points are data points whose current values are values that you want to keep in the next grid. The fine points are points in the current grid that you want to somehow interpolate into the coarse points so that their values are vaguely represented in the coarse points. In other words, to get a coarser grid while maintaining similar properties, fine point values will be combined into coarse values. Let r_f refer to the determined fine-grid points and r_c refer to the coarse grid points. So,

$$r = \begin{pmatrix} r_f \\ r_c \end{pmatrix} \tag{2.7}$$

(Keep in mind, 2.7 is not to be taken literally in that the coarse points are the bottom portion of the vector and the fine points are just the top part of the vector. 2.7 is used to represent that values in x can be divided into coarse points and fine points.)

The restriction matrix takes on the block form:

$$\begin{pmatrix} \omega^t & I \end{pmatrix} \tag{2.8}$$

Where ω^t is an interpolant that will somehow transfer values of fine-grid points into the coarse grid. Applying the restriction matrix to the original fine grid r, the problem becomes:

$$\begin{pmatrix} \omega^t & I \end{pmatrix} \cdot \begin{pmatrix} r_f \\ r_c \end{pmatrix} = \hat{r}_c$$

where $\hat{r}_c$ represents the newly created coarse grid that immediate proceeds the fine grid r. This preceding explanation may be a bit vague, so hopefully, the following examples will help clarify the concepts behind restriction.

Example: 1-D restriction to a generic vector of length 7

Let's say we want to restrict the vector r defined below.

$$r = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix} \qquad \text{Restriction matrix} = \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 & & & & \\ & & 1 & 2 & 1 & & \\ & & & & 1 & 2 & 1 \end{pmatrix}$$

Applying the restriction matrix:

$$\frac{1}{4}\begin{pmatrix} 1 & 2 & 1 & & & & \\ & & 1 & 2 & 1 & & \\ & & & & 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix} = \begin{pmatrix} \hat{v}_1 \\ \hat{v}_2 \\ \hat{v}_3 \end{pmatrix}$$

Notice that now, instead of having 7 points in our vector, we now have a coarser 3 point vector.

Prolongation:

Prolongation is essentially the opposite of restriction; it brings the coarse grid back to the fine grid. Since the prolongation matrix and the restriction matrix are not inverses we do not get back the same grid point values, instead what we are getting back is the original

grid points. The prolongation process simply takes the transpose of the restriction matrix multiplied by a constant to get:

$$c \cdot \begin{pmatrix} \omega \\ I \end{pmatrix}$$

If you are currently on the coarse grid $\hat{u}_c$, applying the coarse grid with the prolongation matrix gets:

$$c \begin{pmatrix} \omega \\ I \end{pmatrix} \cdot \hat{u}_c = \begin{pmatrix} u_f \\ u_c \end{pmatrix}$$

where c is a real constant which is dependent on the restriction scheme that has been used. (Mathematicians have already chosen, for many problems, optimal restriction matrices and their corresponding prolongation matrix.)

Prolongation essentially pulls out the fine-point data from the coarse-grid data while maintaining the essential values of the coarse points.

Example: 1-D prolongation of a length 3 vector

Let's say we want to prolongate the vector $\hat{u}_c$ defined below:

$$\hat{u}_c = \begin{pmatrix} \hat{v}_1 \\ \hat{v}_2 \\ \hat{v}_3 \end{pmatrix} \qquad \text{Prolongation matrix} = \frac{1}{2} \begin{pmatrix} 1 & & \\ 2 & & \\ 1 & 1 & \\ & 2 & \\ & 1 & 1 \\ & & 2 \\ & & 1 \end{pmatrix}$$

Applying the prolongation matrix:

$$\frac{1}{2} \begin{pmatrix} 1 & & \\ 2 & & \\ 1 & 1 & \\ & 2 & \\ & 1 & 1 \\ & & 2 \\ & & 1 \end{pmatrix} \begin{pmatrix} \hat{v}_1 \\ \hat{v}_2 \\ \hat{v}_3 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix}$$

Notice that we started with only 3 points in our vector but after prolongation, the result was a vector of 7 points.

The following figures give a pictorial description of the prolongation and restriction processes in both 1-D and 2-D:

1-D

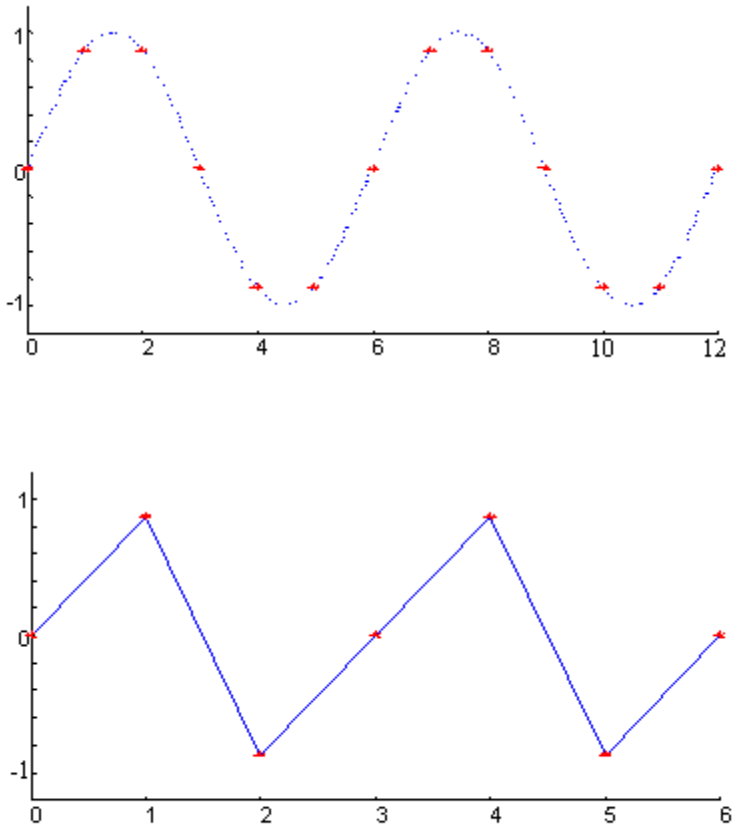


Fig. 2.5

2-D

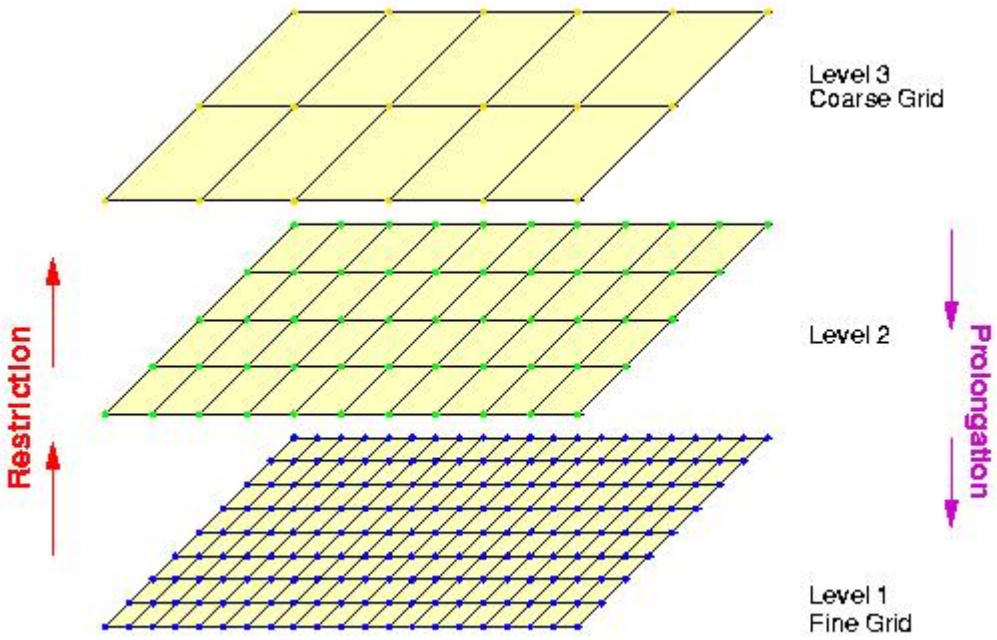


Fig. 2.6

We have now answered the questions of the different types of grids and how to get from one grid to another, but what's the point of all this? The question that still remains is how this gets rid of the low frequency error. We will explain this using a verbal explanation and a visual explanation.

Verbal explanation:

By coarsening the fine grid, the low frequency error on the fine grid has been strategically interpolated into high frequency error on the coarse grid. The key to dealing with high frequency error is utilizing the smoothing process. Since high frequency error will continually arise in the following coarse grid upon interpolating each grid, you need to apply the smoothing process. After smoothing on the final coarse grid, prolongation then takes place in order to get back to the original grid size.

Visual explanation:

Referring to Figure 2.5, the top picture represents some arbitrary error in a fine grid. The bottom picture represents the corresponding coarse grid after interpolation. Notice that

there are now only half as many points in the coarse grid, so the error is now in a higher frequency, which can be dealt with by utilizing the smoothing process.

Various Multigrid Schemes:

There exist multiple schemes in which Multigrid can be implemented. These combinations involve variations in the number of restriction and prolongation phases desired and the pattern they take on. What this means is that you can choose if you wanted to restrict, restrict again, restrict again,… and once you are at the desired coarse grid, you prolongate, prolongate again, prolongate again… until you get back to the original grid size. There are also other patterns of restriction and prolongation. You can choose to restrict once and prolongate once, then restrict twice and prolongate twice, … (note: Remember that following each restriction, smoothing takes place in order to get rid of the high frequency error)

There are three main types of Multigrid cycles: V-cycle, W-cycle, and Full Multigrid scheme. While the number of levels is an individual's preference, the essential patterns lie as follows:

V-cycle:

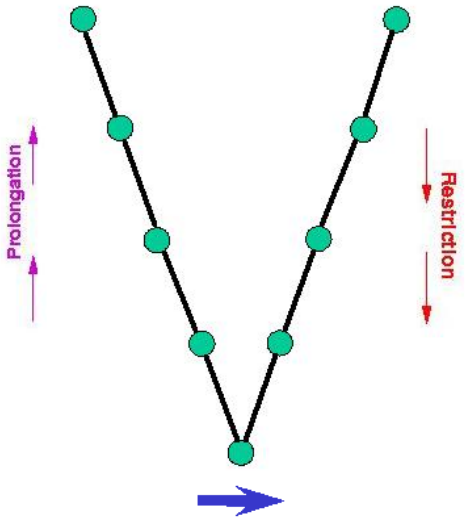


Fig. 2.8

W-cycle:

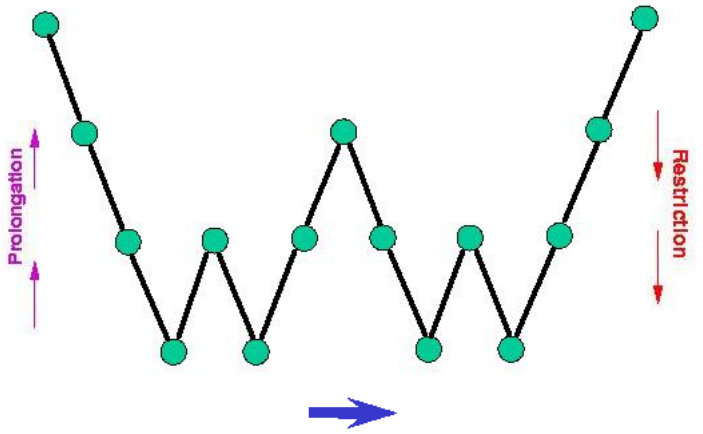


Fig 2.9

FMG-cycle:

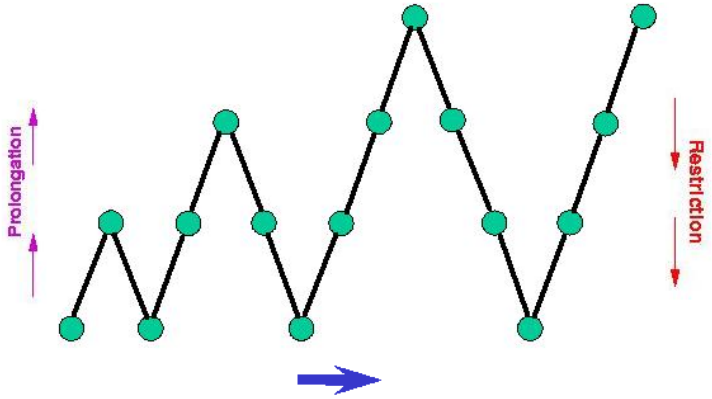


Fig. 2.10

Now that we've discussed all the elements of Multigrid, let's look at some problems.

3 Problem

Upon understanding how Multigrid works, I studied the following problem:

One-dimensional Poisson's equation with Dirichlet boundary conditions

$$u_{xx} = f \text{ in } \Omega=(0,n);\ u(0) = u(n) = 0$$

I took on this problem in two different fashions:

1. A direct approach – examining convergence directly (programming the algorithm into Matlab)
2. An analytical approach – examining convergence by proof.

Approach 1:

Using Matlab, I programmed a 2-level V-cycle using the finite differences discretization scheme. My initial guess was a vector of length n made up only of zeros and exact solution is defined as follows (both initial guess and exact solutions were chosen randomly):

$$u(x) = (-1)^x x \sin(\frac{x\pi}{n+1}) \qquad \text{where x=1,…,n}$$

Since, $u_{xx} = f$ (the problem we are solving), the value of f used in the problem can be computed by taking the 2nd derivative of u with respect to x.

I used n=300, meaning the problem had 300 steps where each step was of size 1.

The smoother used was defined as:

$$B = 4I$$

I did 4 different experiments. In the first, I used one smoothing step per cycle, in the second experiment, I used two smoothing steps per cycle, in the third, three smoothing steps per cycle, and in the forth, four smoothing steps per cycle. I did 5 cycles for each experiment. The chart on the following page shows the results I received. Since I cannot write out the resulting approximate solution for each step of each experiment, I took the 2-norm of the resulting vector of each cycle. The 2-norm is a single number representation of the overall error. Next to the error is the convergence rate ("the speed at which a convergent sequence approaches its limit", in this case, the limit of the error should always be 0) of each step. [6]

Cycle	2-norm of the error	Convergence factor
1	0.1031	----
2	0.0521	0.4990
3	0.026	0.4990
4	0.013	0.500
5	0.0065	0.500

(a)

Cycle	2-norm of the error	Convergence factor
1	0.0037	----
2	0.000940	0.2541
3	0.0002346	0.2495
4	0.0000586	0.2500
5	0.0000146	0.2498

(b)

Cycle	2-norm of the error	Convergence factor
1	0.0016	----
2	0.0002	0.1250
3	0.00002508	0.1254
4	0.0000031325	0.1249
5	0.0000003916	0.1250

(c)

Cycle	2-norm of the error	Convergence factor
1	0.0011	----
2	0.0000686	0.0624
3	0.00000405	0.0590
4	0.000000252	0.0622
5	0.0000000157	0.0625

(d)

a – 1 smoothing step
b – 2 smoothing steps
c – 3 smoothing steps
d – 4 smoothing steps

One interesting observation is that for (a), the convergence rate was roughly around a factor of 0.5. (b) had a convergence rate roughly around 0.25. (c) had a convergence rate roughly around 0.125. (d) had a convergence rate roughly around 0.625. Approach 2 takes on a more analytical approach to this problem and will prove that those values are in fact the convergence rates for the corresponding number of smoothing steps.

Approach 2:

Following the results from Approach 1, I took a more analytical approach in determining the convergence of the same problem:

One-dimensional Poisson's equation with Dirichlet boundary conditions

$$u'' = f \text{ in } \Omega=(0,n);\ u(0) = u(n) = 0 \qquad (3.1)$$

The reason this problem is significant is that it can be completely analyzed, meaning that it is able to completely represent how Multigrid works (barring computational error).

After discretization, let

$$N_j = (N_1 + 1)\cdot 2^{j-1} - 1 \text{ for some } N_1 \geq 1 \text{ and set } h_j = (N_j + 1)^{-1}.$$

In words, N_j refers to the size of the discretized problem whereas h_j represents the corresponding stepsize. We also use an arbitrary j and N_1 in order to show that this problem can be completely analyzed meaning that the results that we obtain represent the solution for all (3.1) regardless of size, initial guess, and true solution.

Next, we get that the matrix A in (2.1) can be represented as:

$$A_j = h_j^{-1}[-1,2,-1]$$

and the smoother being used is the damped Jacobi scheme

$$B_j = 4h_j^{-1} I$$

We find that the eigenvalues and eigenvectors of A_j are

$$\lambda_j = 2h_j^{-1}(1-\cos(i\pi h_j)) \text{ and } (\psi_i)_k = (2h_j)^{1/2}\sin(k\pi i h_j) \text{ respectively.}$$

With respect to the eigenvector basis, we look for the transformed matrices $\hat{\mathrm{A}}$ (transformed matrix representing the discretization), $\hat{B}$ (transformed smoothing matrix), $\hat{R}$ (restriction matrix), $\hat{S}$ (error propagation matrix for the smoothing process), and $\hat{C}$ (projector for the coarse grid correction). Here are the following results:

$$\hat{\mathrm{A}}_{\mathrm{i}} = 4h_j^{-1}\begin{pmatrix} x_i & 0 \\ 0 & 1-x_i \end{pmatrix}$$

$$\hat{B}_i = 4h_j^{-1}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\hat{R}_i = \begin{pmatrix} 1-x_i & x_i \end{pmatrix}$$

$$\hat{S}_i = \begin{pmatrix} 1-x_i & 0 \\ 0 & x_i \end{pmatrix}$$

$$\hat{C}_i = \begin{pmatrix} x_i & -(x_i(1-x_i))^{1/2} \\ -(x_i(1-x_i))^{1/2} & 1-x_i \end{pmatrix}$$

$$\hat{S}_i^{m/2}\hat{C}_i\hat{S}_i^{m/2} = \begin{pmatrix} x_i(1-x_i))^m & -(x_i(1-x_i))^{m+1/2} \\ -(x_i(1-x_i))^{m+1/2} & x_i^m(1-x_i) \end{pmatrix}$$

where $x_i = (1-\cos(i\pi h_j))/2$

After computing $\hat{S}_i^{m/2}\hat{C}_i\hat{S}_i^{m/2}$ (the matrix representing the overall error of a 2-level scheme), we examine its spectral radius $\rho(\hat{S}_i^{m/2}\hat{C}_i\hat{S}_i^{m/2})$ to get an idea of its convergence rate. m represents the number of desired smoothing steps.

Number of smoothing steps	$\rho(\hat{S}_i^{m/2}\hat{C}_i\hat{S}_i^{m/2})$
1	½
2	¼
3	1/8
4	1/16

4 Algebraic Multigrid (AMG)

4.1 Introduction to AMG

One problem that consistently arises from geometric Multigrid (the type explained in section 2) is that it has certain restrictions when solving elliptic PDEs. The key problem to the geometric case is that in order to implement Multigrid, the grid needs to be known. The question then arises, what if the grid is relatively unknown? Algebraic Multigrid (AMG) takes on the same essential concepts as the geometric case described above in regards to the need for initialization, smoothing, prolongation, and restriction. The key

difference comes with knowledge of the grid. In the case of AMG, the grid is relatively unknown. Instead of having a set grid, the data points come in a random model as will be shown later. The relationships among the unknowns are similar to that of the geometric case, but the locations are unknown. Because of this variation, there are essentially 2 key components to AMG:

1. defining the MG components (what is the restriction matrix, prolongation matrix,…)
2. performing MG cycles (doing the actual Multigrid)

Let's pretend we want to solve an elliptic PDE with the domain represented in fig. (4.1). Notice that the grid is basically random, but the relationships between points are known.

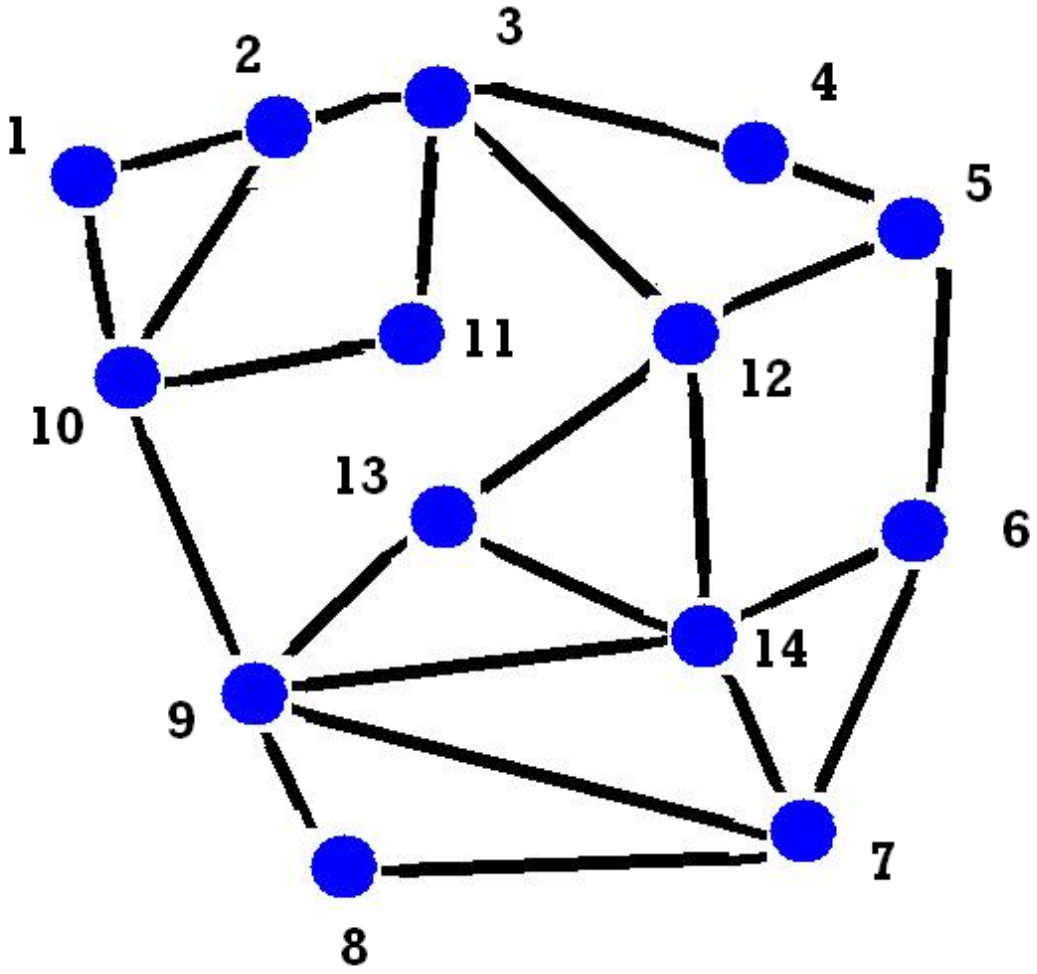


Fig. 4.1

The first thing we need to do is translate this grid into a matrix format that can be solved. Figure 4.2 defines the corresponding matrix. The way to obtain the matrix is to mark the values touching the point in question. So for figure 4.1, point 1 is touching points 2 and 10, so on line 1 (representing point 1) of the matrix, values at 1, 2, and 10 are marked. Point 2 is touching 1, 3, and 10, so line 2 (which representing point 2) of the matrix, values at 1, 2, 3, and 10 are marked. Continue working on this process until all points have been marked.

Corresponding matrix:

$$A=\begin{bmatrix}
x & x & & & & & & & x & & & & \\
x & x & x & & & & & & x & & & & \\
 & x & x & x & & & & & & x & x & & \\
 & & x & x & x & & & & & & & & \\
 & & & x & x & x & & & & & x & & \\
 & & & & x & x & x & & & & & & x \\
 & & & & & x & x & x & x & & & & x \\
 & & & & & & x & x & x & & & & \\
 & & & & & & x & x & x & x & & x & x \\
x & x & & & & & & & x & x & x & & \\
 & & x & & & & & & & x & x & & \\
 & & x & & x & & & & & & & x & x & x \\
 & & & & & & & & x & & & x & x & x \\
 & & & & & x & x & & x & & & x & x & x
\end{bmatrix}$$

Fig. 4.2

(Note: when doing the real problem, each x represents some number; x are only used for simplicity of explaining AMG)

There is little theory in the field of Algebraic Multigrid due to the complexity of the grid. The key to Multigrid as was described in section 2, is smoothing, determining the coarse points/coarse grids, determining how to get to the coarse grids, and then how to get back to the original grid. Initial fine grids, such as the ones in figure (2.5) & (2.6) have coarse grids that can easily be determined, but since the key to AMG is that the grid is relatively free in design, there is no set way of finding a proper coarse grid.

Determining the coarse-grid points is quite open-ended. One method of determining the coarse-grid points utilizes a concept called “strong connections.” A second method is called “Maximal Independent Set.” There are other methods, all based on the users preference, but only these two will be discussed.

Method 1: "Strong Connections"

Note: I did not study this method (I only used method 2). The purpose of putting this in is to show that there are multiple methods in determining the coarse grid but neither can be proved to be more advantageous than the other (reinforcing the reasoning as to why there is little theory behind AMG)

A strong connection is defined as:

$$S_i = \{j : -a_{ij} > \theta \max_{j \neq i} -a_{ij}\} \text{ where } 0 < \theta \leq 1$$

When basing the coarse-grid correction on these strong connections, there are 2 main criteria:

1. For each $i \in F$, each point $j \in S_i$ should either be in C or should be strongly connected to at least one point in C_i.
2. C should be a maximal subset with the property that no two C-points are strongly connected to each other.

C refers to the coarse-grid variables, F refers to the fine-grid variables, and C_i refers to the set of interpolatory coarse-grid variables used to interpolate fine-grid values.

In this case, sometimes both enforcing both criteria is impossible so criteria #1 is more important than criteria #2.

Method 2: "Maximal Independent Set"

Maximal Independent Set requires the satisfaction of the following 2 criteria:

1. No two coarse points are adjacent to each other
2. The set of coarse points are adjacent to all points in the given domain.

We now use Method 2 (Maximal Independent Set) to determine the coarse points on figure (4.1).

Example: Using Maximal Independent Set on figure (4.1)

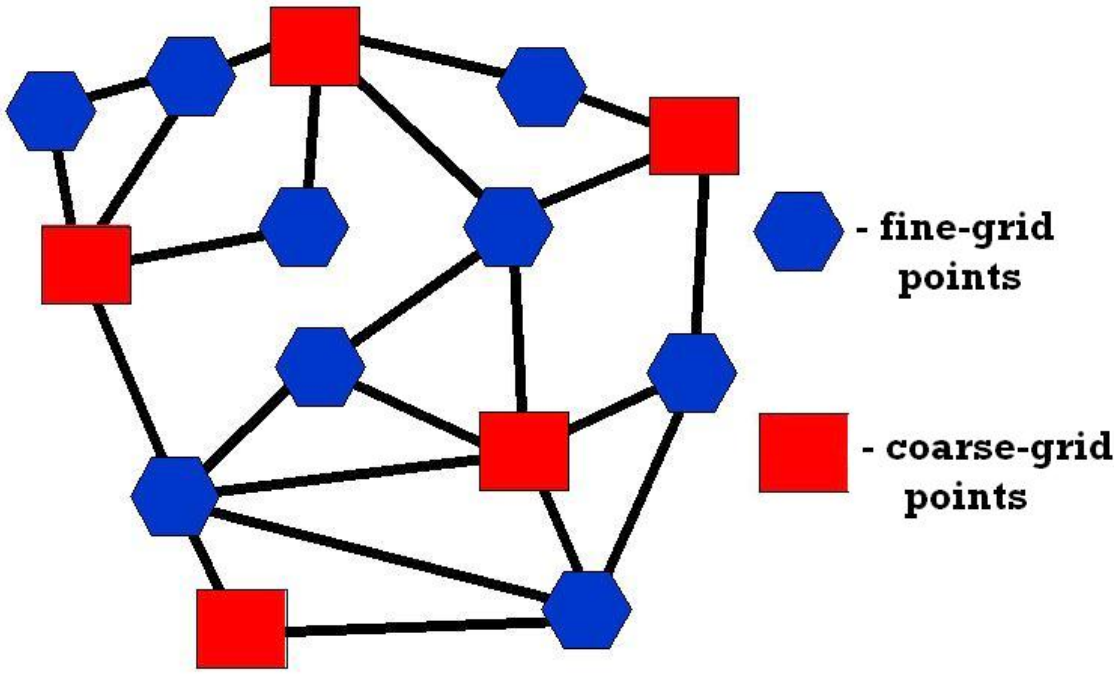


Fig. 4.3

Once coarse-grid points are determined the matrix A is re-ordered with permutation matrices so that all coarse grid points are blocked together as shown:

$$\hat{P}A\hat{P}^t = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix} \tag{4.12}$$

the restriction matrix and prolongation matrix are now:

$$\begin{pmatrix} I & \omega \\ 0 & I \end{pmatrix}, \begin{pmatrix} I & 0 \\ \omega^t & I \end{pmatrix}$$

respectively. ω acts much like ω in (2.8). A requirement is for ω and A_{fc} to have the same sparsity pattern (note: sparsity means "a matrix populated primarily with zeros", such as a diagonal matrix, tridiagonal matrix,…). [6]

4.2 Implementing AMG

Following the determination of all the required components, AMG is essentially the same as the geometric case of Multigrid.

The smoothing process takes on the same form as the geometric Multigrid. But, in this case, A is the permutated matrix defined in (4.1).

$$\begin{aligned} r_{k-1} &= b - Ax_{k-1} \\ B\delta_{k-1} &= r_{k-1} \\ x_k &= x_{k-1} + \delta_{k-1} \end{aligned} \tag{4.13}$$

Again, B is some kind of factorization of A. Since A is generally sparse, the use of the incomplete LU factorization is popular. But other techniques may used such as symmetric Gauss-Seidel.

Following the smoothing step, AMG takes on the same concepts as the geometric case of Multigrid, but instead it is using the AMG components discussed in section 4.1.

For most problems involving AMG, the determination of all the components necessary in Multigrid is what generally takes the most amount of work. This is largely in part because there is no set formula/theory for determining the coarse grid and the corresponding interpolating matrices.

4.3 Experimentation

Prof. Randolph Bank developed Multigraph [1], which computes the solutions for elliptic PDEs using AMG. Multigraph utilizes the Maximal Independent Set for determining coarse grid points. At this point, I am currently using Multigraph to study various PDEs with the goal of trying to understand trends, rates of convergence, accuracy,… using AMG combined with a number of various options (choosing the factorization for the smoother B in (4.13), choosing the discretization method,…) The following are some examples of the use of AMG on various problems, with brief commentary:

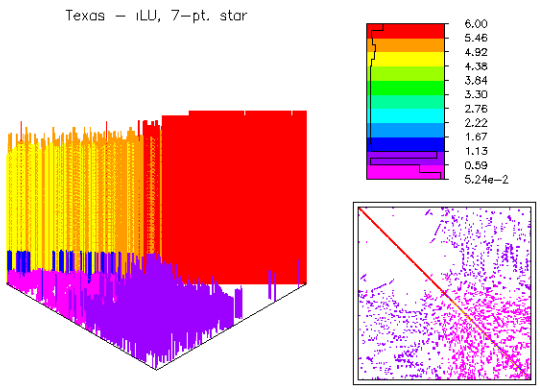


Figure 4.4

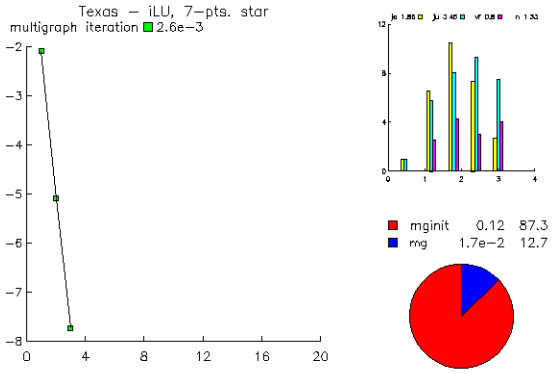


Figure 4.5

Figure 4.4 and 4.5 represent the results of the problem:

$$-.001u_{xx} - u_{yy} = 1$$

with a domain that looks like the shape of Texas (boundary conditions). On the left side, Figure 4.4 shows the matrix solution prior to being redistributed to the original Texas grid (the domain of Texas as opposed to the matrix solution is analogous to the relationship between fig. 4.1 & 4.2). The top right shows the various numerical values (a legend) for the matrix and the bottom right shows the top-down view of the matrix on the left.

Figure 4.5 is what has more significance. The left hand figure represents the norm of the error (the values on the left of the graph are based on the logarithmic scale). Each iteration is represented by the individual point, so in this case, the problem required 3 iterations. The top right chart represents the sparsity of the restriction/prolongation matrices (yellow), the current grid (cyan), and their ratio (purple). This implicitly represents the number of levels being used; since there are 5 sections, this means that 5

levels were used. (Note: because there are 5 levels, only 4 restriction/prolongation matrices were required so notice that the very left hand side bar only has the sparsity of the current grid.) The bottom right pie chart represents the amount of time required to assemble all the necessary components (red) and the amount of time actually needed to compute the solution (blue). While this pie chart is dependent on the computer processor that is being used, it is significant when being compared to other problems solved on the same computer.

This problem was solved using an incomplete LU-factorization for B (the smoother) and used a 7-point star discretization scheme (this is a variation of the finite differences scheme).

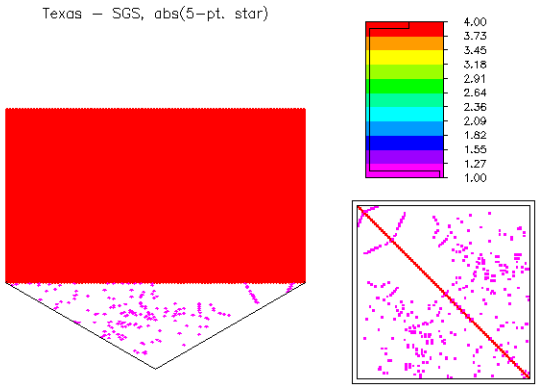


Figure 4.6

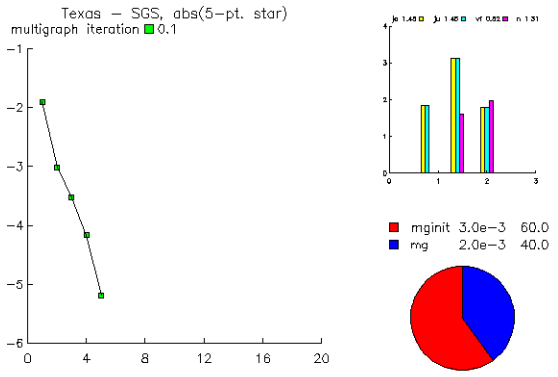


Figure 4.7

Figure 4.6 & 4.7 took on the same problem as 4.4 & 4.5 but this time it was solved in a different manner. The factorization for B was the symmetric Gauss-Seidel scheme and the discretization scheme was an absolute value applied onto the 5-point star method (another variation of the finite differences scheme).

Notice that this time, 5 iterations were required for convergence but more iterations were necessary largely in part because only 3 levels were used (as opposed to the 5 levels used

in the previous experiment). Also notice that in Figure 4.6, the solution appears to be much less complex than the solution in Figure 4.4. While they are essentially the same solutions, there are subtle differences (that are magnified due to the various colors used). At this point, this is what I have been studying; I have been trying to get acquainted with all the variations that can be used in AMG and in the process, trying to figure out what produces the optimal solution.

Acknowledgements

Many thanks to Professor Bank for taking the time to advise me in this year long project. The knowledge and experience that he has given me will not only last a lifetime but has also properly prepared me for the future. I truly appreciate the kindness he has demonstrated to me in agreeing to work on this project.

References

[1] Multigraph 1.0, R.E. Bank, Department of Mathematics, UC San Diego.

[2] William Briggs, Van Emden Henson, and Steve McCormick, *A Multigrid Tutorial*, SIAM, 3600 University City Science Center, Philadelphia, PA 19104-2688.

[3] R.E. Bank and R.K. Smith, *An algebraic multilevel multigraph algorithm*, SIAM J. On Scientific Computing.

[4] R.E. Bank and R.K. Smith, *Multigraph algorithms based on sparse gaussian elimination*, in Thirteenth International Symposium on Domain Decomposition Methods for Partial Differential Equations, Domain Decomposition Press, Bergen.

[5] R.E. Bank and C.G. Douglas, *Sharp estimates for multigrid rates of convergence with general smoothing and acceleration*, SIAM J., Numerical Analysis 22, pg. 617-699, 1985.

[6] Internet resource – www.wikiepdia.org – iterative method, discretization, partial differential equations, rate of convergence, sparse.