Joseph Kovac
18.086 Final Project
Spring 2005
Prof. Gilbert Strang

The Fundamentals and Advantages of Multi-grid Techniques

Introduction

The finite difference method represents a highly straightforward and logical approach to approximating continuous problems using discrete methods. At its heart is a simple idea: substitute finite, discrete differences for derivatives in some way appropriate for a given problem, make the time and space steps the right size, run the difference method, and get an approximation of the answer.

Many of these finite difference methods can ultimately be written in a matrix form, with a finite difference matrix multiplying a vector of unknowns to equal a known quantity or source term. In this paper, we will be examining the problem $\boldsymbol{A}u=f$, where $\boldsymbol{A}$ represents a finite difference matrix operating on u, a vector of unknowns, and f represents a time-independent vector of source terms. While this is a general problem, we will specifically examine the case where $\boldsymbol{A}$ is the finite difference approximation to the centered second derivative. We will examine solutions arising when f is zero (Laplace's equation) and when it is nonzero (Poisson's equation).

The discussion would be quite straightforward if we wanted it to be; to find u, we would simply need to multiply both sides of the equation by $\boldsymbol{A}^{-1}$, explicitly finding $u= \boldsymbol{A}^{-1}f$. While straightforward, this method becomes highly impractical as the mesh becomes fine and $\boldsymbol{A}$ becomes large, requiring inversion of an impractically large matrix. This is especially true for the 2D and 3D finite difference matrices, whose dimensions grow as the square and cube of the length of one edge of the square grid.

It is for this reason that relaxation methods became both popular and necessary. Many times in engineering applications, getting the *exact* answer is not necessary; getting the answer right to within a certain percentage of the actual answer is often good enough. To this end, relaxation methods allow us to take steps toward the right answer. The advantage here is that we can take a few iterations toward the answer, see if the answer is good enough, and if it is not, iterate until it is. Oftentimes, using such an approach, getting an answer "good enough" could be done with orders of magnitude less time and computational energy than with an exact method.

However, relaxation methods are not without their tradeoffs. As will be shown, the error between the actual answer and the last iteration's answer ultimately will decay to zero. However, not all frequency components of the error will get to zero at the same rate. Some error modes will get there faster than others. What we seek is to make all the error components get to zero as fast as possible by compensating for this difference in decay rates. This is the essence of multi-grid; multi-grid seeks to allow the error modes of the solution to decay as quickly as possible by *changing the resolution of the grid* to let the error decay properties of the grid be an advantage rather than a liability.

Basic Theory of the Jacobi Relaxation Method

Before going into the theory of the method, I first want to state that much of the following closely comes from an explanation in *A Multi-grid Tutorial* by William Briggs et al. This text explained the material as clearly and concisely as one could hope for. To a large extent, much of the "theory section" following will be reiteration of their explanation, but with emphasis on concepts which will be validated in the numerical experiments later. In no way do I claim these derivations as my own. The following is a derivation of the Jacobi method in matrix form, which is the relaxation method which will be used for the rest of the paper.

We can first express the matrix $\boldsymbol{A}$ as a sum of its diagonal component D and lower and upper triangular components L and U:

$$\mathbf{A} = D + L + U \quad (1)$$

so

$$(D + L + U)u = f \quad (2)$$

We can move the upper and lower triangular parts to the right side:

$$Du = -(L + U)u + f \quad (3)$$

We can then multiply both sides by D^{-1}:

$$u = D^{-1}(-(L + U)u + f) \quad (4)$$

We can define

$$R_J = -D^{-1}(L + U) \quad (5)$$

Therefore, we have defined the iteration in matrix form, and can write, in the notation of Briggs's chapter in *Multi-grid Methods*:

$$u^{(1)} = R_J u^{(0)} + D^{-1} f \quad (6)$$

Weighed Jacobi takes a fraction of the previous iteration and adds it to a fraction of the previous iteration with the Jacobi iteration applied:

$$u^{(1)} = [(1 - \omega)I + \omega R_J]u^{(0)} + \omega D^{-1} f \quad (7)$$

We can rewrite the above as

$$u^{(1)} = R_\omega u^{(0)} + \omega D^{-1} f \quad (8)$$

where

$$R_\omega = [(1-\omega)I + \omega R_J] \quad (9)$$

This iteration and its characteristics on the grid is the focus of this paper. Before attempting to implement this method, it is first good to predict the behavior we expect to see theoretically. One way to do this is to look at the eigenvalues of the matrix R_ω. The following again stems from Briggs, but some of the following was not explicitly explained and left as an "exercise" in the text.

We first note that, by the properties of eigenvalues and by eq. 9,

$$\lambda_{R\omega} = (1-\omega) + \omega\lambda_{RJ} \quad (10)$$

Therefore, we first need to find λ_{RJ}. We observe that:

$$L + U = A - 2I \quad (11)$$

Therefore,

$$\lambda_{L+U} = \lambda_A - 2 \quad (12)$$

Noting that, for the 1D case,

$$D^{-1} = \frac{1}{2}I \quad (13)$$

So, using eq. 5 and properties of eigenvalues,

$$\lambda_{RJ} = \frac{-1}{2}(\lambda_A - 2) = \frac{-\lambda_A}{2} + 1 \quad (14)$$

Therefore, remembering eq. 10,

$$\lambda_{R\omega} = 1 - \frac{\omega\lambda_A}{2} \quad (15)$$

The k^{th} eigenvalue of the matrix A is:

$$\lambda_k(A) = 4\sin^2(\frac{k\pi}{2n}), 1 \le k \le n-1 \quad (16)$$

So, by eq. 15, the eigenvalues λ_ω are:

$$\lambda_k(R_\omega) = 1 - 2\omega \sin^2\left(\frac{k\pi}{2n}\right), 1 \le k \le n-1 \quad (17)$$

And the j[th] component of the k[th] eigenvector is:

$$\omega_{k,j} = \sin\left(\frac{jk\pi}{n}\right), 1 \le k \le n-1, 0 \le j \le n \quad (18)$$

We can make two quick observations here. First, for ω between 0 and 1, the eigenvalues will always lie between -1 and 1, implying stability to the iteration. Second, we remember that all vectors in the space of the matrix ***A*** can be represented as a weighed sum of the eigenvectors:

$$u^{(0)} = \sum_{k=1}^{n-1} c_k \omega_k \quad (19)$$

In this case, since the eigenvectors are Fourier modes, there is an additional useful interpretation of the weighting coefficients c_k of the linear combination of eigenvectors; these are analogous to the Fourier series coefficients in a periodic replication of the vector *u*. The other key point to see here is that varying the value of ω allows us to adjust how the eigenvalues of ***A*** vary with the frequency of the Fourier modes. Plotted below is the eigenvalue magnitude versus k, for n=32. We can easily see that varying ω significantly changes the relative eigenvalue magnitude at various frequencies.

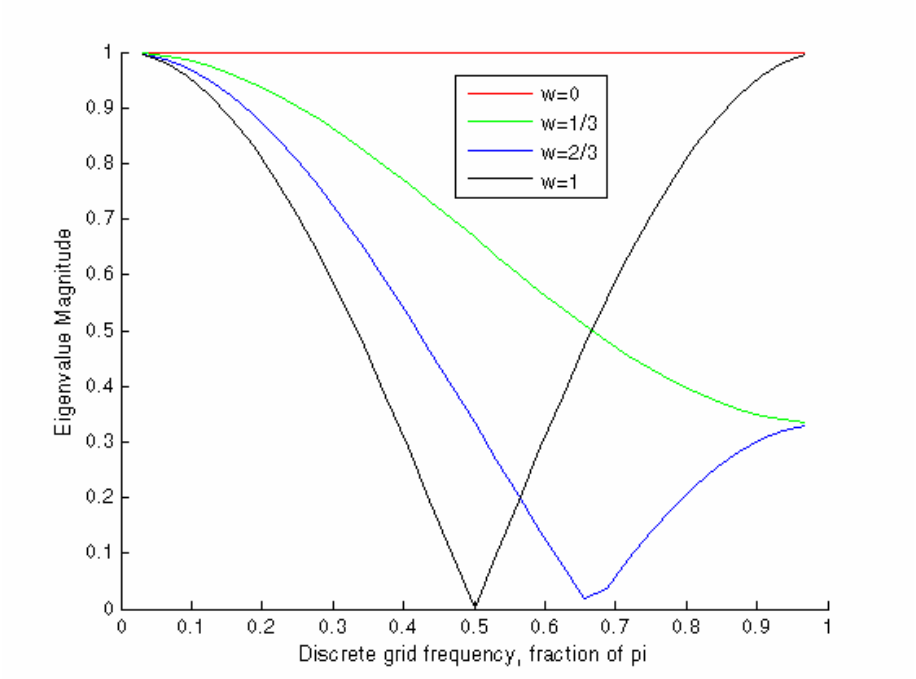


Figure 1: Distribution of eigenvalue magnitude as ω is varied

The implications of the graph above manifest themselves when we think of the homogenous case of $\boldsymbol{A}u=0$. If we were to use the Jacobi iteration in this case, and started from a vector as our "guess" at the final answer:

$$u^{(0)} = \sin\left(\frac{jk\pi}{n}\right) (20)$$

where k is between 1 and n-1, we would have an error made of only one mode, i.e. the error would lie perfectly along a single eigenvector of the matrix $\boldsymbol{A}$ and c_k would be zero for all k except the k which the vector u lay along. A priori, in this simple case, we know that the solution should converge to 0 with enough steps, as there are no source or sink terms.

In practical situations, we won't know the correct u, and the error will not be simply along one eigenvector. Now, the importance of Figure 1 becomes clear: adjusting ω allows us to decide which error frequencies on the grid we want to decay quickly relative to others. Picking the correct ω is somewhat application specific. In Briggs' example, he picks to have the eigenvalue magnitudes of the middle frequency and highest frequency match, so that the grid we work on will be decidedly favored towards either decay of high frequency modes or low frequency modes. The motive for this choice will become apparent later. For this condition, ω=2/3. This value of ω will be used throughout the numerical experiments.

Basic Theory of Multi-grid

There is an obvious disadvantage to the relaxation method so far: while high frequency error components can quickly decay, the eigenvalues of lower frequency components approach 1, meaning that these lower frequency error components take many more iterations to be damped out than higher frequencies. The essence of multi-grid is to use this feature to our advantage rather than to our detriment. What if we were to somehow take a few iterations to first smooth out the high-frequency components on the fine grid, then downsample the problem onto a coarser grid where the lower frequency components would decay faster, then somehow go back up to the fine grid?

First, consider what happens to the k^{th} mode when downsampled onto a grid half as fine as the original vector (i.e. downsampling by a factor of 2). The k^{th} mode on the fine grid becomes the k^{th} mode on the coarse grid. This also implies that the "more oscillatory" modes on the fine grid become aliased on the coarse grid. A rigorous argument complete with Fourier series plots could be made here, but that is not the point. The implication is that now the error that refused to decay quickly on the fine grid has been frequency-shifted so that it has become high-frequency error on the coarse grid and will decay quickly.

All that is left to do is to define what it means to move from a fine grid to a coarse grid and eventually come back again, and how to correctly state the problem so that the answer achieved is accurate. First, a few basic relationships need to be established. Again, this is not original thought, and closely follows the Briggs text. First, the algebraic error of the current iteration is defined as

$$e^{(n)} = u - u^{(n)} \quad (21)$$

where u is the correct value of u, and $u^{(n)}$ is the resulting approximation after n steps.

The residual is defined as the amount by which the current guess at $u^{(n)}$ fails to satisfy $\boldsymbol{A}u=f$:

$$r^{(n)} = f - \mathbf{A}u^{(n)} \quad (22)$$

Given these relationships, we can also state that

$$\mathbf{A}e^{(n)} = r^{(n)} \quad (23)$$

This fact lets us make the following statement about relaxation, as quoted from Briggs:

"Relaxation on the original equation $\boldsymbol{A}$u=f with an arbitrary initial guess v is equivalent to relaxing on the residual equation $\boldsymbol{A}e=r$ with the specific initial guess e=0."

This makes intuitive sense by eqs. 21-23: We don't know the error, but we know that the error will be zero when the residual is zero. Therefore, we can either iterate to solve $\boldsymbol{A}u=f$ or we can ask, what would the error vector have to be to yield the current residual? If we know the error, we know the answer by simple rearrangement of eq. 21.

In more mathematical terms, what the above statements are saying is the following: if we take a few iterations to get the current value of r, we could then reformulate the problem by taking that value of r, then solving the new problem $\boldsymbol{A}$e=r using Jacobi iteration, and read off the value of e after a few iterations. This will give us a guess at what the error was before the problem was restated. Rearrangement of eq. 21 would then imply that if we just added the calculated value of e to the $u^{(n)}$ we had before restating the problem, we would get a refined guess at the true vector u.

Putting this fact together with the idea of moving from grid to grid, we can combine the overall idea into the following:

1) Relax the problem for a few steps on the fine grid with $\boldsymbol{A}u=f$
2) Calculate the residual $r=f-\boldsymbol{A}u^{(n)}$
3) Downsample the residual onto a coarser grid
4) Relax on $\boldsymbol{A}e=r$ for a number of steps, starting with a guess of $e=0$
5) Upsample and interpolate the resulting e onto the fine grid
6) Refine our guess at u by adding e on the fine grid to the original value of $u^{(n)}$

The above method is the central theory of multi-grid and variations of it will show that there are significant gains to be made by changing the grid.

Implementing a Multi-grid Solver – 1D

Up to this point, the paper has mostly been a reiteration and thorough explanation of the Briggs text, specifically highlighting points which will be of importance later. At

this point, however, the subtleties and difficulties of actually implementing a multi-grid solver arise, and while a few of the upcoming points were explained in the Briggs text, much of the actual implementation required original struggling on my part. It was quite difficult despite the clarity of the theoretical basis of multi-grid. I also consulted with two students in Prof. Jacob White's group to help me think about aspects of boundary conditions more clearly.

In the 1-D case, I sought to implement as simple of a solver as possible; I was far more interested in developing a more feature-rich 2D solver. Therefore, in the 1-D case, I developed a solver which would solve Laplace's equation only, and with zero boundary conditions. In other words, I wanted to solve only the homogenous case to demonstrate that the error decays faster on the fine grid for high frequencies versus low frequencies, and that an inter-grid transfer would make error decay faster.

Since I only needed to deal with zero boundary conditions in this case, I was able to use the standard, second finite difference matrix with zero boundary conditions from class. To demonstrate the multi-grid method, I designed one solver and its associated finite difference matrix for a 16 point grid problem, and another which would operate on an 8 point grid. The finite difference method was the standard one from class.

The inter-grid transfers between the fine and coarse grids were the trickier parts. Briggs implements downsampling from the fine grid to the coarse grid by the following "full weighting" definition:

$$v_j^{2h} = \frac{1}{4}\left(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h\right) \quad 1 \le j \le \frac{n}{2} - 1 \qquad (24)$$

For a vector 7 components long, this operation can be implemented by a multiplication by the following matrix:

$$\frac{1}{4} * \begin{matrix} 1\,2\,1\,0\,0\,0\,0 \\ 0\,0\,1\,2\,1\,0\,0 \\ 0\,0\,0\,0\,1\,2\,1 \end{matrix} \qquad (25)$$

Such a matrix would move the vector from a grid of seven points to a grid of three points. This takes care of the coarsening operation; a scaled transpose of this matrix performs linear interpolation, and allows us to transfer data from the coarse grid to the fine grid. That fact is the primary motivation for using the *full weighting* method rather than simply downsampling by taking every other point from the fine grid.

Unfortunately, practical, non-ideal interpolators will also introduce error through the interpolation; this error will need to be smoothed out by relaxing again on the fine grid as it will likely have some higher-frequency components in the interpolation error best smoothed by the finer grid. If one transposes the above matrix and scales it by 2, the linear interpolation scheme would be realized.

As stated before, I only sought to confirm the idea that the higher frequency error will decay faster on the fine grid than the low frequency error. In the graph below, I defined the initial "guess" as the sum of a low frequency (discrete frequency $\pi/8$) and a higher frequency (discrete frequency $15\pi/16$). It is obvious that the high frequency component decays much faster than the low frequency component.

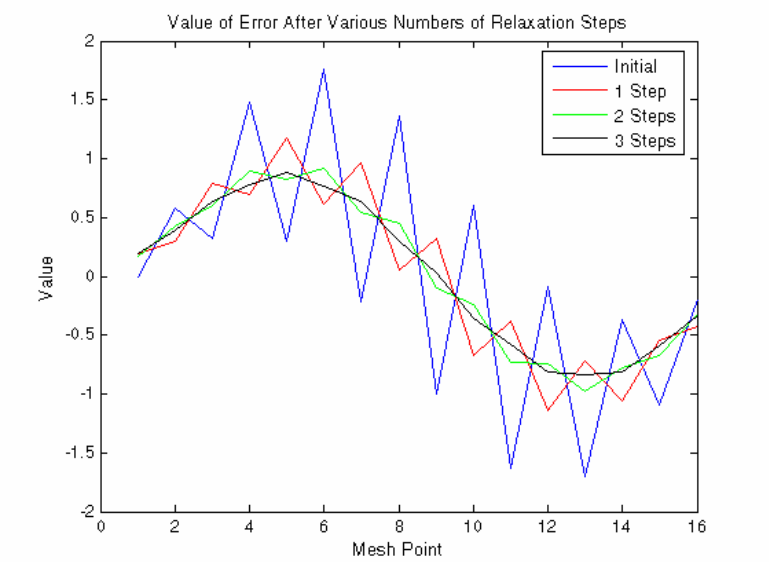


Figure 2: High-frequency component of error decays faster than low frequency component

The only other thing left to confirm in the 1D case was that a multi-grid approach showed some promise of benefit. To demonstrate this, I used the same initial function and compared a relaxation of thirty steps on the fine grid with a relaxation of ten steps on the fine grid, ten on the coarser grid, and ten more to smooth out interpolation error at the end on the fine grid, giving both approaches the same total number of steps. The results for the single grid approach versus the multi-grid approach are shown below.

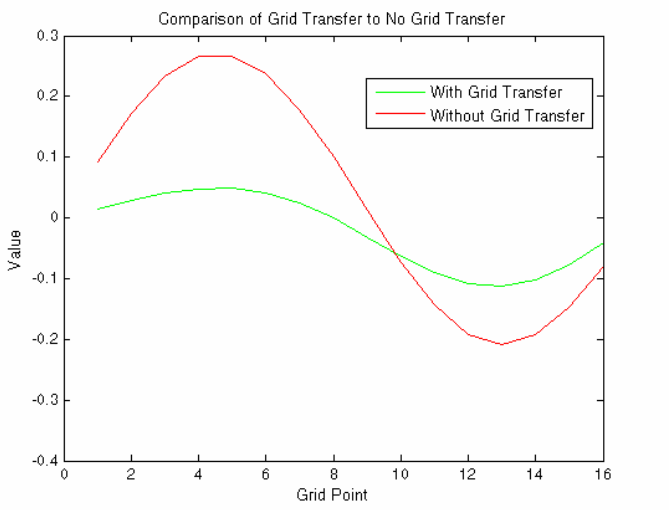


Figure 3: The advantage of the grid transfer quickly becomes apparent

I must qualify the above plot with the following information. There was a bit of a discrepancy with the definition of h in the finite difference method (i.e. the $1/h^2$ term in front of the matrix K). Intuitively, as the grid coarsens, h should change. This change was necessary and gave the best results in the 2D case. However, in the 1D case I had to tweak this factor a bit; I had to multiply the proper K on the coarse grid by 4 to get the

expected advantage working with the grid transfer. I couldn't find the source of the discrepancy, and it might be a subtlety that I missed somewhere. Nonetheless, even with this mysterious "gain factor," the above experiment proves that faster convergence to the zero error state can happen with a grid transfer rather than simply staying on the fine grid for all steps.

Implementing a Multi-grid Solver – 2D

The 2D case shares a number of similarities with the 1D case, but it carries a number of subtleties with it that make implementation of the method significantly more difficult than the 1D case. The most difficult aspect to attack was getting the boundary conditions right. I decided that I would stick to Dirichlet boundary conditions for this project, as their implementation was significant work, let alone think about Neumann conditions.

The 1D case was implemented minimally, only thoroughly enough to demonstrate the relative rates at which the different modes of the error in the homogenous case decayed and that grid transfers showed a hint of promise. In the 2D case, I wanted to implement a more useful and practical solver. Specifically, I wanted to be able to specify Dirichlet boundaries, source terms in the grid, and boundaries within the grid. In the electrostatics case, this would be like saying that I wanted to be able to specify the boundary voltages of my simulation grid, any charge source in the medium, and the voltages of any electrodes existing internal to the grid.

Specifying charge sources is very easy: just specify them in *f*. However, specifying boundary conditions is more difficult. I decided to incorporate the boundary values by altering both the matrix ***A*** and the right-hand side *f*. As we learned, the 2D finite difference matrix generally has the following form:

$$K = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \cdot & \cdot & \cdot \\ & & -1 & 2 \end{bmatrix} \qquad \mathbf{K_{2D}} = \begin{bmatrix} K+2I & -I & & \\ -I & K+2I & -I & \\ & \cdot & \cdot & \cdot \\ & & -I & K+2I \end{bmatrix}$$

Figure 4:The K matrices from Prof. Strang's new text

In order to properly implement the boundary condition, we must remember the equations underlying the K2D matrix: we are simply solving an N^2 by N^2 system of linear equations. Therefore, if we fix *u(p)=b* for some value *p* and constant *b*, this means that in our system of linear equations, whenever the value *u(p)* shows up in one of the simultaneous equations, its value must be *b*. The way to accomplish this is simple; we must alter ***A****u=f* to reflect this fact. If we simply set the p^{th} row of ***A*** to zero, and then set the p^{th} column of that row to be 1 (i.e. set the p^{th} diagonal entry to 1), the value at *u(p)* will be forced to *f(p)*. Therefore, assuming *f* was originally the zero vector, we must now satisfy that the p^{th} entry of *f* now be equal to *u(p)*, so now *f(p)=b*. This has forced *u(p)=b*.

One might wonder if we should also set the p^{th} column to zero. We should not, as the columns allow the forced value of *u(p)* to propagate its information into other parts of

the grid. Physically, at least in electrostatics, there is no intuition of having a source at a point where there is a boundary condition, because the boundary manifests itself as a source in this implementation. Therefore, if there is a boundary at $u(p)$, $f(p)$ will be zero at that point before we put the constraint on the point.

The above method works excellently for interior boundary points. The actual grid boundaries, where Dirichlet conditions were sought, are not as straightforward. Some finite difference methods deal with these points implicitly by never explicitly defining grid points at the edges. Instead, I decided to explicitly define these points and alter the $\boldsymbol{A}$ matrix, creating a matrix format which deviated from that in the figure above.

The difficulties in the above implementation arise when the difference matrix of "K2D" "looks" outside the grid implicitly when it touches the edges of the grid. This is easier to see in the 1D K matrix. The first and last rows of K are missing -1's in that matrix. Implicitly, this means that the finite difference operator looked "off the grid" and found zero, unless a nonzero value shows up to make a non-zero entry in f. I decided to explicitly define the boundary values instead of trying to keep up with these issues.

First, the ordering definition of u and $\boldsymbol{A}$ must be defined. For my implementation, $u(0)$ was the upper-left corner of the grid and $u(N)$ was the lower-left corner. $u(N+1)$ was the point right of $u(0)$, and $u(2N)$ was the point to the right of $u(N)$. $u(N^2-N+1)$ was the upper-right corner, and $u(N^2)$ was the lower-right corner.

Therefore, to define explicit boundaries, I needed to set the first N values of u to the Dirichlet conditions. Therefore, when constructing $\boldsymbol{A}$, by the reasoning from the interior boundary points described above, the upper-left corner of $\boldsymbol{A}$ was a block identity matrix of size N, and $f(1\ldots N)$ was set to the boundary value. This construction dealt with the left edge easily. I constructed the rest of $\boldsymbol{A}$ by using the traditional 2D stencil from class. In order to account for the top and bottom edges, I made sure to set those corresponding rows in $\boldsymbol{A}$ to zero, except with a 1 on the diagonal and the corresponding value of f to the boundary condition. When I reached the lower-right corner, I augmented $\boldsymbol{A}$ with another block identity matrix of size N, and set f to the boundary condition at those points. A matrix density plot is shown below to illustrate this construction. Approaching the boundaries explicitly made them easier to track, but an algorithm to construct a general $\boldsymbol{A}$ for a given mesh size was quite difficult; that is the tradeoff.

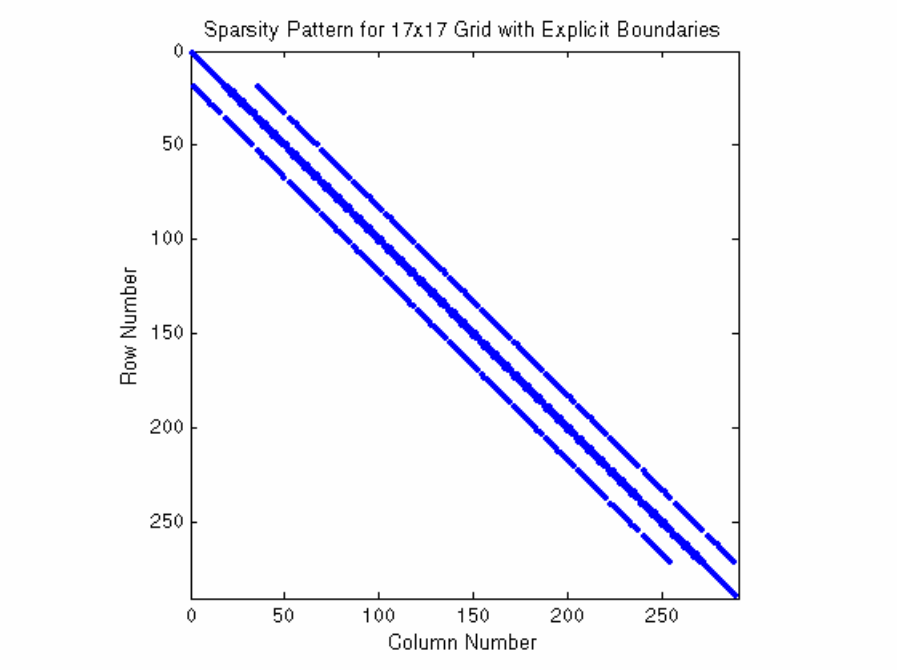


Figure 5: Sparsity pattern of my altered finite difference matrix which allows for explicit boundary definition. Notice the periodic gaps along the diagonal representing the top and bottom edges of the grid.

With boundary conditions properly incorporated, the last topic to address was that of inter-grid transfers: what matrix downsamples the original data to a coarser grid? Which matrix transfers from the coarse grid to the fine grid? The proper way to phrase the question is this: what is the proper way to transfer data from one grid to another?

In going from a coarse grid to a fine grid, the central problem is *interpolation*. The central ideas of interpolation and downsampling were discussed in the 1-D section. The 2D implementation is highly similar, but with a little more complexity than the 1D case due to slightly trickier boundaries on the edges. I decided that I would again seek to do downsampling as a weighted averaging of neighboring points rather than by injection. Again, the reason for this approach was so that simply transposing the downsampling matrix would yield the linear interpolation matrix for upsampling and linear interpolation.

Such a downsampling matrix was rather straightforward to implement for the interior points of the grid. Incorporating the edges would have been somewhat trickier, and the averaging scheme used, if simply allowed to include the edges, would have changed the boundary values themselves, which is to be avoided at all costs. Therefore, I took the following approach.

Downsampling

1) Calculate the residual
2) Remove the edges from the fine grid residual data
3) Design a downsampling matrix to transform the inner grid residual data from the fine grid to the twice-as-coarse grid
4) Apply the downsampling matrix to the interior residual data

5) Append zeros around downsampled residual data grid to represent the fact that the residuals are by definition zero at the edges where we have defined the exterior boundaries.

Upsampling

1) Remove the zeros from the coarse grid's edges (this is after we have relaxed the residual problem on the coarser grid)
2) Apply the scaled, transposed downsampling matrix to the interior points to get the interpolated guess at the error on the fine grid
3) Pad the resulting upsampled interior points with zeros since there is no refinement in the error at the known boundaries
4) Add the upsampled guess at the error to the original guess at u

The downsampling operator was defined explicitly in Briggs, though in an index form rather than matrix form. I implemented the operation as a matrix in order to speed up computation in MatLab. Briggs defines the downsampling operation as follows in 2D (v^{2h} is the vector represented on the coarse grid, v^h is the grid on the fine grid):

$$v_{ij}^{2h} = \frac{1}{16}\begin{bmatrix} v_{2i-1,2j-1}^h + v_{2i-1,2j+1}^h + v_{2i+1,2j-1}^h + v_{2i+1,2j+1}^h \\ +2\left(v_{2i,2j-1}^h + v_{2i,2j+1}^h + v_{2i-1,2j}^h + v_{2i+1,2j}^h\right) \\ +4v_{2i,2j}^h \end{bmatrix}, 1 \le i, j \le \frac{n}{2} - 1 \quad (26)$$

I implemented it instead as a matrix operator. In order to implement the above operation, the following stencil was used:

$$\frac{1}{16}((1 \quad 2 \quad 1)((M-3)zeros)(2 \quad 4 \quad 2)((M-3)zeros)(1 \quad 2 \quad 1)) \qquad (27)$$

M was the number of interior points in one edge of the fine grid from which the mapping was supposed to take place. As a final point, the stencil was not simply replicated along the diagonal, rather to implement the correct operation it was implemented in a staggered pattern (similar to 1D) as shown in the sparsity pattern below.

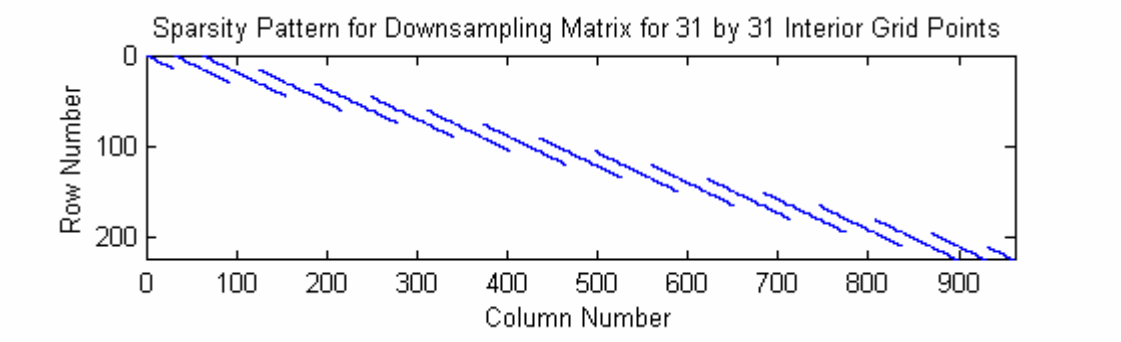


Figure 6: Sparsity pattern for the downsampling matrix; stencil is replicated in the matrix in a staggered fashion

Briggs describes the indexed form of the 2-D linear interpolation operator, and it is simply implemented by transposing the above matrix and scaling by 4.

Numerical Experiments in 2-D

Finally, with all the tools in place for 2-D, numerical experiments could be undertaken. A convincing example that the system was working properly would be solution of a problem with a known solution. To this end, I decided to compare the multi-grid solver's solution to the actual solution to Laplace's equation with the following boundary conditions:

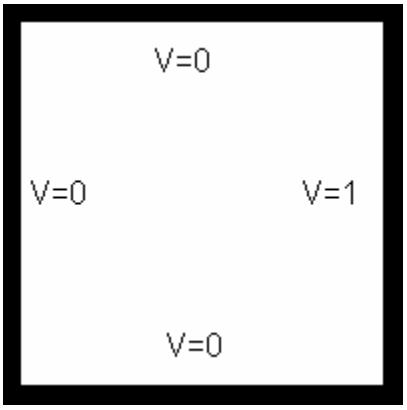


Figure 7: Boundary conditions for the known solution to Laplace's equation

The solution to Laplace's equation in this case can be expressed as a sum of sines and hyperbolic sines. I will not go through the derivation here for that answer, but I wrote a loop in MatLab to compute a partial sum of the relevant terms to produce the correct answer so that it could be compared to the multi-grid solver's output. The two solutions produced are very similar. The Gibbs phenomenon is apparent in the partial sum of sines. They are plotted below.

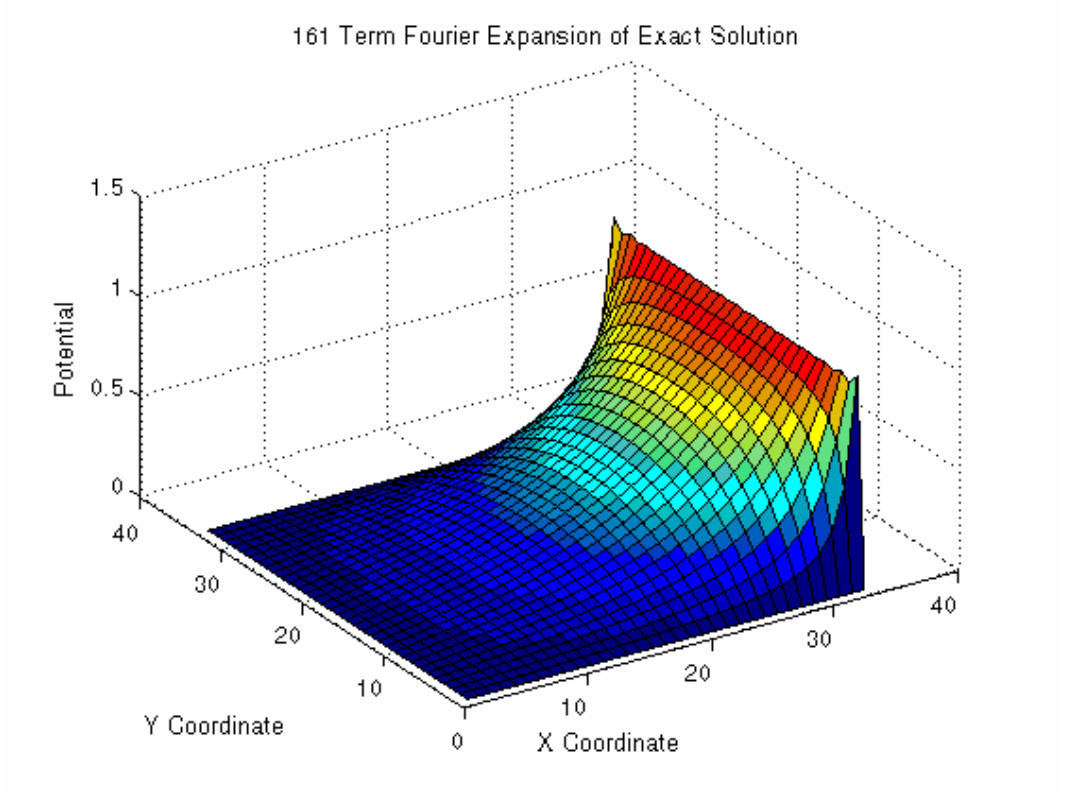


Figure 8: Fourier expansion of actual solution. Right edge converges to 1; perspective of plot is misleading.

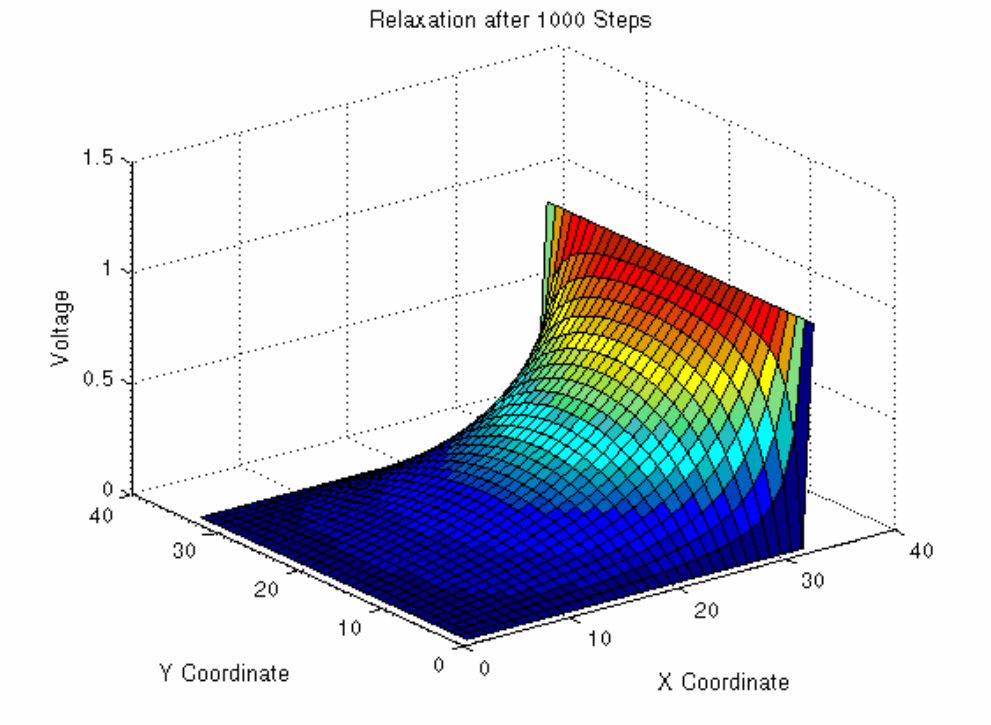


Figure 9: My solver's output after 1000 steps on the fine grid

The next obvious experiment is to see how quickly the relaxation error decays to zero. The decay of pure modes was examined for the 1-D case. Now however, the solver was considerably more powerful, so examining more sophisticated problems would be interesting. In general, we don't know the final solution; we only know the residual after a step. So, from now on, instead of discussing error, we will examine how the norm of the residual decays.

An interesting case to examine would be a unit spatial impulse. The Fourier expansion of an impulse has equal weights on all frequencies, so examining how an initial guess of an "impulse" decays to zero everywhere in homogenous conditions would be insightful. The following plots show a unit impulse located at 67,67 on a 133 by 133 grid after various numbers of steps.

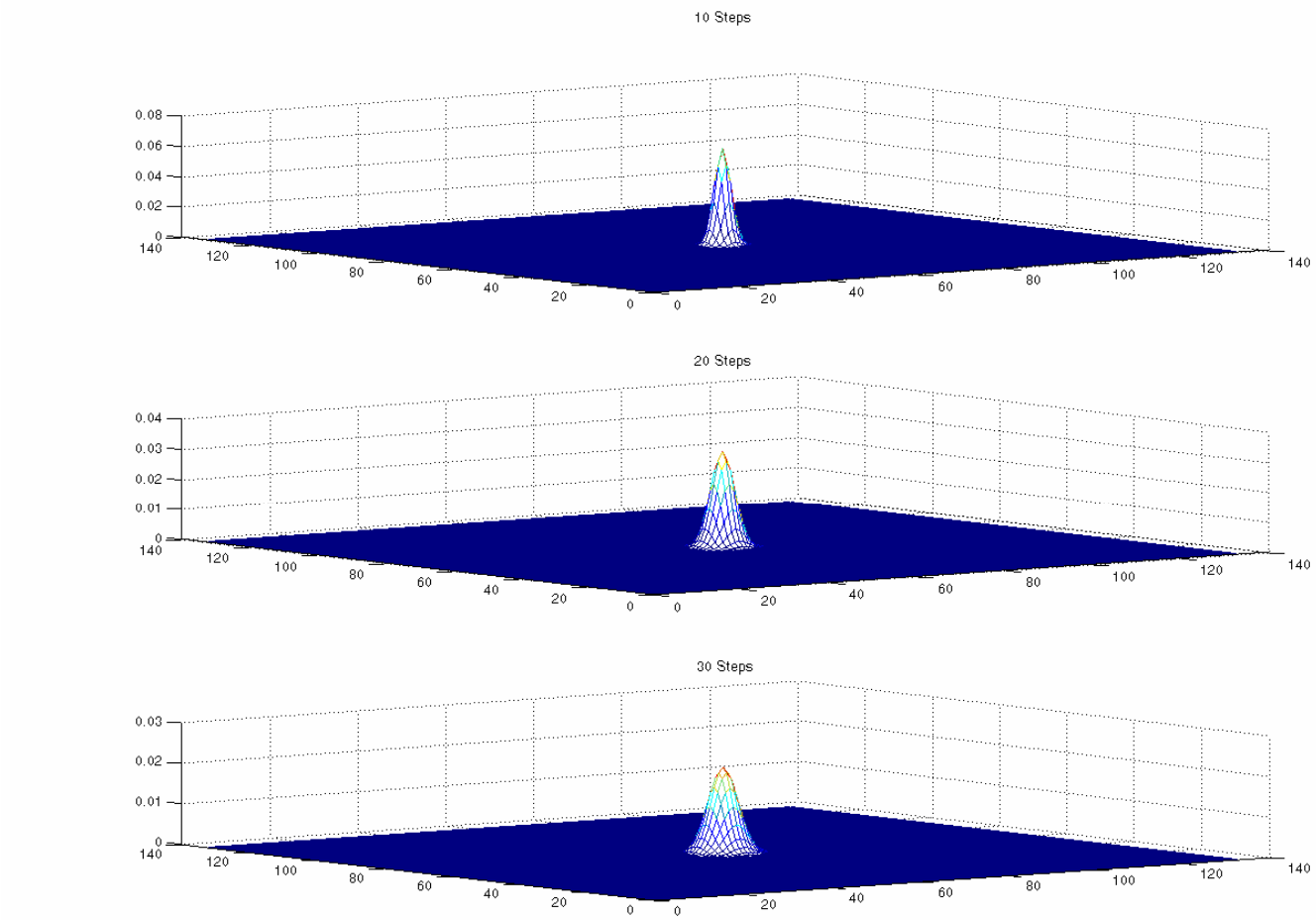


Figure 10: Decay of a unit impulse. Notice that after 30 steps, the solution is much "smoother" than after 10. This is because the higher-frequency modes have been filtered out by the fine grid.

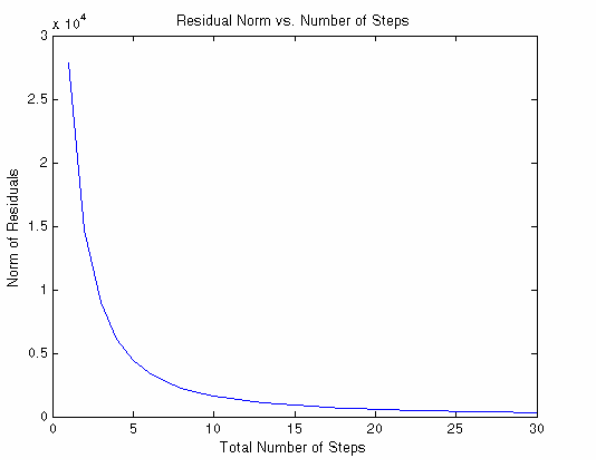


Figure 11: Stalling of residual decay on the fine grid

We can see that the residual decays very quickly initially, but the decay rate then stalls. This is because the error that is left is lower-frequency error which does not decay quickly on the fine grid. This is seen in the figure, as after twenty and thirty iterations, the solution looks very smooth.

The question to ask now is, how much better could the answer be after a number of steps if we employ a multi-grid approach? In the following experiment, three grid coarseness levels were available. Grid 1 was the fine grid. Grid 2 was the "medium" grid, and was twice as coarse as Grid 1. Grid 3 was the "coarse" grid, and was twice as coarse as grid 2.

An experiment similar to the one in Figure 10 was attempted with the unit impulse. Three relaxations were performed, starting with homogenous conditions and a unit impulse initial condition.

Trial 1: Relax with 2500 steps on Grid 1

Trial 2:

a) Relax with 534 steps on Grid 1
b) Move to Grid 2 and relax for 534 steps
c) Move back to Grid 1, incorporate the refinement from (b), and relax for 1432 steps for a total of 2500 steps

Trial 3:

a) Relax with 300 steps on Grid 1
b) Relax with 300 steps on Grid 2
c) Relax with 300 steps on Grid 3
d) Move back to Grid 2, incorporate refinement from (c) and relax for 800 steps
e) Move back to Grid 1, incorporate refinement from (d) and relax for 800 steps for a total of 2500 steps

This scheme was chosen because it gave all methods the total number of steps. Additionally, for trial 2 and trial 3, the ratio of forward relaxations (i.e. relaxation after moving from fine to coarse) to backwards relaxation was constant at 3/8. The detail after 2500 steps is shown below for all three cases.

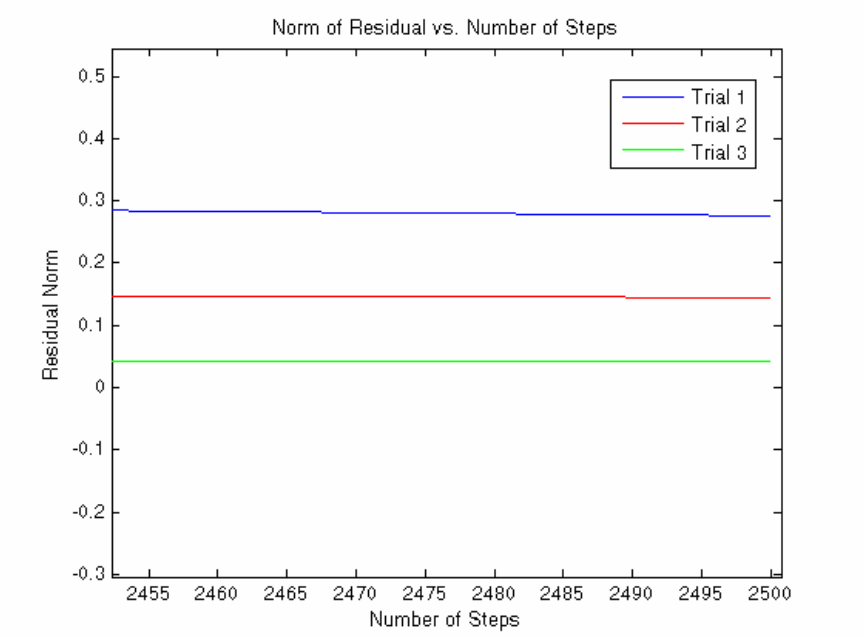


Figure 12: The three-grid scheme outperforms both the single and dual grid schemes.

It is clearly visible that given the same number of steps, the three-grid scheme outperforms the single grid scheme and the dual grid scheme. However, it is not a given that this result will always be the case. If the error, for example, was known to be almost purely high-frequency, the advantage of the grid transfers might be outweighed by the computation power necessary to keep making the transfers and interpolations.

The case shown above for the unit impulse is a case where the frequencies are equally weighted in the initial conditions. As a second trial, I examined how the residuals decayed for an initial condition with more low-frequency content. This case was again homogenous with boundary conditions of zero, but the initial "guess" was 1 everywhere except at the boundaries. I repeated the experiment with these initial conditions, and the results are shown below.

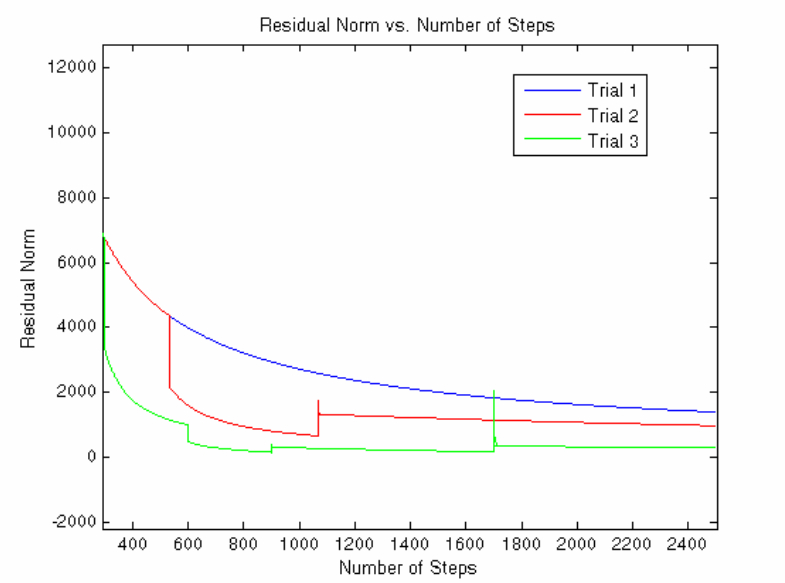


Figure 13: Decay of residuals for the three schemes. The only fair comparison across methods is after all three trials have made it back to the fine grid (i.e. after the last spike in residual on the green line which comes from the interpolation error). The three-grid method is the most accurate after 2500 steps.

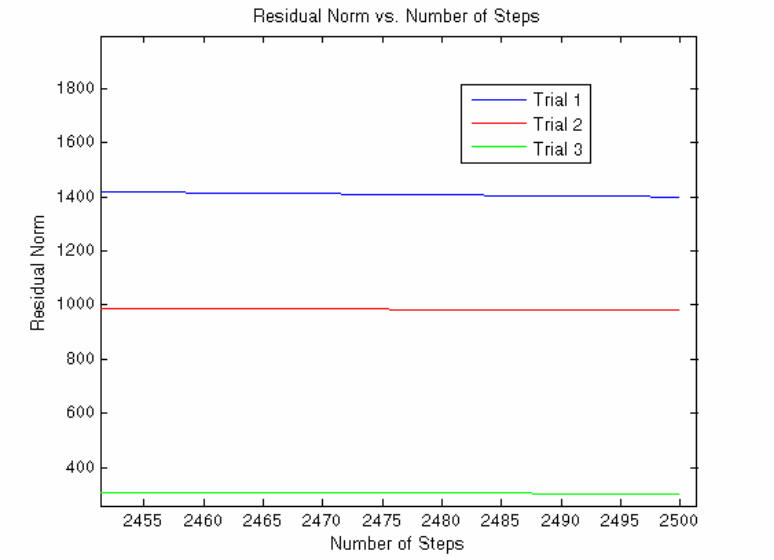
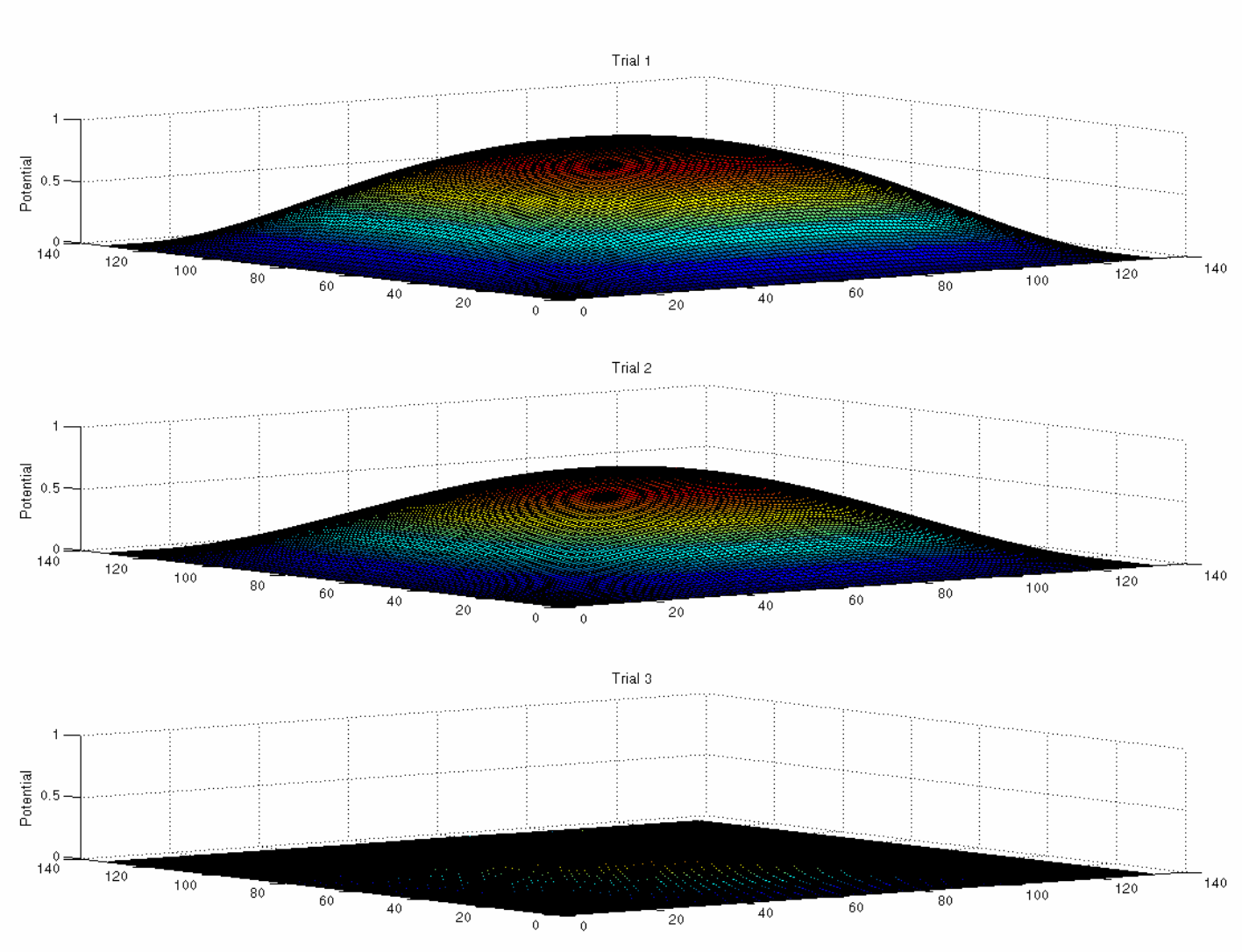


Figure 14: Detail of the final residual values for the three methods. The three-grid method clearly wins out over the others. This is a more drastic out-performance than before since the initial condition contained more low frequency error, which was better managed on the coarser grids.

Once again, the three-grid scheme wins. It is important to note that in the first figure, the "norm" during the steps while the problem's residual resides in the coarser grid is not comparable to the norm of vectors in other grids, as the norm is vector-size dependent. Therefore, the only truly fair points on the graph to compare the methods are when all methods are on the same grids, namely the very beginning and very end (shown in detail in Figure 14).

There are two ways to interpret the results. We can get a better answer with the same number of steps by using the three-grid cycle. Alternatively, we could stop earlier with the three-grid cycle and settle for the value that the other methods would have produced with more steps. The tradeoff is completely problem dependent.

I was suspicious as to how much difference the above residuals made in the appearance of the answer, especially given the much higher initial values of the residuals. The difference after trials 1, 2 and 3 is stark and is shown below. Remember, with an infinite number of steps, all three methods would converge to a value of zero everywhere.



The results are obviously different. Trial 3 yielded an answer visually very close to the correct answer of 0. It is clear that going beyond simply one coarsening operation yielded great benefits. The natural next step would be to try a fourth, coarser grid, and continue coarsening. One could coarsen the grid all the way to a single point. Also,

trying a multitude of different step distribution schemes in order to maximize efficiency of steps at each grid could be tried too.

One could easily write a book about these concerns, but going far down either of these paths would step outside the scope of this introduction to multi-grid and its benefits. Instead, it would be more appropriate to confirm this limited-multi-grid system on other problems.

As stated earlier, a key goal of my 2D implementation was the ability to impose boundary conditions within the grid. I designed my Jacobi relaxer, as described earlier, to support internal boundaries as well. I implemented the system so that I could simply use Windows Paint ® to draw whatever regions I wanted to specify as at a particular "voltage." As an appropriate example, I decided to determine the potential distribution resulting from having an electrode in the shape of the letters "MIT" in the grid, with 0 volt boundary conditions on the edge of the grid. The bitmap used to create the boundary conditions is shown below. The black letters are defined to be 1 volt, the white area zero volts.

Shown below is a plot of the relaxation solution (still staying all the time on the fine grid) of the solution to the problem.

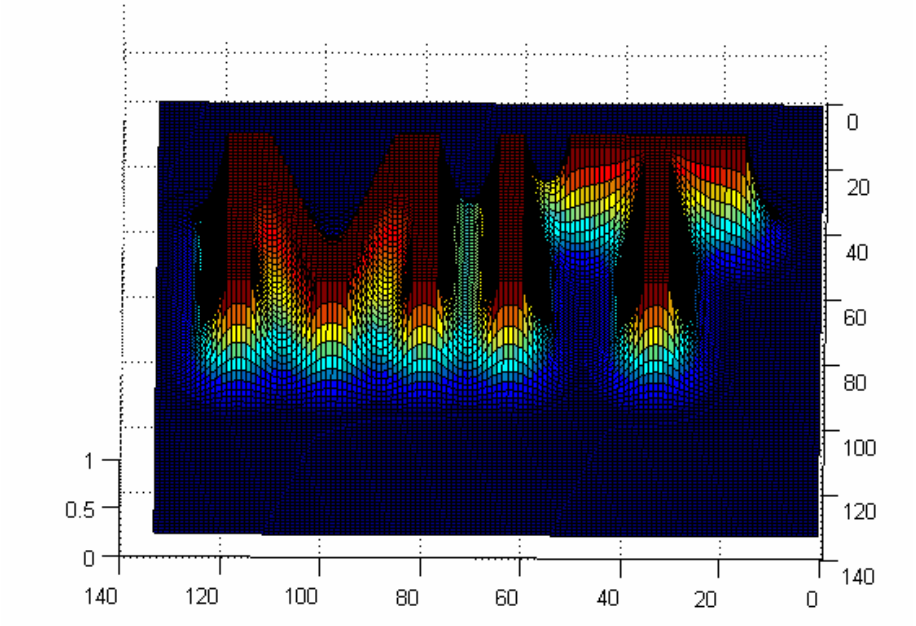


Figure 15: "Electrodes" in shape "MIT" relaxed on fine grid

Finally, just to prove that the ability to add charge into the simulation was added, I added a line of charge the under the "electrodes" used to write "MIT" to underline the word.

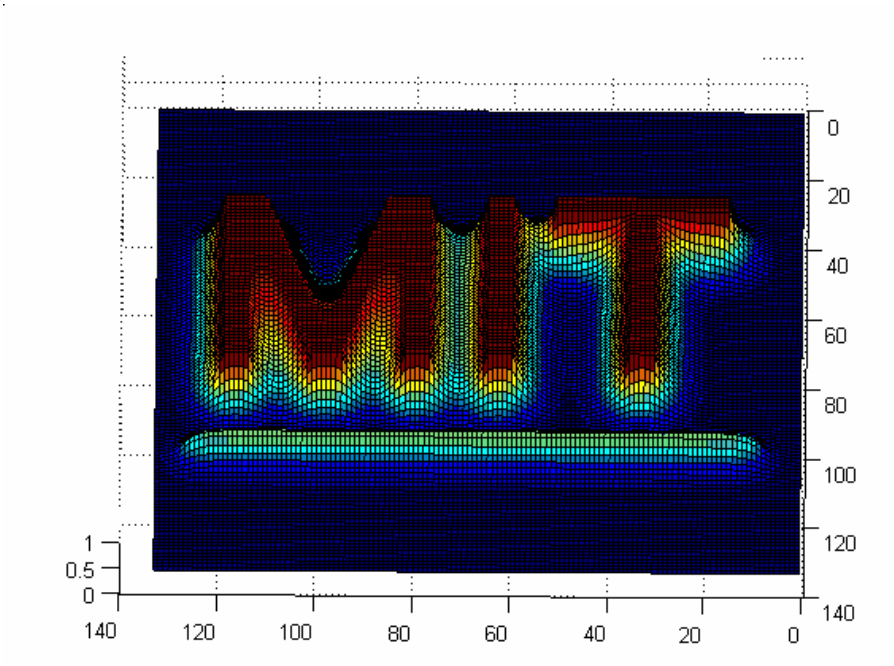


Figure 16: MIT electrodes with a line of charge underlining them

Placement of arbitrary charge within the medium with arbitrary boundary conditions was supported as well. The figure below shows the gains made with a 930 step double-grid method vs. a single grid method; the point is to show that the charge placement was supported across multiple grids.

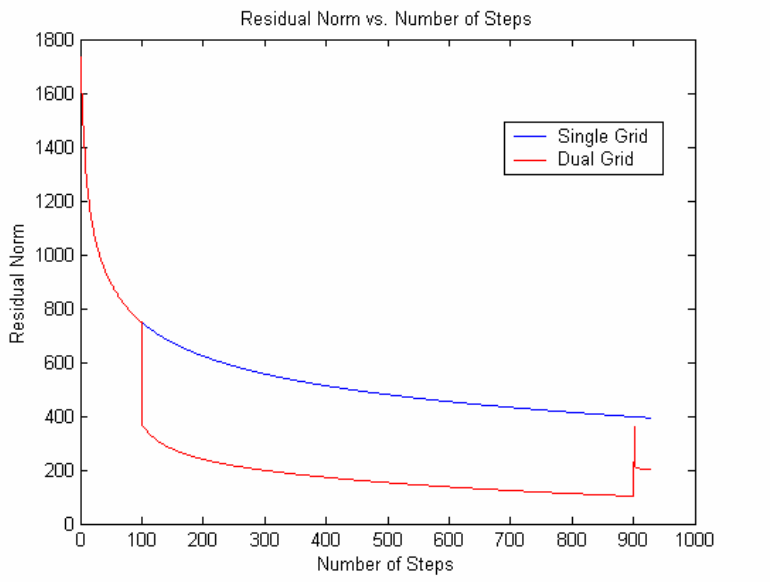


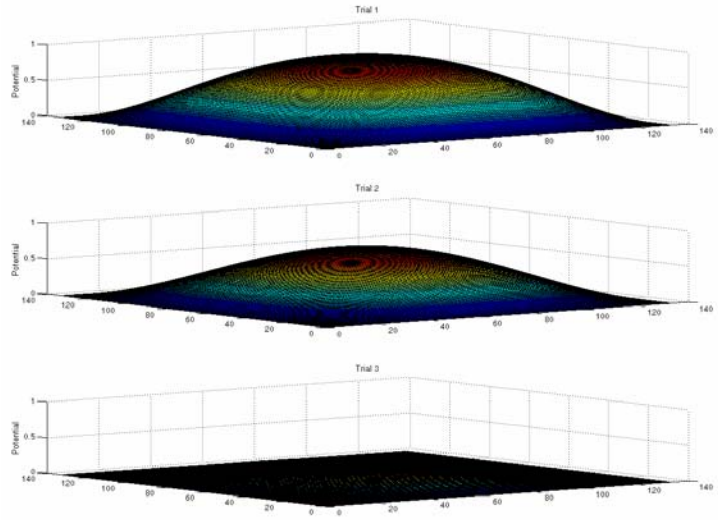
Figure 17: Multi-grid support included for arbitrary charge distributions as well

As for multi-grid support of internal boundary conditions (i.e. if we wanted to relax the MIT electrode problem with multi-grid), I did not quite get around to that. I thought I had it done, but I discovered too late that I had forgotten a subtle aspect. When relaxing on $\boldsymbol{A}e=r$, I forgot to pin internal values of e at the boundaries to zero, as by definition there would never be error at one of the internal boundaries. Without doing this, the compensated error approximation from the coarse grid will attempt to force the value at the boundary to deviate from the correct internal boundary condition.

This could be fixed by changing the matrix $\boldsymbol{A}$ by making the rows corresponding to these points zero, except for a 1 on the diagonal. Additionally, r at that point would need to be 0, but I had already thought of and taken care of that and had implemented that aspect. As simple as the fix sounds, I had an elaborate setup in the algorithm for the current system, and making the change would have meant tearing the whole system down and building it back up, which was unrealistic as late as I found the problem. However, I did determine the source of the problem and its likely fix.

Conclusion

The most convincing figure of the paper is replicated below.



This figure truly sums up the power of multi-grid. In the same number of steps, the approach with the largest utilization of coarsening got closest to the right answer. One can be more quantitative about the true efficiency of the method: what is the computational tradeoff between doing a few more iterations on the fine grid and moving to a coarse grid? Do the benefits of moving outweigh the computational costs of downsampling and interpolation? What is the best way to design a multi-grid cycle scheme? How long should one spend on a coarse grid versus a fine one? These are all excellent questions of multi-grid, and there is no definitive right answer.

As for the tradeoff between interpolation and downsampling versus spending time on a fine grid, making an absolutely definitive answer is difficult. However, multiplying by the Jacobi matrix for an iteration and multiplying by an upsampling or downsampling matrix consist of matrix multiplications of relatively the same size and density, making the intergrid transfers relatively cheap and insignificant compared to large numbers of

steps of relaxation computation. It is more likely that the tradeoffs will come from determining the proper amount of time to spend at each grid. A possible way to do this would be to look at the FFT of the residual, try to predict the spectral content of the error, and adaptively decide which grid to move to based on that result. Other ways would be to look for patterns in the particular class of data being examined. Such design issues would make excellent projects in and of themselves.

What is definite, however, is that multi-grid can yield astonishing results in the right circumstances and can give excellent answers in a fraction of the time that a single-grid relaxation would need. If an exact solution is not necessary, and the grid is huge, multi-grid is an excellent way to go.

References

Briggs, William, et al. *A Multigrid Tutorial, Second Edition*. © 2000 by Society for Industrial and Applied Mathematics.

Jaydeep Bardhan and David Willis, two great advisors from Prof. Jacob White's group.

Prof. Gilbert Strang, for his draft of his new textbook.