Introduction

- Interested in large linear systems of the form Ax=b.
- The core idea is that low frequency errors on a fine mesh appear as high frequency errors on a coarse mesh. Point iterative solvers are very effective at eliminating high frequency errors but slow to eliminate low frequency errors. Consequently, point iteration solvers are very slow to converge for problems with large numbers of unknowns.
- Multigrid takes advantage of these properties by employing the iterative “smoothing” solver over a sequence of grid densities from fine to coarse.
- These methods can be very effective, with the computational cost scaling directly with the number of unknowns, which is the best one could expect.
- Problems of this type are common in scientific computing applications such as computational fluid dynamics (CFD).
- In fact, most commercial CFD solvers employ multigrid methods.
- We will first examine the properties of two common iterative methods, and then look at a geometric multigrid implementation.

Jacobi Iterative Method

In general, we wish to solve

$$A\boldsymbol{x} = \boldsymbol{b}$$

iteratively using the Jacobi method. We can decompose the matrix A as:

$$A = L + U + D$$

Then our iterative scheme can be written:

$$\boldsymbol{x}^{k+1} = D^{-1}(b - (L + U)\boldsymbol{x}^{k})$$

****

Let's look at a simple example. Consider the steady state temperature distribution ϕ(x) along an insulated rod with temperatures held fixed at 0 and 1 on the left and right ends, respectively (where $0 \leq x \leq 1$).

The governing equation is:

$$\frac{d^2\phi}{dx^2} = 0$$

We can discretize this equation using a second order accurate central finite difference approximation as:

$$\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} = 0$$

Let's use 5 total grid points, with 3 interior unknowns. Our index notation specifies i=0 at the left boundary and i=4 at the right boundary. The three discretized equations then become, after including the boundary conditions:

$$\phi_2 - 2\phi_1 = 0$$

$$\phi_3 - 2\phi_2 + \phi_1 = 0$$

$$1 - 2\phi_3 + \phi_2 = 0$$

Or, written in matrix form:

$$\begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

Where

$$D = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

$$L = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{\phi}^{k+1} = D^{-1}(\boldsymbol{b} - (L + U)\boldsymbol{\phi}^k)$$

$$\begin{bmatrix} \phi_1^{k+1} \\ \phi_2^{k+1} \\ \phi_3^{k+1} \end{bmatrix} = \begin{bmatrix} -1/2 & 0 & 0 \\ 0 & -1/2 & 0 \\ 0 & 0 & -1/2 \end{bmatrix} \left(\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \phi_1^k \\ \phi_2^k \\ \phi_3^k \end{bmatrix} \right)$$

Or, written out as individual equations:

$$\phi_1^{k+1} = \frac{1}{2}\phi_2^k$$

$$\phi_2^{k+1} = \frac{1}{2}\left(\phi_1^k + \phi_3^k\right)$$

$$\phi_3^{k+1} = \frac{1}{2}\left(\phi_2^k + 1\right)$$

As expected, these are thus the same equations as we originally derived directly from the finite difference approximation.

Now, we want to consider convergence. That is, given an initial guess, with the iterative procedure converge?

Whether or not it converges depends on the spectral radius of the iteration matrix:

$$D^{-1}(L+U)$$

Or, for our specific case

$$\begin{bmatrix} -1/2 & 0 & 0 \\ 0 & -1/2 & 0 \\ 0 & 0 & -1/2 \end{bmatrix}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \equiv \begin{bmatrix} 0 & -1/2 & 0 \\ -1/2 & 0 & -1/2 \\ 0 & -1/2 & 0 \end{bmatrix}$$

The spectral radius is equal to the maximum value of the magnitudes of the 3 eigenvalues for this 3x3 matrix (or n eigenvalues for an n x n matrix). In other words, the largest absolute value of the eigenvalues of the iteration matrix.

For the above matrix, the eigenvalues (λ) are given as:

$$\lambda_1 = -1/\sqrt{2} = -0.7071\ldots$$

$$\lambda_2 = 1/\sqrt{2} = 0.7071\ldots$$

$$\lambda_3 = 0$$

The spectral radius is thus 0.7071 which is less than one, so the iterative procedure will converge. (We can also look at our original

matrix form of the discretized equation (the A matrix) and note it is diagonally dominant, hence we already knew the iterative procedure would converge.) **The rate of convergence is proportional to the spectral radius.**

Specifying 2 unknowns results in a spectral radius of 0.5.

Specifying 3 unknowns results in a spectral radius of 0.707.

Specifying 4 unknowns results in a spectral radius of 0.809.

Specifying 5 unknowns the spectral radius is 0.866.

What we see is that as the number of unknowns increases, the spectral radius increases.

Hence, increasing the number of unknows slows down the convergence rate.

Free online sources for computing eigenvalues:

https://www.intmath.com/matrices-determinants/eigenvalues-eigenvectors-calculator.php

https://www.wolframalpha.com/input/?i=eigenvalues

Gauss-Seidel Iterative Method

In this case out iterative equations appear as:

$$\phi_1^{k+1} = \frac{1}{2}\phi_2^k$$

$$\phi_2^{k+1} = \frac{1}{2}\left(\phi_1^{k+1} + \phi_3^k\right)$$

$$\phi_3^{k+1} = \frac{1}{2}\left(\phi_2^{k+1} + 1\right)$$

The difference from Jacobi is that we are using the most recently available solutions in the update procedure. (The above assumes iterating from 1 to 3, rather than 3 to 1.)

We can again decompose the original matrix A as:

$$A = L + U + D$$

Then the Gauss-Seidel method can be written as:

$$\boldsymbol{\phi}^{k+1} = (L + D)^{-1}(\boldsymbol{b} - U\boldsymbol{\phi}^k)$$

If we write these equations out, we find:

$$\phi_1^{k+1} = \frac{\phi_2^k}{2}$$

$$\phi_2^{k+1} = \frac{\phi_2^k}{4} + \frac{\phi_3^k}{2}$$

$$\phi_3^{k+1} = \frac{\phi_2^k}{8} + \frac{\phi_3^k}{4} + \frac{1}{2}$$

If we remove ϕ_2^k from the second equation using the first equation, the second equation becomes:

$$\phi_2^{k+1} = \frac{\phi_1^{k+1}}{2} + \frac{\phi_3^k}{2}$$

Similarly, these terms,

$$\frac{\phi_2^k}{8} + \frac{\phi_3^k}{4}$$

In the third equation can be replaced by

$$\frac{\phi_2^{k+1}}{2}$$

From the second equation resulting in the third equation as

$$\phi_3^{k+1} = \frac{\phi_2^{k+1}}{2} + \frac{1}{2}$$

which is what we expected…

We should now be convinced that the Gauss-Seidel iteration matrix is given as:

$$(L + D)^{-1}U$$

For our case above with 3 unknowns it may be expressed as:

$$\begin{bmatrix} 0 & -1/2 & 0 \\ 0 & -1/4 & -1/2 \\ 0 & -1/8 & -1/4 \end{bmatrix}$$

The eigenvalues for this matrix are found as:

$$\lambda_1 = 0$$

$$\lambda_2 = 0$$

$$\lambda_3 = -1/2$$

Consequently, the spectral radius is 1/2, which is smaller than the value of $1/\sqrt{2}$ for the Jacobi method. Hence, we expect the Gauss-Seidel method to converge in fewer iterations. In fact, we would expect two iterations using Jacobi to reduce the error to that using one iteration of Gauss-Seidel.

So why do we care about convergence rates? We see for these iterative methods we can expect that fine grids will result in slow convergence rates. Consequently, they are not ideal for large systems of unknowns that might arise from, say, a large computational fluid dynamics problem.

Next, we take a closer look at how Gauss-Seidel or Jacobi methods reduce errors during the iteration process.

Error Reduction Using Jacobi and Gauss-Seidel Methods

Let's look at the finite-difference solution to the one-dimensional Laplace equation we discussed previously.

Again, the governing equation is:

$$\frac{d^2\phi}{dx^2} = 0$$

Which is discretized using a second order accurate central finite difference approximation as:

$$\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} = 0$$

Hence our iterative equation is simply given as:

$$\phi_i = 0.5(\phi_{i-1} + \phi_{i+1})$$

We need to apply boundary conditions. Let's make things easy and set the conditions as:

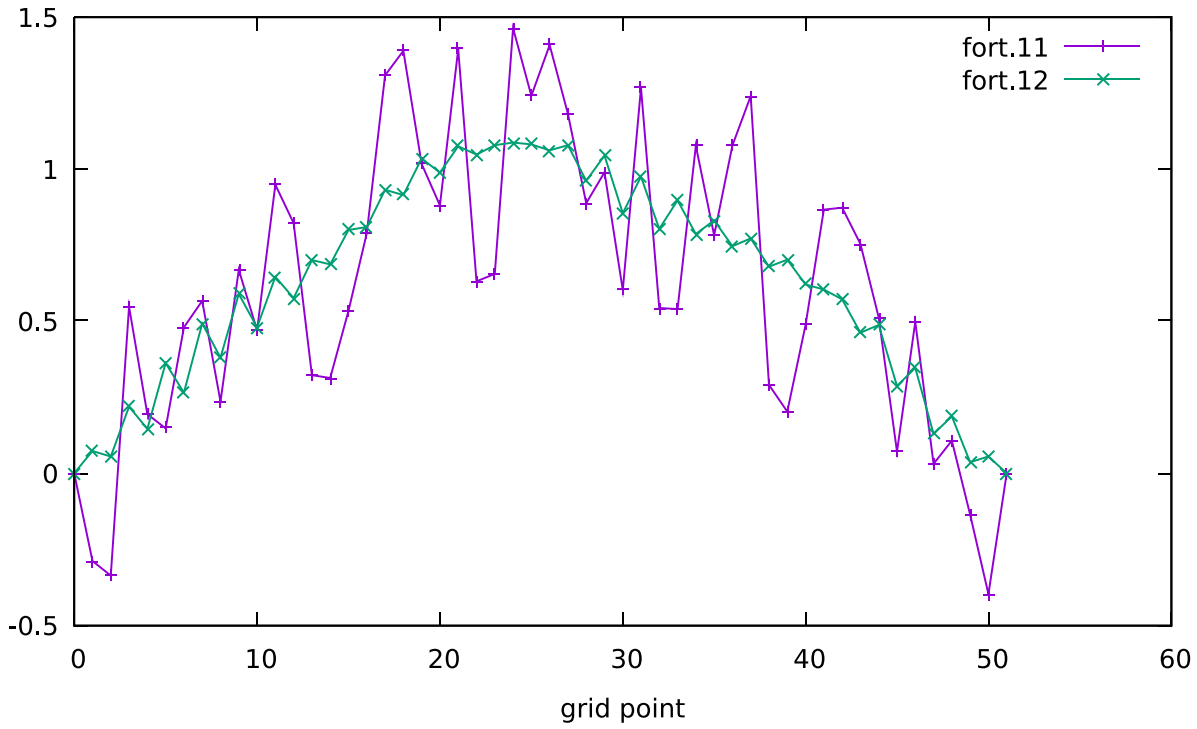
$$\phi(0) = 0, \quad \phi(1) = 0$$

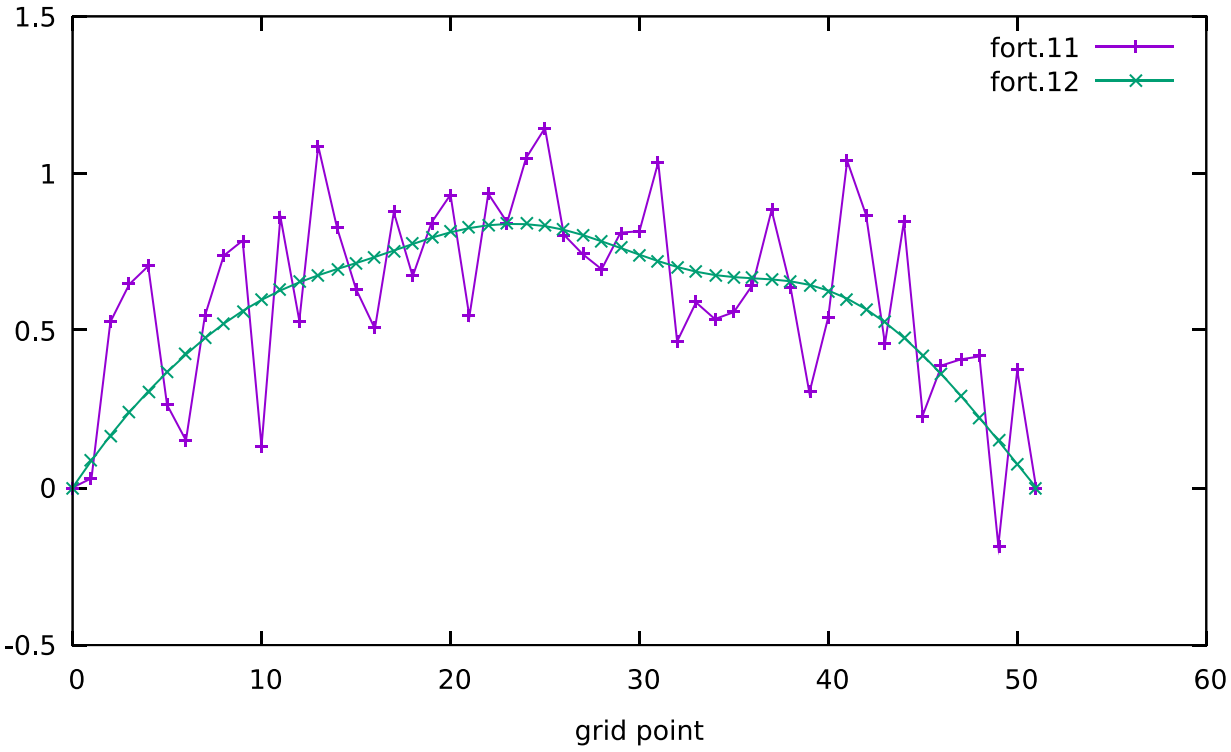
Hence, we know the exact solution is ϕ(x)=0. Seems like a trivial case and it is, but to solve numerically we also need to impose initial conditions. Consequently, the rate at which our iterative method reduces our initial guess to zero indicates the convergence rate.

We will superimpose results from a random number generator onto a sine wave to represent our initial guess (which is all error).

We want to see how quickly Jacobi and Gauss-Seidel iterative methods are able to reduce these errors.

The first plot below includes the initial condition (fort.11) and Jacobi results (fort.12) after 10 iterations with n=50. The second plot is the Gauss-Seidel result.

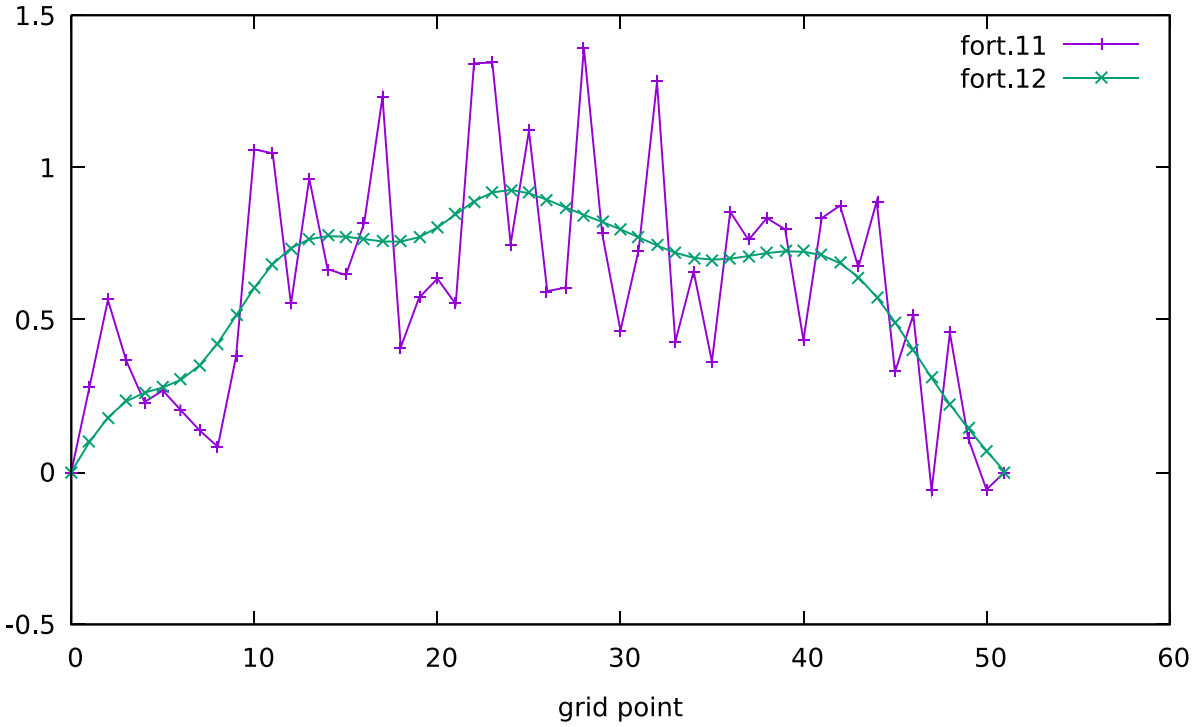




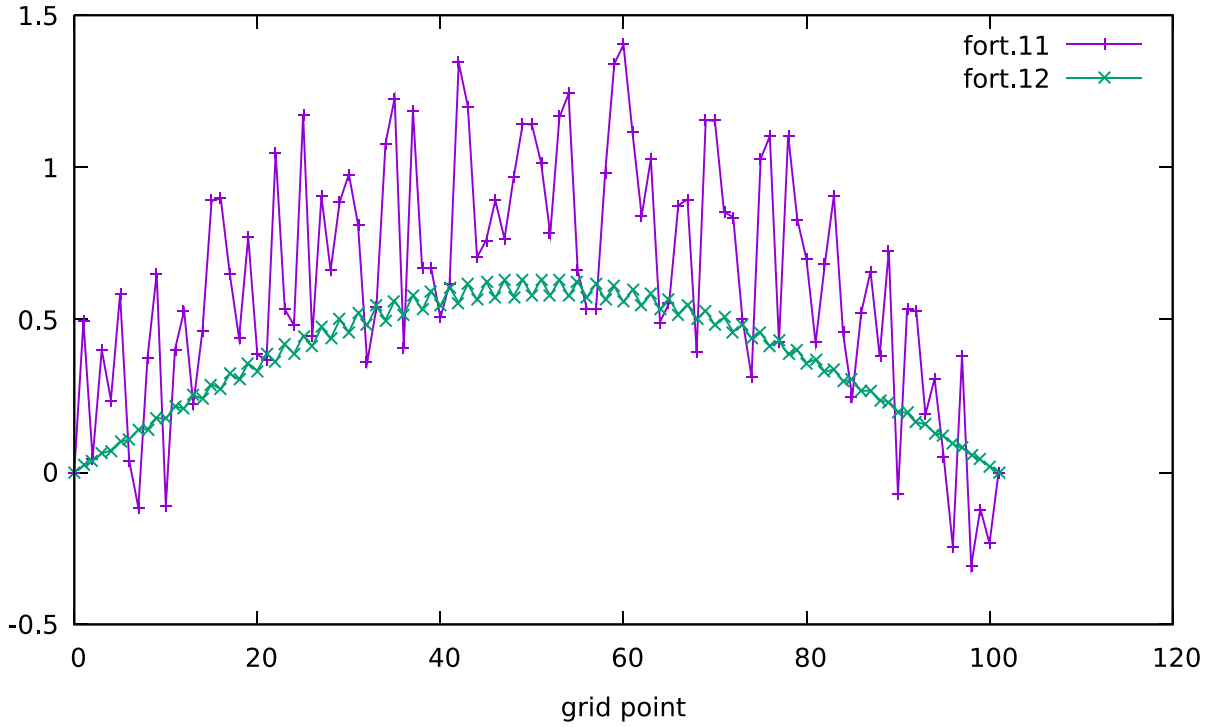
Quite a significant difference in terms of smoothing the high frequencies. We can improve on high frequency response for the Jacobi method using underrelaxation of the form:

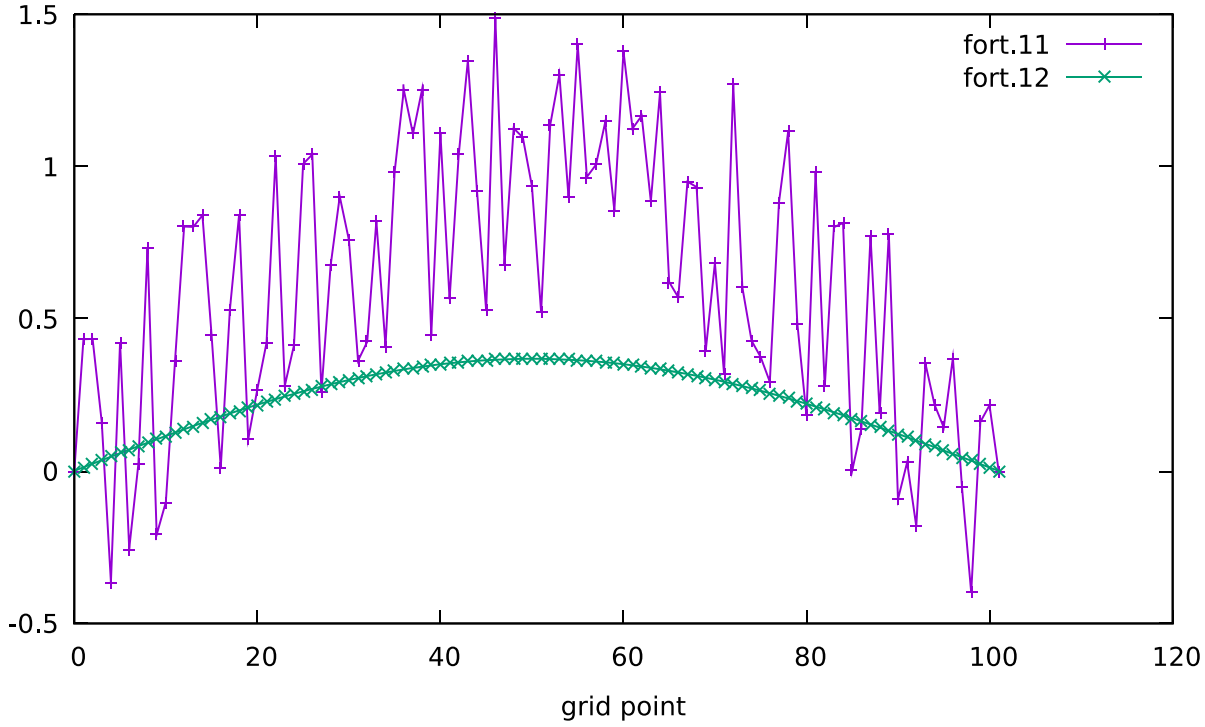
$$\phi^{k+1} = \omega\phi^{k+1} + (1-\omega)\phi^{k}$$

If we take ω=2/3, after 10 iterations with n=50 we find the following.



Finally, we show Jacobi and Gauss-Seidel results, respectively, after 1000 iterations.


fort.11
fort.12
1.5
1
0.5
0
-0.5
0
20
40
60
80
100
120
grid point


fort.11
fort.12
1.5
1
0.5
0
-0.5
0
20
40
60
80
100
120
grid point

What can we conclude?

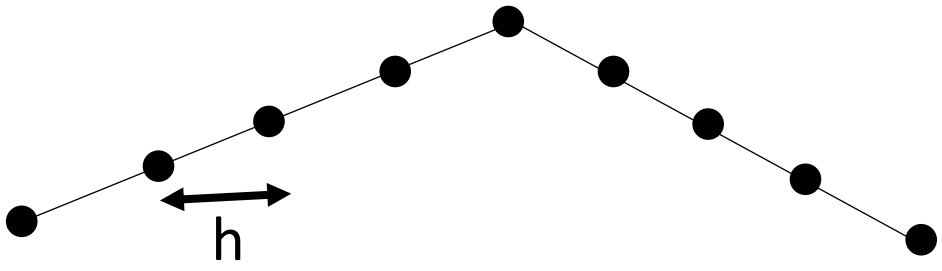
- Jacobi is not very effective at removing high frequency error components.
- Gauss-Seidel does a good job of removing high frequency error components.
- If we under-relax Jacobi, it does a good job of removing high frequency error components.
- Both methods do a poor job of reducing low frequency errors.
- The rate of low frequency error reduction for the under-relaxed Jacobi method is slower than that of Gauss-Seidel.
- Consequently, these methods can be slow to converge on fine meshes.

We will next look at multi-grid basics, and how the error reduction properties of the Gauss-Seidel or under-relaxed Jacobi methods can be exploited to dramatically accelerate convergence.

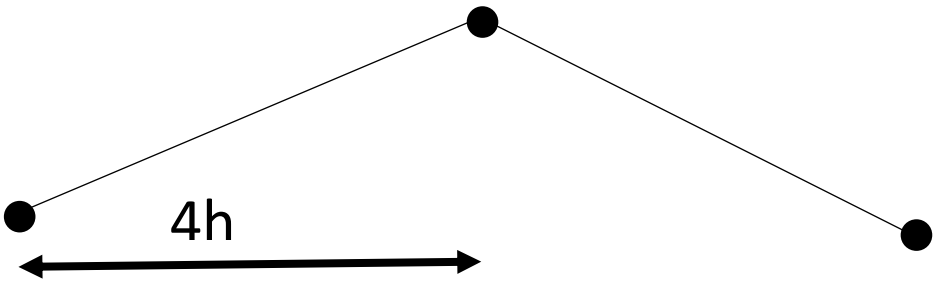
Note that multigrid requires a good "smoother" as the iterative method.

Multigrid Basics

In short, the idea behind multigrid is that what appears as a low frequency error on a fine grid will look like high frequency error on a coarse grid.



low frequency error



high frequency error

Multigrid takes advantage of the properties of iterative “smoothers” like Gauss-Seidel by cycling through different grid levels to eliminate what appear as high frequency errors on each level.

Geometric Multigrid – Typically used when the coefficient matrix A arises from the discretization of a PDE. The individual coefficients are recomputed as necessary based on the grid spacing at different multigrid levels.

Algebraic Multigrid – A more general procedure where information about the development of the coefficient matrix A, or the discretization process, is unavailable. In this case the coefficients on coarser grids are approximated as combinations of those on finer grids.

General Procedure

We wish to solve the system:

$$A\boldsymbol{x} = \boldsymbol{b} \qquad (1)$$

Where A is the coefficient matrix, **b** is a right-hand side vector, and **x** is a vector representing the unknowns. In this equation, **x** is the true solution to the system of equations.

Now consider an iterative solution to this equation, at some intermediate number of iterations we have an approximate solution, call it **y**. The vector **y** will satisfy exactly an equation given by:

$$A\boldsymbol{y} = \boldsymbol{b} - \boldsymbol{r} \qquad (2)$$

where **r** is a "residual" vector.

Now we can define an error vector (**e**) as the difference between the actual solution (**x**) and the approximate solution (**y**) as:

$$\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{y} \qquad (3)$$

We can develop a system that can be solved for the error vector itself by subtracting the system for **y** from the system for **x**:

$$A(\boldsymbol{x}-\boldsymbol{y})=\boldsymbol{b}-(\boldsymbol{b}-\boldsymbol{r}) \quad (4)$$

or

$$A\boldsymbol{e}=\boldsymbol{r} \quad (5)$$

To solve for **e** at any point during the iteration procedure we need the residual vector, which can be obtained directly from Eq. 2:

$$\boldsymbol{r}=\boldsymbol{b}-A\boldsymbol{y} \quad (6)$$

Below we outline a procedure for a <u>two-stage</u> multi-grid process. In other words, we use two meshes, say the fine mesh with n points, and a coarse mesh with n/2 points. The idea is that low frequency errors on the fine mesh will look like high frequency errors on a coarse mesh.

1) Perform (a small number of) iterations on finest mesh with spacing h to remove high frequency errors. This then lets us compute the residual vector using Eq. 6.

2) Transfer the residual vector (Eq. 6) onto the coarse mesh and use Eq. 5 to solve for the error vector on the coarse mesh.

3) Use linear interpolation to transfer the error vector back to the fine mesh.

4) Correct our initial, intermediate result for the solution vector using $\boldsymbol{y}^{improved}=\boldsymbol{y}^{previous}+\boldsymbol{e}$

5) Carry out a few more iterations on the improved solution vector.

Prototype Problem

We are going to solve the problem

$$\frac{d^2\phi}{dx^2} = 0$$

$$\phi(0) = 0, \quad \phi(1) = 0$$

We use a second order accurate central finite difference approximation to find:

$$\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} = 0$$

Our Gauss-Seidel iterative equation is then given as:

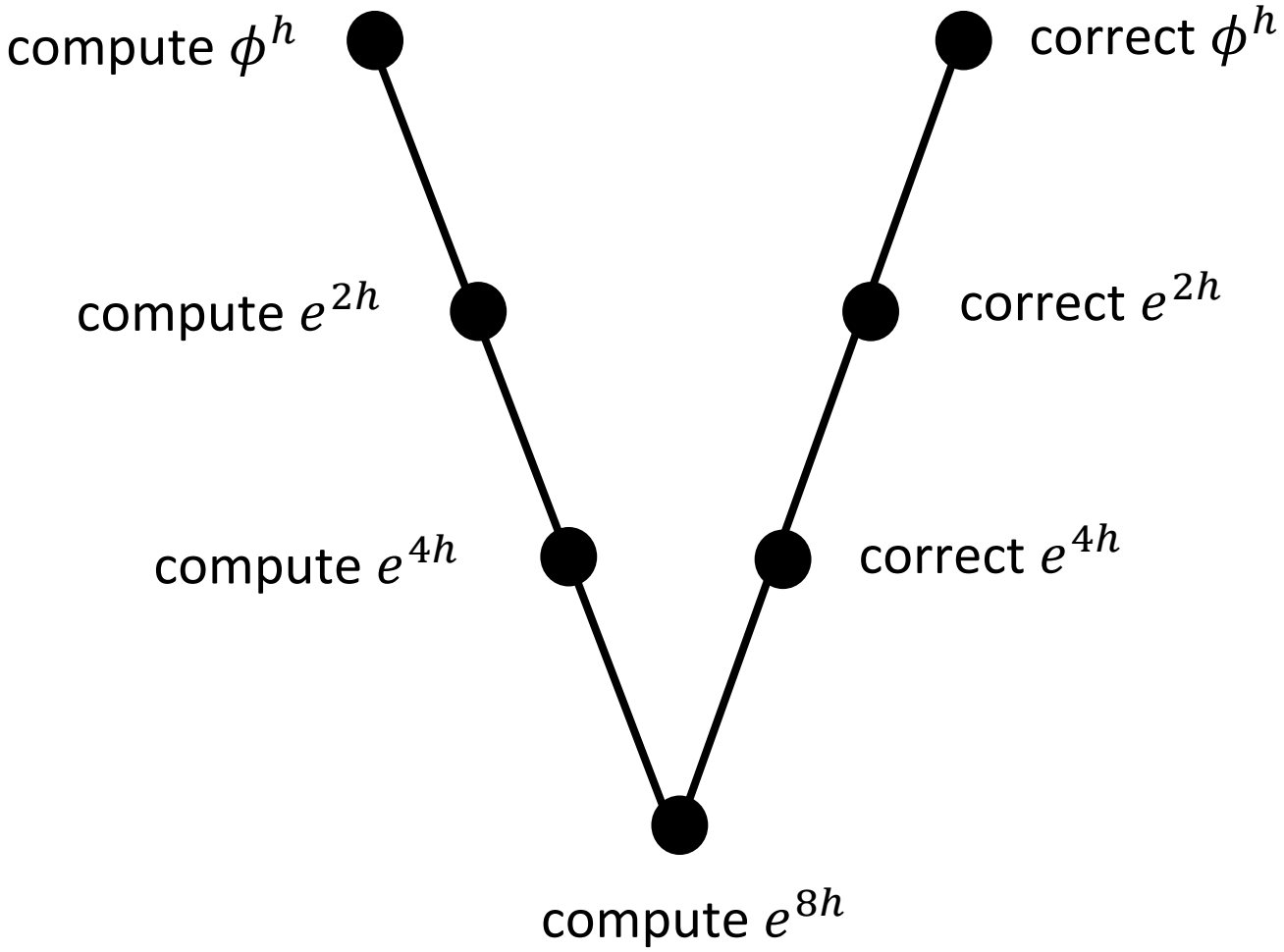
$$\phi_i^{k+1} = 0.5(\phi_{i-1}^{k+1} + \phi_{i+1}^{k})$$

$$\phi(0) = 0, \quad \phi(1) = 0$$

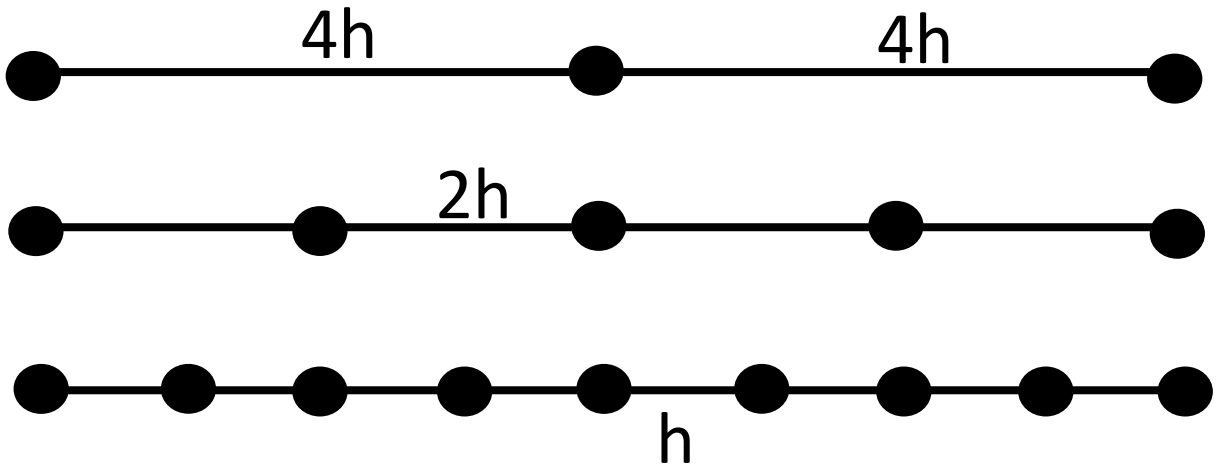
(Note we cleared the denominator in our discretized equation. Had we not, our coefficient matrix A would indeed have grid spacing information included in the coefficients. Or, had the right side been some value other than zero, we would have grid information in the right-side vector.)

Why this problem? Because we know the solution is $\phi(x) = 0$ and hence we can simply observe how quickly our multigrid method reduces the initial guess for ϕ to zero.

Representation of a V-cycle levels that we will use.



Our grid generation approach for a finite difference discretization:



The end points are boundary conditions.

We halve the grid spacing going from the coarsest mesh to finer meshes.

Consequently, using the above, where n=3 (grid points) at the coarsest level, our possible grid points must follow the pattern:

3, 5, 9, 17, 33, 65, 129, 257, … etc.

Or, the relation

n = 2**level +1 or

level = log2(n-1)

- To define grids, start with coarse mesh and sequentially half the grid spacing until the desired fine grid mesh is reached.

- This then fixes the maximum number of levels we may utilize.

- Next, we will look at a set of results for our prototype problem in one-dimension.

- After that, we will look at a 2D problem, then go over the code(s).

One-Dimensional Prototype Problem Results

Multigrid parameters:

65 or 1025 grid points at finest level

6 or 10 multigrid levels

10 gauss-seidel iterations per level going down

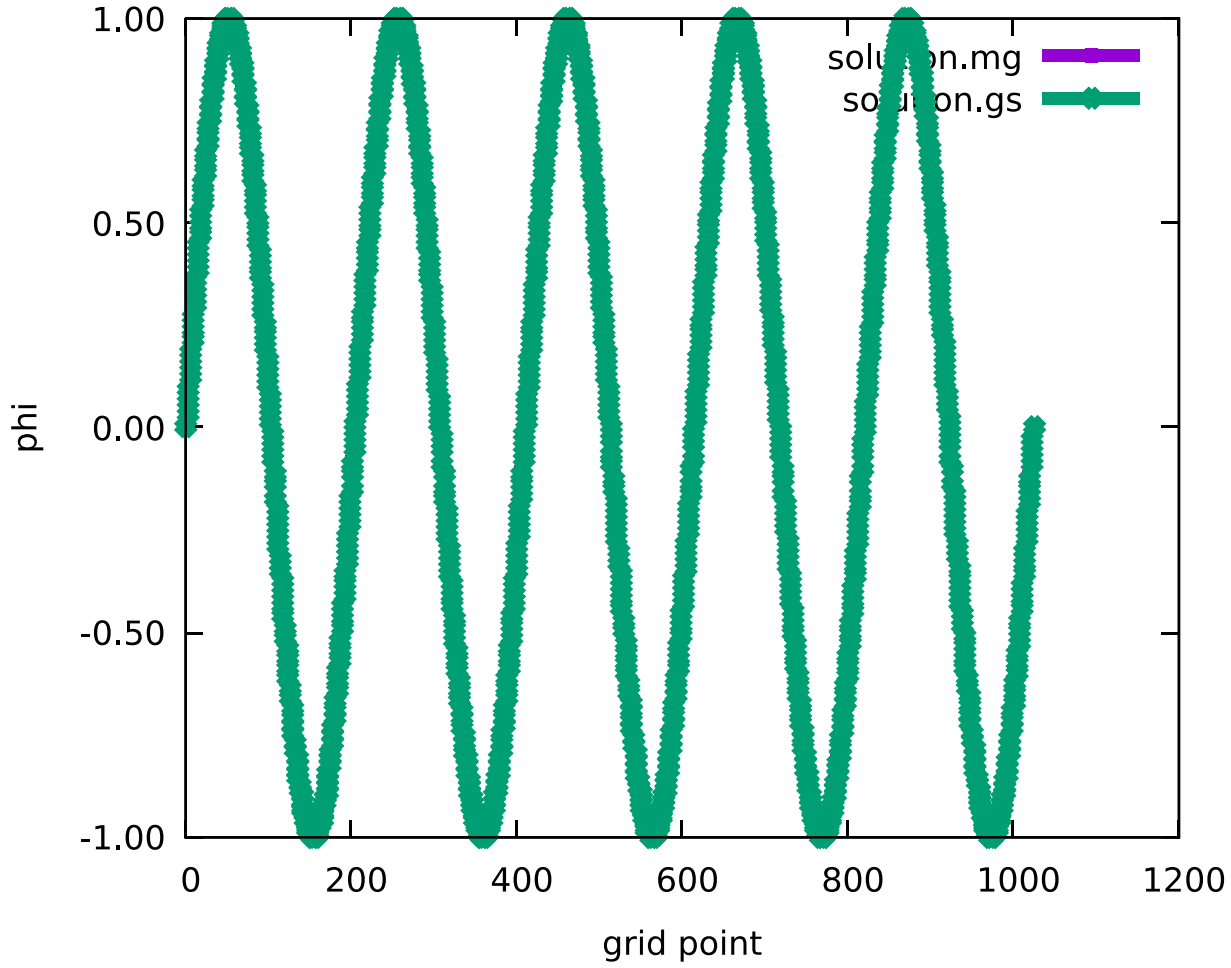
3 gauss-seidel iterations per level going up

Gauss-Seidel parameters (using the multigrid code):

1025 grid points

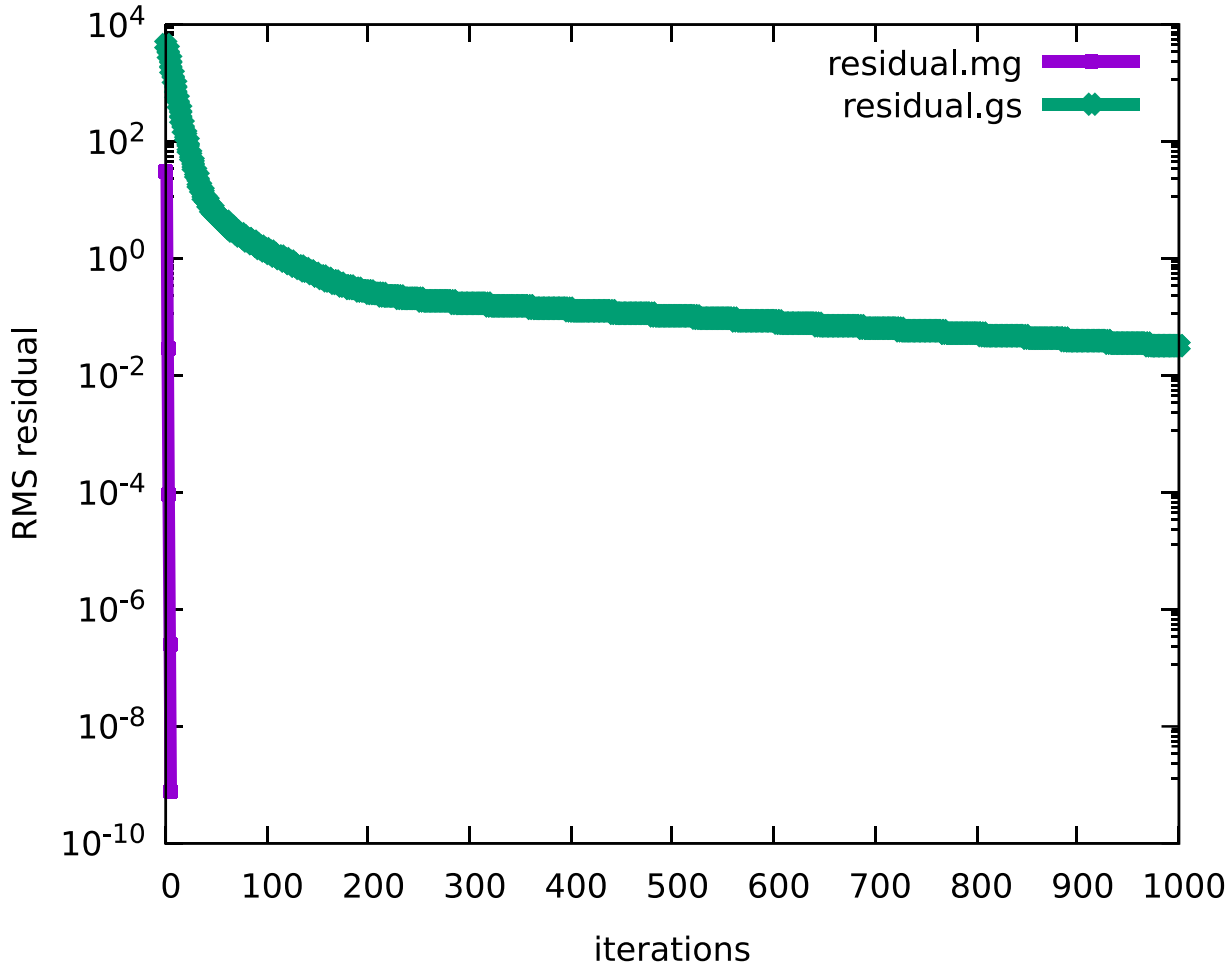
1 multigrid level (i.e., fine grid only)

1000 outer iterations

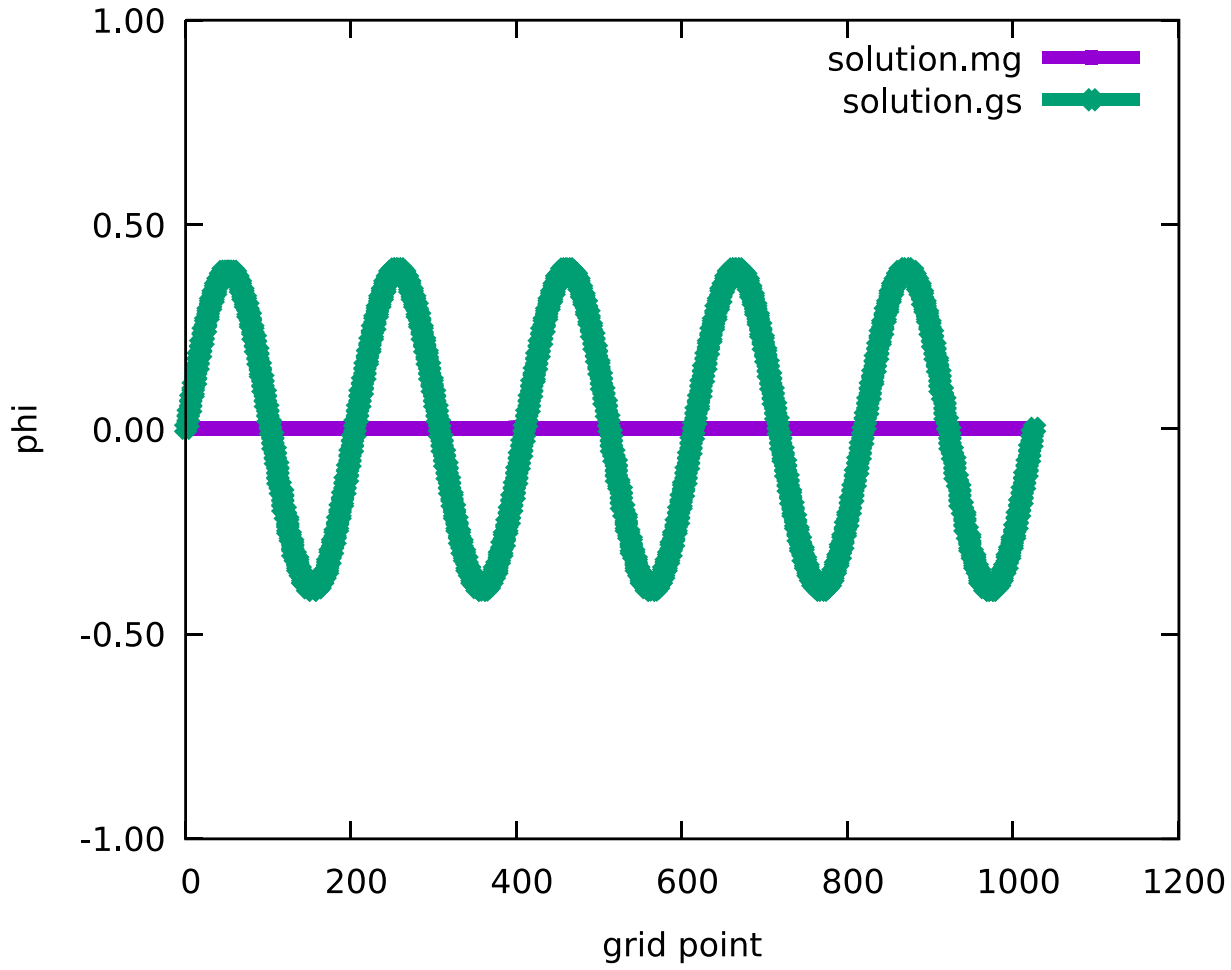


INITIAL GUESS FOR SOLUTION

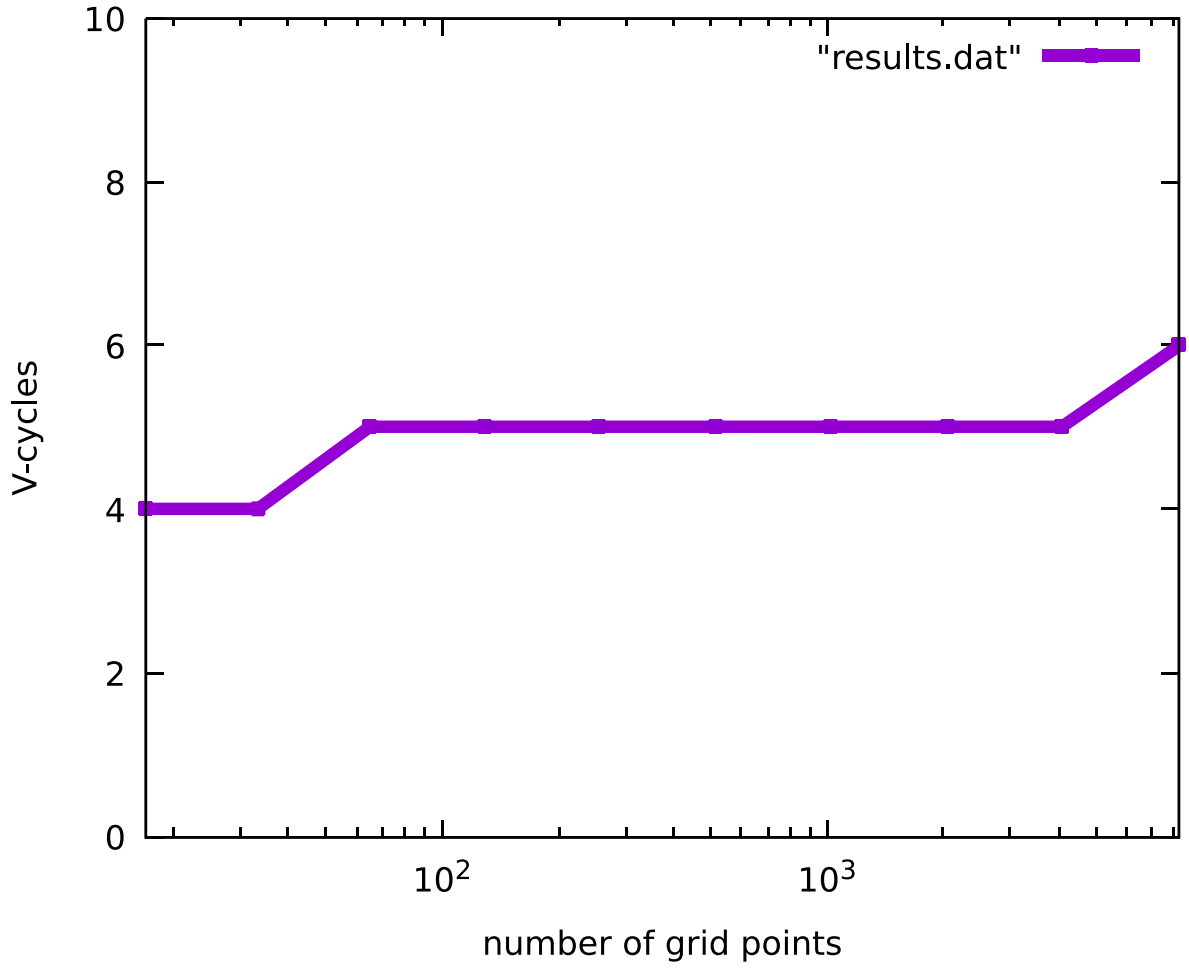
(i.e., initial error distribution)



RMS residuals for N=65 grid points.

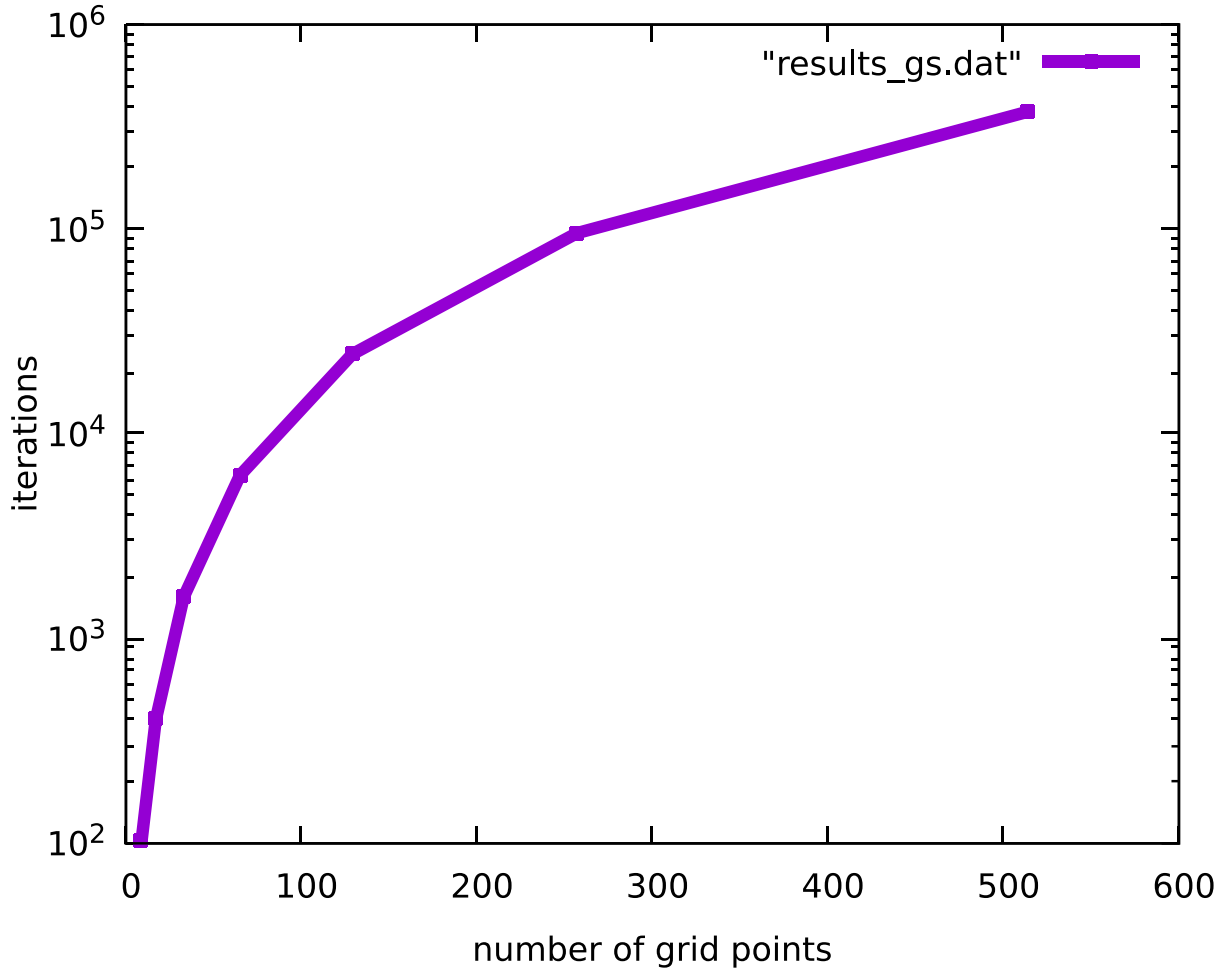


Comparing solutions for both multigrid (converged after 5 cycles) and Gauss-Seidel (after 1000 iterations) on a grid with 1025 points.



Number of multigrid cycles to drive residuals below 10^-7 as a function of number of grid points.

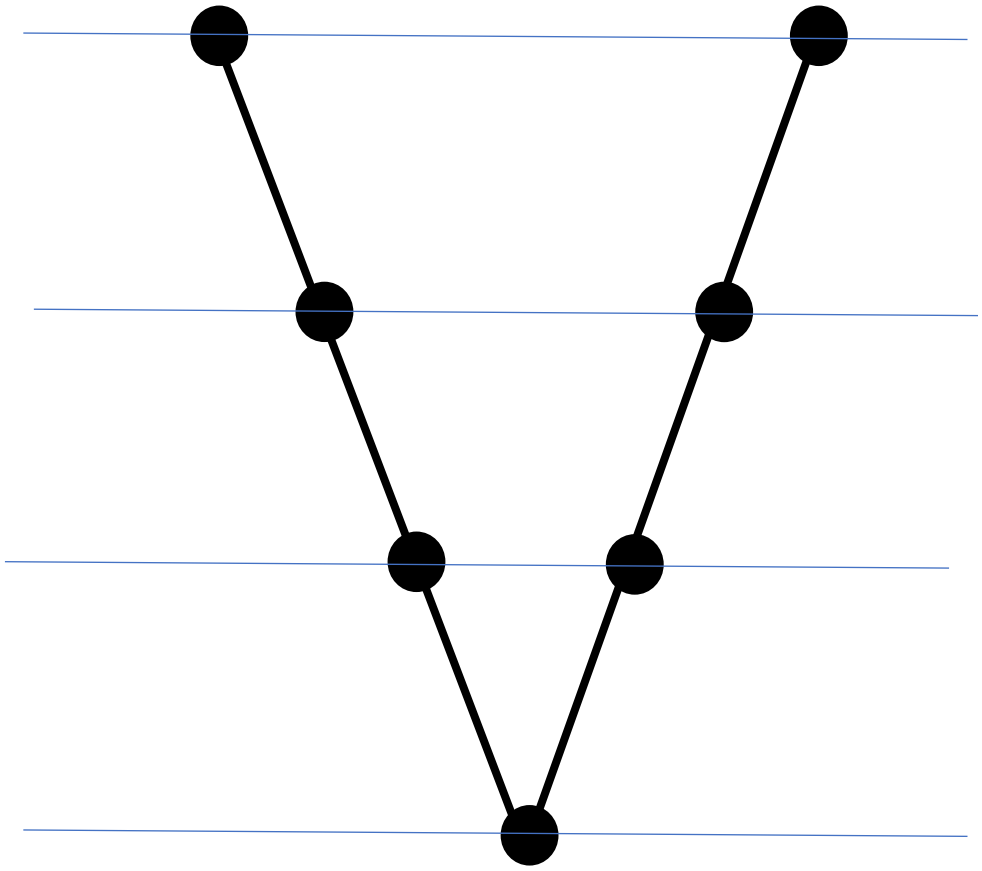
- Number of iterations nearly independent of grid size.
- Hence, cost is approximately proportional to number of grid points N – the best one can expect.



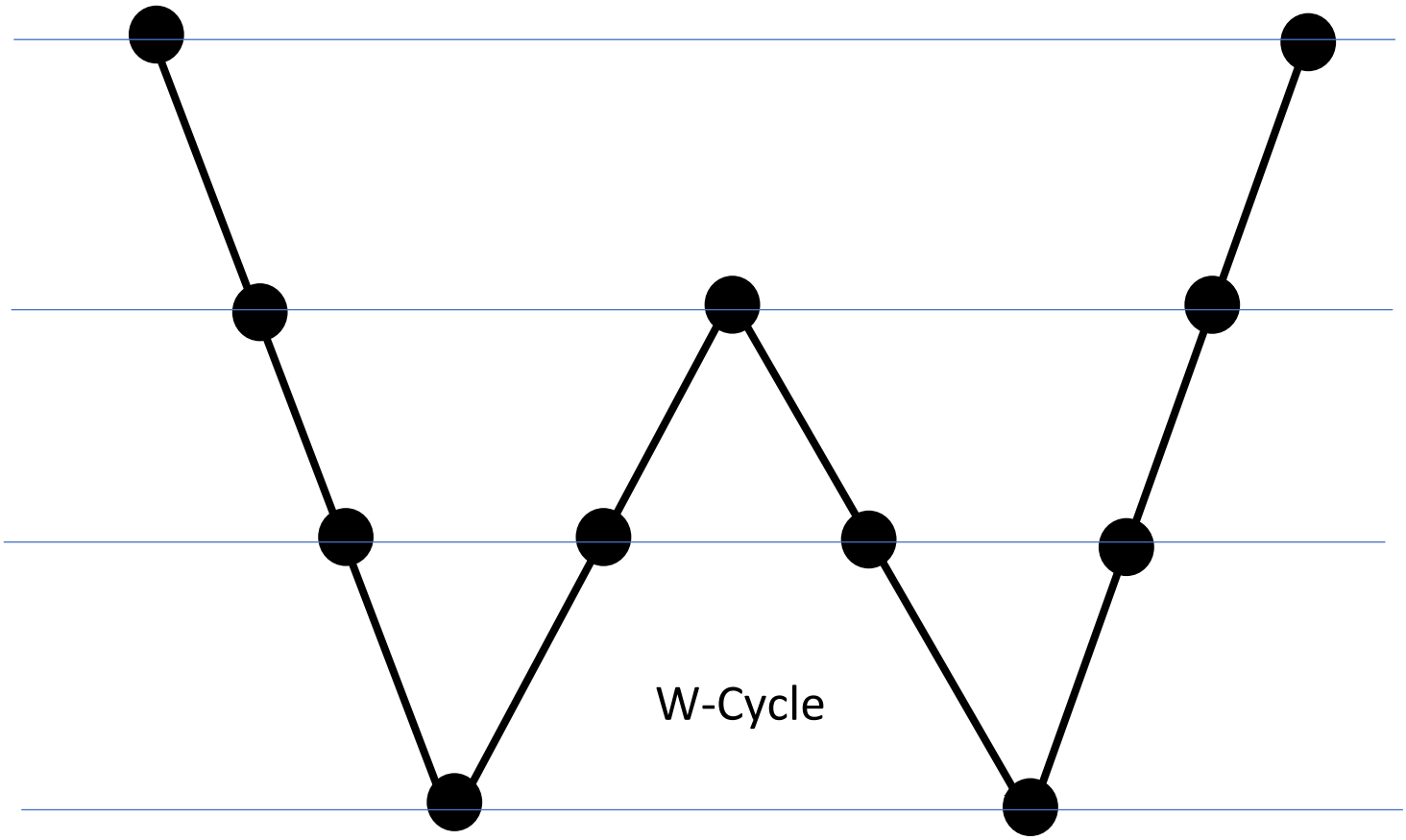
Number of Gauss-Seidel iterations to drive residuals below 10^-7 as function of number of grid points.

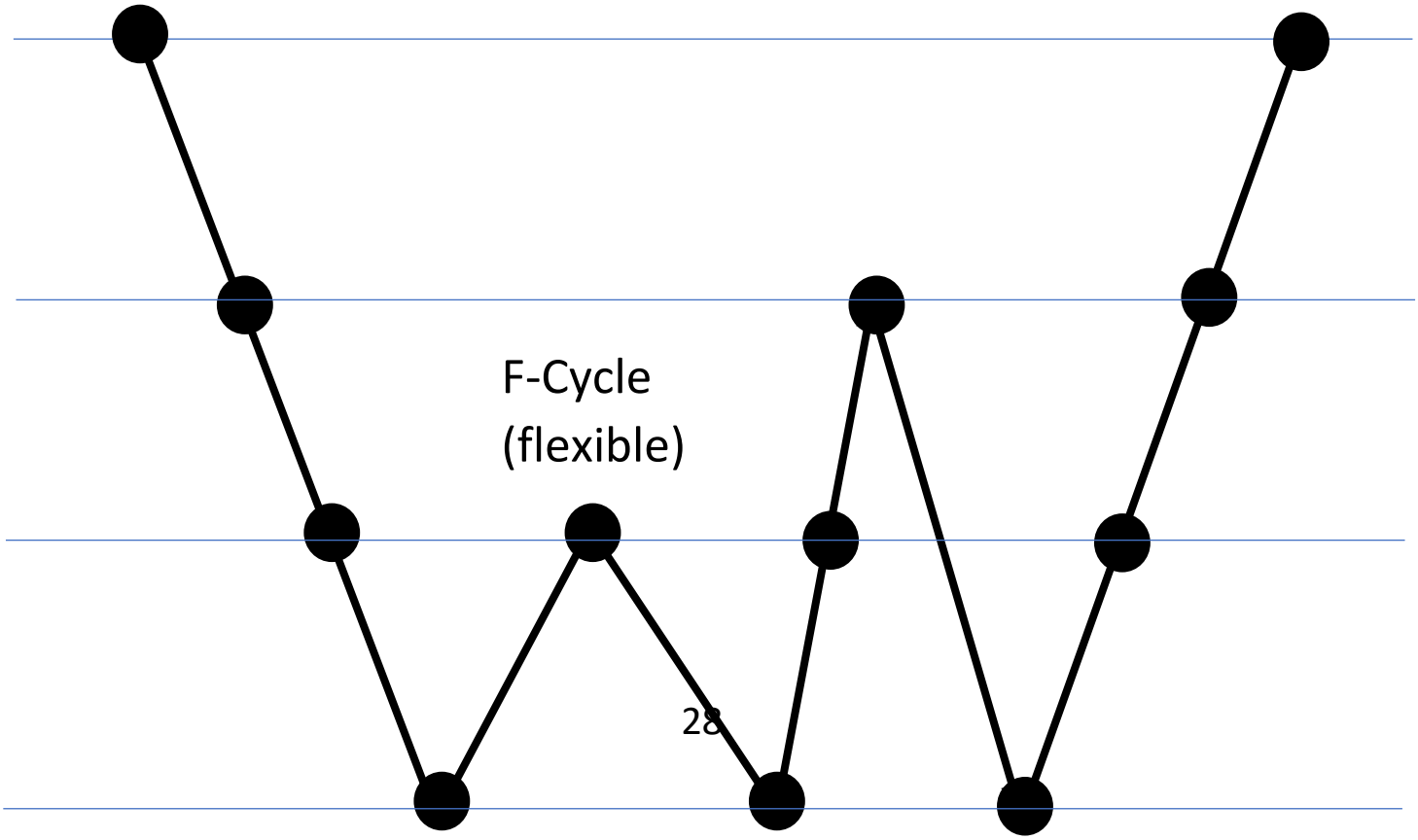
- Number of iterations increases as ~N^2 (i.e., double # grid points means 4X as many iterations).
- Hence cost is proportional to N^3.

Multigrid Cycle Options and Terminology



V-Cycle


W-Cycle

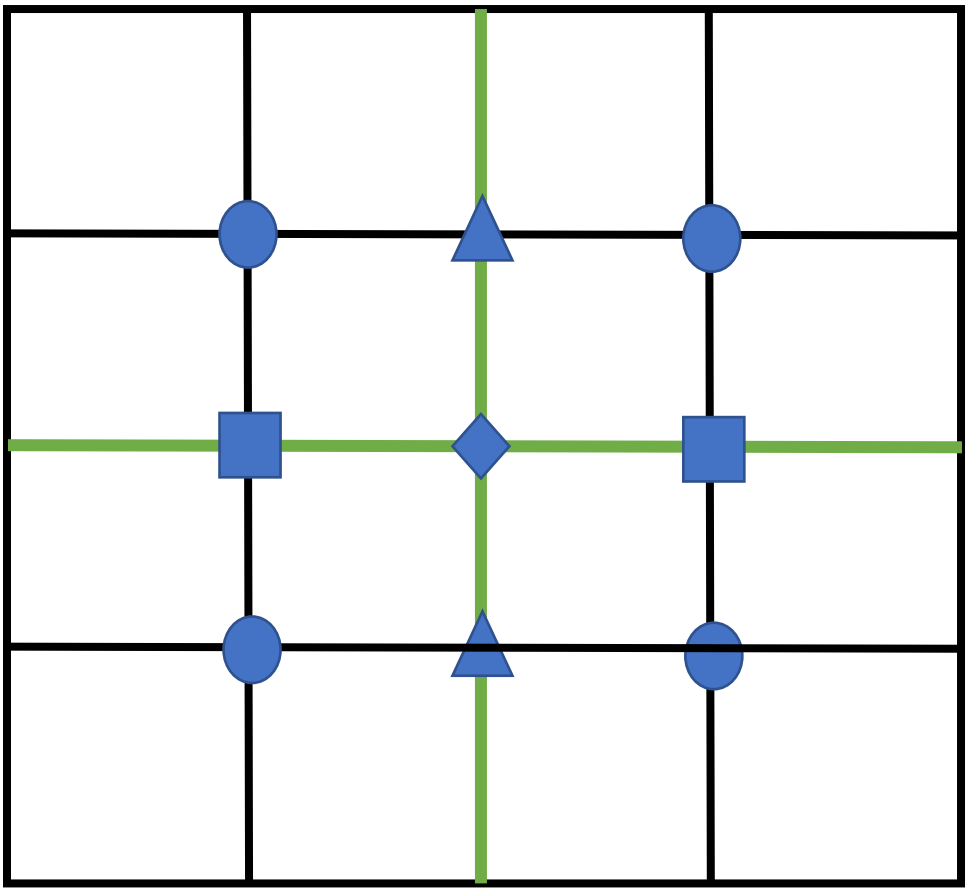

F-Cycle
(flexible)

- W- and F-Cycles use additional restriction (going down) and prolongation (going up) sweeps at coarser levels. This helps remove long wavelength errors.

Other Options and Choices to be Made:

- Schedules for restriction and prolongation – different numbers of solution iterations on each level.
- A few smoothing iterations on the way back up.
- Different iterative methods for the smoother (i.e., Gauss-Seidel, underrelaxed Jacobi, etc.).
- Interpolation schemes.

Two-Dimensional Problem on a Square Domain



Averaging process on interpolating errors from (green) coarse to (black) fine mesh in prolongation step:

Circle – average 4 surrounding points from coarse mesh
Square – average 2 horizontal points from coarse mesh
Triangle – average 2 vertical points from coarse mesh
Diamond – direct transfer, no averaging

Problem Statement

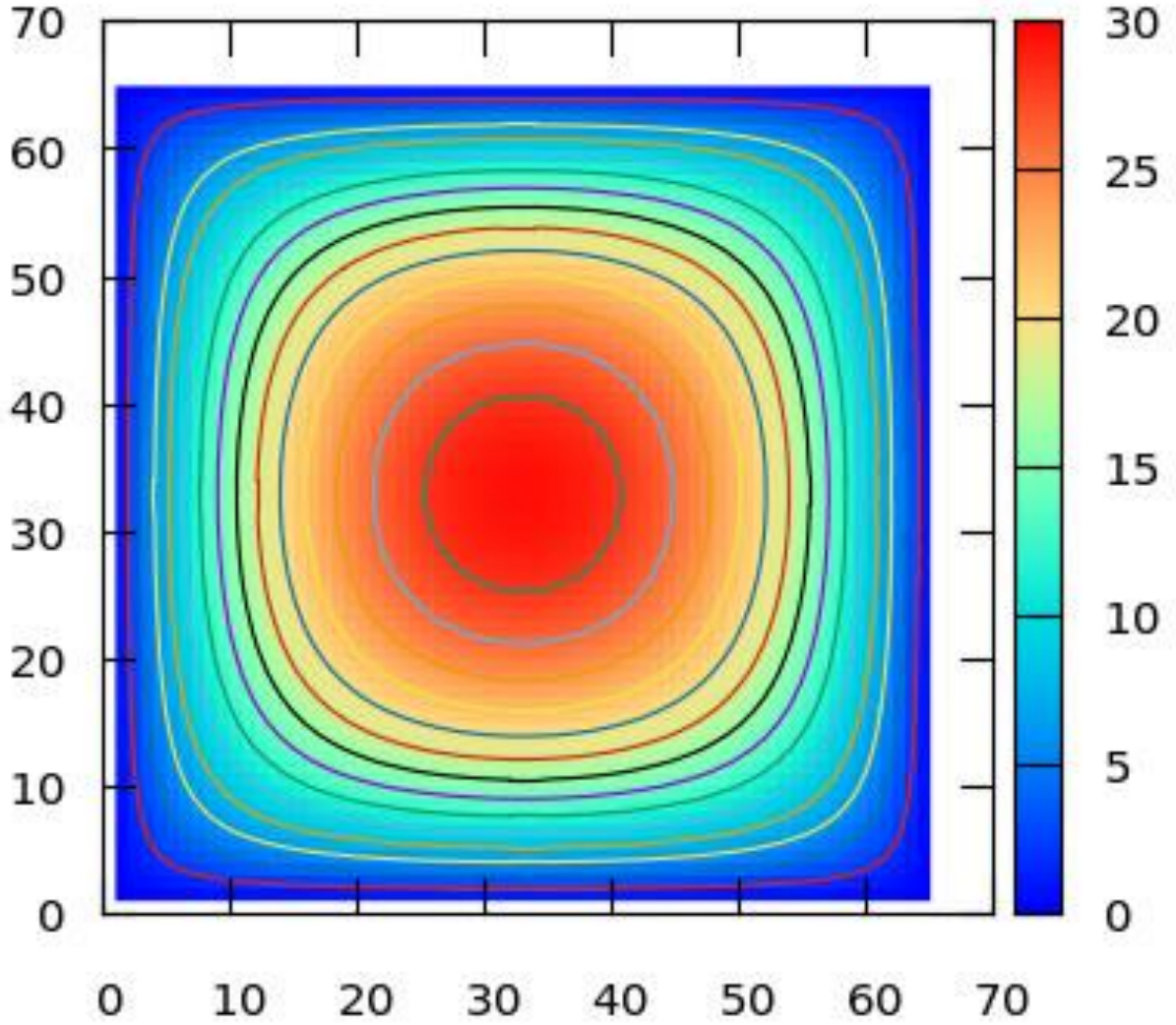
$$\nabla^2\phi + 100 = 0$$

$$\phi(-1, y) = \phi(1, y) = \phi(x, -1) = \phi(x, 1) = 0$$

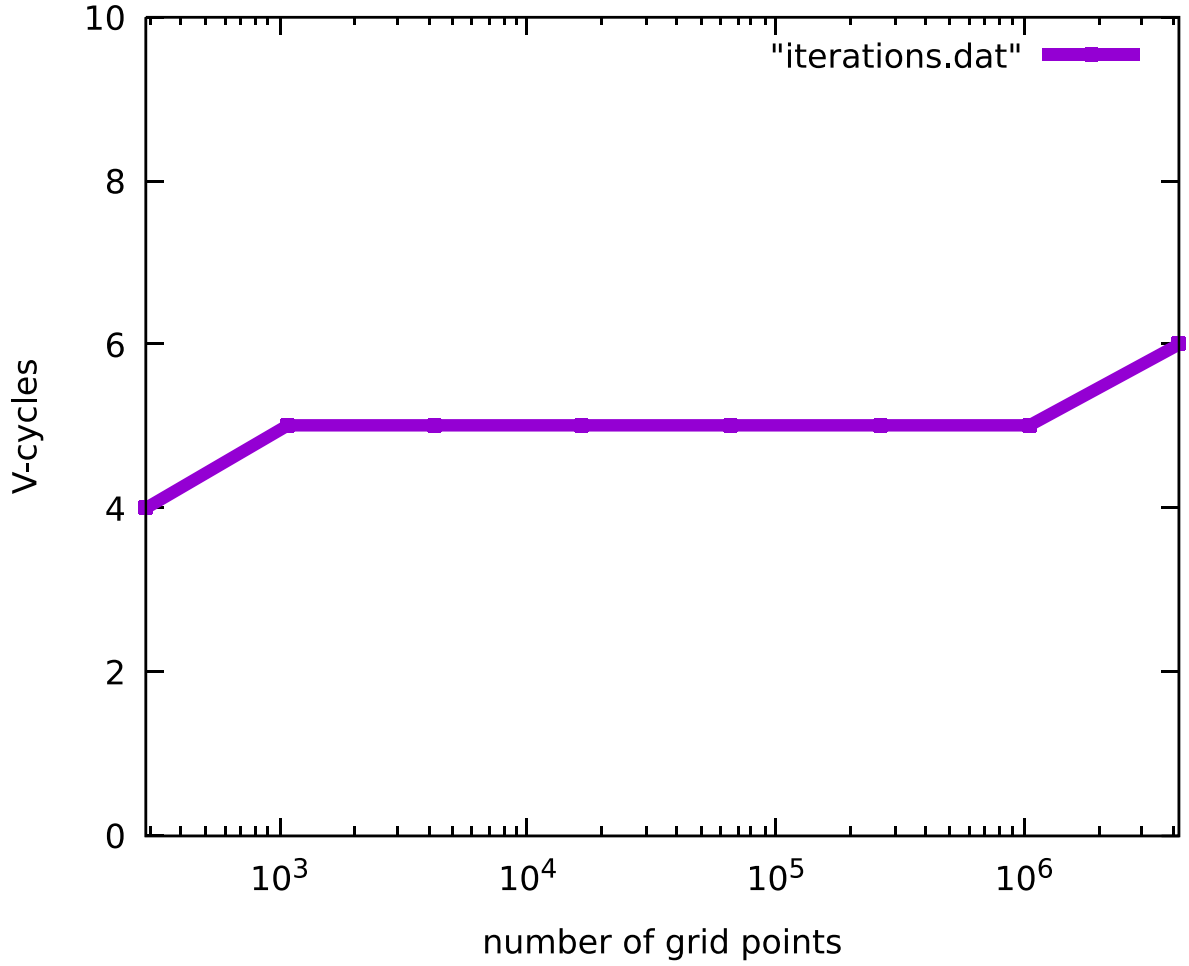
$$-1 \leq x \leq 1 \qquad -1 \leq y \leq 1$$

A separation of variables solution results in:

$$\phi(0,0) = 29.4685 \ldots$$



Results for distribution of ϕ on a 65x65 mesh.



Number of multigrid V-cycles to reduce residuals to below 10^-5 based on initial guess $\phi=0$, with 10 Gauss-Seidel iterations per level.

- Again, number of iterations nearly independent of number of grid points, N^2.

Next, we look at the 1D and 2D codes.

Summary

- We have solved a system of the form Ax=b that resulted from a finite-difference discretization to a Laplace or Poisson equation where the number of unknowns is too large for a direct solution.
- Iterative methods converge very slowly on fine grids but quickly on coarse grids. (Think of the problem as one of transmitting boundary information throughout the domain.)
- The multigrid method takes advantage of the fact that low frequency errors on a fine mesh become high frequency errors on a coarse mesh.
- Multigrid requires an iterative method that rapidly eliminates high frequency errors. (The speed at which it reduces low frequency errors is much less important.)
- Gauss-Seidel satisfies this requirement. Jacobi does not, but under-relaxed Jacobi does.
- Using multigrid can result in one or more orders of magnitude reduction in the number of iterations required to drive down residuals to a given level, relative to typical point iteration methods alone.

- Rate of convergence is nearly independent of the number of grid points. Hence computational cost is proportional to the number of grid points.
- Most commercial computational fluid dynamics solvers utilize multigrid methods to accelerate convergence, given that the number of unknowns can often be in the tens or hundreds of millions.

Basic choices to be made implementing the method include:

1) The specification of the different grid levels.
2) The type of cycle (i.e., V, W, F) or more generally the order at which each level is visited.
3) Number of "smoothing" iterations at each multigrid level both moving down and moving up.
4) The specific restriction and interpolation schemes used to move data between levels.
5) The smoother itself (i.e., Gauss-Seidel, etc.).

Fortunately, one only needs make "reasonable" choices in the above to achieve excellent performance.