An Introduction To Geometric Multigrid

Presented to TTU Math 5344 - Fall 2020

Nicholas Moore

Goal

- We follow the analysis from “A Multigrid Tutorial” by William Briggs
- Focus on Multigrid as solver for 1D Poisson problem
- Concepts and ideas can be applied to higher dimensions or used as a preconditioner

1D Poisson

- For simplicity, conduct experiments on the 1D Poisson problem
- Dirichlet boundary conditions
- Divide interval $(0, 1)$ into N sub-intervals
- Using Central Differences,

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

- $h = \frac{1}{N}$ and A is $N - 1 \times N - 1$

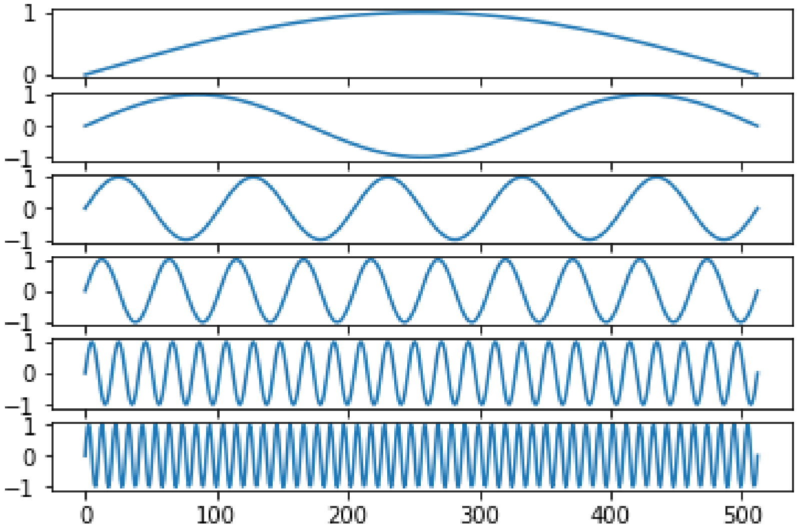
Analyzing the Jacobi Method

- For this analysis, we use $b = 0$
- True solution is 0 so the error and the current iterate are the same
- Generate initial iterates corresponding to sine waves with varying frequencies:

$$x_0 = \sin(w\pi x) \quad \text{for } w \in \{1, 3, 10, 20, 50, 100\}$$

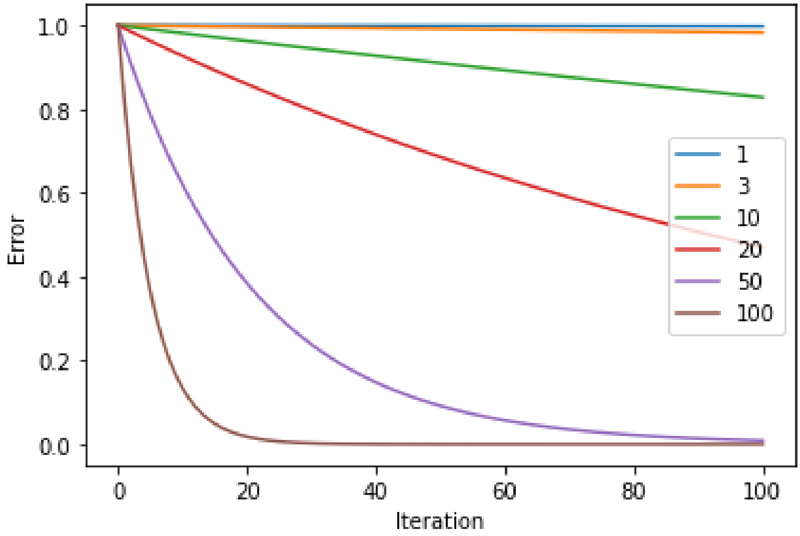
- Notice these initial iterates are also the “initial errors”

Initial Iterates



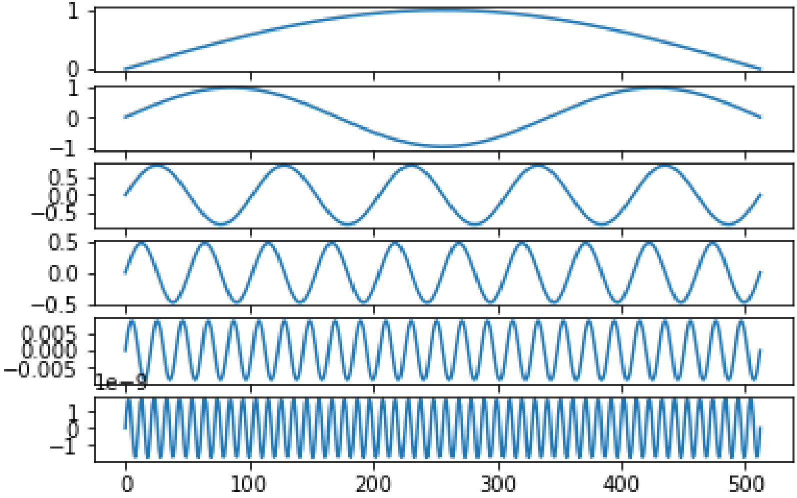
Analysis of Jacobi

Now let's run 100 Jacobi iterations on each of the initial conditions, tracking the error at each iteration.



Analysis of Jacobi

We can also look at our iterates:

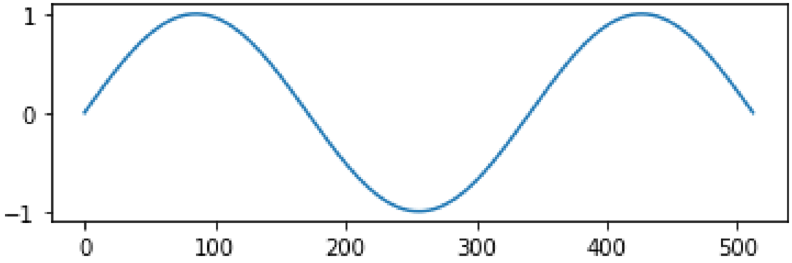


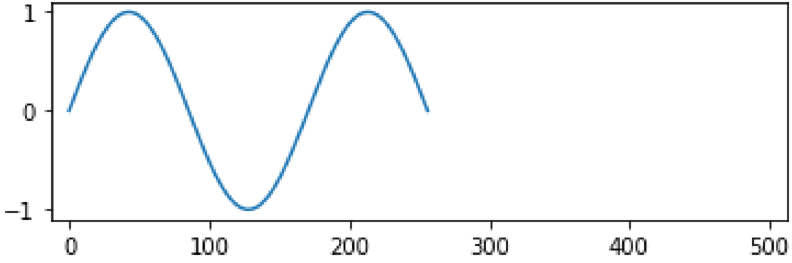
Why Multigrid Works

Here we see the key to understanding the effectiveness of multigrid: Jacobi iteration is much better at eliminating high frequency error than low frequency error.

How Do We Use This?

Notice what happens if we start with a low frequency wave and approximate it with half as many points:





How Do We Use This?

- The same number of waves now fit into half the length
- Jacobi iteration is not aware of the physical structure of the problem
- For Jacobi, the shorter vector has higher frequency error than the longer vector
- Jacobi iteration could be more effective on this new shorter vector

Multigrid

As the name suggests, the multigrid method is about leveraging grids of different resolutions to take better advantage of Jacobi's convergence properties and computational effort.

Notation

- The Ω^h be the grid on which we wish to have a final approximation.
- Superscripts with interval length will be used to denote the grid a quantity is defined on
- For example A^h, e^h, b^h, r^h are all used for the finest grid
- A^{2h}, e^{2h}, b^{2h}, r^{2h} are all used for quantities on a grid with half the number of subintervals (thereby making their length $2h$)

Basic Multigrid Idea

- Start with k_1 Jacobi iterations (k_1 small)
- We don't expect this iteration x^h to be the exact solution
- Write exact solution with the form $x^* = x^h + e^h$
- Re-write the matrix system

$$\begin{aligned} A^h(x^h + e^h) &= b^h \\ b^h - A^h x^h &= A^h e^h = r^h \end{aligned}$$

- On a coarser grid to solve $A^{2h}e^{2h} = r^{2h}$ and use that solution to estimate e^h
- Update x^h as $x^h \leftarrow x^h + e^h$
- Run another k_2 iterations of Jacobi to smooth the error

Finding e^h

- Want to solve $A^{2h}e^{2h} = r^{2h}$
- A^{2h}, e^{2h}, r^{2h} are "coarse grid versions" of A^h, e^h, r^h.
- Smaller matrix $\Rightarrow$ less computation
- Low frequency error in $e^h \Rightarrow$ higher frequency error in e^{2h}
- Then "interpolate" e^{2h} back up to the finer grid

Moving Between Grids

- In previous slides, I mentioned using the "coarse grid versions" of our quantities.
- We need to discuss how these quantities are obtained
- Assume coarse grid spacing which is twice as large as finer grid
- Almost universal practice - not evidence that any other ratio is more effective

Restriction Operator

- For transforming fine grid vectors into coarse grid vectors
- Denoted as I_h^{2h}
- Multiple Options
    - Could simply remove every other entry
    - More common - **full weighting**:
      Produce coarse grid vectors according to the rule $I_h^{2h} x^h = x^{2h}$ where

$$x_j^{2h} = \frac{1}{4}\left(x_{2j-1}^h + 2x_{2j} + x_{j+1}^h\right)$$

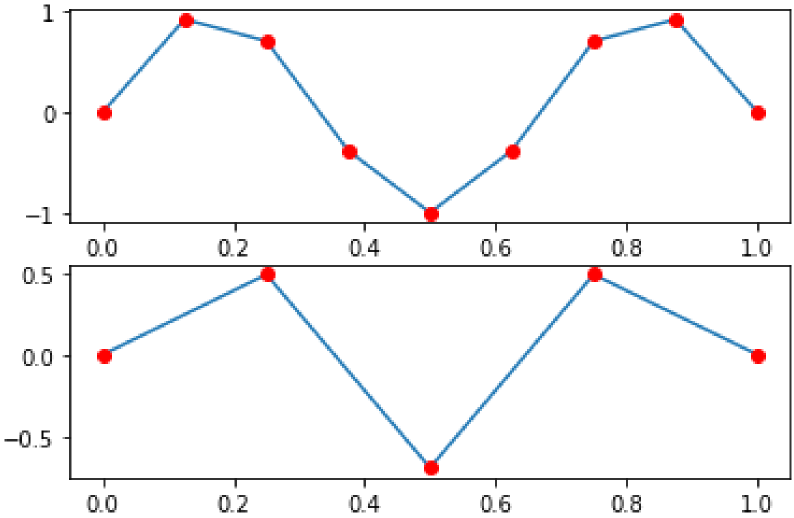
Full Weighting

For example, if we have 7 interior nodes in the fine grid, and 3 interior nodes in the coarse grid, then we have the following:

$$I_h^{2h} x^h = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 & & & & \\ & & 1 & 2 & 1 & & \\ & & & & 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}_h = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{2h} = x^{2h}$$

This opreator also has another advantage that we'll mention later.

Full Weighting



Interpolation Operator

- Also called the **Prolongation** operator
- Transforms coarse grid vectors into finer grid vectors
- Denoted as I_{2h}^{h}
- Rule:

$$x_{2j}^{h} = x_{j}^{2h}$$
$$x_{2j+1}^{h} = \frac{1}{2}\left(x_{j}^{2h} + x_{j+1}^{2h}\right)$$

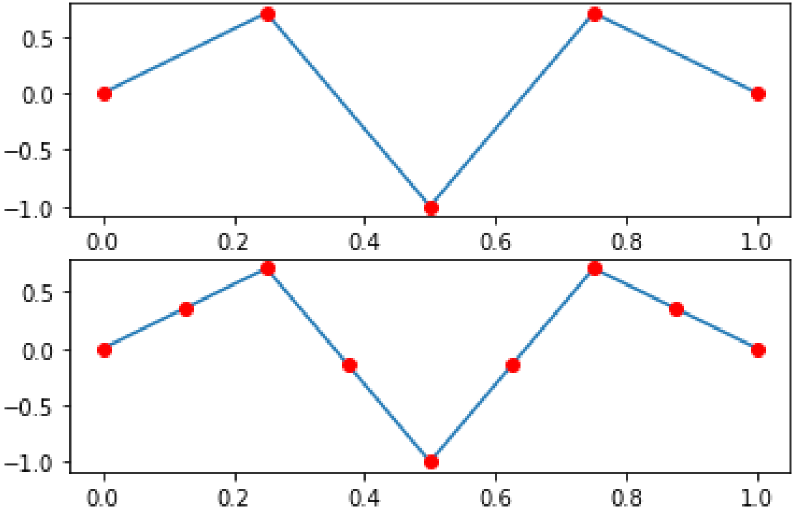
- For shared grid points, let values coinside
- Generate additional fine grid points as average of surrounding coarse grid points

Prolongation Operator

$$I_{2h}^{h} x^{2h} = \frac{1}{2} \begin{bmatrix} 1 & & \\ 2 & & \\ 1 & 1 & \\ & 2 & \\ & 1 & 1 \\ & & 2 \\ & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{2h} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}_{h} = x^{h}$$

- Another advantage: $I_{2h}^{h} = c(I_{h}^{2h})^{T}$
- Important property for theory of multigrid

Prolongation Operator



Coarse Grid A

- Two main methods:
  - Generate discretization of the coarse grid problem
  - **Galerkin Projection:**

$$A^{2h} = I_h^{2h} A^h I_{2h}^h$$

- With full weighting on 1D, both options are the same

Galerkin Projection

First, let e_j^{2h} denote the vector on the coarse grid with a 1 in the jth entry, and zeros elsewhere. Then $A^{2h}e_j^{2h} = I_h^{2h}A^h I_{2h}^h e_j^{2h}$ will be the jth column of A^{2h}. We will calculate this column in steps:

$$I_{2h}^h e_j^{2h} = \frac{1}{2}\begin{bmatrix} 1 & & & \\ 2 & & & \\ 1 & 1 & & \\ & 2 & & \\ & 1 & 1 & \\ & & 2 & \\ & & 1 & \ddots \\ & & & \ddots \\ & & & \ddots \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \frac{1}{2} \\ 1 \\ \frac{1}{2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Galerkin Projection

Notice, this vector now lies in the fine grid so we can now apply the fine grid operator A^h to this vector:

$$A^h I_{2h}^h e_j^{2h} = \frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \frac{1}{2} \\ 1 \\ \frac{1}{2} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \frac{-1}{2h^2} \\ 0 \\ \frac{1}{h^2} \\ 0 \\ \frac{-1}{2h^2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Galerkin Projection

Finally, we apply the restriction operator to this vector to obtain a vector in the course grid space:

$$I_h^{2h}A^hI_{2h}^he_j^{2h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & & & & & \\ & & 1 & 2 & 1 & & & \\ & & & & 1 & 2 & 1 & \\ & & & & & \ddots & \ddots & \ddots \end{bmatrix}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ \frac{-1}{2h^2} \\ 0 \\ \frac{1}{h^2} \\ 0 \\ \frac{-1}{2h^2} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \frac{-1}{4h^2} \\ \frac{1}{2h^2} \\ \frac{-1}{4h^2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Notice that this is exactly the same column we obtain from creating discretization on the coarse grid.

Galerkin Projection

This projection will not be the same as the coarse grid discretization in a 2D problem or if full weighting is not used. Nevertheless, it is a common practice and has been shown to produce good results. It also has the advantage that it requires no extra effort on the part of the user, it can simply be another step in the algorithm.

Formal Two-Grid Cycle

(in Briggs, this is called a Coarse Grid Correction Scheme)

1. Relax ν_1 times on $A^h x^h = b^h$ on Ω^h with initial guess x^h
2. Compute $r^{2h} = I_h^{2h}(b^h - A^h x^h)$
3. Solve $A^{2h} e^{2h} = r^{2h}$ on Ω^{2h}
4. Correct fine grid approximation: $x^h \leftarrow x^h + I_{2h}^h e^{2h}$
5. Relax ν_2 times on $A^h x^h = b^h$ on Ω^h with initial guess x^h

Two-Grid Cycle

```python
def TwoGridScheme(A_fine, b, numPreRelax, numPostRelax, numiters=1):
    I_Restrict = BuildFullWeighting(A_fine.shape[0])
    I_Prolong = 2*I_Restrict.T

    x = np.zeros_like(b)

    A_coarse = I_Restrict.dot(A_fine.dot(I_Prolong))

    for i in range(numiters):
        # First we relax on the fine grid:
        x = Jacobi(x, A_fine, b, numiters=numPreRelax)

        # Now compute the restricted residual
        r_coarse = mvmult(I_Restrict, b - mvmult(A_fine, x))

        # Now we solve the coarse problem Ae = r using CG
        e_coarse = PCG(A_coarse, r_coarse, maxiter=100000)

        # Correct the fine-grid x with the prolongated residual
        x += mvmult(I_Prolong, e_coarse)

        # Relax again
        x = Jacobi(x, A_fine, b, numiters=numPostRelax)
    return x
```

Testing Two-Grid Cycle

- $N = 2^{16}$
- Generate random x^* to manufacture b
- Start with initial $x_0 = 0$

Two Grid Results

Algorithm	Iter	Rel Error	Time (sec)
Jacobi	100	0.87381	37.455
Two Grid (1 pre, 1 post)	1	0.29484	8.900
Two Grid (1 pre, 1 post)	3	0.23544	32.962
Two Grid (3 pre, 3 post)	1	0.23544	11.599

- Can't read too much into the results yet
- Using CG for coarse level - maybe that's the speedup
- Last two lines give some hope though

Let's try:

- More relaxations pre and post
- CG for the fine grid problem

More Results

Algorithm	Iter	Rel Error	Time (sec)
Jacobi	100	0.87381	37.455
Two Grid (1 pre, 1 post)	1	0.29484	8.900
Two Grid (1 pre, 1 post)	3	0.23544	32.962
Two Grid (3 pre, 3 post)	1	0.23544	11.599
Two Grid (5 pre, 5 post)	1	0.20961	14.517
CG	46078	0.21183	28.235

- 2x speed-up to use Two Grid over straight CG
- More we can do to improve this

Recursion

Look at the Two-Grid Algorithm again:

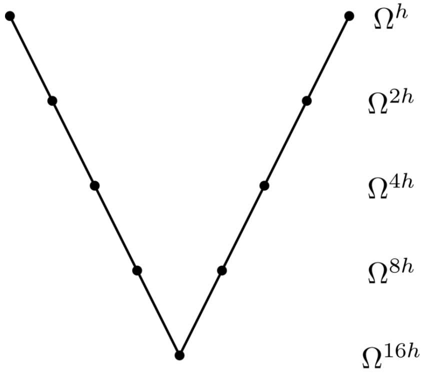
1. Relax ν_1 times on $A^h x^h = b^h$ on Ω^h with initial guess x^h
2. Compute $r^{2h} = I_h^{2h}(b^h - A^h x^h)$
3. Solve $A^{2h} e^{2h} = r^{2h}$ on Ω^{2h}
4. Correct fine grid approximation: $x^h \leftarrow x^h + I_{2h}^h e^{2h}$
5. Relax ν_2 times on $A^h x^h = b^h$ on Ω^h with initial guess x^h

Notice in step 3, we are just solving a smaller linear system. We can replace this solve with another course-grid correction, applying the same process to solve on that grid.

Applying this idea **recursively** is a major concept behind the Multigrid method. At each grid level, our matrix system gets exponentially smaller, allowing faster computation of the coarser levels.

V-Cycle

There are several ways to construct a recursive multigrid pattern. The most common is the **V-Cycle**:



V-Cycle

Here is the code - it's an adapted version of the Two-Grid above

```python
def VCycle(A_fine, b, numPreRelax, numPostRelax, coarsest_N, numiters=1):
    N = A_fine.shape[0]
    I_Restrict = BuildFullWeighting(N)
    I_Prolong = 2*I_Restrict.T

    A_coarse = I_Restrict.dot(A_fine.dot(I_Prolong))
    N_coarse = A_coarse.shape[0]

    for i in range(numiters):
        # First we relax on the fine grid:
        x = Jacobi(x, A_fine, b, numiters=numPreRelax)

        # Now compute the restricted residual
        r_coarse = mvmult(I_Restrict, b - mvmult(A_fine, x))

        # If not on the "bottom of the V", we call recursively
        if N_coarse > coarsest_N:
            # only 1 iteration to get the V-Cycle
            e_coarse = VCycle(A_coarse, r_coarse, numPreRelax, numPostRelax,
      coarsest_N, 1)
        else: # If on the bottom of the V, we solve the coarsest matrix exactly
            e_coarse = PCG(A_coarse, r_coarse, maxiter=100000)

        # Correct the fine-grid x with the prolongated residual
        x += mvmult(I_Prolong, e_coarse)

        # Relax Again
        x = Jacobi(x, A_fine, b, numiters=numPostRelax)

        return x
```

V-Cycle Results

Algorithm	Iter	Rel Error	Time (sec)
Jacobi	100	0.87381	37.455
Two Grid (1 pre, 1 post)	1	0.29484	8.900
Two Grid (1 pre, 1 post)	3	0.23544	32.962
Two Grid (3 pre, 3 post)	1	0.23544	11.599
Two Grid (5 pre, 5 post)	1	0.20961	14.517
CG	46078	0.21183	28.235
V-Cycle (3 pre, 3 post, 127x127 coarse)	1	0.23201	4.7490
V-Cycle (3 pre, 3 post, 127x127 coarse)	3	0.18222	13.906
V-Cycle (5 pre, 5 post, 127x127 coarse)	1	0.20767	7.6766

These are looking pretty good, but the coarse grid size above was chosen arbitrarily. Let's test and see if we can find an optimal coarse grid size

Stopping Analysis

All trials are single V-Cycle run with 5 pre and 5 post relaxations

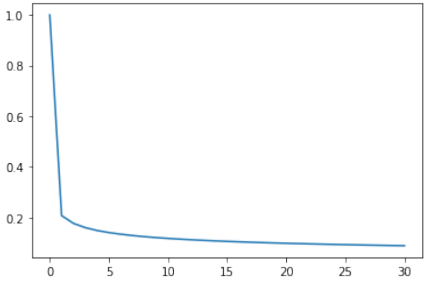
Coarse Matrix Size	Rel Error	Time (sec)
3x3	0.20863	7.64522
7x7	0.20801	7.70789
15x15	0.20784	7.62153
31x31	0.20774	7.64096
63x63	0.20768	7.68061
127x127	0.20767	7.65519
255x255	0.20766	7.82747
511x511	0.20767	7.94081
1023x1023	0.20770	8.07504
2047x2047	0.20777	7.64995
4095x4095	0.20790	7.38173
8191x8191	0.20816	7.14767
16383x16383	0.20868	8.89681
32767x32767	0.20961	14.0686

Overall Results

Algorithm	Iter	Rel Error	Time (sec)
Jacobi	100	0.87381	37.455
Two Grid (1 pre, 1 post)	1	0.29484	8.900
Two Grid (1 pre, 1 post)	3	0.23544	32.962
Two Grid (3 pre, 3 post)	1	0.23544	11.599
Two Grid (5 pre, 5 post)	1	0.20961	14.517
CG	46078	0.21183	28.235
V-Cycle (3 pre, 3 post, 127x127 coarse)	1	0.23201	4.7490
V-Cycle (3 pre, 3 post, 127x127 coarse)	3	0.18222	13.906
V-Cycle (5 pre, 5 post, 127x127 coarse)	1	0.20767	7.6766
V-Cycle (5 pre, 5 post, 8191x8191 coarse)	1	0.20816	7.1476

More Iterations

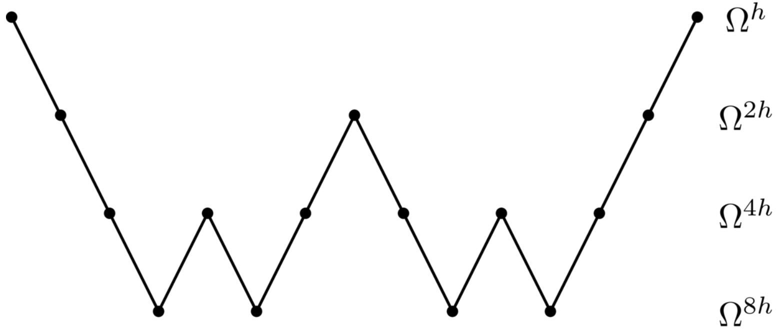
Run 5 pre, 5 post 8191x8191 coarse grid V-Cycle for 30 iterations, tracking error:



Notice the large decrease in error (80% reduction) after the first iteration. This is one of the primary reasons why multigrid tends to make a good preconditioner.

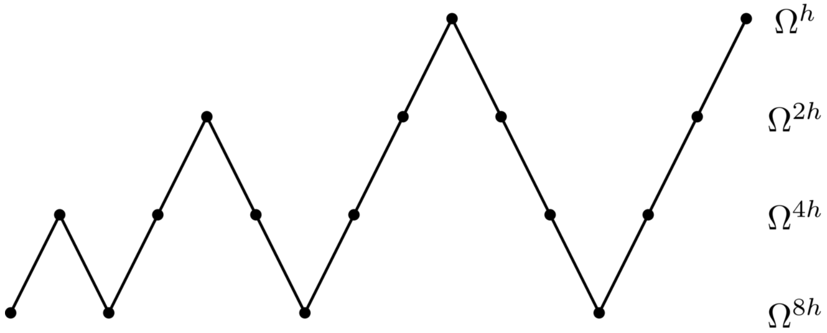
W-Cycle

Complete two V-Cycles on every recursive call:



Generalizes to what Briggs calls μ-Cycles, where you complete μ V-Cycles on every recursive call. Values of $\mu \geq 3$ are not common.

Full Multigrid Cycle



1. Get good initial guess on coarsest grid, correct next finer grid
2. Run a V-Cycle to get a better initial guess for next level up
3. Keep doing V-Cycles until the finest grid is solved

Choices to Make

- Relaxation method
  - Weighted Jacobi (or other variations of Jacobi)
  - Gauss-Seidel (or variations of it)
  - Red-Black versions of Jacobi and GS have been studied
  - Block Jacobi and GS are options as well
- Coarse grid solver
- Number of relaxations (typically 3-5 pre and post)
- Cycle to use (V-Cycle is most common)

Algebraic Multigrid

- Geometric multigrid is useful for gaining intuition into multigrid methods, but not often used in practice
- For general meshes, much tougher to create the restriction and prolongation operators
- Only the physical dimensions are reduced on coarser grid

Algebraic multigrid is a similar method that depends on the coefficient matrix, not the underlying geometric structure.

Algebraic Multigrid

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

- Interpret the magnitudes of the values in the matrix as their "contribution" in calculating the element on the diagonal.
- Use this idea of "significance" to determine which unknowns in the matrix can be "merged" to obtain a "coarser" matrix problem
- This process creates prolongation and restriction operators that only depend on the matrix, not the geometric structure

Algebraic Multigrid

Algebraic multigrid can be programmed in a much more general way and can be more easily extended to other problems. This also makes it more useful as a preconditioner.

Questions?