AMSC 600/CMSC 760
Fall 2007
Solution of Sparse Linear Systems
Multigrid, Part 1
Dianne P. O'Leary

What is Multigrid?

- Originally, multigrid algorithms were proposed as an iterative method to solve linear systems of equations arising from elliptic partial differential equations.
- They have been extended to solve a wide variety of other problems, linear and nonlinear.

In this part of the notes we study a couple of basic examples arising from elliptic partial differential equations.

The Plan

A Simple Example

Suppose we want to solve the differential equation

$$-u_{xx}(x) = f(x)$$

on the domain $x \in [0,1]$, with $u(0) = u(1) = 0$.

Define a mesh $x_j = jh$, where $h = 1/(n+1)$ for some integer n.

Then we can determine approximate values $u_j \approx u(x_j)$, $j = 1, \ldots, n$ using finite difference or finite element approximations. If we choose finite differences, then we have

$$-u_{xx}(x_j) \approx \frac{-u_{j-1} + 2u_j - u_{j+1}}{h^2}.$$

We obtain a system of equations

$$\mathbf{Au} = \mathbf{f}$$

with

$$\mathbf{u} = [u_1, \ldots, u_n]^T,$$
$$\mathbf{f} = [f(x_1), \ldots, f(x_n)]^T,$$

and

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

Applying Gauss-Seidel

Recall that in the G-S method we take an initial guess $\mathbf{u}^{(0)}$ for the solution and then update the guess by cycling through the equations, solving equation i for the ith variable u_i, so that given $\mathbf{u}^{(k)}$, our next guess $\mathbf{u}^{(k+1)}$ becomes

$$u_i^{(k+1)} = (f_i - \sum_{j=1}^{i-1} a_{ij} u_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} u_j^{(k)})/a_{ii}, \ i = 1, \ldots, n.$$

In our case, this reduces to

$$u_i^{(k+1)} = h^2(f_i + u_{i-1}^{(k+1)} + u_{i+1}^{(k)})/2, \ i = 1, \ldots, n,$$

where we define $u_0^{(k)} = u_{n+1}^{(k)} = 0$ for all k.

It is easy to see how G-S can be very slow on a problem like this. Suppose, for example, that we take $\mathbf{u}^{(0)} = \mathbf{0}$, and that $\mathbf{f}$ is zero except for a 1 in its last position. Then $\mathbf{u}^{(1)}$ is zero except for its last entry, $\mathbf{u}^{(2)}$ is zero except for its last two entries, and it takes n iterations to get a guess that has a nonzero first entry. Since the true solution has nonzeros everywhere, this is not good!

The problem is that although G-S is good at fixing the solution locally, the information is propagated much too slowly globally, across the entire solution vector.

So if we are going to use G-S effectively, we need to couple it with a method that has good global properties.

A Multigrid Algorithm

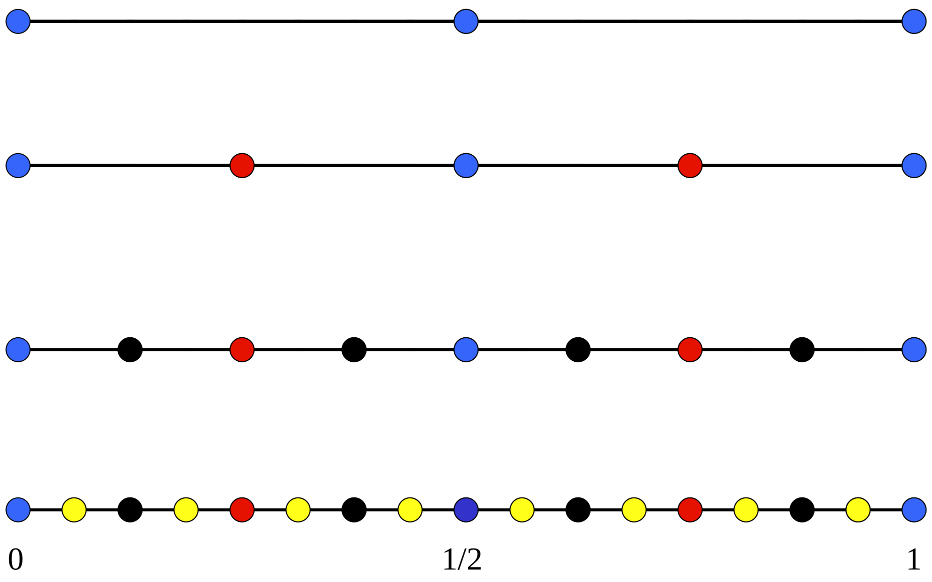

Figure 1: Four levels of nested grids on the interval $[0, 1]$. The coarsest grid, with $h = 1/2$, consists of the blue points. Adding the red points gives $h = 1/4$. Including the black points gives $h = 1/8$, and including all of the points gives the finest grid, with $h = 1/16$.

We chose a value of n, probably guided by the knowledge that the error in the finite difference approximation is proportional to h^2.

There is a whole family of finite difference approximations, defined by different choices of h, and we denote the system of equations obtained using a mesh length $h = 1/(n + 1)$ by

$$\mathbf{A}_h \mathbf{u}_h = \mathbf{f}_h \ .$$

- A large value of h gives a coarse grid. The dimension n of the resulting linear system of equations is very small, though, so we can solve it fast using either a direct or an iterative method. Our computed solution $\mathbf{u}_h$ has the same overall shape as the true solution u but loses a lot of local detail.
- In contrast, if we use a very fine grid with a small value of h, then the linear system of equations is very large and much more expensive to solve, but our computed solution $\mathbf{u}_h$ is very close to u.

In order to get the best of both worlds, we might use a coarse-grid solution as an initial guess for the G-S iteration on a finer grid.

Interpolation

To do this, we must set values for points in the finer grid that are not in the coarse grid. If someone gave us a solution to the system corresponding to h, then we could obtain an approximate solution for the system corresponding to $h/2$ by interpolating those values:

- For points in the finer mesh that are common to the coarser mesh, we just take their values.
- For points in the finer mesh that are midpoints of two points in the coarser mesh, we take the average of these two values.

This defines an interpolation operator $\mathbf{P}_h$ that takes values in a grid with parameter h and produces values in the grid with parameter $h/2$.

For example, because our boundary conditions are zero,

$$P_{1/8} = \begin{bmatrix} 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 \end{bmatrix}.$$

Nested iteration

The process of solving the problems on the sequence of nested grids gives us a Nested iteration algorithm for our sample problem.

```
Nested Iteration
Set $k = 1$, $h = 1/2$, and $\tilde{\mathbf{u}}_h = 0$.
while the approximation is not good enough,
    Set $k = k + 1$, $n = 2^k - 1$, and $h = 1/(n + 1)$.
    Form the matrix $\mathbf{A}_h$ and the right-hand side $\mathbf{f}_h$, and use the G-S iteration,
    with the initial guess $\mathbf{P}_{2h}\tilde{\mathbf{u}}_{2h}$, to compute an approximate solution $\tilde{\mathbf{u}}_h$ to
    $\mathbf{A}_h\mathbf{u}_h = \mathbf{f}_h$.
end
```

The termination tolerance for the norm of the residual $\mathbf{f}_h - \mathbf{A}_h\tilde{\mathbf{u}}_h$ on grid h should be proportional to h^2, since that matches the size of the local error.

This algorithm runs from coarse grid to finest and is useful (although rather silly for one-dimensional problems). But there is a better way.

The V-Cycle

We can do better if we run from finest grid to coarsest grid and then back to finest. This algorithm has 3 ingredients:

- An iterative method that converges quickly if most of the error is high frequency – oscillating rapidly – which happens when the overall shape of the solution is already identified. G-S generally works well.
- A way to transfer values from a coarse grid to a fine one – interpolation or prolongation.
- A way to transfer values from a fine grid to a coarse one – restriction. We let $\mathbf{R}_h$ be the operator takes values on grid $h/2$ and produces values on grid h.

We already have matrices $\mathbf{P}_h$ for interpolation, and (for technical reasons related to preserving the self-adjointness of the problems considered here) we choose $\mathbf{R}_h = \mathbf{P}_h^T$.

We define the V-Cycle idea recursively.

```
V-Cycle
$\mathbf{v}_h$ = V-Cycle($\mathbf{v}_h$, $\mathbf{A}_h$, $\mathbf{f}_h$, $\eta_1$,$\eta_2$)
Perform $\eta_1$ G-S iterations on $\mathbf{A}_h\mathbf{u}_h = \mathbf{f}_h$ using $\mathbf{v}_h$ as the initial guess,
obtaining an approximate solution that we still call $\mathbf{v}_h$.
if $h$ is the coarsest grid parameter then
    compute $\mathbf{v}_h$ to solve $\mathbf{A}_h\mathbf{v}_h = \mathbf{f}_h$ and return.
else
    Let $\mathbf{v}_{2h}$ = V-Cycle($\mathbf{0}$,$\mathbf{A}_{2h}$,$\mathbf{R}_{2h}(\mathbf{f}_h - \mathbf{A}_h\mathbf{v}_h)$, $\eta_1$,$\eta_2$).
    Set $\mathbf{v}_h = \mathbf{v}_h + \mathbf{P}_{2h}\mathbf{v}_{2h}$.
end
Perform $\eta_2$ G-S iterations on $\mathbf{A}_h\mathbf{u}_h = \mathbf{f}_h$ using $\mathbf{v}_h$ as the initial guess,
obtaining an approximate solution that we still call $\mathbf{v}_h$.
```

In using this algorithm, we can define $\mathbf{A}_{2h} = \mathbf{R}_{2h}\mathbf{A}_h\mathbf{P}_{2h}$. This definition is key to extending the multigrid algorithm beyond problems that have a geometric grid. We'll discuss these algebraic multigrid methods later. But for now, let's see how it works on our original problem.

Unquiz 1. Work through the V-Cycle algorithm to see exactly what computations it performs on our simple example for the sequence of grids defined in Figure 1. Estimate the amount of work, measured by the number of floating-point multiplications performed.

The standard multigrid algorithm solves $\mathbf{A}_h\mathbf{u}_h = \mathbf{f}_h$ by repeating the V-Cycle until convergence. We start the iteration by initializing $\mathbf{u}_h = \mathbf{0}$. Then, until convergence, we do the following:

- Compute $\mathbf{\Delta u}_h =$ V-cycle$(\mathbf{0}, \mathbf{A}_h, \mathbf{r}_h, \eta_1, \eta_2)$, where $\mathbf{r}_h = \mathbf{f}_h - \mathbf{A}_h\mathbf{u}_h$.
- Update $\mathbf{u}_h = \mathbf{u}_h + \mathbf{\Delta u}_h$.

Cost of Multigrid

We can estimate the work for multigrid.

- One step of the G-S iteration on a grid of size h costs about $nz(h)$ multiplications, where $nz(h)$ is the number of nonzeros in A_h. We'll call $nz(h)$ multiplications a work unit.
- Note that $nz(h) \approx 2nz(2h)$ since A_{2h} has about half as many rows as A_h.
- So performing 1 G-S step on each grid $h, h/2, \ldots, 1$ costs less than $nz(h)(1 + 1/2 + 1/4 + \ldots) = 2nz(h)$ multiplications $\equiv$ 2 work-units.
- So the cost of a V-Cycle is at most 2 times the cost of $(\eta_1 + \eta_2)$ G-S iterations on the finest mesh plus a modest amount of additional overhead.

Unquiz 2. Convince yourself that the storage necessary for all of the matrices and vectors is also a modest multiple of the storage necessary for the finest grid.

We know that standard iterative methods like G-S are usually very slow (take many iterations), so the success of multigrid relies on the fact that we need only a few iterations on each grid, because the error is mostly local. Thus the total amount of work to solve the full problem to a residual of size $O(h^2)$ is a small number of work-units.

It is rather silly to use anything other than sparse Gauss elimination to solve a system involving a tridiagonal matrix. Note, though, that our algorithm readily extends to higher dimensions; we just need to define $\mathbf{A}_h$ and $\mathbf{P}_h$ for a nested set of grids in order to use the multigrid V-Cycle algorithm.

Multigrid for 2-d Problems

Our first challenge in applying multi-grid to 2-dimensional problems is to develop a sequence of nested grids. Since we discussed finite difference methods for the 1-dimensional problem, let's focus on finite element methods for the 2-dimensional problem, using a triangular mesh and piecewise-linear basis functions.

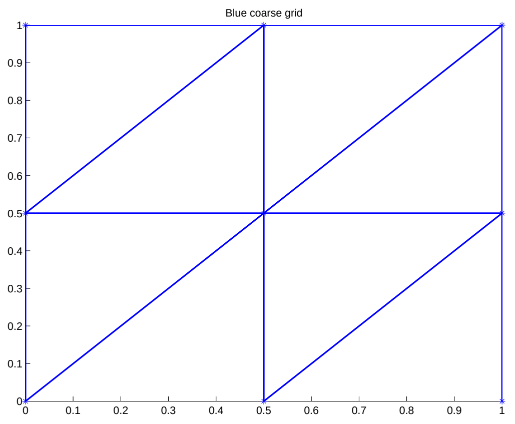

Figure 2: The blue gridpoints define the coarse mesh.

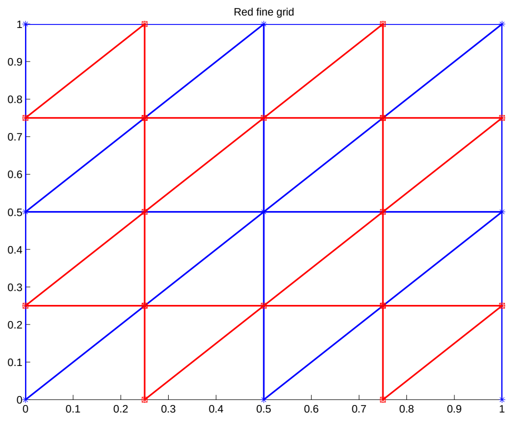

Figure 3: The blue and the red gridpoints define the finer mesh.

It is most convenient to start from a coarse grid and obtain our finest grid through successive refinements. Consider the initial grid in Figure **??**, which divides the unit square into 8 triangles with height $h = 1/2$. The grid points are marked in blue.

Consider taking the midpoints of each side of one of triangles, and drawing the triangle with those points as vertices. If we do this for each triangle, we obtain the red grid points in Figure **??** and the red triangles. Each of the original blue triangles has been replaced by 4 triangles, each having 1 or 3 red sides, and each triangle has height $h = 1/4$.

If we repeat this process, we obtain the black grid points of Figure **??** and a mesh length $h = 1/8$.

Writing a program for refining a general grid is complicated.

Interpolating from one grid to the next finer one is easy. For example, given the

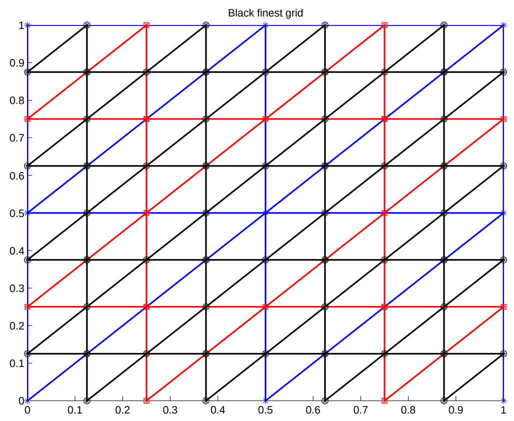

Figure 4: The blue, red, and black gridpoints define the finest mesh.

blue grid values, we obtain values for the blue and red grid by following two rules: blue gridpoints retain their values, and red grid values are defined as the average of the nearest two values on the blue edge containing it. As before, we take the restriction operator to be the transpose of the interpolation operator.

So we have all the machinery necessary to apply multigrid to 2-dimensional problems.

If the partial differential equation is elliptic, it is not too hard to achieve convergence in a small number of work-units. In fact, multigridders would say that if you don't achieve it, then you have chosen either your iteration or your interpolation/restriction pair "incorrectly".

What if the pde is not elliptic?

For problems that are not elliptic, though, things get a bit more complicated, as can be seen using the Helmholtz equation

$$-\Delta u + \kappa u = f.$$

The problem is much harder to solve for negative values of κ. There are two reasons for this:

- First, the matrix $\mathbf{A}_h$ is no longer positive definite, so we lose a lot of nice structure,
- Second, finer grids are necessary to represent the solution accurately.

In order to restore convergence in a small number of work units for the non-elliptic problem, we must make the algorithm more complicated – for example, we might use multigrid as a preconditioner for a Krylov subspace method.

References

The multigrid idea dates back to R. P. Fedorenko in 1964. A good introduction is given in a tutorial book by Briggs, Henson, and McCormick.

It is also useful to use multigrid if only a portion of the grid is refined from one level to the next; for example, we might want to refine only in regions in which the solution is rapidly changing, so that the current grid cannot capture its behavior accurately enough. These adaptive methods are also discussed in Briggs et al.

One multigrid approach to solving the Helmholtz equation with negative κ is given in a paper by Elman, Ernst, and O'Leary.