AMSC/CMSC 661 Scientific Computing II
Spring 2005
Solution of Sparse Linear Systems
Part 3: Preconditioning and multigrid

Dianne P. O'Leary

The plan:

So far:

- Iterative methods:
  - Basic (slow) iterations: Jacobi, Gauss-Seidel, SOR.
  - Krylov subspace methods: the algorithms

Next:

- Krylov subspace methods: convergence theory
- Preconditioning (where direct meets iterative)
- A special purpose method: Multigrid

The practical form of GMRES

First, how the algorithm is used to solve $\hat{G}x^* = c$, given a positive integer m (the restart parameter) and with $B = I$.

Initially, $x^{(m)} = 0$ and $k = m$.
Until termination,

- Set $k = k + 1$. If $k = m + 1$, then set $k = 1$ and $x^{(0)} = x^{(m)}$.
- Increase the dimension of the Krylov subspace to dimension k using the starting vector $c - \hat{G}x^{(0)}$ and the matrix $\hat{G}$, giving a matrix P_k of directions and H_k of coefficients.
- Solve $H_k^T H_k y^{(k)} = H_k^T P_{k+1}^T c$, and set $x^{(k)} = x^{(0)} + P_k y^{(k)}$.
- If $\|\hat{G}x^{(k)} - c\|$ is small enough, set $x_{final} = x^{(k)}$ and terminate.

Note: Ideally, $m = n$, but since the storage is $O(mn)$, and the time to **solve the systems involving $H_1, \ldots H_m$ is $O(m^3)$ (using matrix updating techniques and Cholesky decomposition)**, in practice we keep m to 20 or 100 at most.

Convergence results for GMRES(m)

- **Convergence on positive definite matrices.** (Saad p.205, Thm 6.30) If $(\hat{G} + \hat{G}^T)/2$ is positive definite, then GMRES(m) converges for any $m \geq 1$.

- **Convergence on diagonalizable matrices.** (Saad p.206, Prop 6.32) If $\hat{G} = X\Lambda X^{-1}$ where Λ is the matrix of eigenvalues, then for $k = 1, \ldots, m$,

$$\|r^{(k)}\|_2 \leq \kappa(X)\epsilon_k \|r^{(0)}\|_2,$$

where
$r^{(i)} = c - \hat{G}x^{(i)}$ is the residual,
$\kappa(X)$ is the square-root of the ratio of the largest eigenvalue of $X^T X$ to the smallest,
and

$$\epsilon_k = \min_{\rho \in \mathcal{P}_k} \max_{j=1,\ldots,n} |\rho(\lambda_j)|.$$

where $\mathcal{P}_k$ is the set of all polynomials of degree at most k satisfying $\rho(0) = 1$.

- **For normal matrices:** If $\hat{G}\hat{G}^T = \hat{G}^T\hat{G}$, then $\hat{G}$ is normal and $\kappa(X) = 1$. Otherwise the bound is not very useful.

- (Saad p.206, Cor 6.33) If **all of the eigenvalues of $\hat{G}$ are in an ellipse** that doesn't contain the origin, and the ellipse **is centered at $(c, 0)$ in the complex plane, with** focal distance d **(pure real or pure imaginary)** and semimajor axis a, then

$$\epsilon_k \leq \frac{\left(a/d + \sqrt{(a/d)^2 - 1}\right)^k + \left(a/d + \sqrt{(a/d)^2 - 1}\right)^{-k}}{\left(c/d + \sqrt{(c/d)^2 - 1}\right)^k + \left(c/d + \sqrt{(c/d)^2 - 1}\right)^{-k}}$$

Convergence result for CG

(Saad, p.205, eqn (6.128)) In the **energy norm**,

$$\|x^{(k)} - x^*\|_{\hat{G}} \leq 2\left(\frac{\sqrt{\kappa(\hat{G})} - 1}{\sqrt{\kappa(\hat{G})} + 1}\right)^k \|x^{(0)} - x^*\|_{\hat{G}}$$

where $\kappa(\hat{G})$ is the ratio of the largest and smallest eigenvalues of $\hat{G}$.

Preconditioning GMRES and CG

- For fast iterations, we need to be able to solve linear systems involving M very quickly, since this must be done once per iteration.
- To make the number of iterations small, we want M to be a good approximation to A so that the eigenvalues are in a small ellipse (GMRES) or a small interval (CG).
- **For CG, we need to require that M be symmetric and positive definite.**
- Note that the linear system

$$Mz = r$$

  is typically solved using a **direct method**, so the better M is, the closer we are to solving $Ax = b$ using a direct method.

Some common choices of preconditioning matrices M

- $M =$ the diagonal of A.
- $M =$ a banded piece of A.
- $M =$ an incomplete factorization of A, leaving out inconvenient elements (the **ILU preconditioner**).
- $M^{-1} =$ a sparse approximation to A^{-1}. (the **sparse approximate inverse preconditioner** (SAIP))
- $M =$ a related matrix; e.g., if A is a discretization of a differential operator, M might be a discretization of a related operator that is easier to solve. Or M might be the block diagonal piece of the matrix after ordering for nested dissection.
- M might be the matrix from any stationary iterative method (SIM) or from **multigrid** (to be discussed).

- In some situations, it is a good idea to let M change at each iteration. The resulting algorithm is called **flexible-GMRES**. It is sometimes useful, but we won't discuss it here.

How to form $M^{-1}r$ for an SIM preconditioner

Consider your favorite stationary iterative method (Jacobi, Gauss-Seidel, SOR, etc.),

$$Mx^{(k+1)} = Nx^{(k)} + b$$

or

$$x^{(k+1)} = M^{-1}Nx^{(k)} + \mathbf{M^{-1}}b.$$

Manipulating this a bit, we get

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + (M^{-1}N - I)x^{(k)} + M^{-1}b \\ &= x^{(k)} + M^{-1}(N - M)x^{(k)} + M^{-1}b \\ &= x^{(k)} + M^{-1}(b - Ax^{(k)}) \\ &= x^{(k)} + M^{-1}r^{(k)}\,. \end{aligned}$$

Therefore, we compute the "preconditioned residual" by taking one step of the SIM starting from the latest CG or GMRES iterate, and returning Δx for $M^{-1}r^{(k)}$. It is this matrix M^{-1}, that represents the multiple of the residual that we add on to x, that preconditions CG or GMRES.

Multigrid methods

The idea behind multigrid methods

(Idea: Fedorenko 1964, Brandt 1977, Nicolaides, Hackbusch, ...)
Consider our simplest problem

$$-u'' = f(x)$$

on the interval $x \in (0, 1)$, with $u(0) = u(1) = 0$.

There are three ingredients to the idea.

- If we use a very **coarse grid** for finite elements, with $h = .25$, for example, then
  - The linear system of equations is very small ($n = 3$) so we can solve it fast using either a **direct** or an **iterative** method.
  - We expect our computed solution u_h to have the same overall shape as the true solution u but to lose a lot of local detail.

- If we use a very **fine grid**, then
  - The linear system of equations is much more expensive to solve.
  - We expect our computed solution u_h to be very close to u.
- If an iterative method is started with an initial guess that is **close to the true solution**, we hope to need a very small number of iterations.

– In particular, if we consider Jacobi, Gauss-Seidel, or SOR, we adjust the solution based on very **local information**, so these methods are good at filling in the fine details once the overall shape of the solution is known.

Making use of multiple grids

An idea: **Nested iteration**

Set $k = 0$, $h = 1$, and $u_h = 0$.
While the approximation is not good enough,

Set $k = k + 1$, $n = 2^k - 1$, and $h = 1/(n + 1)$.

Form the matrix A_h and the right-hand side b_h, and solve the matrix problem for the finite element approximation u_h using **GS**, with the initial guess formed from $\mathbf{u_{2h}}$ evaluated at the mesh points.

The termination tolerance for the residual on grid h should be proportional to h^2, since that is the size of the local error.

This algorithm runs from coarse grid to finest and is useful (although rather silly for 1D problems).

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.
- A way to transfer values from a fine grid to a coarse one – **restriction**. We let $\mathcal{R}_h$ be the operator that goes from grid h to grid $2h$.
- A way to transfer values from a coarse grid to a fine one – **interpolation** or **prolongation**.
  We let $\mathcal{I}_h$ be the operator that goes from grid $2h$ to grid h.

Gauss-Seidel gives us the first ingredient, while our finite element formula for the solution as a sum of basis function components gives us the last two. (For

technical reasons, though, restriction should be the adjoint of interpolation, rather than using the finite element choices for both.)

For finite differences, interpolation and restriction can also be defined.

We'll define the V-Cycle idea recursively.

v_h = V-Cycle(v_h,b_h, η_1,η_2)

1. Perform η_1 GS iterations on $A_h u_h = b_h$ using v_h as the initial guess, obtaining an approximate solution that we still call v_h.
2. If h is not the coarsest grid parameter,

   Let v_{2h} = V-Cycle(0,$\mathcal{R}_h(b_h - A_h v_h)$, η_1,η_2).
   Set $v_h = v_h + \mathcal{I}_h v_{2h}$.
3. Perform η_2 GS iterations on $A_h u_h = b_h$ using v_h as the initial guess, obtaining an approximate solution that we still call v_h.

The standard multigrid algorithm is to solve $A_h u_h = b_h$ by repeating the V-Cycle until convergence.

Cost per V-Cycle

A GS iteration on a grid of size h costs about $nz(h)$ multiplications, where $nz(h)$ is the number of nonzeros in A_h. Note that $nz(h) \approx 2nz(2h)$ since A_{2h} has about half as many rows as A_h.

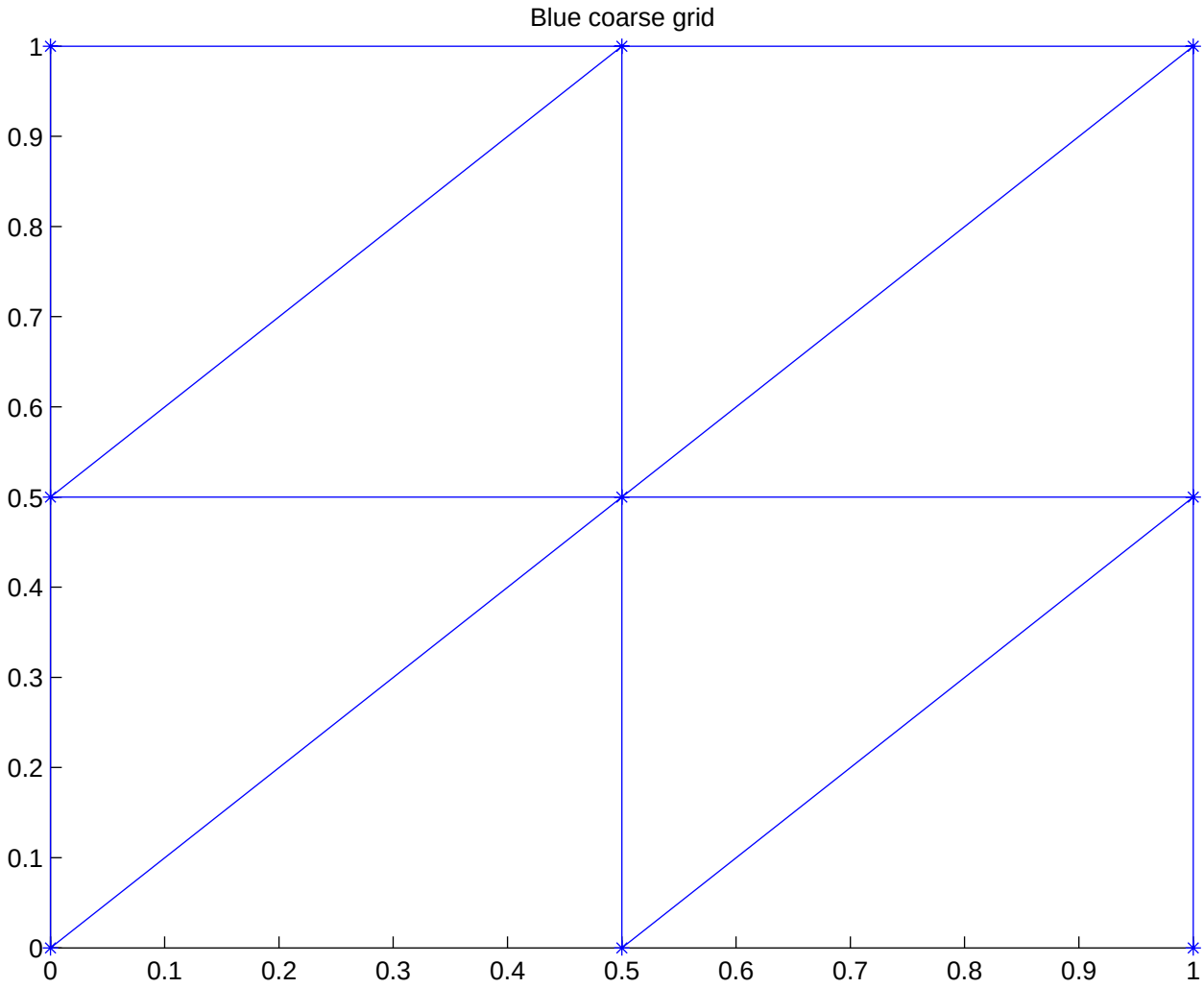
So performing 1 GS iteration on each grid $h, h/2, \ldots, 1$ costs less than $nz(h)(1 + 1/2 + 1/4 + \ldots) \approx 2nz(h)$ multiplications $\equiv$ 2 **work-units**.

So the cost of a V-Cycle is at most 2 times the cost of $\eta_1 + \eta_2$ GS iterations on the finest mesh.

Unquiz: 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. []

Convergence rate for multigrid

We know that standard iterative methods like GS are very slow (take many iterations), but on our simple problem, we need only a few iterations on each grid, and 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.



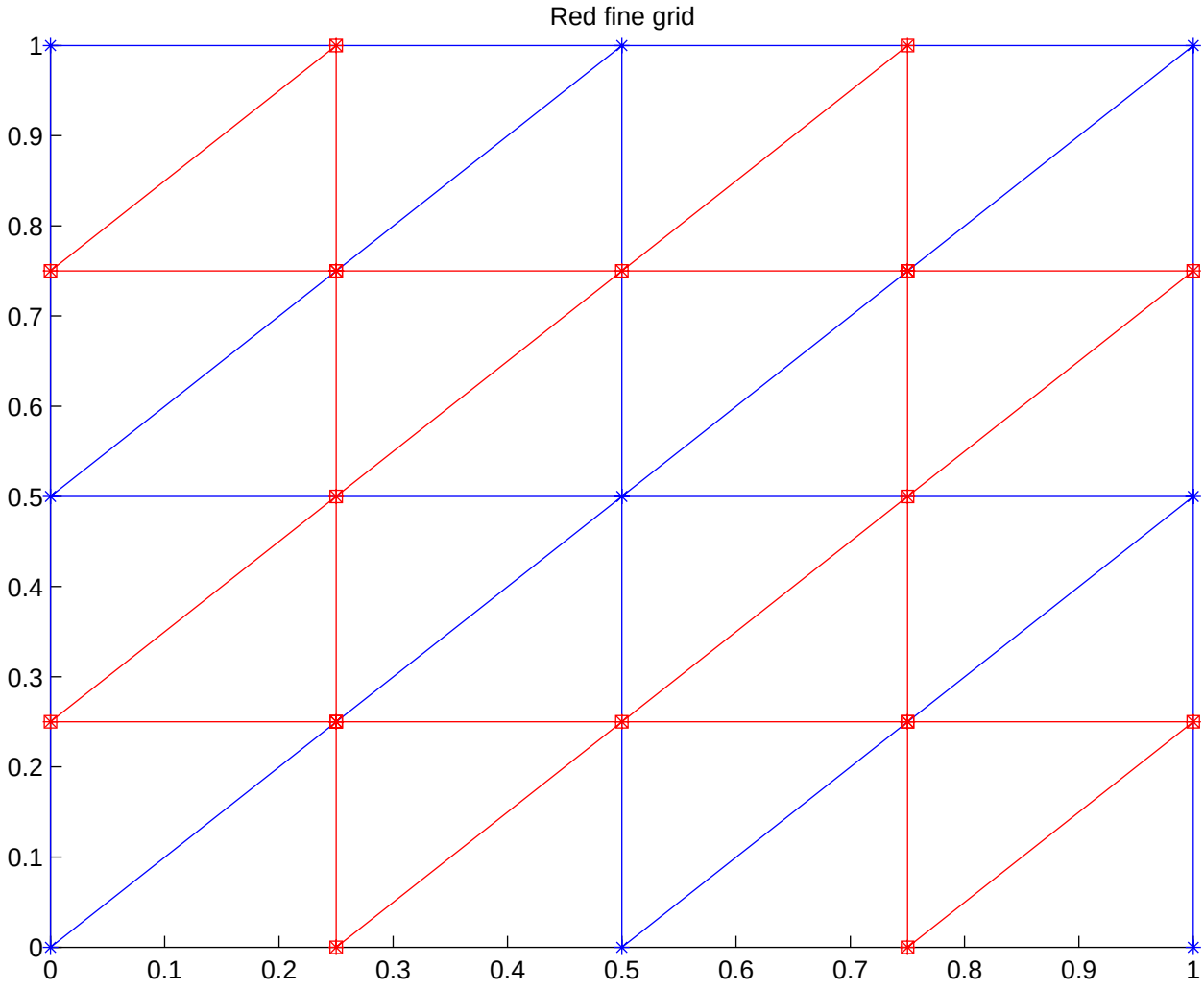
Multigrid for 2-d problems

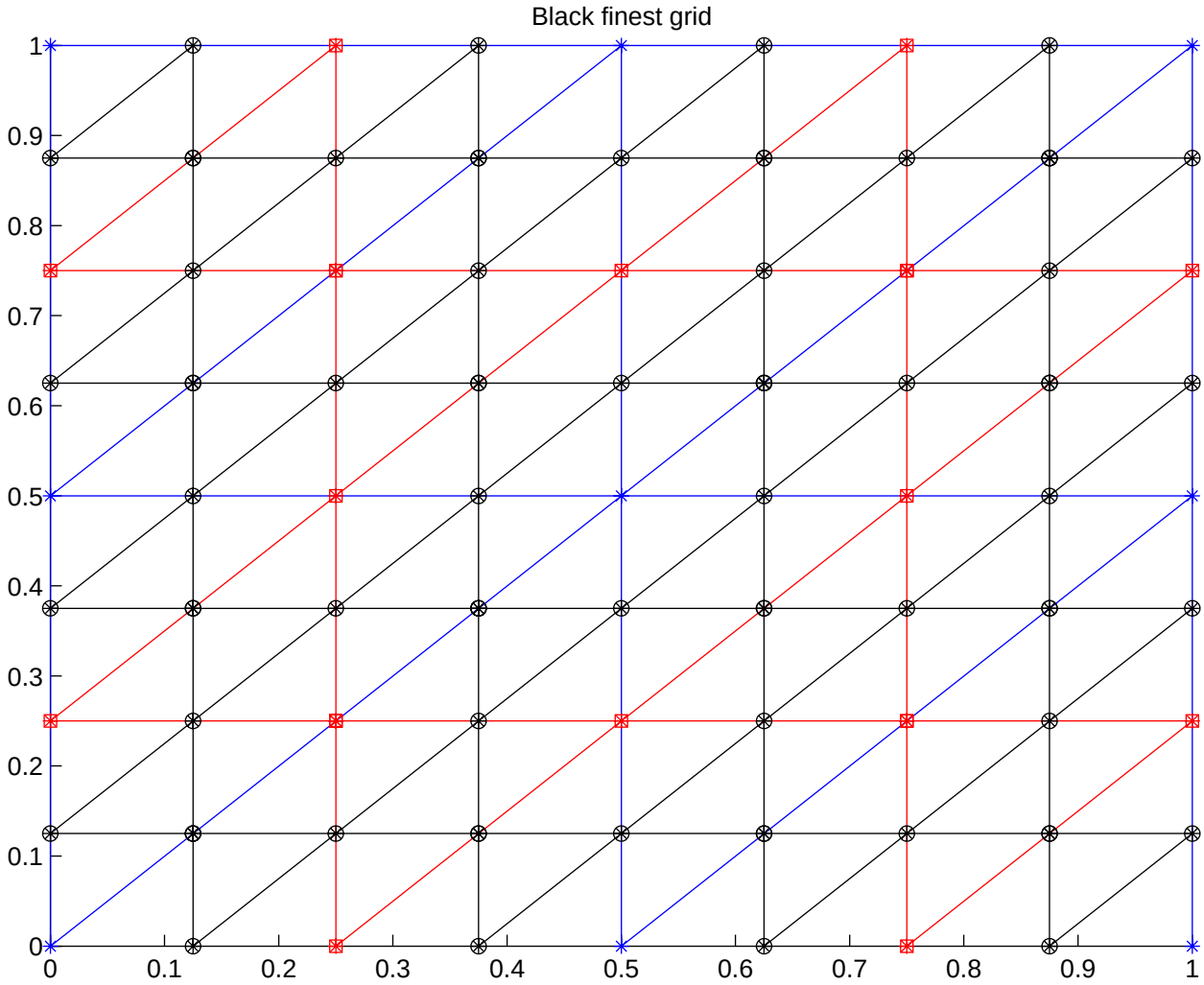
Develop a sequence of **nested grids**.

If the PDE is elliptic, it is not to hard to achieve convergence in a small number of work-units.

Multigridders would say that if you don't achieve it, then you have chosen either your iteration or your interpolation/restriction pair "incorrectly".

For the domain $(0,1) \times (0,1)$, we might use the three grids shown here:


Red fine grid
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Black finest grid
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1