Document: Math 228A (Fall 2011)
Professor: Guy
Latest Update: April 2, 2012
Author: Jeff Irion
`http://www.math.ucdavis.edu/~jlirion`

Contents

1 Guy's Notes (Part 1)

2 Guy's Notes (Part 2)

2.1 Error and Stability

We are dealing with the linear system

$$Au = b$$

In order to measure error, we need to have a norm. The most common are:

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}|$$
$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}|$$
$$\|A\|_2 = \sqrt{\rho(A^*A)}$$

where ρ is the spectral radius (eigenvalue of maximum modulus). We define the error vector as

$$\mathbf{e} = \mathbf{u} - \mathbf{u}_{sol}$$

We also define the truncation error as

$$\begin{aligned}\tau_j &= \frac{1}{h^2}(u(x_{j-1}) - 2u(x_j) + u(x_{j+1})) - f(x_j)\\ &= u_{xx}(x_j) + \frac{h^2}{12}u^{(4)}(x_j) + O(h^4) - f(x_j)\\ &= \frac{h^2}{12}u^{(4)}(x_j) + O(h^4)\end{aligned}$$

since $u_{xx} = f(x)$. Thus, we have an expression for the local truncation error:

$$\tau = A\mathbf{u}_{sol} - \mathbf{b}$$

Using simple algebra, we can derive that

$$\begin{aligned}A\mathbf{u}_{sol} &= \tau + \mathbf{b}\\ A\mathbf{u}^h &= \mathbf{b}\\ A(\mathbf{u}^h - \mathbf{u}_{sol}) &= -\tau^h\\ A\mathbf{e}^h &= -\tau^h\end{aligned}$$

We know that $\tau = O(h^2)$, so we would like it to be the case that $\mathbf{e}^h = O(h^2)$.

$$\begin{aligned}\mathbf{e}^h &= -A^{-1}\tau^h\\ \|\mathbf{e}^h\| &= \|A^{-1}\tau^h\| \le \|A^{-1}\|\|\tau^h\|\end{aligned}$$

Thus, we want the norm of A^{-1} to be $O(1)$.

To tell if the system is convergent, we look at:

Consistency: $\|\tau^h\| \to 0$ as $h \to 0$

Convergence: $\|\mathbf{e}^h\| \to 0$ as $h \to 0$

Stability: The system $A^h\mathbf{u}^h = \mathbf{f}^h$ is stable if $\|A\| \leq C$ for $h \leq h_0$ and C is a constant independent of h.

For a linear PDE, the Lax-Equivalence Theorem says that if a scheme is stable and consistent, then it is convergent.

2.2 Stability in the 2-Norm

Because A is symmetric,

$$\|A\|_2 = \rho(A) = \max |\lambda_j|$$

A^{-1} is also symmetric, with

$$\|A^{-1}\|_2 = \rho(A^{-1}) = \max |\lambda_j^{-1}| = \min |\lambda_j|$$

Recall that the eigenvalues of $Lu = u_{xx}$ are $u_n = \sin(n\pi x)$. Some arithmetic shows that the largest eigenvalue of A is

$$\lambda_N = \frac{2}{h^2}(\cos(N\pi h) - 1)$$

This will lead to a 2nd order accurate solution.

3 Guy's Notes (Part 3)

3.1 3 Properties

1. Discrete Maximum Principle: if $L^h u \geq 0$ on some region, then the maximum value of u is obtained on the boundary. (If $L^h u \leq 0$ on some region, then the minimum value of u is obtained on the boundary.)

2. If u is a discrete function defined on the regular grid discretizing the unit square with $u = 0$ on the boundary, then $\|u\|_\infty \leq \frac{1}{8}\|L^h u\|_\infty$.

3. Let u_{sol} solve $\Delta u = f$ and the corresponding boundary conditions. Then
$\|\mathbf{e}\|_\infty = \|\mathbf{u}^h - \mathbf{u}_{sol}\|_\infty \leq \frac{h^2}{96}(\|\mathbf{u}_{sol,xxxx}\|_\infty + \|\mathbf{u}_{sol,yyyy}\|_\infty) + O(h^4)$.

3.2 Error and the Residual

Let u^k be an approximate solution to $A\mathbf{u} = \mathbf{f}$ and let u be the exact solution to the discrete problem. We define algebraic error as

$$\mathbf{e} = \mathbf{e}^k = \mathbf{u} - \mathbf{u}^k$$

Then

$$A\mathbf{e} = A\mathbf{u} - A\mathbf{u}^k = \mathbf{f} - A\mathbf{u}^k$$

We define the residual as

$$\mathbf{r} = \mathbf{f} - A\mathbf{u}^k$$
$$\mathbf{r} = A\mathbf{e}$$

The residual is a measure of how much our approximate algebraic solution, obtained via iteration, fails to satisfy the discrete equations. The exact solution is

$$\mathbf{u} = \mathbf{u}^k + \mathbf{e} = \mathbf{u}^k + A^{-1}\mathbf{r}$$

3.3 Jacobi and Gauss-Seidel Iteration Methods

3.4 Analysis of Jacobi and Gauss-Seidel

3.5 Iterations to Reduce the Error by a Factor

3.6 Successive Over-Relaxation

4 Guy's Notes (Part 4)

4.1 Motivation for Multigrid

Relaxation methods slow down as the mesh is refined.

$$\frac{\|\mathbf{e}^{k+1}\|}{\|\mathbf{e}^{k}\|} \approx \rho,\ \rho \to 0 \text{ as } h \to 0$$

We want to find an iterative method such that

$$\frac{\|\mathbf{e}^{k+1}\|}{\|\mathbf{e}^{k}\|} \approx \rho < c < 1 \text{ as } h \to 0$$

Note that the estimate of $\frac{\|\mathbf{e}^{k+1}\|}{\|\mathbf{e}^{k}\|} \approx \rho$ applies for large k. In practice, convergence is much faster at first but it slows down dramatically.

Idea: use Gauss-Seidel Red-Black to smooth the error on a fine grid, then transfer to a coarser grid.

4.2 Coarse Grid Correction

Let $\mathbf{u}_h$ be the algebraic solution to the discrete problem $L_h\mathbf{u}_h = \mathbf{f}$. Then $\mathbf{u}_h$ is an approximate solution to

$$\mathbf{e}^k = \mathbf{u}_h - \mathbf{u}^k$$
$$\mathbf{r}^k = \mathbf{f} - L\mathbf{u}^k$$

We have the residual equation: $L\mathbf{e}^k = \mathbf{r}^k$
If we can solve this equation: $\mathbf{e}^k = L^{-1}\mathbf{r}^k$
Then the algebraic solution is: $\mathbf{u}_h = \mathbf{u}^k + L^{-1}\mathbf{r}^k$

We are effectively correcting the approximation.

4.3 2-Grid Preliminary Scheme

Let Ω_h represent the original grid, and let Ω_{2h} represent a coarse grid with twice the grid spacing. We will require transfer operators.

$I_h^{2h} : G(\Omega_h) \to G(\Omega_{2h})$
This is the restriction operator that maps gridfunctions from the fine grid to the coarse grid.

$I_{2h}^{h} : G(\Omega_{2h}) \to G(\Omega_h)$
This is the interpolation operator that maps gridfunctions from the coarse grid to the fine grid.

Our basic scheme looks like this:

1. Obtain an approximate solution, $\mathbf{u}_h^k$, and solve the error equation, $L_h\mathbf{e}_h^k = \mathbf{r}_h^k$.
2. Transfer $\mathbf{r}_h^k$ to the coarse grid: $\mathbf{r}_{2h}^k = I_h^{2h}\mathbf{r}_h^k$.
3. Solve $L_{2h}\mathbf{e}_{2h}^k = \mathbf{r}_{2h}^k$ on the coarse grid.

4. Transfer $\mathbf{e}_{2h}^k$ back to the fine grid: $\mathbf{e}_h^k = I_{2h}^h \mathbf{e}_{2h}^k$

5. Correct

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{e}_h^k$$
$$\mathbf{u}^{k+1} = \mathbf{u}_h^k + I_{2h}^h L_{2h}^{-1} I_h^{2h}(\mathbf{f} - L_h \mathbf{u}_h^k)$$
$$\mathbf{u}^{k+1} = (I - I_{2h}^h L_{2h}^{-1} I_h^{2h} L_h)\mathbf{u}_h^k + c$$

As an iteration method alone, this will not converge. It seems like it has a good shot if the error does not contain high-frequency components, so we could apply a few steps of smoothing before using coarse grid correction. However, interpolation introduces high-frequency error. Thus, we should apply a few steps of post-smoothing after correcting.

4.4 2-Grid Revised Scheme

Given $\mathbf{u}_h^k$, and approximation to $\mathbf{u}_h$.

1. Pre-Smoothing: apply ν_1 steps of smoothing.

2. Coarse Grid Correction:

   - Compute residual
   - Transfer residual to coarse grid
   - Solve error equation on coarse grid
   - Transfer coarse grid error to fine grid
   - Correct approximation

3. Post-Smoothing: apply ν_2 steps of smoothing to eliminate high-frequency errors introduced by coarse grid correction.

Let S be the smoothing operator. The 2-grid multi-grid iteration has the form

$$M = S^{\nu_2} K S^{\nu_2} = S^{\nu_2}(I - I_{2h}^h L_{2h}^{-1} I_h^{2h} L_h) S^{\nu_2}$$

Questions:

- What are the transfer operators, I_{2h}^h and I_h^{2h}?
- What is the coarse grid operator, L_{2h}^{-1}?
- What to choose for ν_1 and ν_2?
- How well does this work, $\rho(M)$? $\leftarrow$ We will show that $\rho(M) \ll 1$

4.5 Restriction Operators

The simplest restriction operator is injection. A better transfer operator is full-weighting. The stencils of the full-weighting operator are:

- 1-D: $I_h^{2h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$
- 2-D: $I_h^{2h} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

4.6 Interpolation Operators

The full-weighting restriction operators suggest the interpolation operators that we should use.

- 1-D: $I_{2h}^{h} = \frac{1}{2} \left] \begin{matrix} 1 & 2 & 1 \end{matrix} \right[$
- 2-D: $I_{2h}^{h} = \frac{1}{4} \left] \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix} \right[$

4.7 Coarse Grid Operator

We need to be able to solve the error equation, $L_{2h}\mathbf{e}_{2h} = \mathbf{r}_{2h}$, on the coarse grid. We have two options:

1. Discretize the coarse grid problem.
   - 1-D: $(L_{2h}\mathbf{e}_{2h})_j = \frac{1}{(2h)^2}\left(\mathbf{e}_{j-1} - 2\mathbf{e}_j + \mathbf{e}_{j+1}\right)$
   - 2-D: $\frac{1}{(2h)^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}$
2. Galerkin coarse grid operator

$$L_{2h} = I_h^{2h} L_h I_{2h}^h$$

$$\text{For 2-D: } L_{2h} = \frac{1}{(2h)^2}\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

4.8 Choosing ν_1 and ν_2

Typically, use $\nu_1 = 1$ and $\nu_2 = 1$ or $\nu_1 = 2$ and $\nu_2 = 1$.

4.9 From 2-Grid to Multigrid

Rather than solve the coarse grid error equation, $L_{2h}\mathbf{e}_{2h} = \mathbf{r}_{2h}$, we can use the same idea and approximate it by using a coarser grid. We will guess that $\mathbf{e}_{2h} = 0$ and apply multigrid.

<u>A 3-Grid Cycle</u>

- GSRB - Relax $L_h\mathbf{u}_h = \mathbf{f}_h$ ν_1 times
- Residual - Compute $\mathbf{r}_h = \mathbf{f}_h - L_h\mathbf{u}_h$
- Restrict - Compute $\mathbf{f}_{2h} = I_h^{2h}\mathbf{r}_h$
  - GSRB - Relax (pre-smooth) $L_{2h}\mathbf{u}_{2h} = \mathbf{f}_{2h}$ ν_1 times, with initial guess $\mathbf{u}_{2h} = 0$
  - Residual - Compute $\mathbf{r}_{2h} = \mathbf{f}_{2h} - L_{2h}\mathbf{u}_{2h}$
  - Restrict - Compute $\mathbf{f}_{4h} = I_{2h}^{4h}\mathbf{r}_{2h}$
    - GSRB - Solve $L_{4h}\mathbf{u}_{4h} = \mathbf{f}_{4h}$ (Note: the residual is 0.)
  - Interpolate - Correct $\mathbf{u}_{2h} := \mathbf{u}_{2h} + I_{4h}^{2h}\mathbf{u}_{4h}$
  - GSRB - Relax (post-smooth) $L_{2h}\mathbf{u}_{2h} = \mathbf{f}_{2h}$ ν_2 times, with initial guess $\mathbf{u}_{2h}$
- Interpolate - Correct $\mathbf{u}_h := \mathbf{u}_h + I_{2h}^h\mathbf{u}_{2h}$
- GSRB - Relax (post-smooth) $L_h\mathbf{u}_h = \mathbf{f}_h$ ν_2 times, with initial guess $\mathbf{u}_h$

5 Guy's Notes (Part 5)

5.1 Descent Methods

Consider the linear system

$$A\mathbf{e} = \mathbf{f}$$

where A is *symmetric positive definite.*

Symmetric:	$A^* = A$	$\left(A^T = A \text{ for real-valued matrices}\right)$
Positive Definite:	$\mathbf{y}^* A \mathbf{y} > 0 \;\forall\; \mathbf{y} \neq 0$	

Note: positive definite also means that all eigenvalues are strictly positive.

Because A is symmetric, all eigenvalues are real and the eigenvectors are orthogonal.

$$AQ = Q\Lambda \quad \text{and } Q^* = Q^{-1}$$
$$Q^* AQ = \Lambda \;\leftarrow\; \text{all elements are positive}$$

For the discrete Laplacian, $L\mathbf{u} = \mathbf{f}$, with Dirichlet boundary conditions, L is negative definite, so $-L$ is positive definite.

Consider the functional

$$\phi(\mathbf{u}) = \frac{1}{2}\mathbf{u}^* A\mathbf{u} - \mathbf{u}^* f$$

We want to find $\mathbf{u}$ that minimizes ϕ, so we look for critical points, i.e. set

$$\nabla\phi(\mathbf{u}) = 0$$
$$\nabla\phi = A\mathbf{u} - \mathbf{f}$$

We know that ϕ is minimized when $\nabla\phi = 0$ since $\nabla\nabla\phi = A$, i.e. all second derivatives are positive. How do we minimize $\phi(\mathbf{u})$? The method of steepest descent.

1. Guess $\mathbf{u}^k$
2. Obtain $\mathbf{u}^{k+1}$ by travelling in the direction of largest decrease, i.e. $-\nabla\phi(\mathbf{u}^k) = f - A\mathbf{u}^k = \mathbf{r}^k$
3. $\mathbf{u}^{k+1} = \mathbf{u}^k + \alpha_k \mathbf{r}^k \;\Rightarrow$ Choose α_k to minimize $\phi(\mathbf{u}^{k+1})$

$$\phi(\mathbf{u}^{k+1}) = \min_{\alpha_k} \phi(\mathbf{u}^k + \alpha_k \mathbf{r}^k)$$

$$\frac{d}{d\alpha_k}\phi(\mathbf{u}^k + \alpha_k \mathbf{r}^k) = 0$$

$$\text{minimized at} \qquad \alpha_k = \frac{\mathbf{r}^T\mathbf{r}}{\mathbf{r}^T A\mathbf{r}} = \frac{\langle \mathbf{r}, \mathbf{r}\rangle}{\langle \mathbf{r}, A\mathbf{r}\rangle}$$

5.2 Preliminary Steepest Descent Algorithm

- Initialize
- Loop in k
  - $\mathbf{r}^k = \mathbf{f} - A\mathbf{u}^k$
  - Check $\|\mathbf{r}^k\|$
  - $\alpha_k = \frac{\mathbf{r}^T\mathbf{r}}{\mathbf{r}^T A\mathbf{r}}$
  - $\mathbf{u}^{k+1} = \mathbf{u}^k + \alpha_k \mathbf{r}^k$
- End

5.3 More Efficient Steepest Descent Algorithm

Note that

$$\begin{aligned}
\mathbf{r}^{k+1} &= \mathbf{f} - A\mathbf{u}^{k+1} \\
&= \mathbf{f} - A\left(\mathbf{u}^k + \alpha_k \mathbf{r}^k\right) \\
&= \mathbf{r}^k - \alpha_k A\mathbf{r}^k
\end{aligned}$$

- Initialize $\mathbf{u}^0$ and $\mathbf{r}^0 = \mathbf{f} - A\mathbf{u}^0$
- Loop in k
  - $\omega = A\mathbf{r}^k$
  - $\alpha_k = \frac{\mathbf{r}^T\mathbf{r}}{\mathbf{r}^T A\mathbf{r}}$
  - $\mathbf{u}^{k+1} = \mathbf{u}^k + \alpha_k \mathbf{r}^k$
  - $\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k \omega^k$
  - Check $\|\mathbf{r}^k\|$
- End

For a 2×2 matrix, the level sets are ellipses. If the initial guess is on the major or minor axes, it will converge in 1 step. Otherwise, the number of steps depends on the ratio of the eigenvalues of the matrix. We define the *condition number* of a matrix as

$$\kappa(A) = \|A\|_2 \|A^{-1}\|_2$$

For a symmetric matrix,

$$\kappa(A) = \frac{\max\limits_j |\lambda_j|}{\min\limits_j |\lambda_j|}$$

For large κ, steepest descent is very slow.

5.4 Conjugate Gradient

Instead of descending in the direction of $\mathbf{r}^k$, choose a different search direction, $\mathbf{p}^k$. Then

$$\begin{aligned}
\mathbf{u}^{k+1} &= \mathbf{u}^k + \alpha_k \mathbf{p}^k \\
\alpha_k &= \frac{(\mathbf{p}^k)^T \mathbf{r}^k}{(\mathbf{p}^k)^T A\mathbf{p}^k}
\end{aligned}$$

Start with one step of steepest descent: $\mathbf{u}^0,\ \mathbf{r}^0,\ \mathbf{p}^0 = \mathbf{r}^0$. Then choose $\mathbf{p}_1$ such that

$$\mathbf{p}_1^T A\mathbf{p}_0 = 0$$

$\mathbf{p}_0$ and $\mathbf{p}_1$ are orthogonal in the inner product $\langle \mathbf{u}, \mathbf{v} \rangle_A = \mathbf{u}^T A\mathbf{v}$, and we say that they are *A-conjugate*. The main idea of conjugate gradient is to choose search directions that are A-conjugate to all past search directions.

5.5 Conjugate Gradient Algorithm

- Initialize $\mathbf{u}_0$, $\mathbf{r}_0 = \mathbf{f} - A\mathbf{u}_0$, $\mathbf{p}_0 = \mathbf{r}_0$
- Loop in k
  - $\omega = A\mathbf{p}_k$
  - $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\omega^T \mathbf{p}_k}$
  - $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k \mathbf{p}_k$
  - $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \omega$
  - Check $\|\mathbf{r}_{k+1}\| \Rightarrow$ if small enough, we're done
  - $\beta = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$
  - $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta \mathbf{p}_k$
- End loop

5.6 Analysis of Conjugate Gradient

Since A is symmetric positive definite, we define the A-norm as

$$\|\mathbf{u}\|_A = \left(\mathbf{u}^T A \mathbf{u}\right)^{1/2}$$

It is shown (Guy's notes page 11) that

$$\|\mathbf{e}^k\|_A \leq a \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \|\mathbf{e}^0\|_A$$

5.7 Preconditioning

Conjugate gradient can be sped up if the eigenvalues are clustered together. Rather than solve $A\mathbf{u} = \mathbf{f}$, we solve

$$M^{-1} A \mathbf{u} = M^{-1} \mathbf{f}$$

This will have the same solution, and if $M^{-1}A$ is well conditioned then conjugate gradient will converge faster.

How to choose M^{-1}

1. Must be symmetric positive definite
2. $M^{-1}A$ should be better conditioned
3. $M\mathbf{x} = \mathbf{b}$ should be easy to solve, i.e. M^{-1} is easy to apply $\Rightarrow$ ideally, M approximates A

For conjugate gradient, consider transforming the system as

$$B^{-1} A B^{-T} \left(B^T \mathbf{u}\right) = B^{-1} \mathbf{f}$$

$$\tilde{A} = B^{-1} A B^{-T} \qquad \tilde{\mathbf{u}} = B^T \mathbf{u} \qquad \tilde{\mathbf{f}} = B^{-1} \mathbf{f}$$

$$\tilde{A} \tilde{\mathbf{u}} = \tilde{\mathbf{f}}$$

This $\tilde{A}$ is symmetric positive definite, and $M = BB^T$. In terms of these transformed variables, the conjugate gradient method is

$$\mathbf{u}_k = B^{-T}\tilde{\mathbf{u}}_k, \quad \mathbf{p}_k = B^{-T}\tilde{\mathbf{p}}_k \quad \mathbf{r}_k = B\tilde{\mathbf{r}}_k$$
$$\tilde{\mathbf{p}}_{k+1} = \tilde{\mathbf{r}}_{k+1} + \beta\tilde{\mathbf{p}}_k$$
$$B^T\tilde{\mathbf{p}}_{k+1} = B^{-1}\tilde{\mathbf{r}}_{k+1} + \beta B^T\tilde{\mathbf{p}}_k$$
$$\mathbf{p}_{k+1} = B^{-T}B^{-1}\mathbf{r}_{k+1} + \beta\mathbf{p}_k$$
$$\mathbf{p}_{k+1} = M^{-1}\mathbf{r}_{k+1} + \beta\mathbf{p}_k$$

5.8 Preconditioned Conjugate Gradient Algorithm

- $\mathbf{r}_0 = \mathbf{f} - A\mathbf{u}_0$
- Solve $M\mathbf{z}_0 = \mathbf{r}_0$
- $\mathbf{p}_0 = \mathbf{z}_0$
- Loop in k
  - $\omega_k = A\mathbf{p}_k$
  - $\alpha_k = \frac{\mathbf{z}_k^T\mathbf{r}_k}{\mathbf{p}_k^T\omega_k}$
  - $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k\mathbf{p}_k$
  - $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\omega_k$
  - Check $\|\mathbf{r}_{k+1}\|$
  - Solve $M\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$
  - $\beta = \frac{\mathbf{z}_{k+1}^T\mathbf{r}_{k+1}}{\mathbf{z}_k^T\mathbf{r}_k}$
  - $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta\mathbf{p}_k$

5.9 Preconditioners

We would like M to approximate A. Some standard preconditioners for the Poisson equation include

- Symmetric SOR (SSOR) - sweep over the grid forwards and then backwards.
- Incomplete Cholesky Factorization - $A = LL^T$, where L is lower triangular.
  $A \approx \tilde{L}\tilde{L}^T \leftarrow$ don't allow too much fill-in
- Approximate Cholesky - construct $\tilde{L}$ to be easy to solve so that $A \approx \tilde{L}\tilde{L}^T$