Lecture 20
Scientific and Data Computing I

Mike Kirby
School of Computing
Lecture

- Wednesday – Iterative Methods: Steepest Descent and Conjugate Gradient
- Monday – Iterative Methods: MINRES and GMRES
- Wednesday – Practicum

Lecture 20 and 21: Based Upon Material From Professor Uri Ascher, Computer Science, Univ. of British Columbia
Why Not Use A Direct Method?

\[ Ax = b \]

A nonsingular \( n \times n \).

**Iterative method:** Starting from initial guess \( x_0 \), generate iterates \( x_1, x_2, \ldots, x_k, \ldots \), hopefully converging to solution \( x \).

But why not simply use LU decomposition, or

\[ x = A \, \text{“backslash”} \, b \]

in Matlab?

The matrix size \( n \) must be large, and the matrix \( A \) must be somehow special to consider iterative methods.
Why Not Use A Direct Method?

- LU decomposition (Gaussian elimination) may introduce fill-in.
- Want to take advantage when only a rough approximation to $x$ is required.
- Want to take advantage when a good $x_0$ approximating $x$ is known (warm start).
- Sometimes $A$ is not explicitly available (expensive), only matrix-vector products $Av$ for any vector $v$ can be efficiently carried out.
The Need To Discretize

Figure: A 2D cross-section of a 3D domain with a square grid added.
Poisson Problem

The **Poisson equation** is a partial differential equation that in its simplest form is defined on the open *unit square*, $0 < x, y < 1$, and reads

$$- \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = g(x, y).$$

Here $u(x, y)$ is the unknown function sought and $g(x, y)$ is a given *source*.

**Boundary conditions:** assume that the sought function $u(x, y)$ satisfies *homogeneous Dirichlet boundary conditions* along the entire boundary of the unit square, written as

$$u(x, 0) = u(x, 1) = u(0, y) = u(1, y) = 0.$$
Discretizing using centered differences for the partial derivatives we obtain the equations

\[ 4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = b_{i,j}, \quad 1 \leq i, j \leq N, \]
\[ u_{i,j} = 0 \quad \text{otherwise}, \]

where \( u_{i,j} \) is the value at the \((i, j)th\) node of a square planar grid, and \( b_{i,j} = h^2g(ih, jh) \) are given values at the same grid locations. Here \( h = 1/(N + 1) \) is the grid width.
The Discretization
A Linear System

Obviously, these are linear relations, so we can express them as a system of linear equations

$$Au = b,$$

where $u$ consists of the $n = N^2$ unknowns $\{u_{i,j}\}$ somehow organized as a vector, and $b$ is composed likewise from the values $\{b_{i,j}\}$.

The two-dimensional problem has features related to sparsity that are not seen in the one-dimensional problem; we will return to this point soon.
Ordering the Unknoows

How should we order the grid unknowns \( \{u_{i,j}\}_{i,j=1}^{N} \) into a vector \( \mathbf{u} \)? We can do this in many ways. A simple way is lexicographically, say by columns, which yields

\[
\mathbf{u} = \begin{pmatrix}
  u_{1,1} \\
  u_{2,1} \\
  \vdots \\
  u_{N,1} \\
  u_{1,2} \\
  u_{2,2} \\
  \vdots \\
  u_{N,2} \\
  u_{1,3} \\
  \vdots \\
  (u_{N,N})
\end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix}
  b_{1,1} \\
  b_{2,1} \\
  \vdots \\
  b_{N,1} \\
  b_{1,2} \\
  b_{2,2} \\
  \vdots \\
  b_{N,2} \\
  b_{1,3} \\
  \vdots \\
  (b_{N,N})
\end{pmatrix}.
\]
Block Tridiagonal Matrix

The $n \times n$ matrix $A$ (with $n = N^2$) has the form

$$A = \begin{pmatrix}
J & -I \\
-I & J & -I \\
& \ddots & \ddots & \ddots \\
& & -I & J & -I \\
& & & -I & J
\end{pmatrix},$$

where $J$ is the tridiagonal $N \times N$ matrix

$$J = \begin{pmatrix}
4 & -1 \\
-1 & 4 & -1 \\
& \ddots & \ddots & \ddots \\
& & -1 & 4 & -1 \\
& & & -1 & 4
\end{pmatrix},$$

and $I$ denotes the identity matrix of size $N$. 
Example

For instance, if $N = 3$ then

\[
A = \begin{pmatrix}
4 & -1 & 0 & | & -1 & 0 & 0 & | & 0 & 0 & 0 \\
-1 & 4 & -1 & | & 0 & -1 & 0 & | & 0 & 0 & 0 \\
0 & -1 & 4 & | & 0 & 0 & -1 & | & 0 & 0 & 0 \\
-1 & 0 & 0 & | & 4 & -1 & 0 & | & -1 & 0 & 0 \\
0 & -1 & 0 & | & -1 & 4 & -1 & | & 0 & -1 & 0 \\
0 & 0 & -1 & | & 0 & -1 & 4 & | & 0 & 0 & -1 \\
0 & 0 & 0 & | & -1 & 0 & 0 & | & 4 & -1 & 0 \\
0 & 0 & 0 & | & 0 & -1 & 0 & | & -1 & 4 & -1 \\
0 & 0 & 0 & | & 0 & 0 & -1 & | & 0 & -1 & 4 \\
\end{pmatrix}
\]

Note zero-diagonals within the band! In general there are $N - 2$ such diagonals, because neighboring unknowns $u_{i,j}$, $u_{i\pm1,j\pm1}$ are no longer consecutive in vector $u$. 
Sparsity Pattern

The **MATLAB** command `spy(A)` plots the nonzero structure of a matrix $A$. For $N = 10$, we obtain (note value of $\text{nz}$ at bottom)
There is a fundamental difference between the 1D and the 2D & 3D problems.

The 1D problem produces a tridiagonal matrix, for which Gaussian elimination (GE) requires a linear, i.e. $O(n)$, number of floating point operations (optimal order of complexity).

Higher-dimensional problems give rise to matrices that are sparse within the band, and complexity of GE is certainly not linear (it is $O(n^2)$ for 2D). This is in fact where iterative methods can be very effective.
Eigenvalues

For any size $N$ the matrix $A$ is diagonally dominant and nonsingular. It is also a Toeplitz matrix.

It can be verified directly that the $n = N^2$ eigenvalues of $A$ are given by

$$\lambda_{l,m} = 4 - 2(\cos(l \pi h) + \cos(m \pi h)), \quad 1 \leq l, m \leq N$$

(recall $(N + 1)h = 1$). Thus $\lambda_{l,m} > 0$ for all $1 \leq l, m \leq N$, and we see that the matrix $A$ is also positive definite.

Knowing the eigenvalues explicitly is helpful in understanding performance, convergence and accuracy issues related to linear solvers.
Jacobi and Gauss-Seidel Relaxation

Given $A$, denote by $D$ its diagonal part and by $E$ its lower triangular part:

$D = \text{diag}(\text{diag}(A))$ ; $E = \text{tril}(A)$

e.g.

$$
A = \begin{pmatrix}
7 & 3 & 1 \\
-3 & 10 & 2 \\
1 & 7 & -15
\end{pmatrix}, \quad \Rightarrow D = \begin{pmatrix}
7 & 0 & 0 \\
0 & 10 & 0 \\
0 & 0 & -15
\end{pmatrix}, \quad E = \begin{pmatrix}
7 & 0 & 0 \\
-3 & 10 & 0 \\
1 & 7 & -15
\end{pmatrix}
$$

Given iterates $x_0, x_1, x_2, \ldots, x_k, \ldots$ denote residual vector

$$
r_k = b - Ax_k.
$$

**Jacobi’s method**

$$
x_{k+1} = x_k + D^{-1}r_k
$$

**Gauss-Seidel method**

$$
x_{k+1} = x_k + E^{-1}r_k
$$
Jacobi Example

\begin{align*}
7x_1 + 3x_2 + x_3 &= 3 \\
-3x_1 + 10x_2 + 2x_3 &= 4 \\
x_1 + 7x_2 - 15x_3 &= 2 .
\end{align*}

Write as

\begin{align*}
7x_1 &= 3 - 3x_2 - x_3 \\
10x_2 &= 4 + 3x_1 - 2x_3 \\
-15x_3 &= 2 - x_1 - 7x_2 .
\end{align*}

Corresponds to a splitting $A = M - N$ with $M = D$. 
Jacobi Example (continued)

\[
7x_1 = 3 - 3x_2 - x_3 \\
10x_2 = 4 + 3x_1 - 2x_3 \\
-15x_3 = 2 - x_1 - 7x_2 .
\]

Evaluate right hand side at current iterate \( k \) and left hand side as unknown

\[
x_1^{(k+1)} = \frac{3 - 3x_2^{(k)} - x_3^{(k)}}{7} \\
x_2^{(k+1)} = \frac{4 + 3x_1^{(k)} - 2x_3^{(k)}}{10} \\
x_3^{(k+1)} = \frac{2 - x_1^{(k)} - 7x_2^{(k)}}{-15}
\]

for \( k = 0, 1, 2, \ldots \)
Gauss-Seidel Example

\[7x_1 = 3 - 3x_2 - x_3\]
\[10x_2 - 3x_1 = 4 - 2x_3\]
\[-15x_3 + x_1 + 7x_2 = 2.\]

Evaluate right hand side at current iterate \(k\) and left hand side as unknown

\[x_1^{(k+1)} = \frac{3 - 3x_2^{(k)} - x_3^{(k)}}{7}\]
\[x_2^{(k+1)} = \frac{4 + 3x_1^{(k+1)} - 2x_3^{(k)}}{10}\]
\[x_3^{(k+1)} = \frac{2 - x_1^{(k+1)} - 7x_2^{(k+1)}}{-15},\]

Corresponds to a splitting \(A = M - N\) with \(M = E\).
Properties of Jacobi and GS

- Jacobi more easily parallelized
- Jacobi matrix $M$ is symmetric
- GS converges whenever Jacobi converges and twice as fast
- Both methods are simple but slow. Used as building blocks for faster, more complex methods.
General Stationary Methods

Based on splitting $A = M - N$, the (fixed point) iterative method is

$$x_{k+1} = x_k + M^{-1}r_k.$$ 

Can be written equivalently as

$$Mx_{k+1} = Nx_k + b.$$ 

This is called stationary because $M$ is independent of iteration counter $k$. 
Over-Relaxation / Under-Relaxation

Let $x_{k+1}$ be obtained from $x_k$ by Jacobi or GS.
Modify it further by

$$x_{k+1} \leftarrow \omega x_{k+1} + (1 - \omega) x_k$$

where $\omega$ is a parameter.
Two useful variants:

- Based on Gauss-Seidel (GS) and $1 < \omega < 2$, obtain faster successive over-relaxation (SOR).
  $$x_{k+1} = x_k + \omega [(1 - \omega) D + \omega E]^{-1} r_k.$$  

- Based on Jacobi and $\omega \approx 0.8$, obtain slower under-relaxation which is a good smoother in some applications.
  $$x_{k+1} = x_k + \omega D^{-1} r_k.$$  

Example

\[ 4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = b_{i,j}, \quad 1 \leq i, j \leq N \]

On a $15 \times 15$ grid: $N = 15, \ n = 225 \ (NB \ \kappa(A) = n)$

<table>
<thead>
<tr>
<th>relaxation method</th>
<th>$\omega$</th>
<th>Error after 2 itns</th>
<th>Error after 20 itns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
<td>1</td>
<td>7.1e-2</td>
<td>5.4e-2</td>
</tr>
<tr>
<td>GS</td>
<td>1</td>
<td>6.9e-2</td>
<td>3.8e-2</td>
</tr>
<tr>
<td>SOR</td>
<td>1.69</td>
<td>5.6e-2</td>
<td>4.8e-4</td>
</tr>
</tbody>
</table>
Iteration Matrix

\[ Mx_{k+1} = Nx_k + b, \quad k = 0, 1, 2, \ldots. \]

For the exact solution, of course

\[ Mx = Nx + b, \quad N = M - A. \]

Define error in the \( k \) iteration

\[ e_k = x - x_k. \]

Then

\[ Me_{k+1} = (M - A)e_k, \quad k = 0, 1, 2, \ldots. \]

Let \( T = I - M^{-1}A \), the iteration matrix. Then

\[ e_{k+1} = Te_k = T(Te_{k-1}) = \cdots = T^{k+1}e_0. \]

Thus, convergence iff \( T^k \to 0 \).
Convergence of Stationary Methods

Method converges if and only if

$$\rho(T) < 1.$$

Recall spectral radius of square matrix $B$ with eigenvalues $\mu_1, \ldots, \mu_n$ is maximum eigenvalue magnitude

$$\rho(B) = \max_i |\mu_i|.$$

How fast does the iteration converge? Define rate of convergence

$$\text{rate} = -\log_{10} \rho(T).$$

Then number of iterations required to reduce error by factor 10 (i.e., gain a decimal digit) is

$$k \approx \frac{1}{\text{rate}}.$$

Example

\[4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = b_{i,j}, \quad 1 \leq i, j \leq N\]

Eigenvalues

\[\lambda_{l,m} = 4 - 2 \left( \cos \frac{l\pi}{N+1} + \cos \frac{m\pi}{N+1} \right), \quad 1 \leq l, m \leq N.\]

Consider Jacobi: \(M = D\). Then

\[T = I - D^{-1}A = I - \frac{1}{4}A.\]

So eigenvalues of iteration matrix are

\[\mu_{l,m} = 1 - \frac{1}{4} \lambda_{l,m} = \frac{1}{2} \left( \cos \frac{l\pi}{N+1} + \cos \frac{m\pi}{N+1} \right), \quad 1 \leq l, m \leq N.\]
Jacobi Method

\[ \mu_{l,m} = 1 - \frac{1}{4} \lambda_{l,m} = \frac{1}{2} \left( \cos \frac{l\pi}{N+1} + \cos \frac{m\pi}{N+1} \right), \quad 1 \leq l, m \leq N. \]

Spectral radius

\[ \rho(T) = \mu_{1,1} = \cos \frac{\pi}{N+1} \leq 1 - \frac{c}{n} \]

Rate of convergence

\[ \text{rate} = -\log \rho(T) \sim 1/n \]

Thus, \( O(n) \) iterations are required for error reduction by constant factor.
Relaxation Methods

- Jacobi requires $O(n)$ iterations hence $O(n^2)$ flops
- Gauss-Seidel is twice as fast, so same $O$
- SOR with optimal $\omega$ requires $O(N)$ iterations hence $O(n^{3/2})$ flops!
- The best possible method would require at least $O(n)$ flops.
Non-Stationary Methods

Try to take advantage of accumulating knowledge as the iteration proceeds.
Consider $M = M_k$, and more generally

$$x_{k+1} = x_k + \alpha_k p_k.$$ 

The scalar $\alpha_k > 0$ is step size.

Restrict consideration to cases where $A$ is symmetric positive definite.
Then $Ax = b$ is equivalent to

$$\min \phi(x) = \frac{1}{2} x^T Ax - b^T x$$
Gradient Descent / Steepest Descent

Gradient descent: take $p_k = r_k$.
How to choose step size?
Simple-minded, greedy approach: exact line search.

$$\min_{\alpha} \phi(x_k + \alpha r_k) = \frac{1}{2}(x_k + \alpha r_k)^T A(x_k + \alpha r_k) - b^T(x_k + \alpha r_k).$$

Critical point: differentiate wrt $\alpha$ and set to 0. Obtain

$$\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k} = \frac{\langle r_k, r_k \rangle}{\langle r_k, A r_k \rangle}.$$
Steepest Descent Algorithm

Given an initial guess $x_0$ and a tolerance $tol$, set at first $r_0 = b - Ax_0$, $\delta_0 = \langle r_0, r_0 \rangle$, $b_\delta = \langle b, b \rangle$ and $k = 0$. Then:

While $\delta_k > tol^2 \ b_\delta$,

\[
\begin{align*}
  s_k &= Ar_k \\
  \alpha_k &= \frac{\delta_k}{\langle r_k, s_k \rangle} \\
  x_{k+1} &= x_k + \alpha_k r_k \\
  r_{k+1} &= r_k - \alpha_k s_k \\
  \delta_{k+1} &= \langle r_{k+1}, r_{k+1} \rangle \\
  k &= k + 1.
\end{align*}
\]
Conjugate Gradient (CG) Algorithm

Given an initial guess $x_0$ and a tolerance $tol$, set at first $r_0 = b - Ax_0$, $\delta_0 = \langle r_0, r_0 \rangle$, $b_\delta = \langle b, b \rangle$, $k = 0$ and $p_0 = r_0$. Then:
While $\delta_k > tol^2 b_\delta$,

\[ s_k = Ap_k \]
\[ \alpha_k = \frac{\delta_k}{\langle p_k, s_k \rangle} \]
\[ x_{k+1} = x_k + \alpha_k p_k \]
\[ r_{k+1} = r_k - \alpha_k s_k \]
\[ \delta_{k+1} = \langle r_{k+1}, r_{k+1} \rangle \]
\[ p_{k+1} = r_{k+1} + \frac{\delta_{k+1}}{\delta_k} p_k \]
\[ k = k + 1. \]
For Next Time

• Reread Chapter 3, read Chapter 8 and get additional material
• Start working on Practicums
• Start working on Homework 5
• No Office Hours