Day 21: 8/18/2

Going back to $A\mathbf{x} = \mathbf{b}$.

Direct methods (LU, Cholesky, QR) are great, give exact answers.

But other ways to solve problems exist.

Iterative methods - we had to use for eigenvalues, but can use for $A\mathbf{x} = \mathbf{b}$.

Start from guess, improve it, then keep on improving till good enough.

Advantages:
- Can exploit a good guess (since just improving it)
- Can stop early (partway LU is worthless) for rough estimate
- Faster for enormous sparse matrices.

Disadvantages:
- May take many iterations
- Not "exact" (though not really a problem)
- May not converge.

Example 1.1: Where can huge matrices naturally arise?

**Ex. 1-d:** $-U'' = f$ (models forces $f$ on a string pulled taut)

$$U(x) = \frac{-2U(x_i) + U(x_{i+1})}{h^2}$$

on mesh $x_0, x_1, x_2, \ldots, x_n$

Matrix\[
\begin{bmatrix}
2 & -1 & 0 & & \\
-1 & 2 & -1 & \ddots & \\
0 & \ddots & \ddots & \ddots & -1 \\
& \ddots & \ddots & 2 & -1 \\
& & & -1 & 2
\end{bmatrix}
\]

- Halving mesh size doubles # of points, so doubles $h$.

So, LU decomp takes (naively) 8 times the work, but using banded have $O(m)$ flops, so double the work.

**Ex. 2-d:** $-\nabla^2 U = f$ (models forces $f$ on a drum head)

Same idea, but in 2 directions. Won't go into details (time) but to estimate at a pt., use that pt. and the 4 beside it.

So if $m \times m$ grid, $m = \frac{1}{h}$, we want to know value of $U$ at $m^2$ points, so have $m^2$ unknowns, so our matrix is $m^2 \times m^2$.

Q: If we halve the mesh size, how much do we increase the matrix size?

$m$ doubled, so our matrix is 4 times as big, in each direction.

Still super sparse - but banded Cholesky still takes $O(m^4)$ flops, so halving mesh multiplies work by sixteen.

Can't do large problems this way! Iterative methods are only $O(m^2)$, though with a large $C$.

**Ex. 3-d problems:** Similar, but now 3d. So, for $m = \frac{1}{h}$, have $m^3$ points in my 3d grid, so I have $m^3 \times m^3$ matrix! Naively, have semi-bandwidth of $m^2$.

So even that's bad!

$h$ halved $\Rightarrow$ $m$ doubled $\Rightarrow$ matrix 8 times bigger in each direction.

$\Rightarrow 128$ times the flops! unacceptable!

Iterative $\Rightarrow$ 8 times the flops. More reasonable.
8.2 Classical iterative methods.

Solving $Ax = b$. Start from a guess $x^{(0)}$, improve to $x^{(1)}$, $x^{(2)}$, etc. Decide we're good enough when, say, residual $\|Ax - b\|$ is sufficiently small.

Q2: Do we know that if the residual is small that our answer is accurate?

No, but it is if well-conditioned, using the inequality from earlier.

$Ax = b$ : system of eqns. $i^{th}$ eqn is

$$\sum a_{ij} x_j = b_i$$

can be rewritten as

$$x_i = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j \right)$$

Of course, if guess not exactly true. But, we could define the next guess using this:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

Most basic kind of iteration: Take eqn that should be exactly true, then change all but 1 unknown to a previous guess.

Jacobi's method. (Picard iteration in PDEs)

Q3: Do Jacobi's method on $\begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, initial guess $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$x_1^{(1)} = \frac{1}{a_{11}} \left( b_1 - a_{12} x_{2}^{(0)} \right) = \frac{1}{1} \left( 1 - \frac{2}{3} \cdot 1 \right) = \frac{1}{3}$

$x_2^{(1)} = \frac{1}{a_{22}} \left( b_2 - a_{21} x_{1}^{(0)} \right) = \frac{1}{1} \left( 1 - \frac{2}{3} \cdot 1 \right) = \frac{1}{3}$

$x^{(1)} = \begin{bmatrix} 1/3 \\ 1/3 \end{bmatrix}$

True soln is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$, so looking promising.

But, actually, won't converge! We'll talk about convergence in 8.5, but iterative methods don't always converge! $x^{(30)} \approx \begin{bmatrix} -4.36 \\ 1 \end{bmatrix}$

Each iteration is order $O(n^2)$ naively, for full matrix, $O(ns)$ for banded, but can often be improved if careful.

Naively, have to use matrices, but easy to adapt Jacobi method directly to diff eqn.

ex: general eqn is $U(x_{i+1}) - 4U(x_i) + U(x_{i-1}) = f(x_i)$

$$U(x_i) = -\frac{1}{4} \left[ f(x_i) - U(x_{i+1}) - U(x_{i-1}) \right]$$

So, for iteration, do

$$U(x_i)^{(k+1)} = -\frac{1}{4} \left[ f(x_i) - U(x_{i+1})^{(k)} - U(x_{i-1})^{(k)} \right]$$
Two more things: First, can think of Jacobi's as a matrix thing:

\[ X^{(k+1)} = D^{-1} [(D-A)X^{(k)} + b] \]

Don't actually program this way, but useful when we'll prove convergence.

Have smt like this for all these methods.

How to program?

function \([\mathbf{x}, \text{iterations}] = \text{Jacobi}(A, b, \mathbf{x}, \text{maxIter}, \text{tolerance})\)

\[
\begin{align*}
\text{n} &= \text{length}(b); \\
\text{iterations} &= \text{maxIter}; \\
\text{for } k = 1: \text{maxIter} & \text{ each iteration is own } k. \\
\text{oldx} &= \mathbf{x}; \\
\text{b} &= \mathbf{b}; \\
\text{for } i = 1:n & \text{ checks if approximation has settled down.} \\
\text{for } j = 1:i-1 & \\
\text{oldx}(i) &= \text{x}(i) - \text{A}(i,j) \times \text{oldx}(j); \\
\text{end} & \\
\text{for } j = i+1:n & \\
\text{x}(i) &= \text{x}(i) - \text{A}(i,j) \times \text{oldx}(j); \\
\text{end} \\
\text{x}(i) &= \text{x}(i)/\text{A}(i,i); \\
\text{end} \\
\text{if norm(oldx-x)/norm(x) < tolerance} & \text{ saves # of iter.} \\
\text{iterations} &= \text{K}; \\
\text{break} & \text{ if accurate, exits the } K \text{-for loop.} \\
\end{align*}
\]

Show \(A=\begin{bmatrix} 3 & 1 \\ 3 & 4 \end{bmatrix}, b=\begin{bmatrix} 7 \\ 8 \end{bmatrix}, x=\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \text{tol}=10^{-2}, \text{then } 10^{-8}\)

Gauss-Seidel (yes that Gauss) is a slight refinement.

Jacobi Formula: \(X^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i}^{n} a_{ij} x_j^{(k)} \right)\)

Idea: Jacobi always uses old guess.

But once you've calculated, say \(X^{(k+1)}\), why not use it instead? Gauss-Seidel is Jacobi, but always use most up to date info into \(X^{(k+1)}\).

Q1: \[
\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 10 \end{bmatrix}, \ x^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ do G-S - 1 step.} \\
\]

\[
\begin{align*}
x_1^{(1)} &= \frac{1}{a_{11}} (b_1 - a_{12} x_2^{(0)}) = 2 \quad \text{as w/ Jacobi}\ \\
x_2^{(1)} &= \frac{1}{a_{22}} (b_2 - a_{12} x_2^{(0)}) = \frac{1}{4} (10 - 3 \cdot 2) = 1 \\
\end{align*}
\]

Which happens to be correct!

More likely to converge than Jacobi method, still might not.

In fact, we picked a super lucky initial guess that converges, usually wouldn't!

(always converges for positive definite)
Order of corrections is important.
We did $X_1, X_2, X_3, \text{ etc.}$
Could go $X_n, X_{n-1} \ldots X_1$ backwards
"Symmetric" G-S is $X_1 \ldots X_n \ X_{n-1} \ldots X_1$.
Overall, usually converges in $\frac{1}{2}$ the iterations of Jacobi; same work per iteration.
But not inherently parallel:
For Jacobi, can send one entry to each processor.
But G-S, need updated one to find next one, so can't do at same time.
Fix: re-order -> red-black G-S.

Update at center point usually only depends on values of adjacent points. So, if those are updated, can do center one.
So color like chess board.
If you look, each black node only looks at red nodes to update, and vice versa.
So if you: update red, then update black, then red, then black,
can send each red node to its own processor, but still get convergence like G-S.

(SOR) Successive over-relaxation: the updates in G-S are right after each other - "successive,"
and they "relax" the system a bit at a time.

SOR is: Take the change you wanted to do to $X$, then over do it a bit.
Over correct. If you over-correct the right amount, converges faster.
\[
\text{G-S: } X^{(n+1)}_i = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x^{(n)}_j \right)
\]
\[
\text{SOR: } \hat{X} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x^{(n)}_j \right) \leftarrow \text{what G-S would do}
\]
\[
\delta = \hat{X} - X_i \leftarrow \text{change G-S would do}
\]
\[
X_i = X_i - \omega \delta \leftarrow \text{over correct.}
\]
$\omega$ is some $\#$ usually b/w 1 and 2, but could be < 1.

Q2: \[
\begin{bmatrix}
1 & 1 \\
4 & 6 \\
\end{bmatrix}
\begin{bmatrix}
X_1 \\
X_2 \\
\end{bmatrix}
= 
\begin{bmatrix}
3 \\
6 \\
\end{bmatrix}, \\
\]
\[
X^{(0)} = \begin{bmatrix}
1 \\
1 \\
\end{bmatrix}
\]
do 1 step of SOR, with $\omega = \frac{3}{2}$.

\[
X_i^{(0)} = \frac{1}{a_{ii}} \left( b_i - a_{i1} x_1^{(0)} \right) = \frac{1}{4} \left( 3 - 1 \cdot 1 \right) = 2 \quad \delta = \hat{X} - X_i^{(0)} = 2 - 1 = 1
\]
\[
X_1^{(1)} = X_i^{(0)} + \omega \cdot \delta = 1 + \frac{3}{2} \cdot 1 = \frac{5}{2}
\]

\[
X_2^{(1)} = \frac{1}{a_{22}} \left( b_1 - a_{12} x_2^{(0)} \right) = \frac{1}{4} \left( 6 - 1 \cdot \frac{5}{2} \right) = \frac{7}{8} \quad \delta = \hat{X} - X_2^{(0)} = \frac{7}{8} - 1 = -\frac{1}{8}
\]
\[
X_2^{(1)} = X_2^{(0)} + \omega \cdot \delta = 1 + \frac{3}{2} \cdot -\frac{1}{8} = \frac{13}{16}
\]
\[
S \cdot X^{(1)} = \begin{bmatrix}
5/2 \\
13/16 \\
\end{bmatrix}
\]
ugly by hand, but only marginally uglier on computer.
SOR: successive overrelaxation. a few last comments,
- over doing the chaos from Gauss-Seidel.
Slightly more work than G-S, but generally faster convergence.
However, depends on choice of \(w\).

ex: One the book did: G-S (\(w=1\)) took 2000 iterations,
\[\begin{align*}
  w &= 1.95 \quad \text{took 361} \\
  w &= 1.97 \quad \text{took 585}.
\end{align*}\]

For some programs (like the Poisson eqn), know optimum \(w\).
But often you won't. You'd have to guess.
The same variations of G-S (symmetric, red-black) both work,
pretty similarly.

In HW, I'll get you to explain why SOR (usually) converges faster.

### 8.3: Convergence of iterative methods

#### Splittings: Additive decomposition
\[
A = M - N, \quad M \text{ nonsingular.}
\]

\(M\) is splitting matrix/preconditioner.

Solving \(Ax = b\) equiv to \(Mx = Nx + b\)
So can make iterative: \(Mx^{(k+1)} = Nx^{(k)} + b\).
Or, equivalently, \(x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b\).

**Q1:** What if \(M = A\)? \(N = 0\), so get exact solve, \(x = M^{-1}b\).
Competing concerns: Want \(M \approx A\) so converges rapidly (i)
but want \(My = b\) to be easy to solve so iterations fast.

ex: Jacobi: \(M = D - \text{diagonal part of } A\).

Super easy to solve \(My = b\), but \(D \neq A\).

G-S: \(M = D - E\) lower triangular part of \(A\).

SOR: \(M = \frac{1}{\omega} D - E\)

Richardson's: \(M = \frac{1}{\omega} I, \quad N = \frac{1}{\omega} I - A, \quad x^{(k+1)} = (I - \omega A)x^{(k)} + \omega b\).

**Convergence:** How to use splittings to prove convergence?
Want \(0\) error: \(e^{(k)} = x - x^{(k)}\) (true sol'n)
don't know it, but we can use it.
Want \(\|e^{(k)}\| \to 0\) quickly, and at all:
True sol'n satisfies \(Ax = b\), and so \(Mx = Nx + b\).
Combine with \(Mx^{(k+1)} = Nx^{(k)} + b\), get
\[
M e^{(k+1)} = N e^{(k)}.
\]
So \( e^{(k+1)} = G e^{(k)} \) where \( G = M^{-1} N \). Each step, our error is multiplied by \( G \), so
\[
\epsilon^{(k)} = G^k \epsilon^{(0)}.
\]
Looks like power method!

Only, we want it to converge to \( 0 \).

Q2: Remembering the power method, under what conditions does \( G^k \epsilon^{(0)} \to 0 \)?
\[
\epsilon^{(0)} = C_1 v_1 + \ldots + C_n v_n \quad (\text{eigen vectors of } G), \quad \text{(depending on original guess)}.
\]
\[
\epsilon^{(k)} = C_1 \lambda^k v_1 + \ldots + C_n \lambda^k v_n \to 0
\]
if \( |\lambda_i| < 1 \), then \( \lambda_i^k \to 0 \). So need all \( |\lambda_i| < 1 \).

Q3: Would we expect it to converge (on a computer) if, say, \( \lambda_1 = 2 \), but \( C_1 = 0 \)?

No! Rounding error would make \( C_1 \neq 0 \), then the \( \lambda_1 = 2 \) would make it go to \( \infty \).

Def: Spectral radius \( \rho(b) = \max_{\lambda} |\lambda| \) (biggest eigenvalue size)

Thm: An iteration \( M x^{(k+1)} = N x^{(k)} + b \) converges for every initial guess if and only if the spectral radius of \( G = M^{-1} N = I - M^{-1} A \) is less than 1.

Each iteration, the error is reduced by at least \( \rho(b) \).

For big problems, hard to calculate. For model problem (Poisson on unit square) Jacobi method, \( \rho(b) = \cos(m h) \). As \( h \to 0 \) (smaller mesh), \( \rho \to 1 \), so slower convergence.

Q4: At each step, the norm of the error is approx. multiplied by \( \rho(b) \).

How many iterations does it take to reduce the error by a factor of \( \epsilon \)?
\[
\frac{\|e^{(k+1)}\|}{\|e^{(k)}\|} \approx \rho(b), \quad \text{so} \quad \frac{\|e^{(k+j)}\|}{\|e^{(k)}\|} \approx (\rho(b))^j
\]

Want
\[
\approx \frac{1}{\epsilon}, \quad \text{so} \quad j \ln \rho(b) = -1
\]

Def: \( R_{ao}(b) = -\ln(\rho(b)) \) asymptotic convergence rate.

As we just saw, \( R_{ao}(b) \) is \( \approx \) # of iterations to reduce error by factor of \( \epsilon \).