Welcome to Ma221! Lecture 22, Fall 24

Solving Poisson with FFT

Direct Method, not iterative

Start with 2D Poisson

$$T_N \cdot V + V \cdot T_N = F$$

T_N = 1D Poisson

$$= Z \Lambda Z^T$$

$$Z^T \left(Z \Lambda Z^T \cdot V + V \cdot Z \Lambda Z^T = F \right) Z$$

$$= \Lambda \cdot (Z^T V Z) + (Z^T V Z) \cdot \Lambda = (Z^T F Z)$$

$$= \Lambda \cdot V' + V' \cdot \Lambda = F'$$

Λ = diagonal $\Rightarrow$ diagonal Sylvester Eqn.

$$(\Lambda \cdot V')_{ij} + (V' \cdot \Lambda)_{ij} = F'_{ij}$$

$$\lambda_i \cdot V'_{ij} + V'_{ij} \cdot \lambda_j = F'_{ij}$$

$$(*) \quad (\lambda_i + \lambda_j) V'_{ij} = F'_{ij}$$

main costs

1) compute $F' = Z^T F Z$
2) solve (*) $O(n^2)$
3) compute $V = Z V' Z^T$

doing 1) and 3) by dense matmul costs $O(n^3)$

but can do it in $O(n^2 \log n)$ via FFT

Z = imaginary part of FFT

= fast sine transform

$(FFT)(i,j) = e^{2\pi\sqrt{-1}\, ij/n}$

Extend to 3D and higher dimensions via Kronecker products

$$(I_n \otimes T_n + T_n \otimes I_n)^{-1}$$

$$= \left((Z \otimes Z)(I_n \otimes \Lambda + \Lambda \otimes I_n)(Z^T \otimes Z^T)\right)^{-1}$$

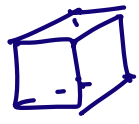
$$= (Z \otimes Z)(\text{diagonal})^{-1}(Z^T \otimes Z^T)$$

For 3D

$$\left((Z \otimes Z \otimes Z)(\text{diagonal})(Z^T \otimes Z^T \otimes Z^T)\right)^{-1}$$

$$= (Z \otimes Z \otimes Z)(\text{diagonal})^{-1}(Z^T \otimes Z^T \otimes Z^T)$$

↑ $\lambda_i + \lambda_j + \lambda_k$ for all triples (i,j,k)

in 3D: 

$Z \otimes Z \otimes Z$ means do FFTs in each direction, of each 1D slice

---

Summary Table of Performance of all algorithms for Poisson, 2D and 3D

Count #flops (all $O(\cdot)$ sense)

memory needed

parallel steps on "perfect" parallel computer with as

many processors as needed

processors needed

Entries sorted in 2 orders

from slowest to fastest on Poisson

(roughly) from most general to most specific (for Poisson)

"Explicit Inverse" = we precompute and store A^{-1} to solve $Ax=b$ ahead of time, only count work to multiply by A^{-1}

SOR(w) = Successive Overrelaxation

(w = tuning parameter)

SSOR(w)/ Chebyshev = Symmetric SOR with Chebyshev acceleration

FFT = Fast Fourier Transform

BCR = Block Cyclic Reduction

Lower Bound = assume 1 flop per component of answer

Sp MV = sparse matrix vector multiply

cost(Sp MV) = # nonzeros

For 2D Poisson on $n \times n$ mesh

$N = n^2$ = # unknowns

For 3D Poisson on $n \times n \times n$ mesh

$N = n^3$ = # unknowns

If table entries for 2D, 3D same only show one else, then 3D cost in parenthrses

Method	direct or Iterative	# flops	Mem	# Parallel steps	# Procs
Dense Cholesky any spd matrix	D	N^3	N^2	N	N^2
Explicit Inverse any matrix	D	N^2	N^2	$\log N$	N^2
Band Cholesky	D	N^2 $(N^{7/3})$	$N^{3/2}$ $(N^{5/3})$	N N	N $(N^{4/3})$
Jacobi (Splitting Method)	I	N^2 $(N^{5/3})$	N N	N $(N^{2/3})$	N N

works on any band matrix cost $= O(N \cdot bw^2)$

2D: $bw = n = N^{1/2}$

3D: $bw = n^2 = N^{2/3}$

$\#flops = O(\#flops\_per\_iteration \cdot \#iterations)$

$= O(\#flops(S_p MV) \cdot \#iterations)$

$= O(\#nnz(A) \cdot \#iterations)$

$= O(\#nnz(A) \cdot cond(A))$ for Poisson

works for any diagonally dominant matrix

building block for faster methods

Gauss-Seidel (Splitting Method)	I	N^2 ($N^{5/3}$)	N N	N ($N^{2/3}$)	N N

cost analysis like Jacobi, better constants

Works for any diagonally dominant matrix or s.p.d. matrix

Sparse Cholesky assumes best ordering of rows/columns	D	$N^{3/2}$ (N^2)	$N \cdot \log N$ ($N^{4/3}$)	$N^{1/2}$ ($N^{2/3}$)	N ($N^{4/3}$)

best ordering for Poisson:
nested dissection
$\Rightarrow$ bottleneck is dense Cholesky on trailing dense submatrix, of size $n \times n$ in 2D, $n^2 \times n^2$ in 3D

Conjugate Gradients (Krylov Subspace method (KSM))	I	$N^{3/2}$ ($N^{4/3}$)	N N	$N^{1/2} \log N$ ($N^{1/3} \log N$)	N N

$$\#\text{flops} = O(\#\text{flops\_per\_iteration} \cdot \#\text{iterations})$$
$$= O(\#\text{flops}(SpMV) \cdot \sqrt{\text{cond}(A)})$$

works on any s.p.d. matrix

SOR(ω) (Splitting Method)	I	$N^{3/2}$ ($N^{4/3}$)	N N	$N^{1/2}$ ($N^{1/3} \log N$)	N N

optimal ω hard to pick in general, known for Poisson

SSOR(ω)/ Chebyshev	I	$N^{5/4}$ ($N^{7/6}$)	N N	$N^{1/4}$ ($N^{1/6}$)	N N

$$\#\text{flops} = O(\#\text{flops}(SpMV) \cdot (\text{cond}(A))^{1/4})$$

choosing optimal ω requires knowing λ_{max} and λ_{min}

works for Poisson, not in general

works on any s.p.d. matrix

FFT	D	$N \cdot \log N$	N	$\log N$	N

works on Poisson — lower bound

BCR	D	$N \cdot \log N$	N	?	?

slightly more general than FFT

Multigrid	I	N	N	$\log^2 N$	N

many variants beyond Poisson

used for FEM, elliptic PDEs, ...

Lower Bound	N	N	$\log N$

Splitting Methods for $Ax = b$

Goal: Given an initial guess x_0 for solution of $Ax = b$, cheaply compute sequence $x_i \to x$

Def: Splitting of $A = M - K$, M nonsingular

$$Ax = b \Rightarrow Mx = Kx + b$$

compute x_{i+1} from x_i by solving

$$Mx_{i+1} = Kx_i + b$$

or $$x_{i+1} = \underbrace{M^{-1}K}_{R} x_i + \underbrace{M^{-1}b}_{c}$$

or (*) $$x_{i+1} = Rx_i + c$$

for this to work well, need

(1) x_i should converge to $x = A^{-1}b$

(2) Solving $Mx_{i+1} = Kx_i + b$ for x_{i+1} should be cheap

Lemma: Let $\|\cdot\|$ be any operator norm then if $\|R\| < 1$, (*) converges to $A^{-1}b$ for any x_0

Proof: Subtract $x = Rx + c$ from (*) to get

$$x_{i+1} - x = R(x_i - x)$$
$$= R^{i+1}(x_0 - x)$$

$$\|x_{i+1} - x\| \leq \|R\|^{i+1} \cdot \|x_0 - x\|$$
$$\to 0 \text{ if } \|R\| < 1$$

Def: The spectral radius of R is

$$\rho(R) = \max_{\lambda \text{ eval of } R} |\lambda|$$

Lemma: For all operator norms, $\rho(R) \leq \|R\|$.

For all R and $\varepsilon > 0$, there exists an operator norm $\|\cdot\|^*$ such that

$$\|R\|^* \leq \rho(R) + \varepsilon$$

Proof: To show $\rho(R) \le \|R\| = \max_{x \neq 0} \frac{\|Rx\|}{\|x\|}$

choose $x = $ evec for λ, $|\lambda| = \rho(R)$

so $\|R\| \ge \frac{\|Rx\|}{\|x\|} = \frac{\|\lambda x\|}{\|x\|} = |\lambda|$

Constructing $\|\cdot\|^*$: depends on R

use Jordan Form of R

$$S^{-1} R S = J = \begin{bmatrix} \square & & \\ & \square & \\ & & \ddots \end{bmatrix} \qquad \square = \begin{bmatrix} \lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix} \text{ or } [\lambda]$$

$$D = \begin{bmatrix} 1 & & & \\ & \varepsilon & & \\ & & \varepsilon^2 & \\ & & & \ddots \\ & & & & \varepsilon^{n-1} \end{bmatrix}, \quad D^{-1} J D \text{ multiplies } J(i,i+1) \text{ by } \varepsilon^{-i+1} \cdot \varepsilon^{i} = \varepsilon$$

$$J_\varepsilon = D^{-1} S^{-1} R S D = \begin{bmatrix} \lambda_1 & \varepsilon & & \\ & \ddots & \ddots & \\ & & \ddots & \varepsilon \\ & & & \lambda_n \end{bmatrix}$$

new vector norm $\|x\|^* = \|(SD)^{-1} x\|_\infty$

$$\|R\|^* = \max_{x \neq 0} \frac{\|Rx\|^*}{\|x\|^*} = \max_{x \neq 0} \frac{\|(SD)^{-1} R x\|_\infty}{\|\underbrace{(SD)^{-1} x}_{y}\|_\infty}$$

$$= \max_{y \neq 0} \frac{\|(SD)^{-1} R (SD)\, y\|_\infty}{\|y\|_\infty}$$

$$= \max_{y \neq 0} \frac{\|J_\varepsilon y\|_\infty}{\|y\|_\infty} = \|J_\varepsilon\|_\infty \le \rho(R) + \varepsilon$$

QED

Thm: $x_{i+1} = Rx_i + c$ converges to $A^{-1}b$ for all x_0 if and only if $\rho(R) < 1$

proof: If $\rho(R) \geq 1$ choose x_0 so that $x_0 - x =$ evec of R for largest eval λ

$\Rightarrow x_i - x = R^i(x_0 - x) = \lambda^i(x_0 - x)$

$|\lambda| \geq 1 \Rightarrow$ does not converge to 0

If $\rho(R) < 1$ use last lemma to construct operator norm $\|R\|^* < \rho(R) + \varepsilon$, choose ε so $\rho(R) + \varepsilon < 1$

$\Rightarrow$ convergence for any x_0 by earlier lemma Q.E.D

Describe Jacobi, Gauss-Seidel (GS) Successive Over relaxation (SOR)

$$A = \begin{bmatrix} D & -U' \\ -L' & \end{bmatrix} = D - L' - U' = D(I - L - U)$$

$L' = DL \quad U' = DU$

Jacobi: In words: (one step)

for $j = 1$ to n, pick $x_{i+1}(j)$ to exactly solve equation j

As a loop: for $j = 1$ to n

$$x_{i+1}(j) = \left(b_j - \sum_{k \neq j} A_{jk} x_i(k)\right) / A_{jj}$$

As a splitting:

$$D x_{i+1} = (L' + U') x_i + b$$

$$x_{i+1} = \underbrace{D^{-1}(L' + U')}_{R_j} x_i + D^{-1} b$$

$$A = M - K = D - (L' + U')$$

$$R_j = M^{-1} K = D^{-1}(L' + U') = L + U$$

For 2D Poisson: $T_N V + V \cdot T_N = h^2 F$

To get V_{i+1} from V_i

for $j = 1$ to n, for $k = 1$ to n

$$V_{i+1}(j,k) = (V_i(j-1,k) + V_i(j+1,k) + V_i(j,k-1) + V_i(j,k+1) + h^2 F(j,k)) / 4$$

= "average" of 4 nearest neighbors and right hand side

Gauss – Seidel

In words: improve on Jacobi by using most recently updated values of x

As a loop: for $j = 1$ to n

$$x_{i+1}(j) = \Big(b_j - \sum_{k<j} A(j,k)\cdot x_{i+1}(k) - \sum_{k>j} A(j,k)\cdot x_i(k)\Big)/A_{jj}$$

... use updated x_{i+1}

.. use old x_i

As a splitting:

$$D x_{i+1} = L' \cdot x_{i+1} + U' x_i + b_i$$

$$A = (D - L') - U' = M - K$$

$$R_{GS} = M^{-1}K = (D-L')^{-1}U'$$

$$= (D(I-L))^{-1}U'$$

$$= (I-L)^{-1}U$$

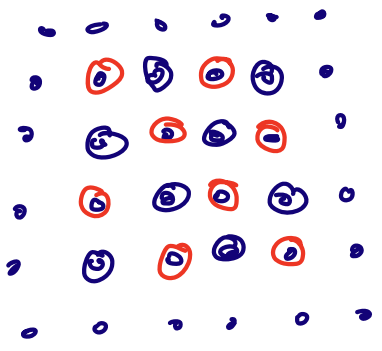
each step of GS is triangular solve with $D - L'$

In constrast to Jacobi, order of update matters

For 2D Poisson

"Natural Order": rowwise or columnwise update of V(j,k)

Red-Black Order



4x4 with boundaries

Red nodes (j+k even)

Black nodes (j+k odd)

Number all Red nodes before Black nodes

Red (or Black) nodes only have Black (or Red) neighbors

⇒ When updating Red, can update in any order, using old data at Black nodes. When updating Black, again in any order, all Red neighbors have updated data

for all Red (j,k) $(j+k$ even$)$

$$V_{i+1}(j,k) = (V_i(j-1,k) + V_i(j+1,k) + V_i(j,k-1) + V_i(j,k+1) + h^2 F(j,k)) / 4$$

old (Black) data

for all Black (j,k) $(j+k$ odd$)$

$$V_{i+1}(j,k) = (V_{i+1}(j-1,k) + V_{i+1}(j+1,k) + V_{i+1}(j,k-1) + V_{i+1}(j,k+1) + h^2 F(j,k)) / 4$$

updated (Red) data