Welcome back to Ma221! Lecture 32, Nov 6

Algorithm for evals only (or just a few)

use bisection, based on Sylvester's Thm:

inertia(A) = (# evals <0, # evals $=0$, # evals >0)

= inertia(XAX^T) any nonsingular X

count # evals in any $[x,y]$

by inertia of $A-x\cdot I$ and $A-y\cdot I$

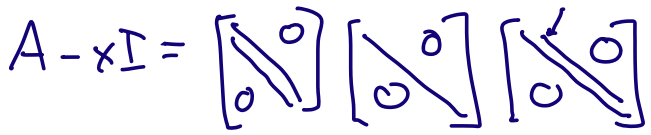
Choosing X so that $X(A-xI)X^T$ = diagonal D

Use Gaussian elimination without pivoting

$A-xI = LDL^T$

ok to divide by 0

still stable!

$A-xI =$ 

function: c = Neg_count(T,x)

... c = # evals of $T<x$

... = # $D_{ii}<0$ in $T-xI=LDL^T$

... diag$(T) = a_1, a_2, \ldots a_n$

... offdiag$(T) = b_1, b_2, \ldots b_{n-1}$

$c=0$, $b_0=0$, $d_0=1$

for $i=1$ to n

$d_i = (a_i - x) - b_{i-1}^2/d_{i-1}$

... obey parentheses!

if $d_{i-1}=0$

$\Rightarrow d_i = -\infty$

$\Rightarrow d_{i+1} = a_{i+1}-x$

if $d_i < 0$, $c = c+1$

end

We don't need $l_i = i^{th}$ off diagonal of L

$$(T - xI)(i, i+1) = b_i$$

$$= (LDL^T)(i, i+1) = d_i \cdot l_i$$

Replace usual inner loop of GE

$$d_i = a_i - x - l_{i-1}^2 \cdot d_{i-1}$$

by $\quad d_i = a_i - x - b_{i-1}^2 / d_{i-1}$

Thm: Assuming we don't divide by 0

Neg_count (T, x) is exactly correct for $T + E$ where $E = E^T$, tridiagonal,

$E(i,i) = 0 \quad |E(i, i+1)| \le 2.5\varepsilon |T(i, i+1)|$

proof: $d_i = \Big(\underbrace{(a_i - x)}_{\cdot(1+\delta_1)} - \underbrace{b_{i-1}^2 / d_{i-1}}_{\cdot(1+\delta_2)(1+\delta_3)} \Big)(1+\delta_4)$

$$d_i \underbrace{(1+\delta_4)^{-1}(1+\delta_1)^{-1}}_{F_i} = (a_i - x) - b_{i-1}^2 \underbrace{(1+\delta)}_{\text{several}} / d_{i-1}$$

G_i

$$d_i \cdot F_i = (a_i - x) - b_{i-1}^2 G_i / d_{i-1}$$

$$d_i \cdot F_i = (a_i - x) - b_{i-1}^2 G_i F_{i-1} / (d_{i-1} \cdot F_{i-1})$$

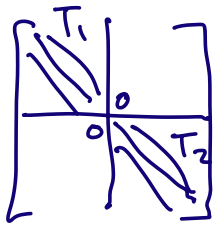
$$d_i' = (a_i - x) - b_{i-1}'^2 / d_{i-1}'$$

exact recurrence for $T + E$ where E changes b_{i-1} to b_{i-1}'

values of d_i' have same signs as d_i

$\Rightarrow$ count is correct for $T + E$

why OK to divide by 0?

if $b_i = 0$ 

solve eig(T_1) and eig(T_2) independently

Chap 6: Iterative Methods for $Ax=b$ (and $Ax=\lambda x$, Chap 7)

Model Problem: Poisson Equation (Lecture 15)

Goals: Contrast direct vs iterative methods

- for $Ax=b$ or least squares: Use iterative methods when direct methods too slow or use too much memory, or you need less accuracy
- for $Ax=\lambda x$ or SVD: same reasons as above, or you need only a few evals and evecs

Choosing best iterative method depends on mathematical structure of $A \Rightarrow$ large diversity of algs + software (links on class webpage)

Details for Model Problem: Poisson, arises in electrostatics, heat flow, quantum mechanics, fluid mechanics, graph theory

Present algorithms from simple to complicated, simple ones are building blocks for others.

Methods covered:

(1) "Splitting Methods" for $Ax=b$

"Split" $A = M - K$, M nonsingular

so $Ax = b$ becomes $Mx = Kx + b$

Iterate: Solve $Mx_{i+1} = Kx_i + b$ for x_{i+1}

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

$-(Mx = Kx + b)$

$Me_{i+1} = Ke_i$ $\quad e_i = x_i - x$

$\Rightarrow e_{i+1} = (M^{-1}K)e_i = (M^{-1}K)^{i+1} e_0$

How fast does $(M^{-1}K)^{i+1} \to 0$?

Consider 3 methods:

Jacobi, Gauss-Seidel,

Successive Over Relaxation (SOR)

eg Jacobi: $M = diag(A)$

(2) Krylov Subspace Methods (KSMs)

What can you do if all you have is a subroutine for $A \cdot x$ for any x?

Or if A so large, can only afford to do $A \cdot x$?

Overview of All KSMs:

Given x_0

1) Compute $x_1 = Ax_0$, $x_2 = Ax_1, \ldots x_k = A x_{k-1}$
   (or a similar set of vectors, spanning same subspace)
2) Choose a linear combination of these vectors that gives a "best" approximation of answer (for some def. of "best")
3) If approx sol. not good enough, increase k, repeat

Depending on
- How x_i are computed
- Properties of A (eg symmetric, spd, ..)
- the def of "best"

you get a large set of algorithms, all used in practice, will show decision tree, give details for Generalized Minimum Residual (GMRES, general A)

and Conjugate Gradients (CG, spd A)

Same idea for $Ax = \lambda x$ or SVD,
find "best" approx evec in same subspace
(Lanczos, Arnoldi, ...)

(3) Preconditioning: How fast do algs converge? Can we accelerate them?

Complicated, but in general depends on $\kappa(A)$, faster if $\kappa(A)$ smaller

if A ill-conditioned, seek matrix M

(1) multiplying by M cheap

(2) MA better conditioned than A

$\Rightarrow$ apply earlier methods to $MAx = Mb$

or $AM(M^{-1}x) = b$ if M^{-1} cheap too

Finding good preconditioner M depends on A: eg for s.p.d A, MA not s.p.d. but CG still works for $MAx = Mb$

(4) Multigrid: Most effective preconditioner, apply to Poisson'

Idea: If A arises from approximating some physical problem, then cheaper approximation available, eg using coarser grid, small approximation cheaper, apply idea recursively

Eg: Poisson on 2D mesh with $n \times n = n^2$ unknowns, cost $= O(n^2)$, optimal cost

(5) Domain Decomposition:
How to hybridize all above algs using different methods for different domains, ie. submatrices

Optimizing Communication! What if bottleneck is sparse $A \cdot x$, no data reuse
Lots of recent work on computing $x_1 = Ax_0, \ldots x_k = Ax_{k-1}$ but only reading A once from slow memory
(works if A sparse enough!)