1/8 - MAT 226B - Lecture 1

Homepage: Math.ucdavis.edu/~freund/226B/

OH: Weds 12:45-2:45pm

Grading:

5 homeworks 50%, Final project 50%

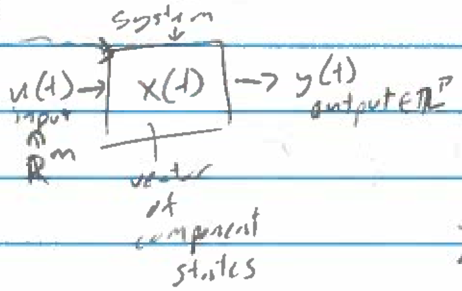
Reference: Matrix Computation - Goldhub?

Typical Matrix Computations

- Linear systems of eqns. $Ax=b$, A $n\times n$
- Eigenvalue problems $Ax=\lambda x$, A $n\times n$
- Linear dynamical systems

$$E\frac{d}{dt}x(t)=Ax(t)+Bu(t)$$

typically desc. by ODEs



A, E $n\times n$, B $n\times m$

+ initial cond'n's $x(t_0)=x_0$.

$m, p << n$

yields: $y(t)=C^T x(t)$, C $n\times p$

Reduced order model: Use SVD or something else to replace A, B, E w/ smaller

Large scale case: n is "large"

For problems arising in practice, the large matrices exhibit special structures
↳ e.g. sparsity

Def: A matrix computation problem is called large-scale if it can only be solved by methods that exploit the problem's special matrix structure.

special structures
- sparsity
- structured dense matrices

We will focus primarily on sparse structures

Sparsity

Def: A matrix $A = [a_{jk}] \in \mathbb{R}^{m \times n}$ is said to be sparse if only a small fraction of its entries a_{jk} are nonzero.

Ex: · Discretization of linear differential equations

$$\underset{\substack{\uparrow \\ \text{deriv. op}}}{L}\,\underset{\substack{\uparrow \\ \text{fn}}}{u} = \underset{\substack{\uparrow \\ \text{deriv. fn.}}}{f} \longrightarrow Ax = b$$

· Network problems, e.g. electrical circuits

· Web search (ranking problem)

⋮

Lecture 2 - MAT226B - 1/10/18

The graph of a (square) matrix

Let $A = [a_{jk}] \in \mathbb{R}^{n \times n}$. We associate with A a directed graph $G(A)$

nodes: $N = \{1, \dots, n\}$

edges: $E = \{(j,k) \mid j,k \in N \text{ s.t. } a_{jk} \neq 0\}$

Ex:

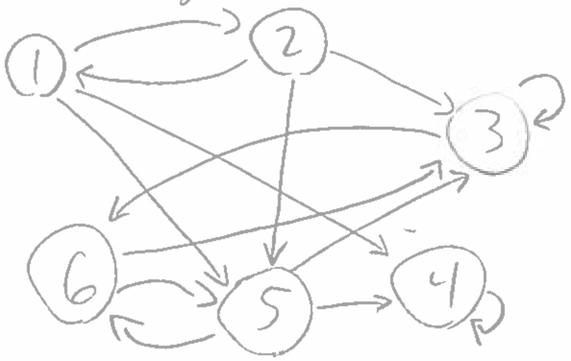
$$A = \begin{bmatrix} 0 & * & 0 & * & * & 0 \\ * & 0 & * & 0 & * & 0 \\ 0 & 0 & * & 0 & 0 & * \\ 0 & 0 & 0 & * & 0 & 0 \\ 0 & 0 & * & * & 0 & * \\ 0 & 0 & * & 0 & * & 0 \end{bmatrix} \in \mathbb{R}^{6 \times 6}$$

$G(A)$: $N = \{1,2,3,4,5,6\}$

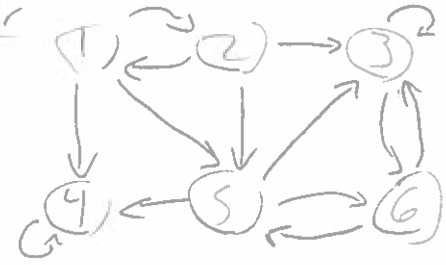
$E = \{(1,2), (1,4), (1,5), (2,1), (2,3), (2,5), (3,3), (3,6), (4,4), (5,3), (5,4), (5,6), (6,3), (6,5)\}$

$*$ = non-zero entry

my try



better



Ex: The WWW matrix: View the www as a graph G

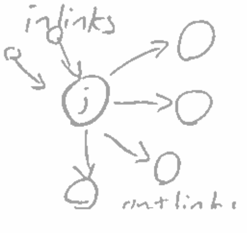
$N = \{1, 2, \dots, n\}$, n = # of websites that are visible to the world

$E = \{(j,k) \mid j,k \in N, j \neq k$ & there is a link from website j to website $k\}$

Corresponding sparse matrix $Q = [q_{jk}] \in \mathbb{R}^{n \times n}$ s.t. $q_{jk} \neq 0 \iff (j,k) \in E$ (hence $G(Q) = G$) Thus, sparsity structure of Q is the connectivity of www.

Values of $q_{jk} \neq 0$?

For each $j \in N$, the outdegree d_j of j is the # of edges (j,k).

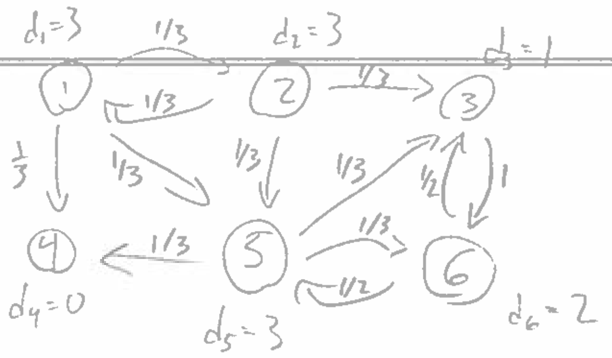


Classical choice:

$$q_{jk} := \begin{cases} 1/d_j & \text{if } (j,k) \in E \\ 0 & \text{o/w} \end{cases}$$

This comes from random walks!

Ex: (n=6)



$$\Rightarrow Q = \begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 1/3 & 0 \\ 1/3 & 0 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 \end{bmatrix}$$

- entries in each row (except row 4) sum up to 1.
- fix this: $A = [a_{jk}] \in \mathbb{R}^{n \times n}$ where $a_{jk} = \begin{cases} q_{jk} & \text{if } d_j > 0 \\ \frac{1}{n} & \text{if } d_j = 0 \end{cases}$

$$\Rightarrow A = \begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 1/3 & 0 \\ 1/3 & 0 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{1}{6} & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 \end{bmatrix}$$ is now a row-stochastic matrix (rows sum to 1)

In general: $A = Q + \frac{1}{n} v e^T$, where $e = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n$ & $v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$

this is easier to store than dense A!

w/ $v_j = \begin{cases} 1 & \text{if } d_j = 0 \\ 0 & \text{o/w} \end{cases}$

This A allows us to rank websites in the www.

Random clicks on links at times $i = 0, 1, 2, \ldots$

$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$ where $x_j^{(i)}$ is the fraction of users at time i staring at website j.

Game: $i \to i+1$. everyone clicks a random link

$$x_k^{(i+1)} = \sum_{j=1}^{n} a_{jk} x_j^{(i)}, \quad k = 1, 2, \ldots, n$$

$$\Rightarrow x^{(i+1)} = A^T x^{(i)} \Rightarrow x^{(i)} = (A^T)^i x^{(0)}$$

One can show: $\lim_{i \to \infty} x^{(i)} = x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$ gives the page rankings.

Lecture 3 - MAT226B - 1/12/18

Recall: Webpage rankings

$x_j^{(i)}$ = fraction of users staring at website j at time i

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} , \quad x^{(i+1)} = A^T x^{(i)}$$

Outgoing links

sparse connectivity matrix

Recall: Can store $A = Q + \frac{1}{n} v e^T$

Thm: $\lim_{i \to \infty} x^{(i)} = x$. There exists a limit point of $x^{(i)}$, which is called the Page rankings.

Note: This implies $x = A^T x$, i.e., x is an eigenvector of A^T with eigenvalue 1.

Def: A square $(n \times n)$ matrix A is row-stochastic if:

1) $a_{jk} \geq 0$ for all $j, k = 1, \ldots, n$

2) $\sum_{k=1}^{n} a_{jk} = 1$ for all $j = 1, \ldots, n$ $\iff$ $Ae = e$, $e = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n$

$\iff$ 1 is an eigenvalue w/ eigenvector e.

But we are interested in eigenvectors of A^T!

Luckily, recall that λ = eigval. of A $\iff$ λ = eigval. of A^T

☆ Can compute eigvector of A^T corr. to $\lambda = 1$.

☆ One can show that there is a corresponding eigenvector x s.t.

$$A^T x = x, \quad x_j \geq 0, \quad \text{and} \quad \|x\|_1 = 1$$

A class of structured dense matrices

Def: A matrix $T \in \mathbb{R}^{n \times n}$ of the form

$$T = \begin{bmatrix} t_0 & t_1 & t_2 & \cdots & \cdots & t_n \\ t_{-1} & t_0 & t_1 & t_2 & & \vdots \\ t_{-2} & t_{-1} & t_0 & t_1 & \ddots & \vdots \\ \vdots & & t_{-1} & \ddots & \ddots & t_2 \\ \vdots & & & \ddots & \ddots & t_1 \\ t_{-n} & \cdots & \cdots & t_{-2} & t_{-1} & t_0 \end{bmatrix}$$

is called a Toeplitz matrix.

(each diagonal is a single repeating value)

Notes: 1) $A = [a_{jk}] \in \mathbb{R}^{n \times n}$ is a Toeplitz matrix

$\iff$ $a_{jk} = t_{k-j}$ for all $j, k = 1, \ldots, n$

2) a $n \times n$ Toeplitz matrix is dense in general, but only need to store $2n-1$ values (as opposed to n^2)

Ex: Let ξ_j be a discrete-time stochastic process

$m_j = \mathbb{E}[\xi_j] =$ mean at time j.

Covariance matrix: $\sigma_{jk} = \mathbb{E}\left[(\xi_j - m_j)(\xi_k - m_k)\right]$

$\Sigma = [\sigma_{jk}] \in \mathbb{R}^{n \times n}$

The stochastic process is said to be weakly stationary (wide-sense stationary or covariance-stationary) if $\sigma_{jk} = t_{k-j}$ for all $j, k = 1, \ldots, n$

$\iff$ Σ is a Toeplitz matrix

Rmk: In this case, $\sigma_{jk} = \sigma_{kj}$, so Σ is symmetric as well.

($\Rightarrow$ $t_{k-j} = t_{j-k}$ $\Rightarrow$ $t_{-j} = t_j \ \forall j = 1, \ldots, n$)

Rmk: Mult. by a Toeplitz matrix can be done in $\mathcal{O}(n \log n)$ flops

(as opposed to $\mathcal{O}(n^2)$) => HW [illegible] on this

Lecture 4 - MAT226B - 1/17

Solution of Linear Systems.

Problem: Given $A \in \mathbb{R}^{n \times n}$, A nonsingular, $b \in \mathbb{R}^n$

solve $Ax = b$. (Standard soln: $A = LU$ Gaussian elim/LU factorization)

Cholesky factorization: for symmetric, positive-definite matrices A (SPD)

can factor $A = LL^T$

⊗ special form of LU factorization

Def: $A \in \mathbb{R}^{n \times n}$ is symmetric positive-definite (SPD) $(A \succ 0)$

if $A = A^T$, $x^T A x > 0 \quad \forall\, x \neq 0 \in \mathbb{R}^n$

Note: $A = [a_{jk}] \succ 0 \implies a_{jj} = e_j^T A e_j > 0$

Thm: Let $A = [a_{jk}] \in \mathbb{R}^{n \times n}$. Then:

1) For any nonsingular $M \in \mathbb{R}^{n \times n}$:

$$A \succ 0 \implies M^T A M \succ 0$$

2) $A \succ 0 \implies \tilde{A} = [a_{jk}]_{j,k \in I} \succ 0$ for any $I \subset \{1, 2, \dots, n\}$

(any subset of rows/cols still forms an SPD matrix)

3) $A \succ 0 \iff \exists!$ lower-triangular matrix $L \in \mathbb{R}^{n \times n}$ w/ $l_{jj} > 0 \ \forall j$

s.t. $A = LL^T$ (*) (Cholesky factorization)

S: This requires no pivoting!

Pf: 1) $A^T = A \Rightarrow M^TA^TM = M^TAM$

$$\Rightarrow (M^TAM)^T = M^TAM$$

$\Rightarrow M^TAM$ is symmetric.

~~M nonsingular allows $M^TAM = (M^TAM)^T = M^TA^TM \Rightarrow A^T = A$.~~

$$x^TAx = \underbrace{x^TM^{-T}}_{\tilde{x}^T}M^TAM\underbrace{M^{-1}x}_{\tilde{x}} = \tilde{x}^TM^TAM\tilde{x}$$

$\tilde{x} \neq 0 \Rightarrow x \neq 0$ since M nonsingular $\Rightarrow \tilde{x}^TM^TAM\tilde{x} > 0 \;\; \forall \tilde{x} \neq 0$

2) There is a permutation matrix P s.t. $P^TAP = \begin{bmatrix} \tilde{A} & * \\ * & * \end{bmatrix} \succ 0$ by 1)

$$\tilde{x}^T\tilde{A}\tilde{x} = \underset{\uparrow}{x}^TP^TAPx > 0 \quad \forall \tilde{x} \neq 0$$

$x = \begin{bmatrix} \tilde{x} \\ 0 \end{bmatrix}$

Note: $\tilde{A}$ symmetric since its the first block in P^TAP which is symmetric!

3) Induction on n

n=1: $A = [a_{11}] \succ 0 \Leftrightarrow a_{11} > 0 \Rightarrow l_{11} = \sqrt{a_{11}} > 0$, $L := [l_{11}]$

$$\Rightarrow A = LL^T = [l_{11}^2] = [a_{11}].$$

n-1 -> n: $A \in \mathbb{R}^{n\times n}$, $A \succ 0$. $A = \begin{bmatrix} a_{11} & w^T \\ w & A_{22} \end{bmatrix}$, where $a_{11} > 0$, $w := \begin{bmatrix} a_{21} \\ \vdots \\ a_{n1} \end{bmatrix}$

$A_{22} := [a_{jk}]_{2 \le j,k \le n}$

$$A = \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{w}{\sqrt{a_{11}}} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \hat{A}_{22} \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & w^T/\sqrt{a_{11}} \\ 0 & I \end{bmatrix}$$

where $\hat{A}_{22} := A_{22} - \frac{ww^T}{a_{11}} \in \mathbb{R}^{n-1\times n-1}$

Part 1 $\Rightarrow \begin{bmatrix} 1 & 0 \\ 0 & \hat{A}_{22} \end{bmatrix} \succ 0 \Rightarrow \hat{A}_{22} \succ 0$ by part 2 $\Rightarrow \hat{A}_{22} = \hat{L}L^T$ by IH

where $\hat{L} = \begin{bmatrix} l_{22} & & 0 \\ \vdots & \ddots & \\ l_{n2} & \cdots & l_{nn} \end{bmatrix}$ & $l_{jj} > 0$. Let $l_{11} := \sqrt{a_{11}}$ & $\begin{bmatrix} l_{21} \\ \vdots \\ l_{n1} \end{bmatrix} := \frac{w}{\sqrt{a_{11}}}$

Then $$A = \begin{bmatrix} l_{11} & 0 \\ l_{21} & \\ \vdots & I \\ l_{n1} & \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{L}\tilde{L}^T \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} \cdots l_{n1} \\ 0 & I \end{bmatrix}$$

$$\Rightarrow \quad A = \begin{bmatrix} l_{11} & 0 \\ l_{21} & \\ \vdots & \tilde{L} \\ l_{n1} & \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} \cdots l_{n1} \\ 0 & \tilde{L}^T \end{bmatrix} = LL^T.$$

□

Notes: 1) $\hat{A}_{22} = A_{22} - \dfrac{ww^T}{a_{11}} = A_{22} - \begin{bmatrix} l_{21} \\ \vdots \\ l_{n1} \end{bmatrix} \begin{bmatrix} l_{21} & \cdots & l_{n1} \end{bmatrix}$

2) $A = [a_{jk}] = A^T \Rightarrow$ we only need the a_{jk}'s with $j \geq k$.

Lecture 5 — 1/19/18.

Proof of Thm 3 $\Longleftrightarrow$ Cholesky factorization algorithm

In step k: · Finalize $\begin{bmatrix} l_{kk} \\ l_{k+1,k} \\ \\ l_{n,k} \end{bmatrix} = l_{k:n,k}$

· Update $\hat{A}_{k+1,k+1} = A_{k+1,k+1} - \begin{bmatrix} l_{k+1,k} \\ l_{k+2,k} \\ \vdots \\ l_{n,k} \end{bmatrix} \begin{bmatrix} l_{k+1,k} & \cdots & l_{n,k} \end{bmatrix}$

Initially: $$\begin{bmatrix} a_{11} & 0 & & & 0 \\ a_{21} & a_{22} & & & \vdots \\ \vdots & & \ddots & & \\ & & & \ddots & 0 \\ a_{n1} & & & & a_{nn} \end{bmatrix} \xrightarrow[k\text{ steps}]{} \begin{bmatrix} l_{11} & & & & & \\ l_{21} & l_{22} & & & & \\ \vdots & & \ddots & & & \\ \vdots & & & l_{kk} & & \\ \vdots & & & \vdots & \begin{bmatrix} l_{k+1,k+1} & & \\ \vdots & \ddots & \\ l_{n,k+1} & \cdots & l_{nn} \end{bmatrix} \\ l_{n1} & \cdots & \cdots & l_{nk} & \end{bmatrix}$$

MATLAB-like notation

$A = [a_{jk}] \in \mathbb{R}^{n\times n}$

$a_{j_1:j_2,k} := \begin{bmatrix} a_{j_1,k} \\ a_{j_1+1,k} \\ \vdots \\ a_{j_2,k} \end{bmatrix}$ for any $1 \le j_1 \le j_2 \le n$,
$1 \le k \le n$

Cholesky Factorization algorithm: "right-looking version"

Input: the elements a_{jk}, $j \ge k$, of $A = [a_{jk}] \in \mathbb{R}^{n\times n}$, $A \succ 0$

```
· Set $l_{jk} = a_{jk}$   $\forall j \ge k$, $j,k = 1,2,\dots,n$   ($L = \text{tril}(A)$)
· For $k = 1,\dots,n$
    · Set $l_{kk} = \sqrt{l_{kk}}$
    · Set $l_{k+1:n,k} = \frac{l_{k+1:n,k}}{l_{kk}}$
    · For $j = k+1,\dots,n$
        · Set $l_{j:n,j} = l_{j:n,j} - l_{j:n,k}\, l_{jk}$
```

Output: Cholesky factor $L = [l_{jk}] \in \mathbb{R}^{n\times n}$ of A.
↳ $A = LL^T$.

Work ~ $\mathcal{O}(n^3)$

This will do fine for small dense matrices but $\mathcal{O}(n^3)$ will catch up quick!

☆ This is a stable algorithm, but can have poor conditioning!

Cholesky factorization of sparse matrices

- Let $A \succ 0$ be sparse.

How can we ensure that the Cholesky factor L is also sparse?

Ex:

$$A = \begin{bmatrix} * & * & * & \cdots & * \\ * & * & 0 & \cdots & 0 \\ * & 0 & * & \ddots & \vdots \\ \vdots & \vdots & \ddots & * & 0 \\ * & 0 & \cdots & 0 & * \end{bmatrix} \succ 0 \Rightarrow L = \begin{bmatrix} * & 0 & \cdots & & 0 \\ * & * & & & \vdots \\ \vdots & * & * & & \\ & & \ddots & \ddots & 0 \\ * & * & \cdots & * & * \end{bmatrix}$$

"arrow"-matrix

<u>Chol.alg</u>: Loses sparsity at first step! Never regained.

<u>Remedy</u>: Reorder the rows & columns (use Pivoting)

$1, 2, 3, \ldots, n-1, n \longrightarrow n, 2, 3, \ldots, n-1, 1$

- $$P^T A P = \hat{A} = \begin{bmatrix} * & 0 & \cdots & 0 & * \\ 0 & * & \ddots & \vdots & * \\ \vdots & \ddots & * & & \vdots \\ & & \ddots & 0 & * \\ 0 & \cdots & 0 & * & * \\ * & * & \cdots & * & * \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & & 0 \\ \vdots & & 1 & & \vdots \\ & & & \ddots & 0 \\ 0 & & & 0 & 1 & \\ 1 & 0 & \cdots & & 0 \end{bmatrix} = P^T$$

<u>Apply Chol. factor alg</u>: $\hat{A} = \hat{L}\hat{L}^T$

$$\hat{L} = \begin{bmatrix} * & & & 0 \\ & * & & \\ & & \ddots & \\ 0 & & & * \\ * & * & \cdots & * & * \end{bmatrix}$$

$\Rightarrow$ preserves sparsity pattern completely!

Sparse Cholesky Factorization Algorithm:

Input: $A \succ 0$

1) (Symbolic factorization)
Determine a permutation matrix P s.t. the Cholesky factor L of

$$P^TAP = LL^T \quad \text{is sparse}$$

and determine the sparsity structure of L

2) (Numerical factorization)
Compute the entries of L
↳ same alg. as before, but modified for sparse structures.

Output: a permutation matrix P
and a lower-triangular matrix L s.t.

$$P^TAP = LL^T.$$

Sparse Cholesky factorization of A.
(not unique, many choices of P)

Optimal P to make L sparse as possible?
NP-hard problem!

Lecture 6 - 1/22/18 - MAT226B

- **Recall:** Sparse Cholesky factorization:

$$(*) \quad P^T A P = L L^T \qquad (A \in \mathbb{R}^{n\times n}, A \succ 0)$$

Notation: For $A = [a_{jk}] \in \mathbb{C}^{m\times n}$

$nnz(A) = \#$ of nonzero entries of A

Optimal choice of P in $(*)$: L s.t. $nnz(L)$ is minimum

Thm: The problem of determining an optimal P is NP-complete.

Consequence: In practice, only heuristics for finding a "good" P are feasible.

- The problem of finding P can be viewed as a graph problem

Let $A = A^T \succ 0$. Then $a_{jk} \neq 0 \Rightarrow a_{kj} \neq 0 \Rightarrow$ use undirected graph to represent A.

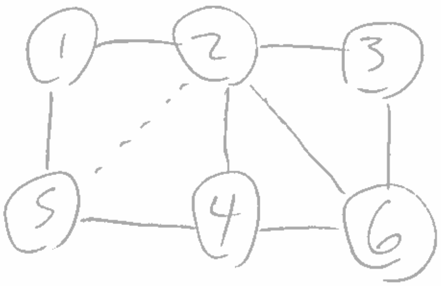
Also $A \succ 0 \Rightarrow a_{jj} > 0$.

Convention: For $A = A^T$, view A as undirected graph

For $A \succ 0$, omit the self-loops, i.e., edges (j,j) corr. to $a_{jj} > 0$

Ex:
$$A = \begin{bmatrix} * & * & 0 & 0 & * & 0 \\ * & * & * & * & 0 & * \\ 0 & * & * & 0 & 0 & * \\ 0 & * & 0 & * & * & * \\ * & 0 & 0 & * & * & 0 \\ 0 & * & * & * & 0 & * \end{bmatrix} \in \mathbb{R}^{6\times 6}$$

$G(A)$:



- First step of Cholesky factorization

$\circledast$ = fill-in element

$$\begin{bmatrix} * & & & & & \\ * & * & & & & \\ 0 & * & * & & & \\ 0 & * & 0 & * & & \\ * & \circledast & 0 & * & * & \\ 0 & * & * & * & 0 & * \end{bmatrix}$$

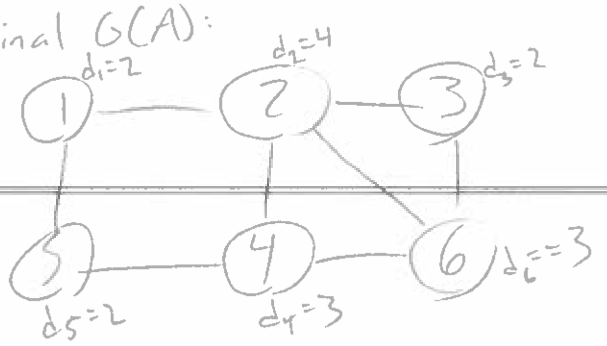
do stuff, fill in zero!

add - - - to graph

① connected to ② & ⑤ but ② & ⑤ not connected. Chol. factorization connects ② & ⑤!

Can we permute the nodes to minimize degree?

Original G(A):



First step of Chol factorization

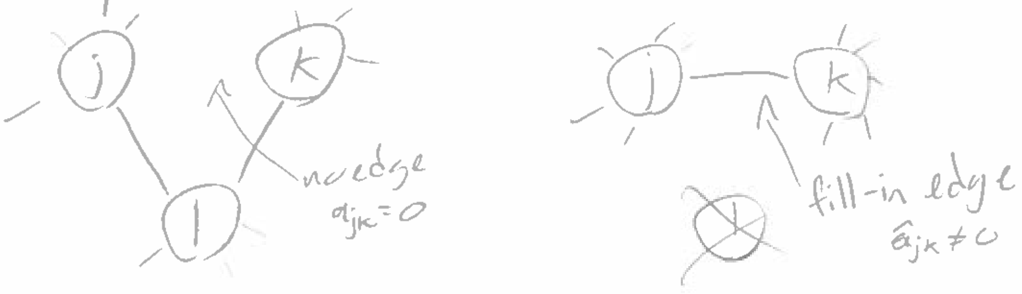
$$A_{22} = [a_{jk}]_{j,k=2,\dots,n} \longrightarrow \hat{A}_{22} = [\hat{a}_{jk}]_{j,k=2,\dots,n}$$

where $\hat{a}_{jk} = a_{jk} - \frac{a_{j1}a_{k1}}{\sqrt{a_{11}}}$

Generic case: $\hat{a}_{jk} \neq 0 \iff a_{jk} \neq 0$ or $(a_{j1} \neq 0$ and $a_{k1} \neq 0)$

Def: $\hat{a}_{jk} \neq 0$ is called a fill-in element if $a_{jk} = 0$ but $a_{j1} \neq 0$ and $a_{k1} \neq 0$.

Interpretation in terms of $G(A)$ and $G(\hat{A}_{22})$:



$G(A)$ $\quad$ $G(\hat{A}_{22})$

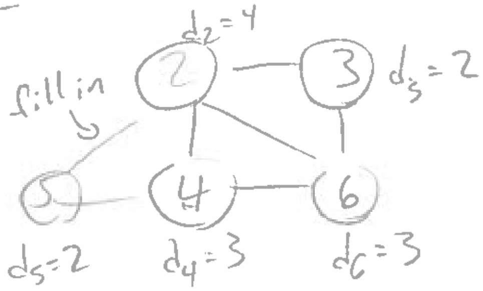
Minimum-degree Algorithm: d_j = degree of node j = #edges w/ node j.

- Order the nodes s.t. the node eliminated in the k^{th} step of Cholesky factorizati has minimum degree
  → (Tiebreaker) use the node w/ smallest index.

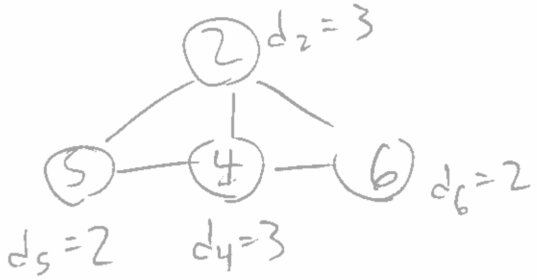
- Track order as we eliminate nodes to create P

- Use Chol. factor alg. on P^TAP.

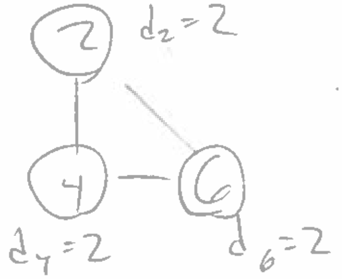
Step 1: Node 1 eliminated



Step 2: Node 3 eliminated



Step 3: Node 5 eliminated



Step 4: Node 2 elim.

Step 5: Node 4 elim.

Step 6: Node 6 elim.

New ordering: 1, 3, 5, 2, 4, 6 $\rightarrow P^TAP$

MATLAB: $p = [1\ 3\ 5\ 2\ 4\ 6]$

$A(p,p)$ is permuted A

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Output: $P^TAP = LL^T$, $L = \begin{bmatrix} * & & & & & \\ 0 & * & & & & \\ * & 0 & * & & & \\ * & * & \circledast & * & & \\ 0 & 0 & * & * & * & \\ 0 & * & 0 & * & * & * \end{bmatrix}$

$\boxed{\text{In this case: } nnz(L+L^T) = nnz(A)+2}$

General case:

Let $G = (N, E)$ be an undirected graph (w/ no self-loops) and $i \in N$.

$G_i = (N_i, E_i)$ where $N_i = N \setminus \{i\}$

$\& \quad E_i = \{(j,k) \in E \mid j \neq i \;\&\; k \neq i\}$

$\cap \{(j,k) \notin E \mid j \neq i, k \neq i, j \neq k, (j,i) \in E, (i,k) \in E\}$

Minimum deg. Alg.

Input: The undirected graph $G^0 = (N^0, E^0)$ assoc. w/ $A \in \mathbb{R}^{n \times n}$, $A \succ 0$.

For $k = 1, \dots, n$:

1) Det. a node $i_k \in N^{k-1}$ of minimum degree in G^{k-1}.
   ↳ (Tiebreaker) Pick smallest index i_k

2) $G^k = (N^k, E^k) = G^{k-1}_{i_k}$

Output: a reordering of the row & cols of A:

$1, 2, 3, \dots, n \longrightarrow i_1, i_2, \dots, i_n$

Notes: 1) In general, the minimum deg. ordering does NOT minimize the Sparsity of the Cholesky factor L, i.e., it is NOT optimal.

2) There are many other heuristics for reordering the rows & cols of A

Lecture 7 - 1/24/18

Sparse Matrices in MATLAB:

Let $A = [a_{jk}] \in \mathbb{C}^{m\times n}$ be sparse, w/ $nnz = nnz(A)$ entries $a_{jk} \neq 0$.

Coordinate (COO) storage format. $f(x,y)$, (x,y)

Store all $a_{jk} \neq 0$ and (j,k) in 3 arrays (vectors)

$$J = \begin{bmatrix} j_1 \\ j_2 \\ \vdots \\ j_{nnz} \end{bmatrix}, \quad K = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_{nnz} \end{bmatrix}, \quad VA = \begin{bmatrix} a_{j_1k_1} \\ a_{j_2k_2} \\ \vdots \\ a_{j_{nnz}k_{nnz}} \end{bmatrix}$$

★ any order is allowed, but J, K, VA have the same order.

Commands:

- A = sparse(J, K, VA) generates a sparse matrix $A \in \mathbb{R}^{m\times n}$ w/ $m = \max_i j_i$, $n = \max_i k_i$.
- A = sparse(J, K, VA, m, n) generates a sparse matrix $A \in \mathbb{R}^{m\times n}$ (error if $m < \max j_i$ or $n < \max k_i$)

Ex:

$$J = \begin{bmatrix} 2 \\ 1 \\ 5 \end{bmatrix}, \quad K = \begin{bmatrix} 3 \\ 4 \\ 2 \end{bmatrix}, \quad VA = \begin{bmatrix} -0.5 \\ 1.5 \\ 7 \end{bmatrix}$$

$$A = \text{sparse}(J, K, VA) \longrightarrow \left.\begin{matrix} (5,2) & 7 \\ (2,3) & -0.5 \\ (1,4) & 1.5 \end{matrix}\right\}$$ different order than input!

$$\text{full}(A) = \begin{bmatrix} 0 & 0 & 0 & 1.5 \\ 0 & 0 & -0.5 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{5\times 4}$$

Since no m, n given as input & $m = \max j_i = 5$, $n = \max k_i = 4$.

A = sparse(J, K, VA, 5, 5)

$$\text{full}(A) = \begin{bmatrix} 0 & 0 & 0 & 1.5 & 0 \\ 0 & 0 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \end{bmatrix}$$

☆ All usual matrix operations work for sparse matrices & sparsity is exploited:

$A+B$, $A*B$, $A*v$, A' (A^H), $A.'$ (A^T)

But not eig(A), unless $A=A^T$ & real

↳ obviously eig(full(A)) works, but doesnt exploit sparsity.

<u>Notes</u>: 1) Operations involving sparse & full matrices will yield a full matrix

<u>Ex</u>: $A \in \mathbb{R}^{n \times n}$ sparse

$B = A + I$ w/ $I = eye(n,n) \Rightarrow B$ full

$B = A + I$ w/ $I = speye(n,n) \Rightarrow B =$ sparse

2) operations involving <u>only</u> sparse matrices will yield a sparse matrix, w/ any new zero entries deleted.

<u>Ex</u>: A sparse, $A - A = 0$ = sparse w/ no elements listed

3) If possible, allocate storage beforehand.

A = spalloc(m,n,nzmax) = sparse m×n zero matrix set up to allow up to nzmax entries.

☆ Compressed Sparse Column-format (CSC) - stores data via columnwise

↳ Shortest way to store sparse matrices

Lecture 8 - 1/26/18

- $A = [a_{jk}] \in \mathbb{C}^{m\times n}$ sparse, $nnz = \#$ of $a_{jk} \neq 0$

COO format: J, K, VA

Compressed Sparse column format (CSC)

Ex: $A = \begin{bmatrix} 1.27 & 0 & -1.5 & 0 & 1.1 \\ 0 & 0.23 & 0 & 0 & -0.7 \\ 0 & 0 & 0 & 0 & 0.3 \\ -7.1 & 0 & 0 & 1.2 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 5}$, $nnz = 8$

$J = \begin{bmatrix} 1 \\ 4 \\ 2 \\ 1 \\ 4 \\ 1 \\ 2 \\ 3 \end{bmatrix}$ $I = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 5 \\ 6 \\ 9 \end{bmatrix}$ $VA = \begin{bmatrix} 1.27 \\ -7.1 \\ 0.23 \\ -1.5 \\ 1.2 \\ 1.1 \\ -0.7 \\ 0.3 \end{bmatrix}$

nnz+1

put there so that $9-6 = nnz$ in last column!

General case: CSC format stores 3 arrays:

VA: nnz values $a_{jk} \neq 0$, stored column by column, within each column order is arbitrary (since we store row # in J)

J: row indices j of the a_{jk}'s, same order as VA.

I: integer vector of length $n+1$:
$I(k)$ = pointer to the beginning of column k in J & VA
$I(n+1) = nnz+1$.

Notes: 1) $I(k+1) - I(k) = \#$ of nonzero a_{jk} in column k.

2) $I(k+1) = I(k) \iff$ the k^{th} column contains only zeros

3) For all $k = 1, \dots, n$

$a_{J(i)k} = VA(i)$, $i = I(k), I(k)+1, \dots, I(k+1)-1$

Ex: $A = \begin{bmatrix} 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.3 & 0 & 0 \\ 0 & 0 & 7.1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{6\times 5}$

$J = \begin{bmatrix} 1 \\ 5 \\ 6 \end{bmatrix}, \; VA = \begin{bmatrix} 0.5 \\ -1.3 \\ 7.1 \end{bmatrix}, \; I = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 4 \\ 4 \\ 4 \end{bmatrix}$

$\text{find}((I(2:n+1) - I(1:n)) = 0)$, finds zero-column indices k very fast.

LU factorization

let $A \in \mathbb{R}^{n\times n}$, A nonsingular

Special case: no pivoting needed

$$A = LU$$

where $L = \begin{bmatrix} 1 & & & 0 \\ \ell_{21} & 1 & & \\ \vdots & & \ddots & \\ \ell_{n1} & \cdots & \ell_{n,n-1} & 1 \end{bmatrix}$ is unit lower-triangular (ones on diag)

& $U = \begin{bmatrix} u_{11} & \cdots & \cdots & u_{1n} \\ & u_{22} & & \vdots \\ & & \ddots & \vdots \\ 0 & & & u_{nn} \end{bmatrix}$ is upper-triangular

Actual Algorithm: $\mathbb{R}^{n\times n} \ni A \to U$

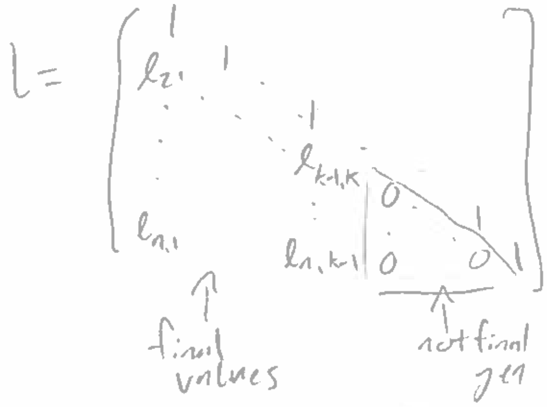
$$I \to L$$

After k-1 steps:

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & & \cdots & u_{1n} \\ & \ddots & & & & \vdots \\ 0 & & u_{k-1,k-1} & \cdots & & u_{k-1,n} \\ & & 0 & u_{k,k} & \cdots & u_{k,n} \\ & & \vdots & \boxed{u_{k+1,k} \; \cdots \; u_{k+1,n}} & & \\ & & \vdots & & & \end{bmatrix}$$

all final values

not final yet



<u>Alg</u>: (LU factorization w/o pivoting)

<u>Input</u>: $A \in \mathbb{R}^{n \times n}$

```
Set $U = A$, $L = I$
For $k = 1, 2, \ldots, n-1$
    If $u_{kk} = 0$: stop, pivoting is needed or A is singular.
    Set $l_{k+1:n,k} = \frac{1}{u_{kk}} u_{k+1:n,k}$ & $u_{k+1:n,k} = 0$
    For $j = k+1, k+2, \ldots, n$
        Set $u_{k+1:n,j} = u_{k+1:n,j} - u_{kj} \cdot l_{k+1:n,k}$
    end j
end k
```

<u>Output</u>: L & U s.t. $A = LU$.

<u>General case</u>: 1) $u_{kk} = 0$ can occur even if A nonsingular

2) numerical instability if $u_{kk} \neq 0$ but $|u_{kk}| << |u_{jk}|$ for some $k+1 \leq j \leq n$

<u>Remedies</u>. <u>Partial pivoting</u>: find $r \in \{k, k+1, \ldots, n\}$ s.t. $|u_{rk}| = \max_{i=k,\ldots,n} |u_{ik}|$

then swap row r & row k

→ Final factorization $PA = LU$

where P is a permutation matrix.

Complete pivoting will search all non-final values for max & then swap rows and columns

Lec 9

1/29

Beginning of k-th step of LU factorization

$$\begin{bmatrix} u_{kk} & \cdots & u_{kn} \\ \vdots & u_{rc} & \vdots \\ u_{nk} & \cdots & u_{nn} \end{bmatrix}$$

Partial pivoting:

$PA = LU$

Complete pivoting: find $r, c \in \{k, k+1, .., n\}$ such that

$$|u_{rc}| = \max_{i,l = k,k+1,..,n} |u_{il}|$$

and interchange rows r and k, along with colums c and k

Find Factorization $PAQ = LU$ where P and Q are permutation matrices

In both cases algorithm does not stop premturely ⇔ A is nonsingular

Sparse LU factorization

Let $A \in \mathbb{R}^{n \times n}$ be sparse

Goal: LU factorization

$PAQ = LU$ where L and U are sparse

Difficulty:

In general, P and Q cannot be determined by symbolic factorization alone, since we also need to pivot for stability

Instead: P and Q are determined during the actual factorization

Note: Symbolic factorization (minimum degree, ..)

can be used as a pre-processing step

$A, G(A) \longrightarrow$ permutation matrices P_0 and Q_0

Raw sparse LU factorization on reordered version $P_0 A Q_0$ of A

Step k of sparse LU factorization

Any entry $u_{il} \neq u$ of the submatrix

$$\begin{bmatrix} u_{kk} & \cdots & u_{kn} \\ \vdots & u_{il} & \vdots \\ u_{nk} & \cdots & u_{nn} \end{bmatrix} (=U^{(k)})$$

Is a candidate for the k-th pivot element

Markowitz Criterion

$r_i = r_i^{(k)} =$ # of nonzero entries in row $u_{i,k:n}$ in $U^{(k)}$

$c_l = c_l^{(k)} =$ # " " " column $u_{k:n,l}$ " "

If $u_{il} \neq 0$ is used as the pivot element, then in the worst case

$(r_i^{(k)} - 1)(c_l^{(k)} - 1)$ (*)

fill-in elements are created in step k

Basic Idea: choose i, l to minimize (*)

Ex

$$\begin{array}{c|cccccc} r_i \backslash c_\ell & 5 & 2 & 3 & 2 & 2 & 3 \\ \hline 3 & * & 0 & * & 0 & 0 & * \\ 4 & * & * & \circledast & * & 0 & * \\ 2 & 0 & \boxed{*} & * & 0 & 0 & 0 \\ 2 & * & 0 & \circledast & 0 & * & \circledast \\ 2 & * & 0 & \circledast & 0 & * & \circledast \\ 4 & * & 0 & * & * & 0 & * \end{array}$$

5 fill-in element

□ = pivot element

$\min_{i,\ell : u_{i\ell} \neq 0} (r_i - 1)(c_\ell - 1) = 1$

with equality for row 3 and column 2

$$\begin{bmatrix} \boxed{*} & 0 & * & 0 & 0 & 0 \\ * & * & \circledast & * & 0 & * \\ 0 & * & * & 0 & 0 & * \\ 0 & * & 0 & 0 & * & 0 \\ 0 & * & 0 & 0 & * & 0 \\ 0 & * & * & * & 0 & * \end{bmatrix}$$

only 1 fill-in elem!

General case

To guarantee numerical stability, we need to make sure $|u_{i\ell}|$ is not "too" small

Practical Markowitz criterion:

Among all $u_{i\ell} \neq 0$, $i, \ell \in \{k, k+1, \ldots, n\}$ with

$$|u_{i\ell}| \geq \alpha \max_{j \geq k} |u_{ij}| \quad \text{or} \quad |u_{i\ell}| \geq \max_{j \geq k} |u_{j\ell}|$$

choose $u_{i\ell}$ such that

choose $u_{i\ell}$ such that $(r_i - 1)(c_\ell - 1)$ is minimum

Tie-breaker: choose smallest i (if still not unique, smallest ℓ)

Here, $0 < \alpha \leq 1$ is a parameter. Typical choose $\alpha = 0.1$

Note There are many variants of this basic criterion

Lec 10

1/31

Matlab's sparse LU factorization

$A \in \mathbb{R}^{n \times n}$, sparse

$[L, U, P, Q, D] = lu(A)$ such that $P(D^{-1}A)Q = LU$

$\uparrow$ diagonal, positive diagonal entries

L, U, P, Q, D all sparse matrices

UMFPACK

Permutation matrices can be stored more compactly as vectors:

$[L, U, p, q, D] = lu(A, \text{'vector'})$ $\qquad I = \text{speye}(n,n)$

$p \leftrightarrow P$ $\qquad P = I(p, :)$

$q \leftrightarrow Q$ $\qquad Q = I(:, q) = (qi, :)$

One-line generation of qi:

$qi(q) = 1:n$

Note: $Q^{-1} = Q^T = I(:, qi) = I(q, :)$

Use to solve $Ax = b$

$$Ax = b \Leftrightarrow \underbrace{PD^{-1}A}_{LU}\overset{=I}{\overbrace{QQ^T}}x = PD^{-1}b$$

$$\Leftrightarrow \underbrace{LU(Q^Tx)}_{=c} = PD^{-1}b \qquad d = Q^Tx \Rightarrow x = Qd$$

Note: $Qd = d(qi)$

$$\Leftrightarrow \begin{cases} Lc = PD^{-1}b \\ Ud = c \\ x = Qd \end{cases}$$

Fast elliptic solvers

Large sparse systems

$Av = b$

often exhibit special structures that are exploited in their solution.

Standard ex: Poisson's equation (on simple domains)

One dimension

$$-\frac{d^2v(x)}{dx^2} = f(x), \quad 0 < x < 1 \qquad (*)$$

$$v(0) = v(1) = 0$$

Centered-difference approximation:

$$-\frac{d^2v(x)}{dx^2}\bigg|_{x=x_j} \approx \frac{2v_j - v_{j-1} - v_{j+1}}{h^2}$$

$v_0 = v(0) = 0$
$v_{m+1} = v(1) = 0$

where $v_j \approx v(x_j)$, $x_j = jh = \frac{j}{m+1}$, $j = 0, 1, \ldots, m+1$, $h = \frac{1}{m+1}$

$\Rightarrow$ approximate version of $(*)$

$$2v_j - v_{j-1} - v_{j+1} = h^2 f_j \quad , j=1,2,\ ,m$$

$$v_0 = v_{m+1} = 0$$

in linear equations for m unknowns $v_1, v_2, \ , v_m$

Compact form $\underbrace{\begin{bmatrix} 2 & -1 & & 0 \\ -1 & 2 & -1 & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix}}_{= T_m} \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix}}_{= v} = h^2 \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}}_{= f}$

$m \times m$

$\boxed{T_m z_\ell = \lambda_\ell z_\ell}$

$\Leftrightarrow T_m v = h^2 f$.

$v_j \approx v(x_j)$, $j = 1, 2, \dots, m$

$v_j - v(x_j) = O(h^2)$

2nd-order accuracy!

<u>Lemma</u>: The eigenvalues λ_ℓ and eigenvectors z_ℓ of T_m are given by

$$\lambda_\ell = 2(1 - \cos \pi h \ell), \quad z_\ell = \sqrt{2h} \begin{bmatrix} \sin(\pi h \ell) \\ \sin(2\pi h \ell) \\ \vdots \\ \sin(m\pi h \ell) \end{bmatrix}, \quad \ell = 1, 2, \ , m$$

$\left(h = \frac{1}{m+1}\right)$

The z_ℓs are orthonormal

$$z_\ell^T z_j = \begin{cases} 0 & \text{if } \ell \neq j \\ 1 & \text{〃 } \ell = j \end{cases}$$

Compact formulation.

$$T_m Z = Z \Lambda, \quad Z^T Z = I = Z Z^T$$

where $Z = [z_1 \ z_2 \ \cdots \ z_m] \in \mathbb{R}^{m \times m}$

and $\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & 0 \\ & & \ddots & \\ 0 & & & \lambda_m \end{bmatrix} \in \mathbb{R}^{m \times m}$

<u>Notes</u>: 1) $0 < \lambda_\ell < 4$

2) $T_m = Z \Lambda Z^T$, $\Lambda = Z^T T_m Z$

Lecture 11 - MAT226B - 2/2/18

- $T_m = \begin{bmatrix} 2 & -1 & & 0 \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{m\times m}, \quad h = \frac{1}{m+1}$

Poisson's eqn

$$\begin{cases} \Delta v = f \\ v = 0 \text{ on } \partial\Omega \end{cases}$$

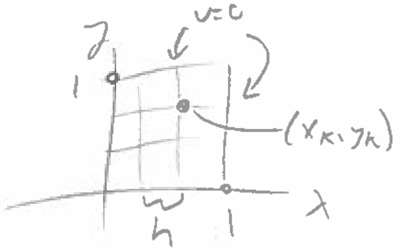
$T_m Z = Z\Lambda$, $Z^T Z = Z Z^T = I$, $\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \\ & & & \lambda_m \end{bmatrix}$.

Z = eigenvector matrix

Notes: 1) $0 < \lambda_\ell < 4$

2) $T_m = Z\Lambda Z^T$, $\Lambda = Z^T T_m Z$

2-D: $\begin{cases} -\dfrac{\partial^2 v(x,y)}{\partial x^2} - \dfrac{\partial^2 v(x,y)}{\partial y^2} = f(x,y), \quad 0 < x,y < 1 \\ v = 0 \text{ on boundary} \end{cases}$



- $(x_k, y_k) = (jh, kh)$, $f_{jk} = f(x_j, y_k)$

$v_{jk} \approx v(x_j, y_k)$

Centered-Diff. Approx:

$$4v_{jk} - v_{j-1,k} - v_{j+1,k} - v_{j,k-1} - v_{j,k+1} = h^2 f_{jk}, \quad j,k = 1,2,\dots,m \qquad (*)$$

where $v_{0k} = v_{m+1,k} = v_{j0} = v_{j,m+1} = 0$.

Compact formulation of $(*)$: $T_m V + V T_m = h^2 F$

where $F = [f_{jk}]$ & $V = [v_{jk}]_{j,k=1,\dots,m} \in \mathbb{R}^{m\times m}$ is unknown.

Recall from 228:

- $v = \begin{bmatrix} v_{11} \\ \vdots \\ v_{1m} \\ v_{21} \\ \vdots \\ v \end{bmatrix}, f = \begin{bmatrix} f_{11} \\ \vdots \\ f_{1m} \\ f_{21} \\ \vdots \\ f \end{bmatrix}$

$$\begin{bmatrix} T_m + 2I & -I & & & \\ -I & T_m + 2I & -I & & \\ & -I & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & T_m + 2I \end{bmatrix} v = h^2 f$$

Soln of $T_m V + V T_m = h^2 F$:

Recall: eigenvectors Z are equally-spaced sin fns => can use fft!

$$\underbrace{Z^T T_m}_{=\Lambda} \overbrace{Z Z^T}^{=I} \underbrace{V Z}_{=:V'} + \underbrace{Z^T V Z}_{=:V'} \underbrace{Z^T T_m Z}_{=\Lambda} = h^2 (\underbrace{Z^T F Z}_{=:F'})$$

$$\Rightarrow \Lambda V' + V' \Lambda = h^2 F'$$

entrywise: $\lambda_j v_{jk}' + v_{jk}' \lambda_k = h^2 f_{jk}'$, $j,k = 1, 2, \ldots, m$

$$\Rightarrow v_{jk}' = \frac{h^2 f_{jk}'}{\lambda_j + \lambda_k} \quad , \; j,k = 1, 2, \ldots m$$

↖ note nothing can go wrong since all $\lambda > 0$.

Alg for solving 2D Poisson's eqn:

1) Compute $F' = Z^T F Z$
2) Set $v_{jk}' = \dfrac{h^2 f_{jk}'}{\lambda_j + \lambda_k}$, $j,k = 1, \ldots, m$
3) Compute $V = Z V' Z^T$

Flop count: (naive implementation)

1) 2 matrix-matrix multiplications in $\mathbb{R}^{m \times m}$ $\approx 4m^3$ flops
2) $3m^2$ flops (2 for loops, 3 flops per it)
3) same as (1) $\approx 4m^3$ flops

$\Rightarrow$ Total $= \mathcal{O}(m^3)$ flops $\rightsquigarrow$ gain compared to $Av = h^2 f$ $\approx \mathcal{O}(m^6$ flo

$\uparrow \mathbb{R}^{n \times n}$, $n = m^2$

We can do better!

Recall: $Z = [z_{jk}]_{j,k=1,2,\dots,m} \in \mathbb{R}^{m\times m}$ eigenvector matrix of T_m

$$z_{jk} = \sqrt{2h}\,\sin(jk\pi h)$$

Notes 1) $Z = Z^T$

2) Z is related to the $(2m+2)\times(2m+2)$ DFT (discrete Fourier transform matrix)

$$\bar{\Phi} = [\varphi_{jk}]_{j,k=0,1,\dots,2m+1} \in \mathbb{C}^{(2m+2)\times(2m+2)}$$

$$\varphi_{jk} = e^{-\frac{jk\pi i}{m+1}} = \cos\left(\tfrac{jk\pi}{m+1}\right) - i\sin\left(\tfrac{jk\pi}{m+1}\right) \quad \rightsquigarrow \frac{1}{m+1} = h$$

$$= \cos(jk\pi h) - i\sin(jk\pi h)$$

In fact,

$$\bar{\Phi} = \begin{array}{c|c|c|c|} & \text{col } 0 & m \text{ cols} & m+1 \text{ cols} \\ \hline \text{row } 0 & * & * \ \cdot & \cdot \quad * \\ \hline m \text{ rows} & * \ \vdots & \hat{Z} & \\ \hline m+1 \text{ rows} & \vdots \ * & & \\ \hline \end{array}$$

where $\hat{Z} = [\hat{z}_{jk}],\ \hat{z}_{jk} = \varphi_{jk}$, $j,k = 1,\dots,m$

$$Z = -\sqrt{2h}\,\mathrm{Im}(\hat{Z})$$

Prop: For any $v \in \mathbb{R}^m$, the matrix-vector product

$$w = Zv = Z^T v \quad \text{can be computed as following}$$

1) $\hat{v} = \begin{bmatrix} 0 \\ v \\ 0 \end{bmatrix} \begin{matrix} \}1 \\ \}m \\ \}m+1 \end{matrix} \in \mathbb{R}^{2m+2}$ $\wedge$ $Z = [\sin(jk\pi h)]_{j,k=1,\dots,m} \in \mathbb{R}^{m\times m}$

2) Compute $\tilde{w} = \bar{\Phi}\tilde{x}$, where $\bar{\Phi}$ is the DFT matrix (use fft)

3) Partition $\tilde{w}$ as $\tilde{w} = \begin{bmatrix} * \\ \hat{w} \\ * \\ * \\ * \\ * \end{bmatrix} \begin{matrix} \}1 \\ \}m \\ \\ \}m+1 \\ \\ \end{matrix}$

4) Set $w = -\sqrt{2h}\,\mathrm{Im}(\hat{w})$

Lecture 12 - 2/5 (Week 5)

Cor. The product $w = Zv = Z^T v$ can be computed with $\mathcal{O}(m \log m)$ flops using the DFT

Notes. 1) In MATLAB, $\hat{w} = \Phi \hat{v} \iff \hat{w} = \text{fft}(\hat{v})$

2) For any $F \in \mathbb{R}^{m \times m}$, we can compute

$$F' = Z^T F Z = Z (ZF)^T$$

w/ $\mathcal{O}(m^2 \log m)$ by using cor. above m times $= \mathcal{O}(n \log n)$ flops

3) With these DFTs, the alg. for 2D Poisson's eqn. requires $\mathcal{O}(n \log n)$ flops

This is almost optimal. An optimal algorithm (multigrid) requires $\mathcal{O}(n)$ flops.

This concludes our discussion of direct methods for linear systems

↳ exact (theoretical) solutions

→ GE, LU factorization, Cholesky factorization, elliptic solvers

Next type of methods are Iterative methods.

↳ Krylov subspace methods, multigrid, ...

Iterative Methods for soln. of linear eqns:

$Ax = b$, $A \in \mathbb{R}^{n\times n}$ nonsingular, $b \in \mathbb{R}^n$

Krylov Subspace Methods

If A is sparse with nnz potentially nonzero entries, the computing of $y = Ax$ for any $x \in \mathbb{R}^n$ is cheap, at most 2nnz flop.

The Conjugate Gradient (CG) method:

(in exact arithmetic, this converges exactly in n iterations)

Assumption: $A \succ 0$

CG is an iterative method. $x_0 \in \mathbb{R}^n \to x_1 \to \dots \to x_k \in \mathbb{R}^n \to \dots$

x_k : k^{th} iterate

$r_k := b - Ax_k$: corresponding residual vector

Note: $r_k = 0 \iff x_k = A^{-1}b =: x^*$ is the soln of $Ax = b$

Goal: construct x_k s.t. $\|r_k\|$ is small, $\|\cdot\|$ is an appropriate norm in $\mathbb{R}^n$

✩ a very specific norm underlies the convergence of CG:

$A \succ 0 \Rightarrow \|x\|_A := \sqrt{x^T A x}$ is a norm in $\mathbb{R}^n$

$\hookrightarrow A^{-1} \succ 0 \Rightarrow \|r\|_{A^{-1}} := \sqrt{r^T A^{-1} r}$ is a norm in $\mathbb{R}^n$

Note: $\|x\|_A = \|Ax\|_{A^{-1}}$ for all $x \in \mathbb{R}^n$.

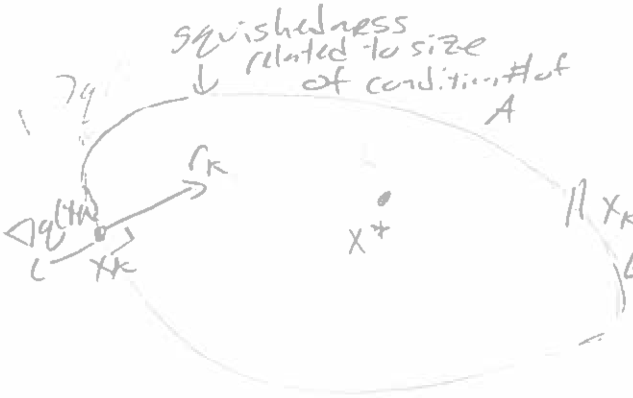
The CG method is based on the error norm

$$\|x^* - x_k\|_A = \|Ax^* - Ax_k\|_{A^{-1}} = \|b - Ax_k\|_{A^{-1}} = \|r_k\|_{A^{-1}}$$

CG method

Suppose have $x_k \in \mathbb{R}^n$ and want to construct

$x_{k+1} = x_k + \alpha_k p_k$, where $p_k \in \mathbb{R}^n$, $p_k \neq 0$ is a <u>search direction</u>



& $\alpha_k \in \mathbb{R}$, $\alpha_k > 0$ is a step size.

$\|x_k - x^*\|_A = \|x^* - x\|_A \iff q(x_k) = q(x)$.

level curves are ellipses since A distorts Euclidean distances

w/ $q(x) := \frac{1}{2}\|x^* - x\|_A^2 = \frac{1}{2}(x^* - x)^T A (x^* - x)$

<u>Steepest Descent</u>: choose $p_k = -\nabla q(x_k) = A(x^* - x_k) = r_k$

↳ inefficient! r_k doesn't point towards center in an ellipse! Will bounce around too much for moderately elliptical spaces (moderate condition # of A)!

Do better! Use <u>Conjugate gradient</u> instead of gradient:

$$\left[\begin{array}{l} p_0 = r_0 = b - Ax_0 \\ \text{For } k = 0, 1, \ldots \\ p_{k+1} = r_{k+1} + \beta_{k+1} p_k, \quad \beta_{k+1} = \dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \end{array}\right.$$

One can show that: $p_j^T A p_k = 0$ for $j \neq k$

Will converge in exact arithmetic in exactly n steps for $x \in \mathbb{R}^n$.

Lecture 13 - 2/7

Notes on CG: 1) choice of $\beta_{k+1} \iff p_{k+1}^T A p_k = 0$

2) choice of α_k : $q(x_k + \alpha_k p_k) = \min_{\alpha} q(x_k + \alpha p_k)$

CG-method Algorithm: (best formulas for numerical implementation)

Input: · the routine to compute $z = Ap$ for any $p \in \mathbb{R}^n$ $(A \in \mathbb{R}^{n \times n}, A \succ 0)$, · convergence tolerance $tol > 0$

· $b \in \mathbb{R}^n$

· $x_0 \in \mathbb{R}^n$ (arbitrary)

```
Set $r_0 = b - A x_0$
    $p_0 = r_0$
for $k = 0, 1, 2, \dots$ :
    If $\frac{\|r_k\|_2}{\|r_0\|_2} \le tol$, stop; $x_k \approx A^{-1} b$
    Set $z = A p_k$
    $\alpha_k = \frac{r_k^T r_k}{p_k^T z} \left( = \frac{\|r_k\|_2^2}{p_k^T A p_k} > 0 \text{ for } A \succ 0 \right)$
    $x_{k+1} = x_k + \alpha_k p_k$
    $r_{k+1} = r_k - \alpha_k z \quad (= r_k - \alpha_k A p_k)$
    $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \left( = \frac{\|r_{k+1}\|_2^2}{\|r_k\|_2^2} \right)$
    $p_{k+1} = r_{k+1} + \beta_{k+1} p_k$
end
```

Notes: 1) In exact arithmetic: $r_k = b - A x_k$, $k = 0, 1, 2, \dots$

2) Each k^{th} iteration involves the following operations:

1-Matrix-vector product $z = A p_k$ . . . . .

Notes: 2) Each k^{th} iteration involves the following operations:

1 matrix-vector product $z = Ap_k$

2 inner-products in $\mathbb{R}^n$: $p_k^T z$ and $r_{k+1}^T r_{k+1}$ ~ 2n-flops

3 SAXPYs: $x_{k+1} = x_k + \alpha_k p_k$ (Scalar-vector mult. followed by vector-vector addition w/ vector-override)

$r_{k+1} = r_k - \alpha_k z$

$p_{k+1} = r_{k+1} + \beta_{k+1} p_k$

Storage: 4 vectors of length n: x, r, p, z

3) CG is a Krylov subspace method

$$x_1 = x_0 + \alpha_0 r_0 \in x_0 + \text{span}\{r_0\}$$

$$x_2 = x_1 + \alpha_1 p_1 = x_0 + \alpha_0 r_0 + \alpha_1(\underset{\uparrow\, r_1 = r_0 - \alpha_0 A r_0}{r_1} + \beta_1 r_0)$$

$$= x_0 + (\alpha_0 + \alpha_1 + \beta_1) r_0 - \alpha_0 \alpha_1 A r_0$$

$$\in x_0 + \text{span}\{r_0, Ar_0\}$$

$$\vdots$$

$$x_k \in x_0 + \underbrace{\text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0\}}_{=: K_k(A, r_0) = k^{th} \text{ Krylov subspace (induced by } A \text{ and } r_0)}$$

Facts about Krylov subspaces

$K_k(A, r) = \text{span}\{r, Ar, A^2 r, \ldots, A^{k-1} r\}$, $k = 1, 2, \ldots,$

is defined for any $A \in \mathbb{C}^{n \times n}$, $r \in \mathbb{C}^n$ ($r \neq 0$ so that $K_k(A, r) \neq \emptyset$)

1) $v \in K_k(A, r) \iff v = c_0 r + c_1 A r + \ldots + c_{k-1} A^{k-1} r$

$$= (c_0 I + c_1 A + \ldots + c_{k-1} A^{k-1}) r$$

$$= \psi(A) r, \text{ where } \psi \in \Pi_{k-1}: \text{[illegible]} \le k-1.$$

$$\text{w/ } \Pi_{k-1} := \{\psi(\lambda) = c_0 + c_1 \lambda + \ldots + c_{k-1} \lambda^{k-1}\}$$

Lecture 14 - 2/9

$K_k(A,r) = \text{span}\{r, Ar, A^2r, \ldots, A^{k-1}r\}$, $A \in \mathbb{C}^{n\times n}$, $r \in \mathbb{C}^n$, $r \neq 0$

Π_{k-1} = polys. of deg. $\leq k-1$

1) $K_k(A,r) = \{\psi(A)r \mid \psi \in \Pi_{k-1}\}$

2) $K_k(A,r)$ is a subspace of $\mathbb{C}^n$ (fact)

$\Rightarrow A^d r = c_0 r + c_1 Ar + \ldots + c_{d-1}A^{d-1}r \in K_d(A,r)$ for some $1 \leq d \leq n$

$d(A,r) :=$ smallest such d, called the <u>grade</u> of A wrt. r

↳ first d s.t. $A^d r$ is linearly dependent on $\text{span}\{r, Ar, \ldots, A^{d-1}r\}$

$$\Rightarrow \dim K_k(A,r) = \begin{cases} k & \text{if } k \leq d(A,r) \\ d(A,r), & \text{if } k \geq d(A,r) \end{cases}$$

and $K_k(A,r) = K_{d(A,r)}(A,r)$ for all $k \geq d(A,r)$

3) If A is diagonalizable: $d(A,r)$ is the number of eigenvectors in an eigendecomposition of r:

$$r = \sum_{j=1}^{d(A,r)} \rho_j z_j \quad , \text{ where } Az_j = \lambda_j z_j, \quad \rho_j \in \mathbb{C}, \ \rho_j \neq 0$$
$$\lambda_j \neq \lambda_\ell \text{ for } j \neq \ell, \qquad z_j \neq 0$$

Suppose $|\lambda_1| > |\lambda_j|$ for $j \geq 2$.

$$\frac{1}{\lambda_1^\ell} A^\ell r = \sum_{j=1}^{d(A,r)} \rho_j \left(\frac{\lambda_j}{\lambda_1}\right)^\ell z_j \xrightarrow{\ell \to \infty} \rho_1 z_1$$

Hence $\{r, Ar, \ldots, A^{k-1}r\}$ is an <u>aconditioned basis</u> of $\mathbb{C}^n$

↳ not a useful basis for $K_k(A,r)$ in practice!

<u>Exception</u>: Power method exploits this limit to compute the dominant eigenvector of A.

4) Let A be nonsingular, $x_0 \in \mathbb{C}^n$, $r_0 = b - Ax_0$

then: $X^* = A^{-1}b \in x_0 + K_d(A, r_0)$, $d = d(A, r_0)$

Pf: $r = r_0$. $A^d r = c_0 r + c_1 A r + c_2 A^2 r + \ldots + c_{d-1} A^{d-1} r$ $\quad$ d is minimal

$\Rightarrow c_0 \neq 0$ (suppose $c_0 = 0$, then can mult. by A^{-1}: $A^{d-1} r = c_1 r + c_2 A r + c_3 A^2 r + \ldots$
hence d not minimal ↯

$$r = A\Big(\underbrace{\tfrac{1}{c_0}\big(-c_1 r - c_2 A r - \ldots - c_{d-1} A^{d-2} r + A^{d-1} r\big)}_{=: z^* \in K_d(A, r)}\Big)$$

$$b - Ax_0 = r = Az^*$$

$$\Rightarrow \quad b = Ax_0 + Az^*$$

$$\Rightarrow \quad \underbrace{A^{-1}b}_{X^*} = \underbrace{x_0 + z^*}_{\in x_0 + K_d(A, r)}$$

□

Back to the case $A \succ 0$:

Thm: 1) In exact arithmetic, CG terminates after finitely many steps:

$x_k \neq x^* = A^{-1}b$ for $k < d(A, r_0)$

$x_k = x^*$ for $k = d(A, r_0)$.

2) For all $k = 1, 2, \ldots, d(A, r_0)$, $x_k \in x_0 + K_k(A, r_0)$ is optimal in the sense that

$$\min_{x \in x_0 + K_k(A, r_0)} \|x^* - x\|_A = \|x^* - x_k\|_A .$$

3) For all $k = 1, 2, \ldots, d(A, r_0)$:

$$\frac{\|x^* - x_k\|_A}{\|x^* - x_0\|_A} \leq 2\left(\frac{\sqrt{\mathcal{K}} - 1}{\sqrt{\mathcal{K}} + 1}\right)^k, \text{ where } \mathcal{K} = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} = \text{condition \# of } A \succ 0 \text{ wrt. } \|\cdot\|_2\text{-norm}$$

note: $0 < \lambda_{\min}(A)$ since $A \succ 0$
& $\mathcal{K} \geq 1$

hence smaller $\mathcal{K} \Rightarrow$ faster convergence

Lecture 15 - 2/12

Preconditioning

Basic idea: $Ax = b \iff A'x' = b' \quad (A' \succ 0)$

s.t. $\kappa(A') \ll \kappa(A)$ to improve CG speed

$A \approx M = M_1 M_2 \succ 0$ where $M_1, M_2 \in \mathbb{R}^{n \times n}$ and linear systems with M_1 and M_2 are "easy" to solve

M is called a <u>preconditioner</u> for A.

For *example*: M_1 = sparse lower-triangular matrix
$M_2 = M_1^T \quad (\Rightarrow M = LL^T = M_1 M_2)$

$$Ax = b \iff \underbrace{M_1^{-1} A M_2^{-1}}_{= M_1^{-1} A M_1^{-T} =: A' \succ 0} \underbrace{M_2 x}_{= M_1^T x = x'} = \underbrace{M_1^{-1} b}_{=: b'} \iff A'x' = b'$$

(Run CG on $A'x' = b'$, obtain x'. Solve $M_1^T x = x'$ for x.)

PCG (Preconditioned CG) w/ preconditioner of the form $M = M_1 M_1^T$:

- Set $b' = M_1^{-1} b$, $x_0' = M_1^T x_0$
- Apply CG to $A'x' = b'$ with initial guess x_0'
- Set $x_k = M_1^{-T} x_k'$

dont actually form A' just apply A & solve w/ M_1, M_2

dont apply inverse just solve lin. system: e.g. $x_k = M_1^T \backslash x_k'$

General form of preconditioned CG (pcg in MATLAB) allows preconditioners of the form $M = M_1 M_2 \succ 0$ where M_2 <u>need not</u> be equal to M_1^T.

<u>Special cases</u>: $M = M_1, M_2 = I$ <u>left</u> preconditioning
$M = M_2, M_1 = I$ right preconditioning

Work per k^{th} iteration of PCG:

1 multiplication w/ A

1 solve with M_1 ⎤
1 solve with M_2 ⎦ – additional work due to preconditioning

2 inner products ~~of vectors of length n~~

3 SAXPYs " " " " "

Extra work per iteration, but good preconditioning cuts down the number of iterations by so much that overall it's less work.

Two preconditioners:

1) Diagonal preconditioning – good for elliptic PDE discretizations

$$A = \begin{bmatrix} a_{11} & & & * \\ & a_{22} & & \\ & * & \ddots & \\ & & & a_{nn} \end{bmatrix} \succ 0 \quad (\Rightarrow a_{jj} > 0 \ \forall j), \quad M = \begin{bmatrix} a_{11} & & & 0 \\ & a_{22} & & \\ 0 & & \ddots & \\ & & & a_{nn} \end{bmatrix}$$

can set $M_1 = M_2 = \sqrt{M} = \begin{bmatrix} \sqrt{a_{11}} & & 0 \\ & \sqrt{a_{22}} & \\ 0 & & \ddots \\ & & & \sqrt{a_{nn}} \end{bmatrix}$

2) Incomplete Cholesky factorization – good for 3D PDEs, where sparse Chol has too much fill-in

Instead of computing L s.t. $A = LL^T$, compute a (much) sparser L w/

$$A \approx LL^T = M$$

Typical approach: prescribe sparsity of L

Choose $E \subset \{(j,k) \mid 1 \le k \le j \le n\}$ s.t. $(j,j) \in E \ \forall j$

· Construct L s.t. $\ell_{jk} \neq 0 \Rightarrow (j,k) \in E$.

Ex: $L = \begin{bmatrix} * & & & & 0 & \\ & * & & & & \\ & & * & & & \\ * & & * & * & & \\ & * & & & * & \\ & & * & & & * \end{bmatrix} \in \mathbb{R}^{6\times 6} \Rightarrow E = \{(1,1), (4,1), (2,2), (5,2), (3,3), (4,3), (6,3), (4,4), (5,5), (6,6)\}$

Efficient (columnwise) storage of E:

$$J = \begin{bmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 4 \\ 6 \\ 4 \\ 5 \\ 6 \end{bmatrix} \qquad I = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 8 \\ 9 \\ 10 \\ 11 \end{bmatrix} \leftarrow \text{nnz}(L)+1$$

In general:

J = integer vector of length nnz(L)

I = integer vector of length n+1

$$E = \{ (J(i), k) \mid i = I(k), I(k)+1, \ldots, I(k+1)-1, \ k = 1, 2, \ldots, n \}$$

Popular choice for E: Allow <u>no</u> fill-in from sparse Chol. factorization.
maintain sparsity structure of A:

$$E = \{ (j, k) \mid 1 \le k \le j \le n \text{ and } a_{jk} \neq 0 \}$$

$\Rightarrow$ L will have same sparsity structure as lowertriangularpart of A.

$\rightarrow$ E = find(tril(A) ≠ 0)

Lec 16 - 2/14

"Politics is sleazy, backroom deals ... not like pure math."

Recall: $A \succ 0$, $A \in \mathbb{R}^{n \times n}$, $A \approx LL^T$

$E \subset \{(j,k) \mid 1 \le k \le j \le n\}$ w/ $(j,j) \in E \ \forall j$.

Construct L s.t. $l_{jk} \neq 0 \Rightarrow (j,k) \in E$.

Construction of "incomplete" Cholesky factor L - Algorithm:

Input: E (e.g. given by J, I)
the elements a_{jk} of $A = [a_{jk}] \in \mathbb{R}^{n \times n}$, $A \succ 0$, for all $(j,k) \in E$

```
· Set $l_{jk} = a_{jk}$ for all $(j,k) \in E$
For $k = 1, 2, \dots, n$ do:
    · If $l_{kk} \le 0$, stop. Algorithm fails
    · Set $l_{kk} = \sqrt{l_{kk}}$
    · Set $l_{jk} = \frac{1}{l_{kk}} \cdot l_{jk}$ for all $(j,k) \in E$ w/ $j > k$
    For all $(j,k) \in E$ with $j > k$ do:
        · Set $l_{ij} = l_{ij} - l_{jk} l_{ik}$ for all $(i,j) \in E$
    end(j)
end(k)
```

· In general, this algorithm can break down due to $l_{kk} \le 0$.
but there are important classes of $A \succ 0$ for which this cannot occur!

Ex: Laplacian in any dimension.

· "Perfect preconditioners don't exi-"

Krylov subspace methods for general nonsingular linear systems:

(*) $Ax = b$, $A \in \mathbb{R}^{n \times n}$ nonsingular, $b \in \mathbb{R}^n$

Want to use same setting: $x_0 \in \mathbb{R}^n$, $r_0 = b - Ax_0$, $x^* = A^{-1}b$

but if $A \nsucc 0$, using CG makes no sense.

Poor man's use of CG to solve (*):

(*) $\Longleftrightarrow$ $A^T A x = A^T b$ (**) (Normal eqns)

<u>Note</u>: $A^T A \succ 0 \Rightarrow$ (**) can be solved with CG for x^* (same soln as to (*));

<u>Resulting method</u>: CGNE (CG on the normal eqns)

<u>Why bad?</u> $\|x^* - x\|_{A^TA} = \sqrt{(x^*-x)^T A^T A (x^*-x)} = \|Ax^* - Ax\|_2 = \|b - Ax\|_2$

$r_k = b - Ax_k$

CGNE iterates: $x_1 \in x_0 + K_k(A^TA, A^Tr_0)$ s.t. ← wrong Krylov space!

$$\|r_k\|_2 = \|b - Ax_k\|_2 = \min_{x \in x_0 + K_k(A^TA, A^Tr_0)} \|b - Ax\|_2$$ ← wrong min!

CGNE error bounds:

$$\frac{\|r_k\|_2}{\|r_0\|_2} = \frac{\|b - Ax_k\|_2}{\|b - Ax_0\|_2} \le \underbrace{2\left(\frac{\sqrt{\kappa(A^TA)} - 1}{\sqrt{\kappa(A^TA)} + 1}\right)^k}_{} \quad k = 0, 1, 2, \dots$$

$$\sqrt{\kappa(A^TA)} = \sqrt{\kappa(A)^2} = \kappa(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)} \Rightarrow \quad 2\left(\frac{\kappa(A) - 1}{\kappa(A) + 1}\right)^k$$

So much slower than standard CG!
For large n, $\sqrt{\kappa} \ll \kappa \Rightarrow$ error bound <u>much</u> closer to 1 for CGNE!

<u>Issue</u>: Wrong Krylov space in CGNE

Want to work with $K_k(A, r_0)$.

Minimal Residual (MR) method for solving (*)

$\mathbb{R}^n \ni x_0 \longrightarrow x_1 \longrightarrow \dots \longrightarrow x_k \longrightarrow \dots \longrightarrow x_d = A^{-1}b = x^*$

$d = d(A, r_0)$

where $x_k \in x_0 + K_k(A, r_0)$ s.t.

$$\|b - Ax_k\|_2 = \min_{x \in x_0 + K_k(A, r_0)} \|b - Ax\|_2, \quad k = 1, 2, \dots, d = d(A, r_0)$$

Recall: CG (for $A \succ 0$) uses short recurrences (just last step's info)

Thm: For general A, it is not possible to implement the MR method w/ short recurrences: to compute x_{k+1}, vectors from all prev. k iterations needed

Lecture 17 - 2/16

Arnoldi process.

Goal: Given $A \in \mathbb{C}^{n \times n}$ (not necc. nonsingular) and $r \in \mathbb{C}^n$, $r \neq 0$, construct orthonormal basis vectors $v_1, v_2, \dots, v_k, \dots, v_{d(A,r)}$ for $K_k(A, r)$

Alg. (Arnoldi process)

```
Input: $r \in \mathbb{C}^n$, $r \neq 0$
       a routine to compute $q = Av$ for $v \in \mathbb{C}^n$

· Set $\beta = \|r\|_2$ and $v_1 = r/\beta$
· For $k = 1, 2, \dots$
    · Compute $q = Av_k$
    For $j = 1, 2, \dots$
        · Set $h_{jk} = v_j^H q$
        · Set $q = q - h_{jk} v_j$
    end(j)
    Set $h_{k+1,k} = \|q\|_2$
    If $h_{k+1,k} = 0$ then $K(A, r)$ has reached the max dimension i.e. $k = d(A, r)$
    else:
        set $v_j = \frac{q}{h_{k+1,k}}$
end (k)

Output:
    · orthonormal vectors $v_1, v_2, \dots, v_k, \dots, v_{d(A,r)}$
    · coefficients $h_{jk}$
```

Properties: 1) $v_k^H v_j = \delta_{kj}$

2) $K_k(A,r) = \text{span}\{v_1, v_2, \dots, v_k\}$, $k = 1, 2, \dots, d(A,r)$

3) $h_{k+1,k}\, v_{k+1} = \underbrace{A v_k}_{q} - \sum_{j=1}^{k} h_{jk} v_j$ $k = 1, 2, \dots, d(A,r)$

Compact formulation of (3):

$$A V_k = V_{k+1} \tilde{H}_k$$

where $V_k = [v_1, v_2, \dots, v_k]$, $V_{k+1} = [v_1, v_2, \dots, v_k, v_{k+1}]$

and $\tilde{H}_k := [h_{j\ell}]_{\substack{j=1,2,\dots,k+1 \\ \ell=1,2,\dots,k}}$

$$= \begin{bmatrix} h_{11} & h_{12} & \cdots & \cdots & h_{1k} \\ h_{21} & h_{22} & \ddots & & \vdots \\ & h_{32} & \ddots & & \vdots \\ & & \ddots & \ddots & \vdots \\ & 0 & & \ddots & h_{kk} \\ & & & & h_{k+1,k} \end{bmatrix} \in \mathbb{C}^{k+1 \times k}$$

$\tilde{H}_k$ is an Upper-Hessenberg matrix

Notes: 1) rank $\tilde{H}_k = k$, $k = 1, 2, \dots, d(A,r) - 1$

2) orthonormality of the v_k's $\iff$ $V_k^H V_k = I \in \mathbb{C}^{k \times k}$

3) A, r real $\Rightarrow$ $v_1, v_2, \dots, v_k$, $\tilde{H}_k$ real

GMRES - (Generalized Minimal Residual)

- Implementation of the MR method based on the Arnoldi process

Assume that $r_0 \neq 0$ (otherwise $x_0 = x^* = A^{-1} b$)

Note: $K_k(A, r_0) = \text{span}\{v_1, v_2, \dots, v_k\} = \{v = V_k z \mid z \in \mathbb{R}^k\}$

Step k of GMRES:

$x \in x_0 + K_k(A, r_0) \iff x = x_0 + V_k z$, $z \in \mathbb{R}^k$

$\Rightarrow b - Ax = \underbrace{b - A x_0}_{} - \underbrace{A V_k z}_{} = V_{k+1}(\beta e_1 - \tilde{H}_k z)$] No A in this form!

$$b - Ax = V_{k+1}(\beta e_1 - \hat{H}_k z)$$

$$\Rightarrow \|b - Ax\|_2 = \|V_{k+1}(\beta e_1 - \hat{H}_k z)\|_2 = \sqrt{(\beta e_1 - \hat{H}_k z)^H \underbrace{V_{k+1}^H V_{k+1}}_{I} (\beta e_1 - \hat{H}_k z)}$$

$$= \|\beta e_1 - \hat{H}_k z\|_2$$

Recall: MR Method wants $\|b - Ax_k\|_2 = \min\limits_{x \in x_0 + K_k(A, r)} \|b - Ax\|_2 = \min\limits_{z \in \mathbb{R}^k} \|\beta e_1 - \hat{H}_k z\|$

Lec 18 - 2/21

Recall: k^{th} GMRES iterate

$x_k \in x_0 + K_k(A, r_0)$ s.t. $\|b - Ax_k\|_2 = \min\limits_{x \in x_0 + K_k(A, r_0)} \|b - Ax\|_2$

$$= \min_{z \in \mathbb{R}^k} \|\beta e_1 - \hat{H}_k z\|_2$$

k steps of Arnoldi process (applied to A & r_0) yields V_k, $\hat{H}_k$, $\beta = \|r_0\|_2$

Computation of the GMRES iterate x_k:

1) Find $z_k \in \mathbb{R}^k$ s.t. $\|\beta e_1 - \tilde{H}_k z_k\|_2 = \min\limits_{z \in \mathbb{R}^k} \|\beta e_1 - \hat{H}_k z\|_2$ $(LS)_k$

2) Set $x_k = x_0 + V_k z_k$

$(LS)_k$ is a Least-Squares problem with a matrix $\hat{H}_k \in \mathbb{R}^{k+1 \times k}$ which has full column rank $k \Rightarrow$ unique soln of $z_k \in \mathbb{R}^k$

Soln of $(LS)_k$? Recall QR $\Rightarrow$ Special case for $\hat{H}_k$:

$$Q_k \hat{H}_k = \begin{bmatrix} R_k \\ 0 \cdots 0 \end{bmatrix}$$

Soln of $(LS)_k$ — exploits $\hat{H}_k$ upper-Hessenberg

Let $Q_k \in \mathbb{R}^{k+1 \times k+1}$ be orthogonal (i.e., $Q_k^T Q_k = I$) s.t.

$$Q_k \hat{H}_k = \begin{bmatrix} R_k \\ 0 \cdots 0 \end{bmatrix},$$ where $R_k \in \mathbb{R}^{k \times k}$ is upper-triangular & nonsingular

Then: $$\min_{z \in \mathbb{R}^k} \|\beta e_1 - \hat{H}_k z\|_2 = \min_{z \in \mathbb{R}^k} \| \underbrace{Q_k \beta e_1}_{\begin{bmatrix} f_k \\ \tau_{k+1} \end{bmatrix} \begin{matrix} \}k \\ \}1 \end{matrix}} - \begin{bmatrix} R_k \\ 0 \cdots 0 \end{bmatrix} z \|_2$$

z can make $f_k - R_k z = 0$ since R_k nonsingular

$\leftarrow$ can't do anything about τ_{k+1}

$\Rightarrow$ $z_k = R_k^{-1} f_k$ is the soln. of $(LS)_k$

and $$\|\beta e_1 - \hat{H}_k z_k\|_2 = \underbrace{|\tau_{k+1}| = \|b - Ax_k\|_2}_{\text{can monitor residual } \underline{\text{w/o}} \text{ computing iterate } x_k!}$$

GMRES Alg: (Solve $Ax=b$ using Krylov spaces for nonsymmetric A)

<u>Input</u>: $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, a routine to compute matrix-vector products $q = Av$, convergence tolerance tol

```
· Set $r_0 = b - Ax_0$ and $\beta = \|r_0\|_2$
· If $\beta = 0$, stop: $x_0 = A^{-1}b$ is the soln. of $Ax=b$.
· Set $v_1 = r_0/\beta$
For $k = 1, 2, \ldots,$ do:
    1) Perform the $k^{th}$ step of the Arnoldi process → get $\hat{H}_k$, $V_k$
    2) Determine $z_k$ and $\tau_{k+1}$ s.t. $|\tau_{k+1}| = \|\beta e_1 - \hat{H}_k z_k\|_2 = \min_{z \in \mathbb{R}^k} \|\beta e_1 - \hat{H}_k z\|_2$
    3) If $\left(\frac{\|r_k\|_2}{\|r_0\|_2} = \right) \frac{|\tau_{k+1}|}{\beta} \le tol$, set $x_k = x_0 + V_k z_k$ & stop: $x_k \approx A^{-1}b$.
end(k).
```

Computation of R_k, f_k, τ_{k+1}: exploit H_k growing from H_{k-1}

Givens rotations: $G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \in \mathbb{R}^{2\times 2}$, $c^2+s^2=1$

(can think of as $c=\cos\theta, s=\sin\theta$)

Notes: 1) G is orthogonal: $G^TG=I$

2) For any $h \in \mathbb{R}^2$, G can be chosen s.t. $Gh = \begin{bmatrix} * \\ 0 \end{bmatrix}$.

$k=1$: $\tilde{H}_1 = \begin{bmatrix} h_{11} \\ h_{21} \end{bmatrix}$, $Q_1 = G_1 = \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}$ s.t.

$$Q_1 \tilde{H}_1 = G_1 \begin{bmatrix} h_{11} \\ h_{21} \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \begin{matrix}]1 \\ ]1 \end{matrix}$$

$$\Rightarrow \quad Q_1 \beta e_1 = G_1 \begin{bmatrix} \beta \\ 0 \end{bmatrix} = \begin{bmatrix} c_1\beta \\ -s_1\beta \end{bmatrix} = \begin{bmatrix} f_1 \\ \tau_2 \end{bmatrix} \begin{matrix}]1 \\ ]1 \end{matrix}$$

$k-1 \to k$: Assume found

$$Q_{k-1}\tilde{H}_{k-1} = \begin{bmatrix} R_{k-1} \\ 0 \cdots 0 \end{bmatrix} \begin{matrix}]k-1 \\ ]1 \end{matrix}, \quad Q_{k-1}\beta e_1 = \begin{bmatrix} f_{k-1} \\ \tau_k \end{bmatrix} \begin{matrix}]k-1 \\ ]1 \end{matrix}$$

Arnoldi process yields

$$\tilde{H}_k = \left[\begin{array}{c|c} \tilde{H}_{k-1} & \begin{matrix} h_{1k} \\ h_{2k} \\ \vdots \\ h_{kk} \end{matrix} \\ \hline 0 \cdots 0 & h_{k+1,k} \end{array}\right]$$

So pick

$$\underbrace{\begin{bmatrix} \overbrace{I_{k-1}}^{k-1\times k-1} & \\ & \underbrace{G_1}_{2\times 2} \end{bmatrix} \left[\begin{array}{c|c} Q_{k-1} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \cdots 0 & 1 \end{array}\right]}_{=:Q_k} \tilde{H}_k = \begin{bmatrix} \overbrace{I_{k-1}}^{k-1\times k-1} & 0 \\ 0 & \underbrace{G_k}_{2\times 2} \end{bmatrix} \left[\begin{array}{c|c} R_{k-1} & r_{k-1} \\ 0 \cdots 0 & \hat{r}_{kk} \\ \hline 0 \cdots 0 & h_{k+1,k} \end{array}\right] \begin{matrix}]k-1 \\ ]1 \\ ]1 \end{matrix}$$

$$= \left[\begin{array}{c|c} R_{k-1} & r_{k-1} \\ 0 \cdots 0 & r_{kk} \\ \hline & 0 \end{array}\right] =: R_k$$

where $G_k = \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$ s.t. $G_k \begin{bmatrix} \hat{r}_{kk} \\ h_{k+1,k} \end{bmatrix} = \begin{bmatrix} r_{kk} \\ 0 \end{bmatrix}$

and also $Q_k B e_1 = \begin{bmatrix} f_k \\ \tau_{k+1} \end{bmatrix}$, with $f_k := \begin{bmatrix} f_{k-1} \\ c_k \tau_k \end{bmatrix}$

$\tau_{k+1} := -s_k \tau_k$

Lec 19 - 2/23

GMRES step $k-1 \to k$:

$\tau_k, f_{k-1}, R_{k-1} \to \tau_{k+1}, f_k, R_k$

update requires only one Givens rotations:

$$G_k = \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$$

In particular, $\tau_{k+1} = -s_k \tau_k$

Note: $\|r_k\|_2 = |\tau_{k+1}| = \underset{\leq 1}{|s_k|} |\tau_k| \leq \|r_{k-1}\|_2$

Convergence of GMRES:

$K_k(A, r_0) = \{\psi(A) r_0 \mid \psi \in \Pi_{k-1}\}$

$x \in x_0 + K_k(A, r_0) \Leftrightarrow x = x_0 + \psi(A) r_0, \ \psi \in \Pi_{k-1}$

$b - Ax = \underbrace{b - Ax_0}_{r_0} - A\psi(A) r_0 = (I - A\psi(A)) r_0 = \varphi(A) r_0$

where $\varphi(\lambda) = 1 - \lambda\psi(\lambda) \in \Pi_k, \ \varphi(0) = 1$

MR property:

$$\|b - Ax\|_2 = \min_{x \in x_0 + K_k(A, r_0)} \|b - Ax\|_2 = \min_{\varphi \in \Pi_k, \varphi(0)=1} \|\varphi(A) r_0\|_2$$

Thm: The iterates x_k obtained in the MR method (e.g. GMRES implementation) satisfy:

1) $\dfrac{\|r_k\|_2}{\|r_0\|_2} = \dfrac{\|b - Ax_k\|_2}{\|b - Ax_0\|_2} \leq \min\limits_{\varphi \in \Pi_k, \varphi(0)=1} \|\varphi(A)\|_2$

Moreover, if A is diagonalizable w/ $A = U \Lambda U^{-1}$, then

2) $\dfrac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(U) \min\limits_{\varphi \in \Pi_k, \varphi(0)=1} \max\limits_{\lambda_j \in \Lambda} |\varphi(\lambda_j)|$

Recall: $\kappa_2(U) = \|U\|_2 \|U^{-1}\|_2$

Proof of (2):

$$\begin{cases} A = U \Lambda U^{-1} \\ A^2 = U \Lambda \cancel{U^{-1} U} \Lambda U^{-1} = U \Lambda^2 U^{-1} \\ \vdots \\ A^k = U \Lambda^k U^{-1} \end{cases}$$

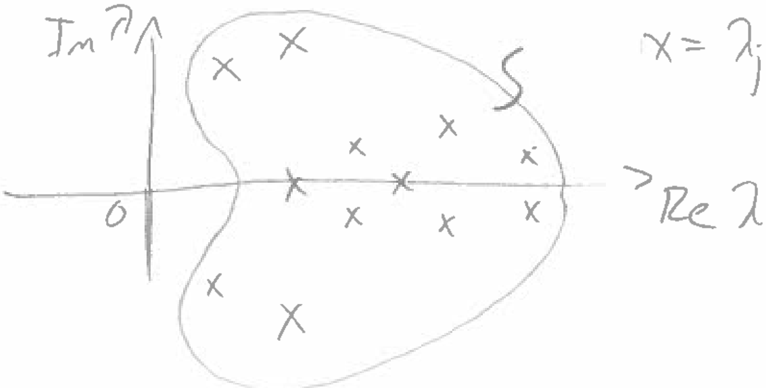
$\Rightarrow \varphi(A) = U \varphi(\Lambda) U^{-1}$ $\forall$ polynomials φ

$$\Rightarrow \|\varphi(A)\|_2 = \|U \varphi(\Lambda) U^{-1}\|_2 \leq \underbrace{\|U\|_2 \|U^{-1}\|_2}_{\kappa_2(U).} \|\varphi(\Lambda)\|_2$$

$$\varphi(\Lambda) = \begin{bmatrix} \varphi(\lambda_1) & & & 0 \\ & \varphi(\lambda_2) & & \\ & & \ddots & \\ 0 & & & \varphi(\lambda_n) \end{bmatrix} \Rightarrow \|\varphi(\Lambda)\|_2 = \max_{\lambda_j \in \Lambda} |\varphi(\lambda_j)|.$$

Cor: If A is diagonalizable, $\sigma(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\} \subset S$, $S \subset \mathbb{C}$ compact, $0 \notin S$, then:

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(U) \min_{\substack{\varphi \in \Pi_k, \\ \varphi(0)=1}} \max_{\lambda \in S} |\varphi(\lambda)|$$



x = λ_j's of A.

Consequence: fast convergence of GMRES if the eigenvalues of A are clustered away from the origin.

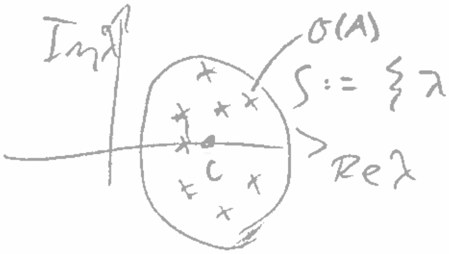
Lec 20 - 2/26

Recall: Convergence rate of GMRES depends on

$\sigma(A) = \{\lambda_1, \lambda_2, \dots, \lambda_n\} \subset S$, S compact, $0 \notin S$

since residuals $\frac{\|r_k\|_2}{\|r_0\|_2} \le \mathcal{K}_2(U) \cdot \min_{\substack{\varphi \in \Pi_k, \\ \varphi(0)=1}} \max_{\lambda \in S} |\varphi(\lambda)|$.

Ex: eigenvals in a disk S



$S := \{\lambda \mid |\lambda - c| \le \rho\}$ w/ $\rho < |c|$ (since $0 \notin S$)

$\varphi(\lambda) = \left(1 - \frac{\lambda}{c}\right)^k \in \Pi_k$, $\varphi(0) = 1$

$\Rightarrow \frac{\|r_k\|_2}{\|r_0\|_2} \le \mathcal{K}_2(U) \max_{\lambda \in S} |\varphi(\lambda)| \underset{\uparrow}{=} \mathcal{K}_2(U) \cdot \left(\frac{\rho}{|c|}\right)^k$

$\partial S = \{\lambda = c + \rho e^{it} \mid 0 \le t \le 2\pi\}$

since $|\varphi(c + \rho e^{it})| = |(-\frac{\rho}{c} e^{it})^k| = \left(\frac{\rho}{|c|}\right)^k$

φ is the optimal polynomial (Rouché Thm), i.e. $\forall \psi \in \Pi_k$, for most S compact (ellipses, intervals, segments)

$\max_{\lambda \in S} |\varphi(\lambda)| \le \max_{\lambda \in S} |\psi(\lambda)|$.

Preconditioned GMRES

(*) $Ax = b$, $A \in \mathbb{R}^{n\times n}$ nonsingular, $b \in \mathbb{R}^n$

Preconditioner: $M \in \mathbb{R}^{n\times n}$ nonsingular s.t. $M = M_1 M_2$, where $M_1, M_2 \in \mathbb{R}^{n\times n}$ and M_1, M_2 easy to solve, $M \approx A$ (in some sense)

↳ doesn't change cond# here, aims to cluster the eigenvalues instead!

Let $x_0 \in \mathbb{R}^n$ be any initial guess for $x = A^{-1}b$:

Then $Ax = b \iff A(x - x_0) = b - Ax_0$

$$\iff \underbrace{M_1^{-1}AM_2^{-1}}_{A'}\underbrace{M_2(x-x_0)}_{x'} = \underbrace{M_1^{-1}(b-Ax_0)}_{b'}$$

Note: $x = x_0 + M_2^{-1}x'$

$M \approx A \iff$ GMRES applied A' converges faster than for A,
e.g., $\sigma(A')$ is clustered away from 0.

Preconditioned GMRES Algorithm:

Input: $b, x_0 \in \mathbb{R}^n$
a routine to compute $q = Av$
" " " solve systems w/ M_1
" " " " " " M_2
convergence tolerance tol

1) Solve $M_1 b' = b - Ax_0$ for b'

2) Set $x_0' = 0 \in \mathbb{R}^n$ and $r_0' = b'$.

3) Run GMRES to solve $A'x' = b'$ (w/ initial guess x_0')
to accuracy $\dfrac{\|r_k'\|_2}{\|r_0'\|_2} \leq tol$

4) Solve $M_2 w = x_k'$ for w & set $x_k = x_0 + w$

Notes: • $r_k = b - Ax_k$, $r_k' = M^{-1} r_k$, so r_k' within tol doesn't guarantee r_k is!

• Step(3) requires matrix-vector products $q' = A'v'$ $(= M_1^{-1} A M_2^{-1} v')$
& each such product requires 1 solve w/ M_2, 1 mult w/ A, 1 solve w/ M_1

Lec21 - 2/28

Solving linear system $Ax = b$, $A \in \mathbb{R}^{n\times n}$, nonsingular

Preconditioner $M = M_1 M_2 \in \mathbb{R}^{n\times n}$

$$A' = M_1^{-1} A M_2^{-1}$$

$$M \approx A \Rightarrow A' \approx I$$

Some preconditioners:

$$A = \underbrace{D_0}_{\text{diagonal}} + \underbrace{F}_{\text{strictly lower triangular}} - \underbrace{G}_{\text{strictly upper triangular}} \in \mathbb{R}^{n\times n}$$

1) Diagonal preconditioning (D_0 nonsingular)

$M = D_0$ $(= M_1 M_2)$

Typically: $M_1 = I$, $M_2 = D_0$ or $M_1 = D_0$, $M_2 = I$

2) SSOR-type preconditioning

Let $D \in \mathbb{R}^{n\times n}$ be diagonal & nonsingular

$$M = (D-F)D^{-1}(D-G)$$

If $D = D_0$, then $M = \underbrace{D_0 - F - G}_{=A} + F D_0^{-1} G$

↑almost "free" preconditioner

3) Incomplete LU factorization

$$\underset{\substack{\uparrow \\ \text{permutation}\\ \text{matrices}}}{PAQ} \approx \underset{\substack{\uparrow\\ \text{lower}\Delta}}{L}\overset{\leftarrow \text{upper}\Delta}{U}$$

L, U nonsingular (incomplete LU)

$$\Rightarrow A \approx \underbrace{P^T L}\underbrace{UQ^T} =: M$$

Restarted GMRES

Note: GMRES w/o restarting is called full-GMRES.

Suppose $A = A'$ is preconditioned. $A \in \mathbb{R}^{n \times n}$.

Recall: k^{th}-step of the Arnoldi process:

$$h_{k+1,k} v_{k+1} = A v_k - h_{1k} v_1 - h_{2k} v_2 - \dots - h_{kk} v_k$$

This requires: 1 multiplication with A ($\mathbb{R}^{n \times n} \cdot \mathbb{R}^n$)

$k+1$ inner products ($\mathbb{R}^n \cdot \mathbb{R}^n$)

k SAXPYs

storage of all vectors $v_1, v_2, \dots, v_{k+1}$

$\Rightarrow$ too expensive for very large n as k increases

Remedy: Restarts

Let k_0 be the largest # of Arnoldi steps one is willing/able to run.

Typical values: $k_0 = 50$, $k_0 = 100$

Algorithm (Restarted GMRES)

Input: a routine to compute $q = Av$

$b, x_0 \in \mathbb{R}^n$

a convergence tolerance tol

the restart parameter k_0

1) Set $\rho_0 := \|b - Ax_0\|_2$

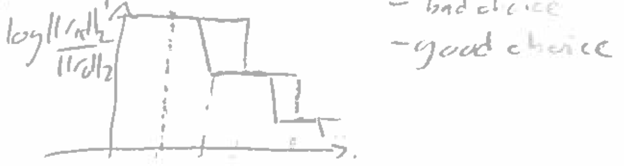
2) Run GMRES until

a) $\frac{\|r_k\|_2}{\rho_0} \le tol$: stop, $x_k \approx A^{-1}b$

or b) $k = k_0$ is reached: set $x_0 = x_{k_0}$ & repeat step (2).

Notes: 1) The bounds derived for (full) GMRES are no longer valid for restarted GMRES since each restart we are looking at a new Krylov space $K(A, r_{k_0})$.

2) Convergence of restarted GMRES can be very sensitive to the choice of k_0:



Lec22 - 3/2

Two other applications of the Arnoldi process:

$A \in \mathbb{C}^{n\times n}$, $r \in \mathbb{C}^n$, $r \neq 0$, after k $(\leq d(A,r))$ steps of Arnoldi:

$$AV_k = V_{k+1}\hat{H}_k = V_k H_k + h_{k+1,k} \cdot \begin{bmatrix} \vec{0} & \cdots & \vec{0} & v_{k+1} \\ | & \cdots & | & | \end{bmatrix}$$

where $\hat{H}_k = \begin{bmatrix} H_k \in \mathbb{C}^{k\times k} \\ \hline 0 \cdots \; 0 \cdot h_{k+1,k} \end{bmatrix} \in \mathbb{C}^{k+1\times k}$, $V_k^H V_k = I$, $V_k^H v_{k+1} = 0$

$$\Rightarrow V_k^H A V_k = H_k + \underbrace{V_k^H \begin{bmatrix} 0 & \cdots & 0 & \top \\ / & & & v_{k+1} \\ / & & & \bot \end{bmatrix}}_{V_k^H v_{k+1} = 0} \cdot h_{k+1,k}$$

$$\Rightarrow V_k^H A V_k = H_k$$

$\hookrightarrow \operatorname{span} V_k = \operatorname{span}\{v_1, v_2, \dots, v_k\} = \mathcal{K}_k(A,r)$

$\hookrightarrow$ This is a projection/restriction of $A \in \mathbb{C}^{n\times n}$ onto $\mathcal{K}_k(A,r)$!

<u>Thm:</u> The upper-Hessenberg matrix H_k produced by k steps of the Arnoldi process is the projection of A onto $\mathcal{K}_k(A,r)$:

$$V_k^H A V_k = H_k$$

1) Approximate eigenpairs of A:

$Ax = \lambda x$, $\lambda \in \mathbb{C}$, $x \in \mathbb{C}^n$, $x \neq 0$.
(λ, x): eigenpair

Approx eigenpair: $(\tilde{\lambda}, \tilde{x})$ s.t. $A\hat{x} \approx \hat{\lambda}\hat{x}$, $\hat{x} \neq 0$. (no exact λ's for $n \geq 5$ by fund. thm. algebra)

- Pick $r \in \mathbb{C}^n$, $r \neq 0$ & use $\hat{x} \in \mathcal{K}_k(A,r)$, $\hat{x} \neq 0$

$$\Rightarrow \hat{x} = V_k z,\ z \in \mathbb{C}^k,\ z \neq 0$$

$$V_k^H \,|\, AV_k z = A\hat{x} \approx \hat{\lambda}\hat{x} = \tilde{\lambda} V_k z \Rightarrow H_k z = \hat{\lambda} z$$

Eigenpair problem $A\hat{x} \approx \hat{\lambda}\hat{x} \iff H_k z = \hat{\lambda} z$

$\Rightarrow$ Compute k eigenpairs of $\mathcal{H}_k$:

$$(\hat{\lambda}_j, z_j), \quad j = 1, 2, \dots, k$$

~~& Then $\hat{x}_j := V_k z_j$.~~

$\Rightarrow$ approx. eigenpairs of A: $(\hat{\lambda}_j, \hat{x}_j)$, $j = 1, 2, \dots, k$

Note: Let $\|\cdot\|$ be a norm in $\mathbb{C}^n$

$\rho_j := \|A\hat{x}_j - \hat{\lambda}_j \hat{x}_j\|$ = measure of quality of approx. eigenpair $(\hat{\lambda}_j, \hat{x}_j)$.

We can compute ρ_j w/o forming $\hat{x}_j$:

$$A\hat{x}_j - \hat{\lambda}\hat{x}_j = \underbrace{AV_k}\, z_j - \hat{\lambda}_j V_k z_j$$

$$= V_k H_k + h_{k+1,k} \begin{bmatrix} 0 & \cdots & 0 & | \\ \vdots & & \vdots & v_{k+1} \\ \vdots & & \vdots & | \end{bmatrix}$$

$$= V_k(\underbrace{H_k z_j - \hat{\lambda}_j z_j}_{0}) + h_{k+1,k} (z_j)_k v_{k+1}$$

$$z_j = \begin{bmatrix} * \\ * \\ \vdots \\ * \\ (z_j)_k \end{bmatrix} \in \mathbb{C}^k$$

$\Rightarrow \rho_j = h_{k+1,k} |(z_j)_k| \cdot \|v_{k+1}\|$

In particular, for $\|\cdot\| = \|\cdot\|_2$, $\|v_{k+1}\|_2 = 1$

$$\Rightarrow \boxed{\rho_j = h_{k+1,k} \cdot |(z_j)_k|, \quad j = 1, 2, \dots, k}$$

Lec 23 - 3/5

2) Large-Scale Matrix functions

Ex: Matrix exponential: $f(\lambda)=e^{\lambda}=\sum_{j=0}^{\infty}\frac{1}{j!}\lambda^{j}\in\mathbb{C}$ converges for any $\lambda\in\mathbb{C}$

$\Rightarrow f(A)=e^{A}:=\sum_{j=0}^{\infty}\frac{1}{j!}A^{j}\in\mathbb{C}^{n\times n}$ converges for any $A\in\mathbb{C}^{n\times n}$

Pick $r\in\mathbb{C}^{n}$, $r\neq 0$, run Arnoldi for k steps to obtain: H_k, V_k $(k\ll n)$

$$H_k = V_k^{H} A V_k \Rightarrow A \approx V_k H_k V_k^{H} \Rightarrow A^{2} \approx V_k H_k \underbrace{V_k^{H} V_k}_{=I} H_k V_k^{H}$$

$$\Rightarrow A^{2} \approx V_k H_k^{2} V_k^{H}$$

$$\Rightarrow A^{j} \approx V_k H_k^{j} V_k^{H} \quad \forall j=0,1,\dots$$

$$\Rightarrow e^{A} \approx V_k\left(\sum_{j=0}^{\infty}\frac{1}{j!}H_k^{j}\right)V_k^{H} = V_k e^{H_k} V_k^{H}.$$

$$\underset{n\times n}{\begin{bmatrix} e^{A} \\ \text{(not sparse in general)} \end{bmatrix}} = \underset{n\times k}{\begin{bmatrix} V_k \end{bmatrix}} \underset{k\times k}{\boxed{e^{H_k}}} \underset{k\times n}{\begin{bmatrix} V_k^{H} \end{bmatrix}}$$

e^{A} not sparse for general sparse A, so don't even form product $V_k e^{H_k} V_k^{H}$!
$\rightarrow$ only compute e^{H_k}.

The Lanczos Process:

Special case: $A=A^{H}\in\mathbb{C}^{n\times n}$, $r\in\mathbb{C}^{n}$, $r\neq 0$

k steps of Arnoldi: $H_k = V_k^{H} A V_k = V_k^{H} A^{H} V_k = (V_k^{H} A V_k)^{H} = H_k^{H}$

Recall: $H_k = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1k} \\ h_{21} & h_{22} & \ddots & \vdots \\ & \ddots & \ddots & h_{k-1,k} \\ 0 & & h_{k,k-1} & h_{kk} \end{bmatrix}$, so in this case $H_k = H_k^{H} = \begin{bmatrix} h_{11} & h_{12} & & 0 \\ h_{21} & h_{22} & \ddots & \\ & \ddots & \ddots & h_{k-1,k} \\ 0 & & h_{k,k-1} & h_{kk} \end{bmatrix}$ is real, symmetric, & tridiagonal!

Set $H_k =: T_k = \begin{bmatrix} \alpha_1 & \beta_2 & & 0 \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_k \\ 0 & & \beta_k & \alpha_k \end{bmatrix}$

Then $AV_k = V_k T_k + \beta_{k+1} \begin{bmatrix} 0 & \cdots & 0 & v_{k+1} \end{bmatrix}$

$\Leftrightarrow Av_j = \beta_j v_{j-1} + \alpha_j v_j + \beta_{j+1} v_{j+1}$

$\Leftrightarrow \boxed{\beta_{j+1} v_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}, \quad j = 1, 2, \dots, k}$

Arnoldi for $A = A^H$ = <u>Hermitian Lanczos Process</u>

<u>Alg - Hermitian Lanczos Process</u>:

<u>Input</u>: a routine to compute $q = Av$
$r \in \mathbb{C}^n$, $r \neq 0$

```
· Set $\beta_1 = \|r\|_2$ and $v_1 = r/\beta_1$
· For $k = 1, 2, \dots$ do:
    · Compute $q = Av_k$
    · If $k > 1$, set $q = q - \beta_k v_{k-1}$
    · Compute $\alpha_k = v_k^H q$
    · Set $q = q - \alpha_k v_k$
    · Set $\beta_{k+1} = \|q\|_2$
    · If $\beta_{k+1} = 0$, stop: $k = d(A, r)$
    · Set $v_{k+1} = q / \beta_{k+1}$.
  end
```

<u>Output</u>: V_k, T_k

<u>Note</u>: Work per k^{th} iteration is constant:
1 product Av, 2 inner products, 2 SAXPYs, 1 division of vector by scalar

<u>The general case</u>: $A \in \mathbb{C}^{n \times n}$. We still have $AV_k = V_k T_k + \beta_{k+1} \begin{bmatrix} 0 & \cdots & 0 & v_{k+1} \end{bmatrix}$ (*v)
but now the v_k's are no longer orthonormal!

We also consider $w_1, w_2, \dots$ s.t. $\text{span}\{w_1, \dots, w_k\} = K_k(A^T, c)$, $k = 1, 2, \dots, d(A^T, c)$
where $c \in \mathbb{C}^n$, $c \neq 0$ is a "left" starting vector

Recurrence relations in compact form:

$$(*w)\quad A^T W_k = W_k \hat{T}_k + \gamma_{k+1} \begin{bmatrix} | & | & & | & | \\ 0 & 0 & \cdots & 0 & w_{k+1} \\ | & | & & | & | \end{bmatrix}$$

where $W_k = \begin{bmatrix} | & & | \\ w_1 & \cdots & w_k \\ | & & | \end{bmatrix}$ and $\hat{T}_k$ is tridiagonal

Notation:

v_k's are called right Lanczos vectors

w_k's " " left " "

Lec 24 - 3/7/18

General-case Lanczos Process (Nonsymmetric A)

$A \in \mathbb{C}^{n \times n}$, $r, c \in \mathbb{C}^n$, $r, c \neq 0$

$\text{span}\{v_1, v_2, \ldots, v_k\} = \mathcal{K}_k(A, r)$

$\text{span}\{w_1, w_2, \ldots, w_k\} = \mathcal{K}_k(A^T, c)$

$$(*v)\quad A V_k = V_k T_k + \beta_{k+1} \begin{bmatrix} | & & | & | \\ 0 & \cdots & 0 & v_{k+1} \\ | & & | & | \end{bmatrix}$$

$$(*w)\quad A^T W_k = W_k \hat{T}_k + \gamma_{k+1} \begin{bmatrix} | & & | & | \\ 0 & & 0 & w_{k+1} \\ | & & | & | \end{bmatrix}, \quad T_k, \hat{T}_k \in \mathbb{C}^{k \times k} \text{ tridiagonal}$$

Note: The Lanczos vectors are constructed to be biorthonormal:

$w_j^T v_k = 0 \quad \forall\, j \neq k, \quad j, k = 1, 2, \ldots$

Convenient normalization: $\|v_k\|_2 = \|w_k\|_2 = 1 \quad \forall\, k.$

Alg (Nonsymmetric Lanczos Process):

on next page

Alg 1 Nonsymmetric Lanczos Process

Input: routine to compute $q = Av$ for $v \in \mathbb{C}^n$

" " " $s = A^T w$ for $w \in \mathbb{C}^n$

$r, c \in \mathbb{C}^n$, $r, c \neq 0$

```
· Set $\beta_1 = \|r\|_2$, $\gamma_1 = \|c\|_2$, $v_1 = r/\beta_1$, $w_1 = c/\gamma_1$.
For k = 1, 2, ... do:
    · Compute $\delta_k = w_k^T v_k$, if $\delta_k = 0$, stop: "breakdown" of the algorithm
    · Compute $q = Av_k$ and $s = Aw_k$
    · If $k > 1$, set $q = q - (\gamma_k \frac{\delta_k}{\delta_{k-1}}) v_{k-1}$   $(\xrightarrow{\text{ensures}} w_{k-1}^T q = 0)$
            & set $s = s - (\beta_k \frac{\delta_k}{\delta_{k-1}}) w_{k-1}$   $(\rightarrow s^T v_{k-1} = 0)$
    · Set $\alpha_k = \frac{w_k^T q}{\delta_k}$ and $q = q - \alpha_k v_k$   $(\rightarrow w_k^T q = 0)$
                                $s = s - \alpha_k w_k$   $(\rightarrow s^T v_k = 0)$
    · Set $\beta_{k+1} = \|q\|_2$ and $\gamma_{k+1} = \|s\|_2$
    · If $\beta_{k+1} = 0$, stop: $\mathcal{K}_k(A, r)$ has reached its maximal dimension $k = d(A, r)$.
    · If $\gamma_{k+1} = 0$, stop: $\mathcal{K}_k(A^T, c)$ " " " " " $k = d(A^T, c)$.
                                                In general, these are dif
    · Set $v_{k+1} = q/\beta_{k+1}$, $w_{k+1} = s/\gamma_{k+1}$.
end
```

Properties: 1) In exact arithmetic: If no breakdown occurs, the algorithm will stop for $k = \min(d(A,r), d(A^T, c)) \leq n$

2) The Lanczos vectors satisfy the 3-term recurrences

$$\beta_{k+1} v_{k+1} = A v_k - \alpha_k v_k - \underbrace{\boxed{\gamma_k \frac{\delta_k}{\delta_{k-1}}}}_{=: \eta_k} v_{k-1}$$

$$\gamma_{k+1} w_{k+1} = A^T w_k - \alpha_k w_k - \underbrace{\boxed{\beta_k \frac{\delta_k}{\delta_{k-1}}}}_{=: \varphi} w_{k-1}$$

These recurrences are just (*v) & (*w) with

$$T_k = \begin{bmatrix} \alpha_1 & \eta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \eta_k \\ 0 & & \beta_k & \alpha_k \end{bmatrix}, \quad \hat{T}_k = \begin{bmatrix} \alpha_1 & \xi_2 & & \\ \gamma_2 & \ddots & \ddots & \\ & \ddots & \ddots & \xi_k \\ & & \gamma_k & \alpha_k \end{bmatrix}$$

3) Biorthonormality $\iff W_k^T V_k = \begin{bmatrix} \delta_1 & & & 0 \\ & \delta_2 & & \\ & & \ddots & \\ 0 & & & \delta_k \end{bmatrix} =: D_k$

4) $W_k^T A V_k = \underbrace{W_k^T V_k}_{D_k} T_k + \underbrace{\cancel{\beta_{k+1} W_k^T \begin{bmatrix} | & & | & \\ 0 & \cdots & 0 & v_{k+1} \\ | & & | & \end{bmatrix}}}_{=0 \text{ by biorthonormality}}$

$\Rightarrow W_k^T A V_k = D_k T_k$

Similarly, $V_k^T A^T W_k = V_k^T W_k \hat{T}_k = D_k \hat{T}_k$

$\Rightarrow \boxed{\hat{T}_k = D_k^{-1} T_k^T D_k}$

5) $T_k, \hat{T}_k$ have the same eigenvalues

6.) The algorithm can be modified so that one can continue in the case of breakdown: <u>look-ahead Lanczos</u>

7) The matrix T_k represents an <u>oblique projection</u> of A onto $\mathcal{K}_k(A, r)$ & orthogonal to $\mathcal{K}_k(A^T, c)$:

$$(W_k^T V_k)^{-1} W_k^T A V_k = T_k$$

Lec 25 - 3/9

Proof of 7: $W_k^T A V_k = \underbrace{W_k^T V_k}_{= D_k \text{ nonsingular}} T_k + \beta_{k+1} \left[\begin{matrix} 0 \\ | \end{matrix} \cdots \begin{matrix} 0 \\ | \end{matrix} \; \underbrace{\cancel{W_k^T v_{k+1}}}_{0} \right]$

$\Rightarrow (W_k^T V_k)^{-1} W_k^T A V_k = T_k$. //

8) Like Arnoldi process, the Lanczos process has many applications in large-scale matrix computations

One application: approximate eigentriples

$A \in \mathbb{C}^{n \times n}$. $Ax = \lambda x$, $x \neq 0 \to x$ is a right eigenvector of A
λ is eigenvalue of A

$A^T y = \lambda y \;\; \to \; y$ is a left eigenvector of A
$(\Leftrightarrow y^T A = \lambda y^T)$

(λ, x, y): eigentriple of A (for symmetric A, $x = y$)

Approximate eigentriple $(\hat\lambda, \hat x, \hat y)$: $A\hat x \approx \hat\lambda \hat x$, $\hat x \neq 0$
$A^T \hat y \approx \hat\lambda \hat y$, $\hat y \neq 0$

Run k steps of Lanczos process:

$\hat x = V_k z$, $z \neq 0$
$\hat y = W_k u$, $u \neq 0$

$\Rightarrow W_k^T A V_k z = W_k^T A \hat x \approx W_k^T \hat\lambda \hat x = \hat\lambda \underbrace{W_k^T V_k}_{= D_k} z$

$\Rightarrow \underbrace{D_k^{-1} W_k^T A V_k}_{T_k} z = \hat\lambda z \quad \Rightarrow \quad T_k z = \hat\lambda z$ gives k approx. eigvals. λ_i, z_i

k approximate right-eigvectors: $\hat x_i = V_k z_i$

Similarly, $\hat T_k u = \hat\lambda u$ yields same k eigvals (by prop 5) $\Rightarrow \lambda_i, u_i$

k approx. left-eigvectors $\hat y_i = W_k u_i$

Approx. eigentriples via Lanczos process

For general $A \in \mathbb{C}^{n \times n}$ run k steps of Lanczos process

- Compute k eigenvalues of T_k: $\tilde{\lambda}_j$, $j = 1, 2, \dots, k$
- compute corresponding (right) eigenvectors z_j of T_k & corr. (left) eigvectors u_j of $\hat{T}_k$
- Set $\hat{x}_j = V_k z_j$ & $\hat{y} = W_k u_j$

$\Rightarrow$ yields k approx. eigentriples of A: $(\hat{\lambda}_j, \hat{x}_j, \hat{y}_j)$, $j = 1, 2, \dots, k$

Domain Decomposition

Basic idea: Solve PDE $\left(L = -\frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial y^2}\right)$

$$(*) \quad Lu = f \text{ in } R$$
$$u = g \text{ on } \partial R$$

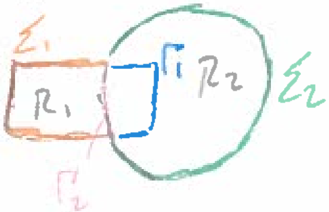
By solving p subproblems

$$Lu_i = f \text{ in } R_i \qquad \text{w/ } R = R_1 \cup R_2 \cup \dots \cup R_p$$
$$u_i = g_i \text{ on } \delta R_i$$

We will only consider $p = 2$.

Classical alternating Schwarz-method (Hermann Schwarz, 1870)

$R_1 \cap R_2 \neq \emptyset$:



solve PDE analytically on R_1 & R_2

Γ_1, Γ_2 = artificial boundaries

$\Sigma_1 \cup \Sigma_2 = \delta R$

$\Sigma_i \cup \Gamma_i = \delta R_i$

Alg (Alternating Schwarz method)

(Solving $Lu = f$ in R, $u = g$ on δR

via $Lu_i = f$ in R_i, $u_i = g$ on δR_i, $i = 1, 2$)

Guess $u_2^{(0)}|_{\Gamma_1} \approx u|_{\Gamma_1}$

```
For $n = 1, 2, \ldots$
    1) Solve $\begin{cases} Lu_1^{(n)} = f \text{ on } R_1 \\ u_1^{(n)} = \begin{cases} g \text{ on } \Sigma_1 \\ u_2^{(n-1)} \text{ on } \Gamma_1 \end{cases} \end{cases}$   for $u_1^{(n)}$
    2) Solve $\begin{cases} Lu_2^{(n)} = f \text{ on } R_2 \\ u_2^{(n)} = \begin{cases} g \text{ on } \Sigma_2 \\ u_1^{(n)} \text{ on } \Gamma_2 \end{cases} \end{cases}$   for $u_2^{(n)}$
end
```

Schwarz proved that $\lim_{n\to\infty} u_i^{(n)} = u$,

true soln of (*).

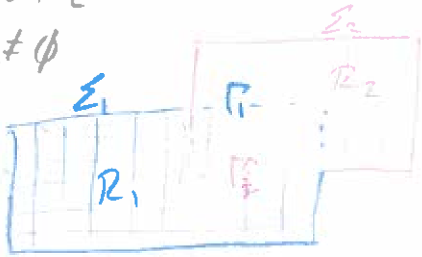
Lec 26 - 3/12

(*) $Lu = f$ in R

$u = g$ on δR

$R = R_1 \cup R_2$

$R_1 \cap R_2 \neq \emptyset$



$\Sigma_i = \delta R_i - \Gamma_i$

L: linear in u, Ex: $Lu = -u_{xx} - u_{yy}$

Discretization of (*): $Av = b$

Discretized Alternating Schwarz Method (nonmatching grids)

Problem on R_1: $Lu_1 = f$

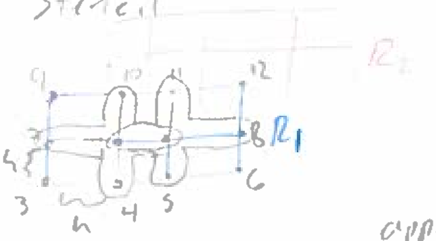
$$u_1 = \begin{cases} g & \text{on } \Sigma_1 \\ u_2 & \text{on } \Gamma_1 \end{cases}$$

" " R_2: $Lu_2 = f$

$$u_2 = \begin{cases} g & \text{on } \Sigma_2 \\ u_1 & \text{on } \Gamma_2 \end{cases}$$

Discretized?

Ex: 5-pt Stencil



nodes 1-2: interior grid points of R_1

" 3-9: "true" boundary gridpoints (on Σ_1)

10-12: artifical " " (on Γ_1)

approx. values of u at these points: $\begin{cases} V_1 \\ V_{\Sigma_1} = g_{\Sigma_1} \\ V_{\Gamma_1} \end{cases}$

$$\left[\begin{array}{cc:ccccccc:ccc} \multicolumn{2}{c}{=A_1} & \multicolumn{7}{c}{=B_{\Sigma_1}} & \multicolumn{3}{c}{=B_{\Gamma_1}} \\ 4 & -1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ -1 & 4 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 \end{array}\right] \begin{bmatrix} V_1 \\ - \\ V_{\Sigma_1} \\ V_{\Gamma_1} \end{bmatrix} = h^2 f_1$$

↖ f eval on interior grid points of R_1

$$\Rightarrow A_1 V_1 + \underline{B_{\Gamma_1} V_{\Gamma_1}} = h^2 f_1 - \underline{B_{\Sigma_1} g_{\Sigma_1}} =: b_1$$

Ideally: $V_{\Gamma_1} = u|_{\Gamma_1}$, but we dont have u, just V_2

Set $V_{\Gamma_1} = I_{R_2}^{\Gamma_1} V_2$ with an interpolation "operator" that uses grid points in R_2 to obtain values on Γ_1.

With this approximation:

$$A_1 V_1 + B_{\Gamma_1} I_{R_2}^{\Gamma_1} V_2 = b_1$$

Same procedure in R_2: $B_{\Gamma_2} I_{R_1}^{\Gamma_2} V_1 + A_2 V_2 = b_2$

Together discretization of (*): $\begin{bmatrix} A_1 & B_{\Gamma_1} I_{R_2}^{\Gamma_1} \\ B_{\Gamma_2} I_{R_1}^{\Gamma_2} & A_2 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$ (**)

Schwarz method:

Splits $A = \begin{bmatrix} A_1 & B_{\Gamma_1} I_{R_2}^{\Gamma_1} \\ B_{\Gamma_2} I_{R_1}^{\Gamma_2} & A_2 \end{bmatrix} = \begin{bmatrix} A_1 & 0 \\ B_{\Gamma_2} I_{R_1}^{\Gamma_2} & A_2 \end{bmatrix} - \begin{bmatrix} 0 & -B_{\Gamma_1} I_{R_2}^{\Gamma_1} \\ 0 & 0 \end{bmatrix}$

Alg (Discretized Alternating Schwarz method):

- Choose initial guess for $v_2^{(0)}$, e.g. $v_2^{(0)} = 0$
- For $n = 1, 2, \ldots$:
    1) Solve $A_1 v_1^{(n)} = b_1 - B_{\Gamma_1} I_{R_2}^{\Gamma_1} v_2^{(n-1)}$ for $v_1^{(n)}$
    2) Solve $A_2 v_2^{(n)} = b_2 - B_{\Gamma_2} I_{R_1}^{\Gamma_2} v_1^{(n)}$ for $v_2^{(n)}$
    3) Set $v^{(n)} = \begin{bmatrix} v_1^{(n)} \\ v_2^{(n)} \end{bmatrix}$. If $\|b - Av^{(n)}\| < tol$, stop.

Notes: 1) This alg. is just block Gauss-Siedel applied to (A)

2) Krylov subspace methods can be used here & are much faster!

lec27 - 3/14

(*) $Lu = f$ in R, $u = g$ on δR

$R = R_1 \cup R_2$, $R_1 \cap R_2 \neq \emptyset$

(A*) $\begin{bmatrix} A_1 & B_{\Gamma_1} I_{R_2}^{\Gamma_1} \\ B_{\Gamma_2} I_{R_1}^{\Gamma_2} & A_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \iff Av = b$

Use $M = \begin{bmatrix} A_1 & 0 \\ B_{\Gamma_2} I_{R_1}^{\Gamma_2} & A_2 \end{bmatrix}$ as a left-preconditioner for $Av=b$

& solve w/ GMRES:

$$Av = b \iff M^{-1}Av = M^{-1}b$$

Alg: (GMRES w/ alternating Schwarz as preconditioner)

1) Choose initial guess for $v^{(0)} = \begin{bmatrix} v_1^{(0)} \\ v_2^{(0)} \end{bmatrix}$, e.g., $v^{(0)} = 0$

2) Apply GMRES to $A'v = b'$

Notes: 1) Each matrix-vector product $q = A'p$ can be computed efficiently w/ 1 solve w/ A_1 and 1 solve w/ A_2:

$$A = \begin{bmatrix} A_1 & A_{12} \\ A_{21} & A_2 \end{bmatrix}, \quad M = \begin{bmatrix} A_1 & 0 \\ A_{21} & A_2 \end{bmatrix}, \quad \begin{matrix} A_{12} = B_{\Gamma_1} I_{R_2}^{\Gamma_1} \\ A_{21} = B_{\Gamma_2} I_{R_1}^{\Gamma_2} \end{matrix}$$

$$A' = M^{-1}A = M^{-1}\left(M + \begin{bmatrix} 0 & A_{12} \\ 0 & 0 \end{bmatrix}\right)$$

$$= I + M^{-1}\begin{bmatrix} 0 & A_{12} \\ 0 & 0 \end{bmatrix}, \text{ where } M^{-1} = \begin{bmatrix} A_1^{-1} & 0 \\ -A_2^{-1}A_{21}A_1^{-1} & A_2^{-1} \end{bmatrix}$$

$$= I + \begin{bmatrix} 0 & A_1^{-1}A_{12} \\ 0 & -A_2^{-1}A_{21}A_1^{-1}A_{12} \end{bmatrix}$$

$$q = A'p = \left(I + \begin{bmatrix} 0 & A_1^{-1}A_{12} \\ 0 & -A_2^{-1}A_{21}A_1^{-1}A_{12} \end{bmatrix}\right)\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = p + \begin{bmatrix} \underbrace{A_1^{-1}A_{12}p_2}_{=:t_1} \\ -\underbrace{A_2^{-1}A_{21}t_1}_{=:t_2} \end{bmatrix}$$

$$= \begin{bmatrix} p_1 + t_1 \\ p_2 - t_2 \end{bmatrix}$$

Don't do by inverse construction & multiplication!

Instead, let t_1 be soln of $A_1 t_1 = A_{12} p_2$
t_2 " " $A_2 t_2 = A_{21} t_1$

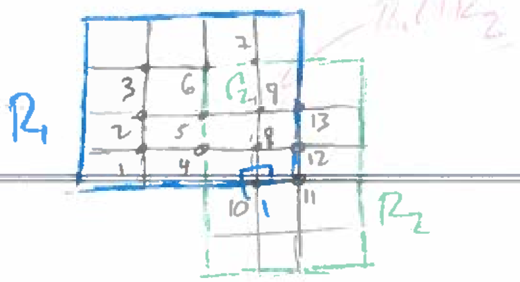
2) The solves w/ A_1 & A_2 cannot be done in parallel!

3) If R_1 & R_2 use matching grids, then $I_{R_2}^{\Gamma_1}$ & $I_{R_1}^{\Gamma_2}$ are just "pruned" identity matrices, e.g.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Discretized Schwarz methods (matching grids)

Ex:



- General problem (**) w/ matching grids on $R_1 \cap R_2$
- Discretization of (*):

$$A v = b$$

b ← f at interior points + values of g on boundary

v ↑ vector of approx. values at interior grid points of R.

We order v s.t.

$$v = \begin{bmatrix} v_{R_1 \setminus \bar{R}_2} \\ v_{\Gamma_2} \\ v_{R_1 \cap R_2} \\ v_{\Gamma_1} \\ v_{R_2 \setminus \bar{R}_1} \end{bmatrix} \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} v_1 : \text{values at grid points in } R_1 \\ \left.\begin{array}{l} \\ \\ \end{array}\right\} v_2 : \text{'' '' '' '' } R_2 \end{array}$$

Note that $v_1 = I_1 v$, $I_1 = [\underset{\substack{\uparrow \\ \text{size = length of } v_1}}{I} \quad 0] : R_1 \to R_1$

$v_2 = I_2 v$, $I_2 = [0 \quad \overset{\substack{\text{size = length } v_2 \\ \downarrow}}{I}] : R \to R_2$

Thus $A_1 = I_1 A I_1^T$ ($\Leftrightarrow$ discretization on R_1)

$A_2 = I_2 A I_2^T$ ($\Leftrightarrow$ '' '' R_2)

$$A = \begin{bmatrix} A_1 & \\ & A_2 \end{bmatrix}$$

w/ $Lu = -u_{xx} - u_{yy}$ via 5-point stencil : $Av = b \Leftrightarrow$

$$I_1 = [\underset{9\times 9}{I} \quad \underset{9\times 6}{0}]$$

$$I_2 = [\underset{8\times 7}{0} \quad \underset{8\times 8}{I}]$$

$$\left[\begin{array}{ccc:cccc|cc|cccc|cc}
4 & -1 & 0 & -1 & 0 & 0 & 0 & & & & & & & & \\
-1 & 4 & -1 & 0 & -1 & 0 & 0 & \multicolumn{2}{c|}{0} & & 0 & & & & \\
0 & -1 & 4 & 0 & 0 & -1 & 0 & & & & & & & & 0 \\
\hdashline
-1 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & & & & & & \\
0 & -1 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & & & & & & \\
0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 & 0 & & 0 & & & & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 4 & 0 & -1 & & & & & & \\
\hline
 & 0 & & -1 & 0 & 0 & 0 & 4 & -1 & -1 & 0 & -1 & 0 & & 0 \\
 & & & 0 & -1 & 0 & -1 & -1 & 4 & 0 & 0 & 0 & -1 & & 0 \\
\hline
 & 0 & & & 0 & & & -1 & 0 & 4 & -1 & 0 & 0 & -1 & \\
 & & & & & & & 0 & 0 & -1 & 4 & -1 & 0 & & \\
 & & & & & & & -1 & 0 & 0 & -1 & 4 & -1 & & \\
 & & & & & & & 0 & -1 & 0 & 0 & -1 & 4 & & 0 \\
\hdashline
 & 0 & & & 0 & & & & 0 & -1 & 0 & 0 & 0 & 4 & -1 \\
 & & & & & & & & & 0 & -1 & 0 & 0 & -1 & 4
\end{array}\right]$$

A_1

A_2

Alg: (Multiplicative Schwarz method).

```
· Choose initial guess for $v^{(0)}$, e.g. $v^{(0)} = 0$
· For $n = 0, 1, 2, \ldots$, do:
                    brings back to R
                              cuts down to R_1
    · $v^{(n+1/2)} = v^{(n)} + I_1^T A_1^{-1} I_1 (b - Av^{(n)})$
    · $v^{(n+1)} = v^{(n+1/2)} + I_2^T A_2^{-1} I_2 (b - Av^{(n+1/2)})$
- end
```

☆ requires 1 solve per it with both A_1 & A_2

☆ cannot be parallelized, has slow, linear convergence rate.

Lec 28 - 3/16

Error at n^{th} step: $e^{(n)} := A^{-1}b - v^{(n)}$ (= error vector)

Residual: $Ae^{(n)} := b - Av^{(n)}$ (= residual ")

$$e^{(n+1/2)} = e^{(n)} - I_1^T A_1^{-1} I_1 A e^{(n)} = (I - \underbrace{I_1^T A_1^{-1} I_1 A}_{=: P_1}) e^{(n)}$$

$$e^{(n+1)} = (I - \underbrace{I_2 A_2^{-1} I_2 A}_{=: P_2}) e^{(n+1/2)} = (I - P_2)(I - P_1) e^{(n)}$$

$\Rightarrow$ error is the product of two matrices & previous error $\Rightarrow$ multiplicative method

For "sufficient" overlap of R_1 and R_2:

$$\|e^{(n)}\| \le \rho^n \|e^{(0)}\|, \quad \rho < 1$$

Multiplicative Schwarz as preconditioner

$$P_i = B_i A \quad , \text{where} \quad B_i = I_i^T A_i^{-1} I_i \, , \, i = 1, 2$$

$$\Rightarrow e^{(n+1)} = (I - P_1 - P_2 + P_2 P_1) e^{(n)} = (I - (B_1 + B_2 - B_2 A B_1) A) e^{(n)}$$

Set $M := (B_1 + B_2 - B_2 A B_1)^{-1} \Rightarrow e^{(n+1)} = (I - M^{-1}A) e^{(n)}$

M preconditioner for A $\iff M \approx A \iff I \approx M^{-1}A \iff I - M^{-1}A \approx 0$)

Use M as a preconditioner for a Krylov subspace method for solving $Av=b$

Need solves w/ M: $Mq=r \iff q = M^{-1}r = (B_1+B_2-B_2AB_1)r$

But $M \neq M^T$ because of the term B_2AB_1 in

$\Rightarrow$ we cannot use CG, even when $A \succ 0$!

Additive Schwarz Method

```
For $n = 0, 1, 2, \ldots$
    $v^{(n+1/2)} = v^{(n)} + B_1(b - Av^{(n)})$
    $v^{(n+1)} = v^{(n+1/2)} + B_2(b - Av^{(n)})$
end
```

can compute in parallel now!

this is [illegible] $(n+\frac{1}{2})$ [illegible]

*Not good as a stand-alone, since messed up the accuracy, but opens doors as a preconditioner!

Now: $e^{(n+1)} = (I - (B_1+B_2)A)e^{(n)}$

Use $M = (B_1+B_2)^{-1}$ as a preconditioner

M is called the additive Schwarz preconditioner.

Notes: 1) If $A \succ 0$, then $M \succ 0$

$\Rightarrow$ use CG with left preconditioner M

2) $Mq = r \iff q = (B_1+B_2)r = B_1r + B_2r$

$\Rightarrow$ subdomain solves for Ω_1 & Ω_2 can be done in //

3) All of this extends to any number of subdomains $\Omega_1, \Omega_2, \ldots, \Omega_p$.