Failure to pivot can result in all subsequent rows looking like multiples of the kth row:

- Consider

$$A = \begin{pmatrix} \epsilon & —\underline{r}_1^T— \\ a_{21} & —\underline{r}_2^T— \\ a_{31} & —\underline{r}_3^T— \\ \vdots & — \vdots — \end{pmatrix}$$

Gaussian elimination leads to

$$\underline{r}_i \longleftarrow \underline{r}_i - \frac{a_{i1}}{\epsilon}\underline{r}_1 \approx -\frac{a_{i1}}{\epsilon}\underline{r}_1.$$

- Matlab example “pivot.m”

End of Lecture 3

pivot_gui.m

1.0e-18	1.0000	2.0000	3.0000	4.0000
1.0000	4.0000	4.0000	6.0000	1.0000
2.0000	8.0000	7.0000	9.0000	2.0000
3.0000	6.0000	1.0000	3.0000	3.0000
4.0000	4.0000	2.0000	8.0000	4.0000

Failure to Pivot, Noncatastrophic Case

- In cases where the nominal pivot is small but > ϵ_M, we are effectively reducing the number of significant digits that represent the remainder of the matrix A.
- In essence, we are driving the rows (or columns) to be ***similar***, which is equivalent to saying that we have nearly parallel columns.
- We will see next time a 2 x 2 example where the condition number of the matrix with 2 unit-norm vectors scales like 2 / θ , where θ is the (small) angle between the column vectors.

Partial Pivoting: Costs

Procedure:

- For each k, pick k' such that $|a_{k'k}| \geq |a_{ik}|$, $i \geq k$.
- Swap rows k and k'.
- Proceed with central update step: $A^{(k+1)} = A^{(k)} - \mathbf{c}_k \, \mathbf{r}_k^T$

Costs:

- For each step, search is $O(n-k)$, total cost is $\approx n^2/2$.
- For each step, row swap is $O(n-k)$, total cost is $\approx n^2/2$.
- Total cost for partial pivoting is $O(n^2)\lambda 2n^3/3$.
- If we use *full pivoting*, total search cost such that $|a_{k'k''}| \geq |a_{ij}|$, $i, j \geq k$, is $O(n^3)$.
- Row and column exchange costs still total only $O(n^2)$.

Notes:

- Partial (row) pivoting ensures that multiplier column entries have modulus ≤ 1. (Good.)
- Full pivoting also destroys band structure, whereas partial pivoting leaves some band structure intact.

Partial Pivoting: LU=PA

- Note: If we swap rows of A, we are swapping equations.
- We must swap rows of $\mathbf{b}$.
- LU routines normally return the pivot index vector to effect this exchange.
- Nominally, it looks like a permutation matrix P, which is simply the identity matrix with rows interchanged.
- If we swap equations, we must also swap rows of L
- If we are consistent, we can swap rows at any time (i.e., A, or L) and get the same final factorization: $LU = PA$.
- Most codes swap $A^{(k+1)}$, but not the factors in L that have already been stored.
- Swapping rows of $A^{(k+1)}$ helps with speed (vectorization) of $A^{(k+1)} = A^{(k)} - \mathbf{c}_k \, \mathbf{r}_k^T$.
- In parallel computing, one would *not* swap the pivot row. Just pass the pointer to the processor holding the new pivot row, where the swap would take place locally.

Pivoting, continued

- Although pivoting is generally required for stability of Gaussian elimination, pivoting is *not* required for some important classes of matrices
  - *Diagonally dominant*

$$\sum_{i=1,\, i\neq j}^{n} |a_{ij}| < |a_{jj}|, \quad j = 1, \ldots, n$$

  - *Symmetric positive definite*

$$\boldsymbol{A} = \boldsymbol{A}^T \quad \text{and} \quad \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} > 0 \ \text{ for all } \ \boldsymbol{x} \neq \boldsymbol{0}$$

Uniqueness of LU Factorization

- Despite variations in computing it, LU factorization is unique up to diagonal scaling of factors
- Provided row pivot sequence is same, if we have two LU factorizations $\boldsymbol{PA} = \boldsymbol{LU} = \hat{\boldsymbol{L}}\hat{\boldsymbol{U}}$, then $\hat{\boldsymbol{L}}^{-1}\boldsymbol{L} = \hat{\boldsymbol{U}}\boldsymbol{U}^{-1} = \boldsymbol{D}$ is both lower and upper triangular, hence diagonal
- If both $\boldsymbol{L}$ and $\hat{\boldsymbol{L}}$ are unit lower triangular, then $\boldsymbol{D}$ must be identity matrix, so $\boldsymbol{L} = \hat{\boldsymbol{L}}$ and $\boldsymbol{U} = \hat{\boldsymbol{U}}$
- Uniqueness is made explicit in LDU factorization $\boldsymbol{PA} = \boldsymbol{LDU}$, with $\boldsymbol{L}$ unit lower triangular, $\boldsymbol{U}$ unit upper triangular, and $\boldsymbol{D}$ diagonal

Storage Management

- Elementary elimination matrices $\boldsymbol{M}_k$, their inverses $\boldsymbol{L}_k$, and permutation matrices $\boldsymbol{P}_k$ used in formal description of LU factorization process are *not* formed explicitly in actual implementation
- $\boldsymbol{U}$ overwrites upper triangle of $\boldsymbol{A}$, multipliers in $\boldsymbol{L}$ overwrite strict lower triangle of $\boldsymbol{A}$, and unit diagonal of $\boldsymbol{L}$ need not be stored
- Row interchanges usually are not done explicitly; auxiliary integer vector keeps track of row order in original locations

Inversion vs. Factorization

- Even with many right-hand sides $\boldsymbol{b}$, inversion never overcomes higher initial cost, since each matrix-vector multiplication $\boldsymbol{A}^{-1}\boldsymbol{b}$ requires n^2 operations, similar to cost of forward- and back-substitution
- Inversion gives less accurate answer; for example, solving $3x = 18$ by division gives $x = 18/3 = 6$, but inversion gives $x = 3^{-1} \times 18 = 0.333 \times 18 = 5.99$ using 3-digit arithmetic
- Matrix inverses often occur as convenient notation in formulas, but explicit inverse is rarely required to implement such formulas
- For example, product $\boldsymbol{A}^{-1}\boldsymbol{B}$ should be computed by LU factorization of $\boldsymbol{A}$, followed by forward- and back-substitutions using each column of $\boldsymbol{B}$

Band Matrices

- Gaussian elimination for band matrices differs little from general case — only ranges of loops change
- Typically matrix is stored in array by diagonals to avoid storing zero entries
- If pivoting is required for numerical stability, bandwidth can grow (but no more than double)
- General purpose solver for arbitrary bandwidth is similar to code for Gaussian elimination for general matrices
- For fixed small bandwidth, band solver can be extremely simple, especially if pivoting is not required for stability

Tridiagonal Matrices

- Consider tridiagonal matrix

$$\boldsymbol{A} = \begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{bmatrix}$$

- Gaussian elimination without pivoting reduces to

$d_1 = b_1$
for $i = 2$ **to** n
  $m_i = a_i / d_{i-1}$
  $d_i = b_i - m_i c_{i-1}$
end

Cost is O(n) !

Tridiagonal Matrices, continued

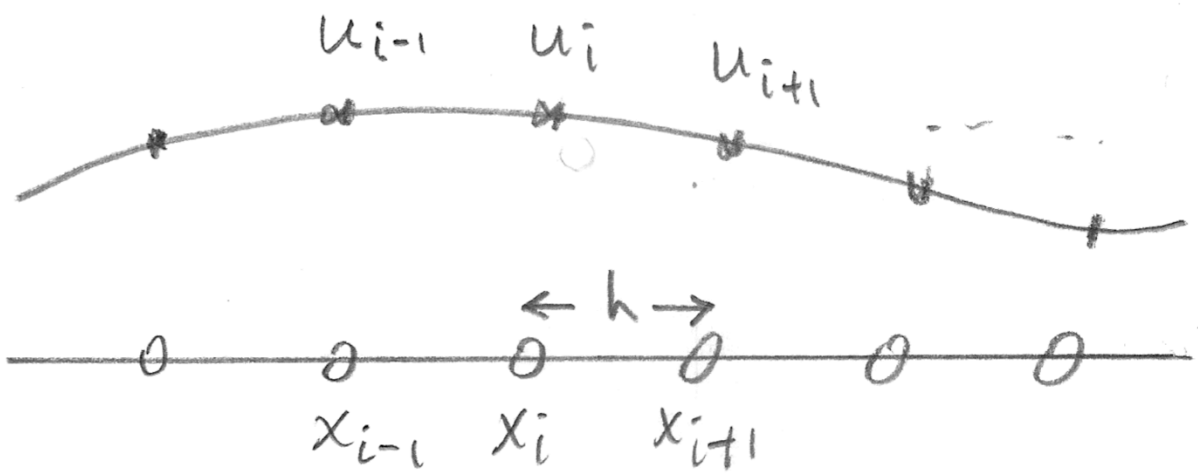
- LU factorization of $\boldsymbol{A}$ is then given by

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ m_2 & 1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & m_{n-1} & 1 & 0 \\ 0 & \cdots & 0 & m_n & 1 \end{bmatrix}, \quad \boldsymbol{U} = \begin{bmatrix} d_1 & c_1 & 0 & \cdots & 0 \\ 0 & d_2 & c_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & d_{n-1} & c_{n-1} \\ 0 & \cdots & \cdots & 0 & d_n \end{bmatrix}$$

Example of Banded Systems

- Graphs (i.e., matrices) arising from differential equations in 1D, 2D, 3D (and higher…) are generally banded and sparse.

- Example:



$$-\frac{d^2u}{dx^2} = f(x) \quad \longrightarrow \quad -\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \approx f_i$$

In Matrix Form

$$-\frac{d^2u}{dx^2} = f(x) \quad \longrightarrow \quad -\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \approx f_i$$

$$A_{1D} = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_m \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_m \end{pmatrix}$$

- Banded, tridiagonal matrix (“1D Poisson Operator”)

Some Hints For HW1

- Consider the tridiagonal matrix system, $A\underline{x} = \underline{f}$,

$$\underbrace{\begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & a_3 & \ddots & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ & & & a_n & b_n \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix}}_{\underline{x}} = \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_n \end{pmatrix}}_{\underline{f}}.$$

- When solving this system, one only needs to store five vectors of length $O(n)$, namely, $\underline{a}$, $\underline{b}$, $\underline{c}$, $\underline{x}$, and $\underline{f}$. (Often, the solution is overwritten onto $\underline{f}$, so you don't actually need $\underline{x}$.) The code provided implements a tridiagonal system solve for this class of problems.

- Gaussian elimination for this system leads to the following pseudocode for the forward solve:

```
for i=2:n
  a_i = a_i/b_{i-1}          % Store row multiplier.
  b_i = b_i - a_i * c_{i-1}  % Update row i of A.
  f_i = f_i - a_i * f_{i-1}  % Update row i of f.
end
```

- The preceding loop factors the matrix A into the product $LU = A$, where L is unit-lower triangular and U is upper triangular. It also maps the original right-hand side to $\underline{f} \longleftarrow L^{-1}\underline{f}$.
- The remaining step is to compute $\underline{x} \longleftarrow U^{-1}\underline{f}$:

$$\underbrace{\begin{pmatrix} b_1 & c_1 & & & \\ & b_2 & c_2 & & \\ & & \ddots & \ddots & \\ & & & \ddots & c_{n-1} \\ & & & & b_n \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix}}_{\underline{x}} = \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_n \end{pmatrix}}_{\underline{f}}.$$

- Pseudocode for this system is

$$\begin{aligned} & x_n \;=\; f_n \,/\, b_n \\ & \text{for i=(n-1):1} \\ & \quad x_i \;=\; \frac{1}{b_i}\,(f_i \;-\; c_i * x_{i+1}) \\ & \text{end} \end{aligned}$$

- For the HW, you are asked to solve a *periodic* matrix, which can be cast in the following form

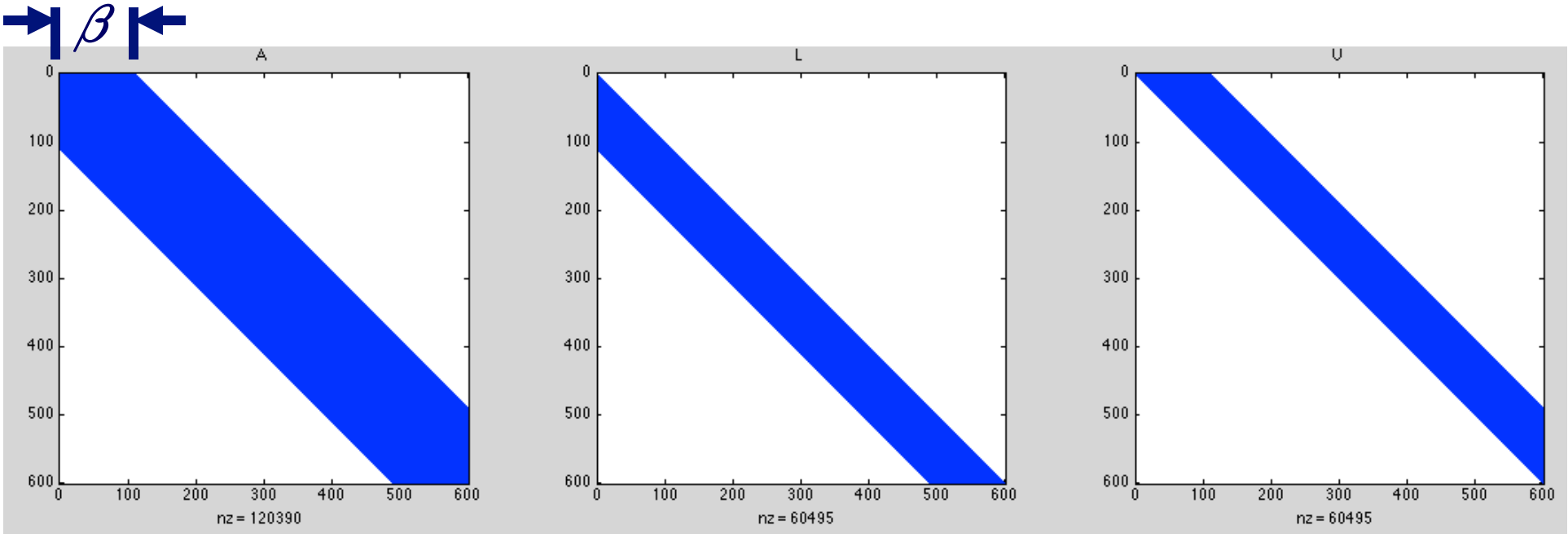
$$\underbrace{\begin{pmatrix} b_1 & c_1 & & & & d_1 \\ a_2 & b_2 & c_2 & & & d_2 \\ & a_3 & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & c_{n-2} & d_{n-2} \\ & & & a_{n-1} & b_{n-1} & d_{n-1} \\ e_1 & e_2 & \cdots & e_{n-2} & e_{n-1} & d_n \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix}}_{\underline{x}} = \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_n \end{pmatrix}}_{\underline{f}}.$$

- Factorization of the principal (leading) $(n-1) \times (n-1)$ tridiagonal submatrix will proceed as before.
- In addition, you'll need to update the last row ($\underline{e}^T$) and column ($\underline{d}$).
- When you get to the final 2×2 block you have interactions between the $\underline{b}$, $\underline{e}$, and $\underline{d}$ vectors that should be treated outside of the *for* loop.
- Proceed with standard Gaussian elimination for this phase and then with backward substitution for the remaining upper triangular system.

General Band Matrices

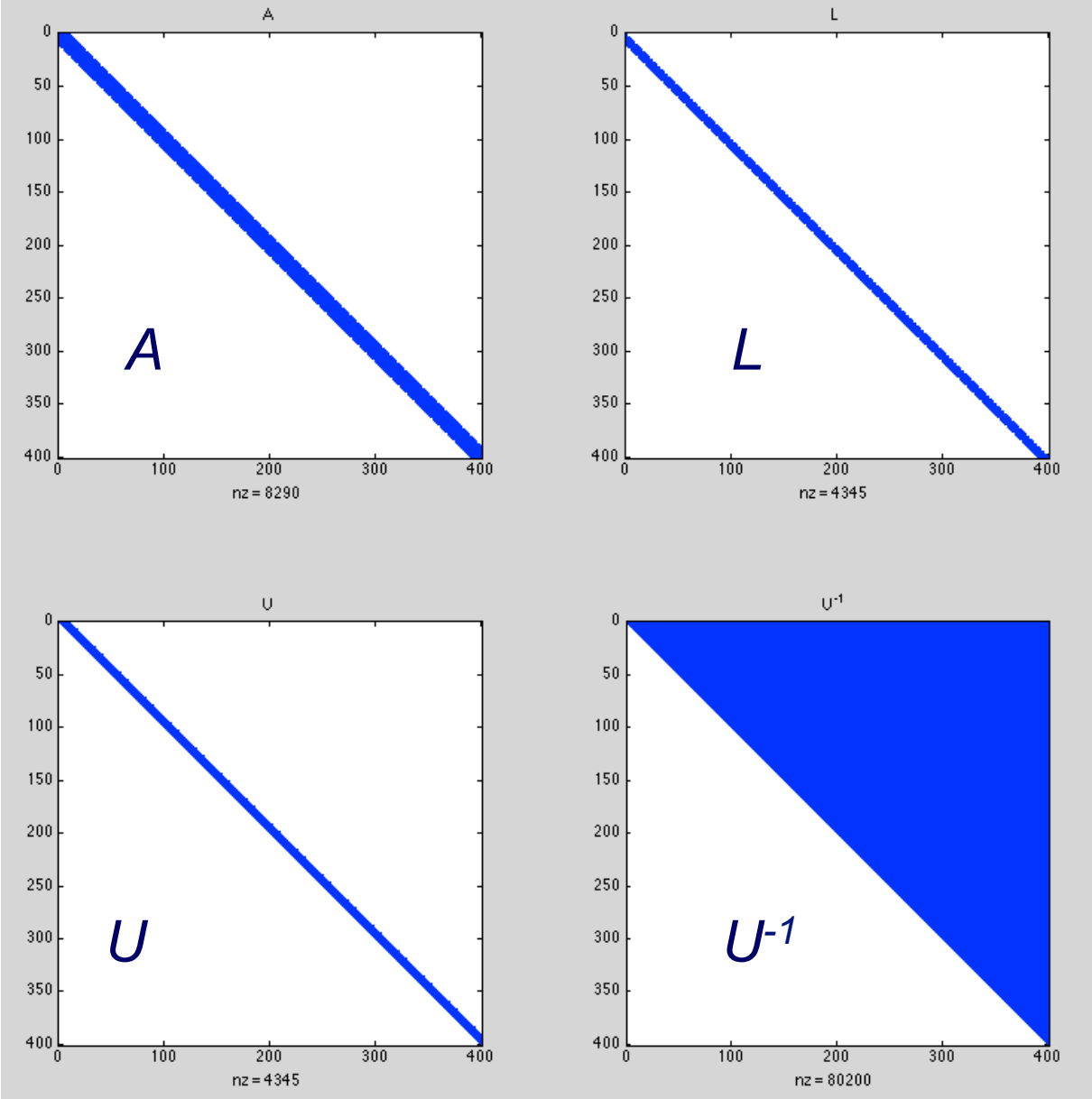
- In general, band system of bandwidth β requires $\mathcal{O}(\beta n)$ storage, and its factorization requires $\mathcal{O}(\beta^2 n)$ work
- Compared with full system, savings is substantial if $\beta \ll n$

Banded Systems

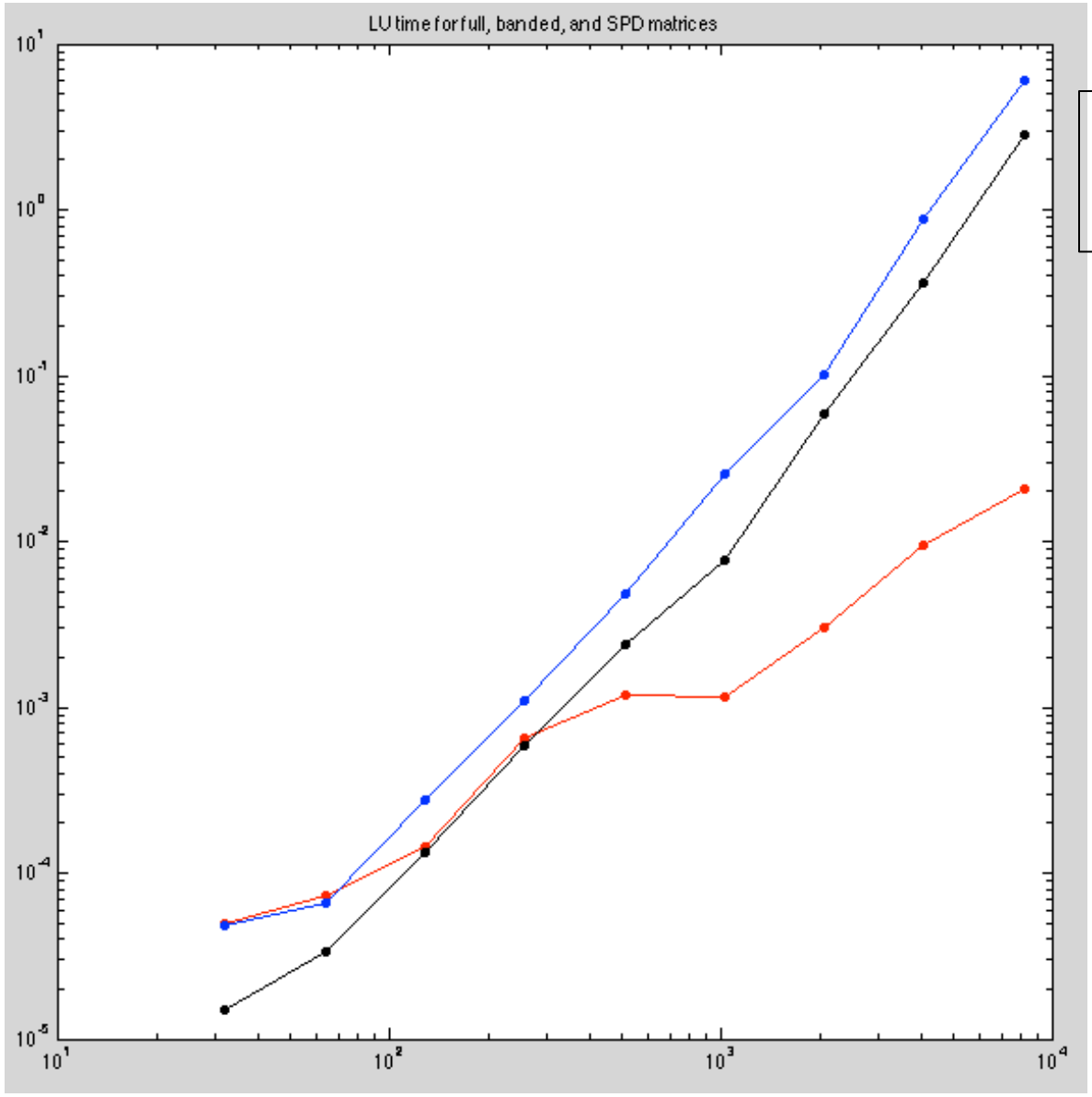


- Significant savings in storage and work if A is banded → $a_{ij} = 0$ if $|i-j| > \beta$
- The LU factors preserve the nonzero structure of A (unless there is pivoting, in which case, the bandwidth of L can grow by at most 2x).
- Storage / solve costs for LU is ~ $2n\beta$
- Factor cost is ~ $n\beta^2$ << n^3

Definitely Do Not Invert A or L or U for ***Banded Systems***



Solver Times, Banded, Cholesky (SPD), Full



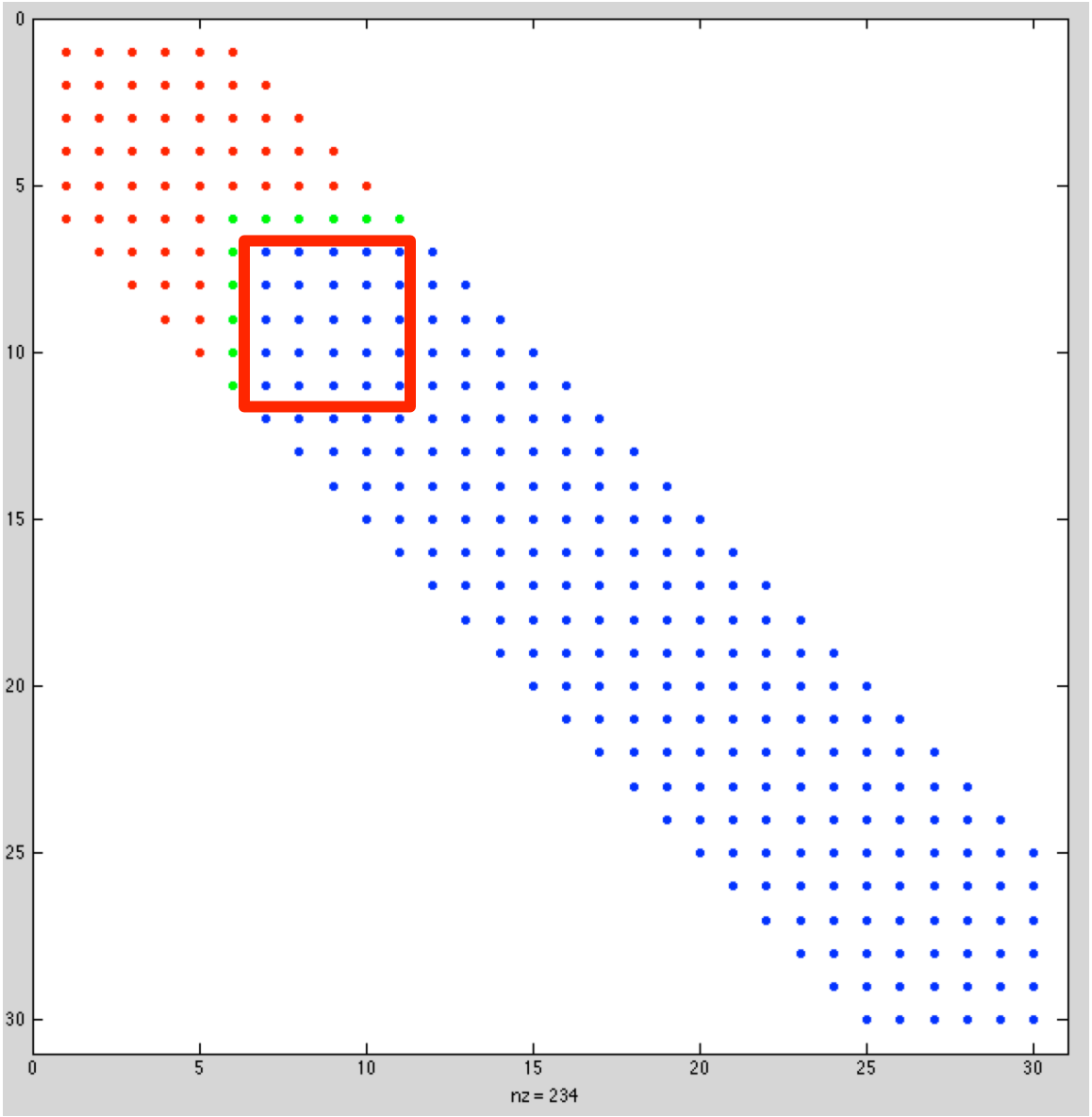
2 x savings

100 x savings

Solver Times, Banded, Cholesky (SPD), Full

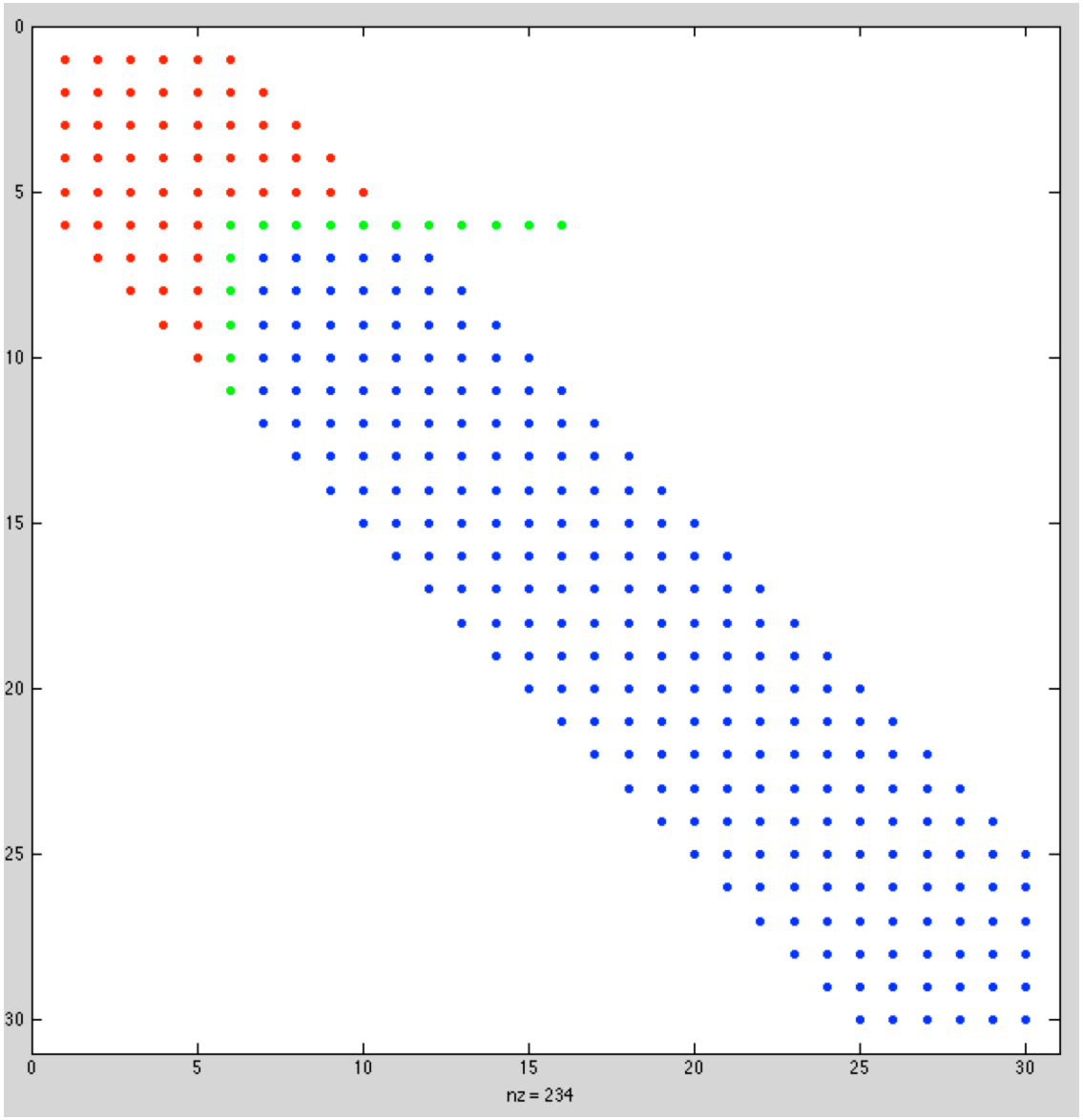
```
% Demo of banded-matrix costs

clear all;

for pass=1:2;
beta=10;

for k=4:13; n = 2^k;

   R=9*eye(n) + rand(n,n); S=R'*R; A=spalloc(n,n,1+2*beta);
   for i=1:n; j0=max(1,i-beta);j1=min(n,i+beta);
       A(i,j0:j1)=R(i,j0:j1);
   end;

   tstart=tic; [L,U]=lu(A); tsparse(k) = toc(tstart);
   tstart=tic; [L,U]=lu(R); tfull(k) = toc(tstart);
   tstart=tic; [C]=chol(S); tchol(k) = toc(tstart);
   nk(k)=n;
   sk(k)= (2*(n^3)/3)/(1.e9*tfull(k)); % GFLOPS
   ck(k)= (2*(n^3)/3)/(1.e9*tchol(k)); % GFLOPS

   [n tsparse(k) tfull(k) tchol(k)]

end;
loglog(nk,tsparse,'r.-',nk,tfull,'b.-',nk,tchol,'k.-')
axis square; title('LU time for full, banded, and SPD matrices')
```

Cost of Banded Factorization



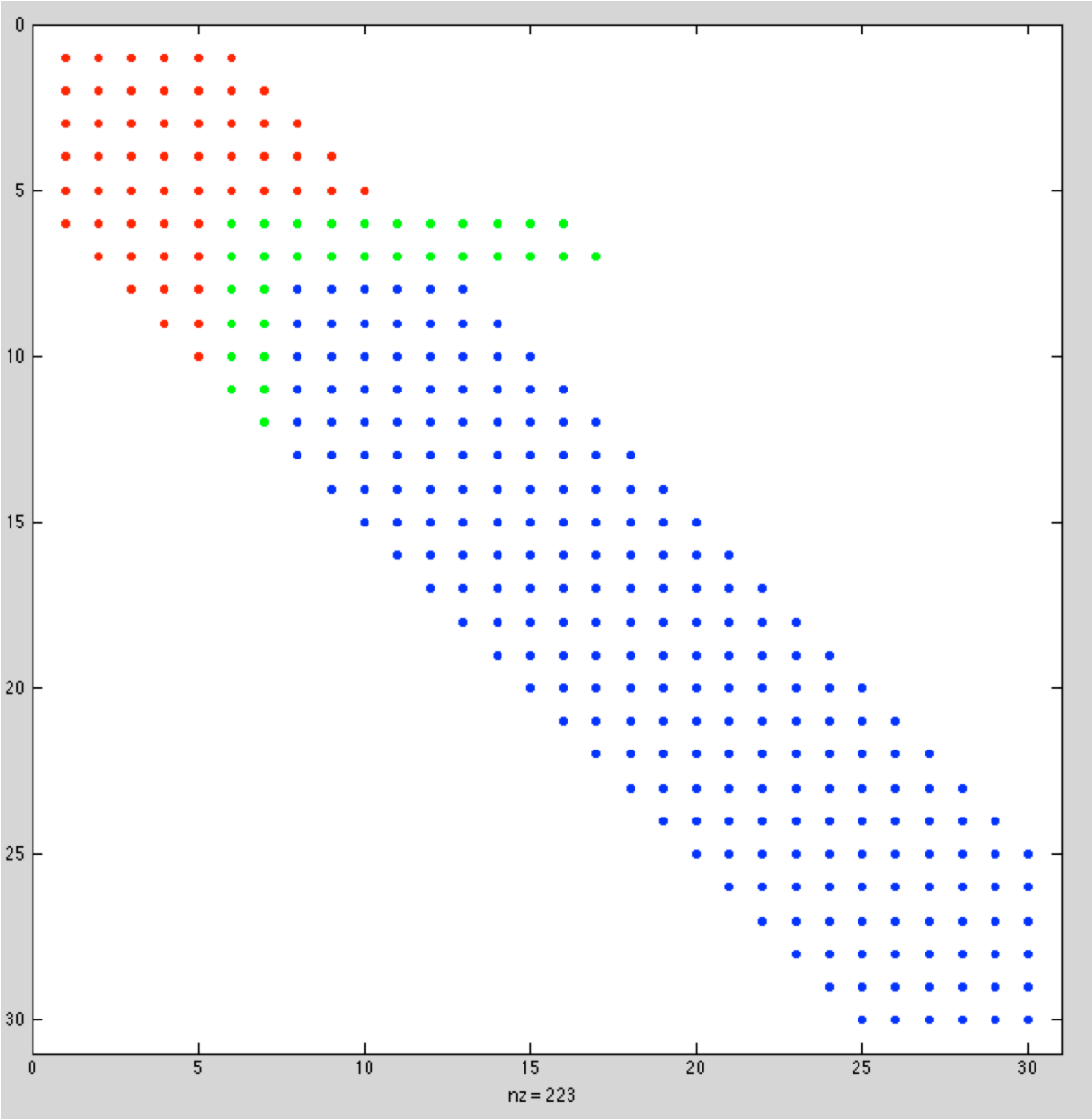
- Active submatrix for matrix with bandwidth b is (b x b).
- Work for outer product is $\mathbf{cr}^T$, which is outer product of two vectors of length b.
- So, total work is ~ n x (b^2) x 2 operations to convert A into LU.
- If we have pivoting, then bandwidth of U can grow by 2x.

Cost of Banded Factorization



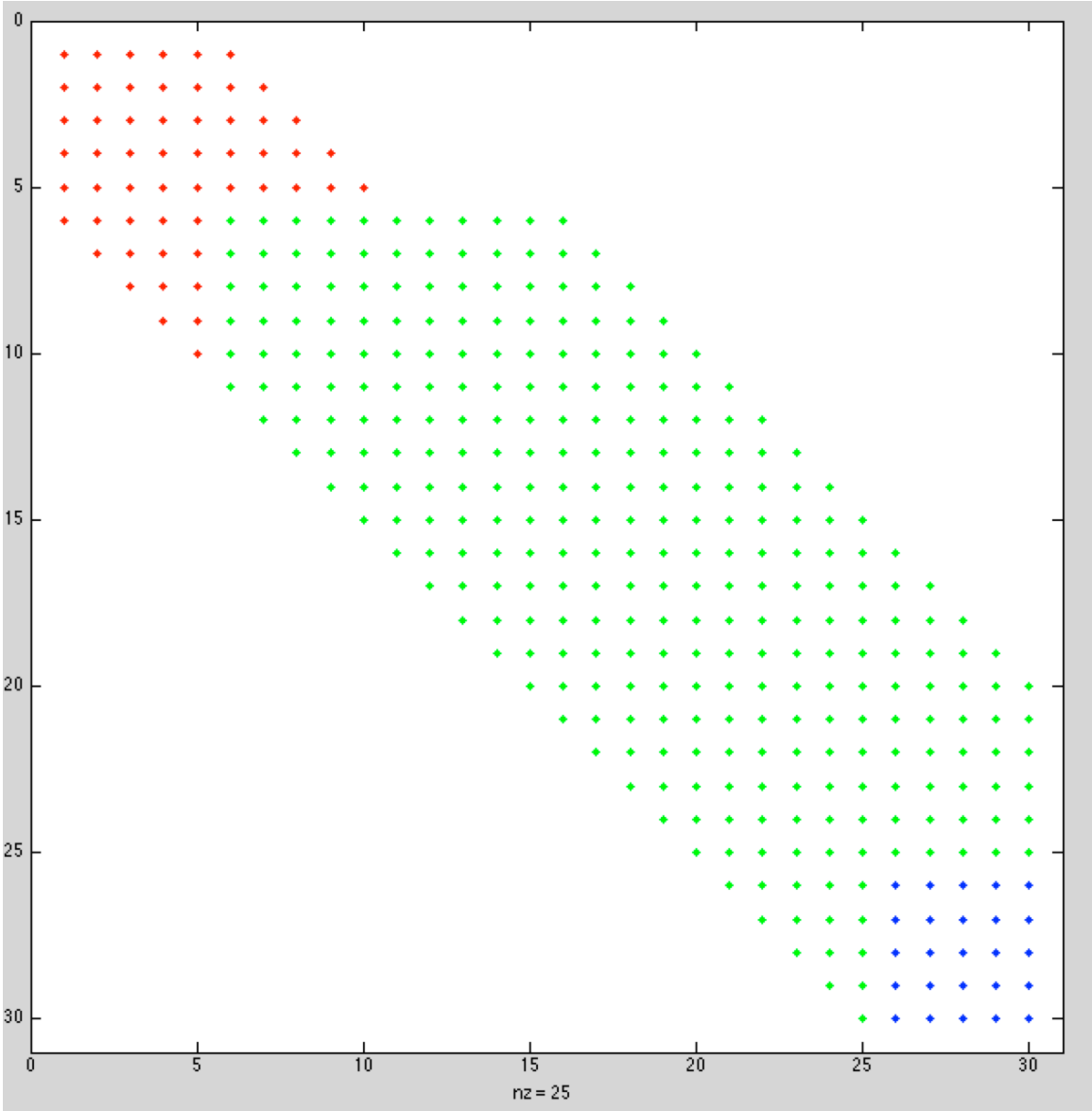
- Pivoting can pull a row that has 2b nonzeros to right of diagonal.
- U can end up with bandwidth 2b.

Cost of Banded Factorization



- Pivoting can pull a row that has 2b nonzeros to right of diagonal.
- U can end up with bandwidth 2b.

Cost of Banded Factorization



- Pivoting can pull a row that has 2b nonzeros to right of diagonal.
- U can end up with bandwidth 2b.

pivot_gui_band demo

0.3808	0.3687	0.9319	0.7159	0	0	0	0	0	0
0	0.6074	0.8979	0.8132	0.8964	0.8443	0	0	0	0
0.0341	0.4704	-0.1058	0.5477	0.2857	-0.3972	0	0	0	0
0.4967	0.2730	-0.0850	-0.5775	-0.2447	-0.2305	0	0	0	0
0	0	0.3564	0.1630	0.1818	0.5544	0.1102	0	0	0
0	0	0	0.0605	0.1366	0.7068	0.0704	0.0576	0	0
0	0	0	0	0.4603	0.5187	0.1690	0.4586	0.1100	0
0	0	0	0	0	0.9951	0.8019	0.8349	0.8467	0.1633
0	0	0	0	0	0	0.4288	0.7628	0.8159	0.2321
0	0	0	0	0	0	0	0.2054	0.3190	0.9207

Partial pivoting

LINPACK and LAPACK

- `LINPACK` is software package for solving wide variety of systems of linear equations, both general dense systems and special systems, such as symmetric or banded
- Solving linear systems of such fundamental importance in scientific computing that `LINPACK` has become standard benchmark for comparing performance of computers
- `LAPACK` is more recent replacement for `LINPACK` featuring higher performance on modern computer architectures, including some parallel computers

Basic Linear Algebra Subprograms

- High-level routines in `LINPACK` and `LAPACK` are based on lower-level Basic Linear Algebra Subprograms (`BLAS`)
- `BLAS` encapsulate basic operations on vectors and matrices so they can be optimized for given computer architecture while high-level routines that call them remain portable
- Higher-level `BLAS` encapsulate matrix-vector and matrix-matrix operations for better utilization of memory hierarchies such as cache and virtual memory with paging
- Generic Fortran versions of `BLAS` are available from `Netlib`, and many computer vendors provide custom versions optimized for their particular systems

Examples of BLAS

Level	Work	Examples	Function
1	$\mathcal{O}(n)$	`saxpy`	Scalar $\times$ vector $+$ vector
		`sdot`	Inner product
		`snrm2`	Euclidean vector norm
2	$\mathcal{O}(n^2)$	`sgemv`	Matrix-vector product
		`strsv`	Triangular solution
		`sger`	Rank-one update
3	$\mathcal{O}(n^3)$	`sgemm`	Matrix-matrix product
		`strsm`	Multiple triang. solutions
		`ssyrk`	Rank-k update

- Level-3 `BLAS` have more opportunity for data reuse, and hence higher performance, because they perform more operations per data item than lower-level `BLAS`

Vector Norms

- Magnitude, modulus, or absolute value for scalars generalizes to *norm* for vectors
- We will use only p-norms, defined by

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

  for integer $p > 0$ and n-vector $\boldsymbol{x}$
- Important special cases
  - 1-norm: $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|$
  - 2-norm: $\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$
  - ∞-norm: $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$

Next Topics

- Conditioning *(This material is in the first part of Chapter 2.)*
  - Measuring errors
  - Defining measures (norms)
  - Condition number *(Be aware of why condition number is important.)*

- Special matrices:
  - Rank-1 updates: Sherman Morrison
  - Tensor-product matrices (online notes)

Vector Norms

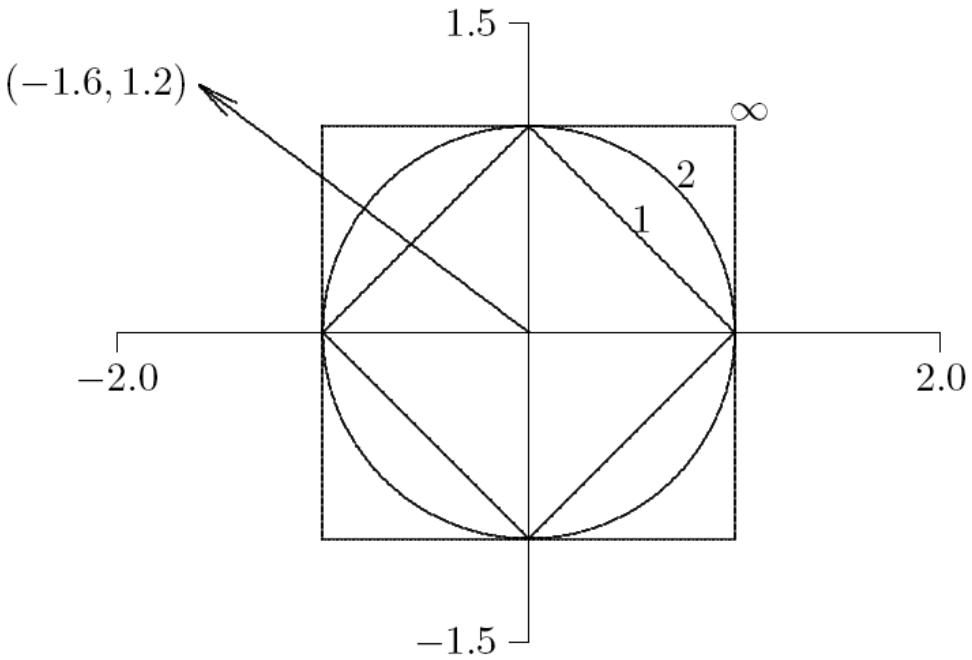
- Magnitude, modulus, or absolute value for scalars generalizes to *norm* for vectors
- We will use only p-norms, defined by

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

  for integer $p > 0$ and n-vector $\boldsymbol{x}$
- Important special cases
  - 1-norm: $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|$
  - 2-norm: $\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$
  - ∞-norm: $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$

Example: Vector Norms

- Drawing shows unit sphere in two dimensions for each norm



- Norms have following values for vector shown

$$\|\boldsymbol{x}\|_1 = 2.8 \quad \|\boldsymbol{x}\|_2 = 2.0 \quad \|\boldsymbol{x}\|_\infty = 1.6$$

Equivalence of Norms

- In general, for any vector $\boldsymbol{x}$ in $\mathbb{R}^n$, $\|\boldsymbol{x}\|_1 \geq \|\boldsymbol{x}\|_2 \geq \|\boldsymbol{x}\|_\infty$
- However, we also have

$$\|\boldsymbol{x}\|_1 \leq \sqrt{n}\,\|\boldsymbol{x}\|_2, \quad \|\boldsymbol{x}\|_2 \leq \sqrt{n}\,\|\boldsymbol{x}\|_\infty, \quad \|\boldsymbol{x}\|_1 \leq n\,\|\boldsymbol{x}\|_\infty$$

- Thus, for given n, norms differ by at most a constant, and hence are equivalent: if one is small, they must all be proportionally small.

❑ *Important Point:* ***Equivalence of Norms*** *(for n fixed):*

For all vector norms $\|\underline{x}\|_m$ and $\|\underline{x}\|_M$ $\exists$ constants c and C such that

$$c\,\|\underline{x}\|_m \leq \|\underline{x}\|_M \leq C\,\|\underline{x}\|_m$$

Allows us to work with the norm that is *most convenient.*

Properties of Vector Norms

- For any vector norm
  - $\|\boldsymbol{x}\| > 0$ if $\boldsymbol{x} \neq \boldsymbol{0}$
  - $\|\gamma \boldsymbol{x}\| = |\gamma| \cdot \|\boldsymbol{x}\|$ for any scalar γ
  - $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ (triangle inequality)
- In more general treatment, these properties taken as *definition* of vector norm
- Useful variation on triangle inequality
  - $| \, \|\boldsymbol{x}\| - \|\boldsymbol{y}\| \, | \leq \|\boldsymbol{x} - \boldsymbol{y}\|$

Matrix Norms

- *Matrix norm* corresponding to given vector norm is defined by

$$\|\boldsymbol{A}\| = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$$

- Norm of matrix measures maximum stretching matrix does to any vector in given vector norm

Matrix Norms

For any vector norm $||\underline{x}||_*$, define

$$||A||_* = \max_{\underline{x} \neq 0} \frac{||A\underline{x}||_*}{||\underline{x}||_*} = \max_{||\underline{x}||_* = 1} ||A\underline{x}||_*$$

- Often called the induced or subordinate matrix norm associated with the vector norm $||\underline{x}||_*$

Q: Can a matrix norm be less than 1 ?

Matrix Norms

- Matrix norm corresponding to vector 1-norm is maximum absolute *column* sum

$$\|\boldsymbol{A}\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$

- Matrix norm corresponding to vector ∞-norm is maximum absolute *row* sum

$$\|\boldsymbol{A}\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$

- Handy way to remember these is that matrix norms agree with corresponding vector norms for $n \times 1$ matrix

Matrix Norm Example

- Questions:
  - What is the 1-norm of the matrix below ?
  - What is the ∞-norm ?

$$B = \begin{bmatrix} 1 & 4 & -2 \\ 4 & 2 & -5 \\ 0 & -5 & 3 \end{bmatrix}$$

Matrix Norms: 2-norm

- The 2-norm of a symmetric matrix is $\max_i |\lambda_i|$
- Here, λ_i is the ith eigenvalue of A
- We say A is symmetric if $a_{ij} = a_{ji}$ for $i,j \in \{1,2,\ldots,n\}^2$
- That is, $A = A^T$ (A is equal to its transpose)

Properties of Matrix Norms

- Any matrix norm satisfies
  - $\|\boldsymbol{A}\| > 0$ if $\boldsymbol{A} \neq \boldsymbol{0}$
  - $\|\gamma \boldsymbol{A}\| = |\gamma| \cdot \|\boldsymbol{A}\|$ for any scalar γ
  - $\|\boldsymbol{A} + \boldsymbol{B}\| \leq \|\boldsymbol{A}\| + \|\boldsymbol{B}\|$
- Matrix norms we have defined also satisfy
  - $\|\boldsymbol{A}\boldsymbol{B}\| \leq \|\boldsymbol{A}\| \cdot \|\boldsymbol{B}\|$
  - $\|\boldsymbol{A}\boldsymbol{x}\| \leq \|\boldsymbol{A}\| \cdot \|\boldsymbol{x}\|$ for any vector $\boldsymbol{x}$

Matrix Norm Example

- Matrix norms are particularly useful in analyzing *iterative solvers.*
- Consider the system $A\mathbf{x} = \mathbf{b}$ to be solved with the following iterative scheme.
- Start with initial guess $\mathbf{x}_0 = 0$ and, for k=0, 1, . . . ,

$$\mathbf{x}_{k+1} \;=\; \mathbf{x}_k \;+\; M\,(\,\mathbf{b} - A\mathbf{x}_k\,)\,. \tag{1}$$

- Let $G := I - MA$. We can use the matrix norm of G to bound the error in the above iteration and determine its rate of convergence.
- Begin by defining the error to be $\mathbf{e}_k := \mathbf{x} - \mathbf{x}_k$.
- Note that $\mathbf{b} - A\mathbf{x}_k = A\mathbf{x} - A\mathbf{x}_k = A(\mathbf{x} - \mathbf{x}_k) = A\mathbf{e}_k$.
- Using the preceding result and subtracting (1) from the equation $\mathbf{x} = \mathbf{x}$ yields the error equation

$$\mathbf{e}_{k+1} \;=\; \mathbf{e}_k \;-\; M\,A\,\mathbf{e}_k \;=\; [\,I - MA]\;\mathbf{e}_k \;=\; G\,\mathbf{e}_k.$$

Matrix Norm Example

- Error equation

$$\mathbf{e}_{k+1} \;=\; \mathbf{e}_k \;-\; M\,A\,\mathbf{e}_k \;=\; [\,I - MA]\;\mathbf{e}_k \;=\; G\,\mathbf{e}_k.$$

- From the definition of the matrix norm, we have

$$||\mathbf{e}_k|| \;\leq\; ||G||\,||\mathbf{e}_{k-1}|| \;\leq\; ||G||^2\,||\mathbf{e}_{k-2}|| \;\ldots\; \leq\; ||G||^k\,||\mathbf{e}_0||$$

- With $\mathbf{x}_0 = 0$, we have $\mathbf{e}_0 = \mathbf{x}$ and thus the *relative error*

$$\frac{||\mathbf{e}_k||}{||\mathbf{x}||} \;\leq\; ||G||^k$$

- If $||G|| < 1$, the scheme (1) is convergent.
- By the equivalence of norms, if $||G|| < 1$ for *any* matrix norm, it is convergent.
- **Q:** Suppose $||G|| \leq 0.25$. What is the bound on the number of iterations required to converge to machine precision in IEEE 64-bit arithmetic? (Hint: Think carefully. What is the best base to use in considering this question?)

Matrix Norm Example

- Consider the following example:

$$\begin{aligned} A &= nI + 0.1\,R, \quad R = \text{rand}(n,n)\ r_{ij} \in [0,1] \\ M &= \text{diag}(1/a_{ii}) \end{aligned}$$

- In this case,

$$\begin{aligned} g_{ii} &= 0 \\ g_{ij} &= 0.1\,\frac{-r_{ij}}{n + 0.1 r_{ii}} \end{aligned}$$

- The ∞-norm for G is given by

$$||G||_\infty = \max_i \sum_{j=1}^{n} |g_{ij}| \le \max_i \sum_{i \neq j} M^* = (n-1)M^*,$$

  where

$$M^* := \max_{i \neq j} |g_{ij}| < \frac{0.1}{n}.$$

- In this case, we have a relative error bounded by $||G||_\infty^k \le (0.1)^k$.

- **Q:** Estimate the number of iterations required to reduce the error to machine epsilon when using IEEE 64-bit floating point arithmetic.

Matrix Norm Example

- Recall, the algorithm is computable:

$$\mathbf{x}_k \;=\; \mathbf{x}_{k-1} \;+\; M\,(\,\mathbf{b} - A\mathbf{x}_{k-1}\,)\,.$$

- The error, *not computable*, obeys:

$$||\mathbf{e}_k|| \;\leq\; ||G||^k||\mathbf{e}_0|| \;\leq\; ||G||^k||\mathbf{x}|| \qquad (\text{if } \mathbf{x}_0 = 0),$$

$$G \;:=\; I \;-\; M^{-1}A.$$

- Which is why we are interested in the norm of G.

- These types of iterative solvers (and better versions) are of particular interest when
  - A is sparse and has large bandwidth or
  - Formation of A is much more expensive than evaluation of matrix-vector products of the form $\mathbf{w} = A\mathbf{x}_{k-1}$.
- Of course, we don't need to save the $\mathbf{x}_k$s. We simply overwrite $\mathbf{x}$ until we converge.

Condition Number

- *Condition number* of square nonsingular matrix $\boldsymbol{A}$ is defined by

$$\operatorname{cond}(\boldsymbol{A}) = \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^{-1}\|$$

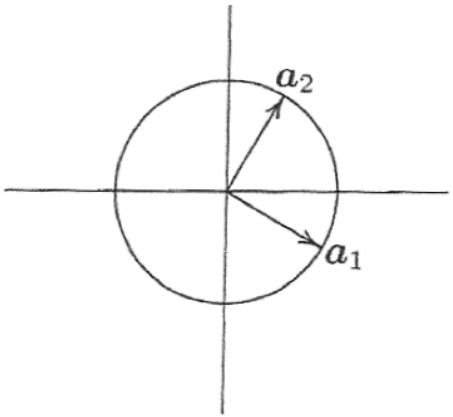
- By convention, $\operatorname{cond}(\boldsymbol{A}) = \infty$ if $\boldsymbol{A}$ is singular
- Since

$$\|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^{-1}\| = \left(\max_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \right) \cdot \left(\min_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \right)^{-1}$$

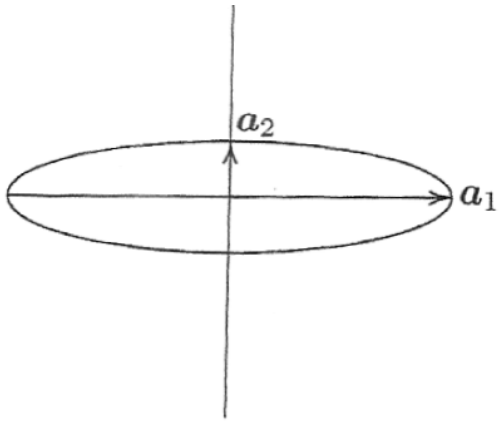
  condition number measures ratio of maximum stretching to maximum shrinking matrix does to any nonzero vectors

- Large $\operatorname{cond}(\boldsymbol{A})$ means $\boldsymbol{A}$ is *nearly singular*

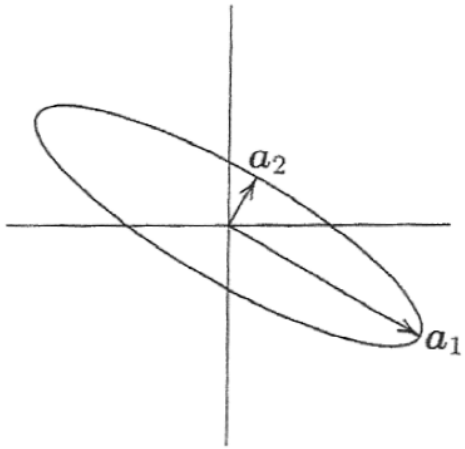
Condition Number Examples



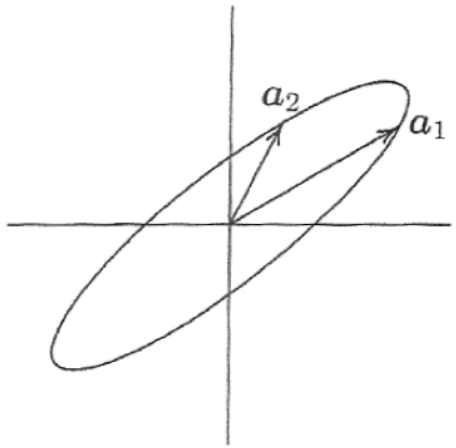
$$A_1 = \begin{bmatrix} 0.87 & 0.5 \\ -0.5 & 0.87 \end{bmatrix}, \quad \text{cond}_2(A_1) = 1$$



$$A_2 = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad \text{cond}_2(A_2) = 4$$



$$A_3 = \begin{bmatrix} 1.73 & 0.25 \\ -1 & 0.43 \end{bmatrix}, \quad \text{cond}_2(A_3) = 4$$



$$A_4 = \begin{bmatrix} 1.52 & 0.91 \\ 0.47 & 0.94 \end{bmatrix}, \quad \text{cond}_2(A_4) = 4$$

Properties of Condition Number

- For any matrix $\boldsymbol{A}$, $\mathrm{cond}(\boldsymbol{A}) \geq 1$
- For identity matrix, $\mathrm{cond}(\boldsymbol{I}) = 1$
- For any matrix $\boldsymbol{A}$ and scalar γ, $\mathrm{cond}(\gamma \boldsymbol{A}) = \mathrm{cond}(\boldsymbol{A})$
- For any diagonal matrix $\boldsymbol{D} = \mathrm{diag}(d_i)$, $\mathrm{cond}(\boldsymbol{D}) = \dfrac{\max |d_i|}{\min |d_i|}$

Computing Condition Number

- Definition of condition number involves matrix inverse, so it is nontrivial to compute
- Computing condition number from definition would require much more work than computing solution whose accuracy is to be assessed
- In practice, condition number is estimated inexpensively as byproduct of solution process
- Matrix norm $\|\boldsymbol{A}\|$ is easily computed as maximum absolute column sum (or row sum, depending on norm used)
- Estimating $\|\boldsymbol{A}^{-1}\|$ at low cost is more challenging

Computing Condition Number, continued

- From properties of norms, if $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{y}$, then

$$\frac{\|\boldsymbol{z}\|}{\|\boldsymbol{y}\|} \leq \|\boldsymbol{A}^{-1}\|$$

  and bound is achieved for optimally chosen $\boldsymbol{y}$

- Efficient condition estimators heuristically pick $\boldsymbol{y}$ with large ratio $\|\boldsymbol{z}\|/\|\boldsymbol{y}\|$, yielding good estimate for $\|\boldsymbol{A}^{-1}\|$

- Good software packages for linear systems provide efficient and reliable condition estimator

Error Bounds

- Condition number yields error bound for computed solution to linear system
- Let $\boldsymbol{x}$ be solution to $\boldsymbol{Ax} = \boldsymbol{b}$, and let $\hat{\boldsymbol{x}}$ be solution to $\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{b} + \Delta\boldsymbol{b}$
- If $\Delta\boldsymbol{x} = \hat{\boldsymbol{x}} - \boldsymbol{x}$, then

$$\boldsymbol{b} + \Delta\boldsymbol{b} = \boldsymbol{A}(\hat{\boldsymbol{x}}) = \boldsymbol{A}(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{Ax} + \boldsymbol{A}\Delta\boldsymbol{x}$$

which leads to bound

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \operatorname{cond}(\boldsymbol{A}) \frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}$$

for possible relative change in solution $\boldsymbol{x}$ due to relative change in right-hand side $\boldsymbol{b}$

Condition Number and Relative Error: $A\mathbf{x} = \mathbf{b}$.

- Want to solve $A\mathbf{x} = \mathbf{b}$, but computed rhs is:

$$\mathbf{b}' \;=\; \mathbf{b} \;+\; \Delta\mathbf{b},$$

  where we anticpate

$$\frac{||\Delta\mathbf{b}||}{||\mathbf{b}||} \;\approx\; \leq \epsilon_M.$$

- Net result is we end up solving $A\mathbf{x}' = \mathbf{b}'$ and want to know how large is the relative error, $\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$,

$$\frac{||\Delta\mathbf{x}||}{||\mathbf{x}||}?$$

- Since $A\mathbf{x}' = \mathbf{b}'$ and (by definition) $A\mathbf{x} = \mathbf{b}$, we have:

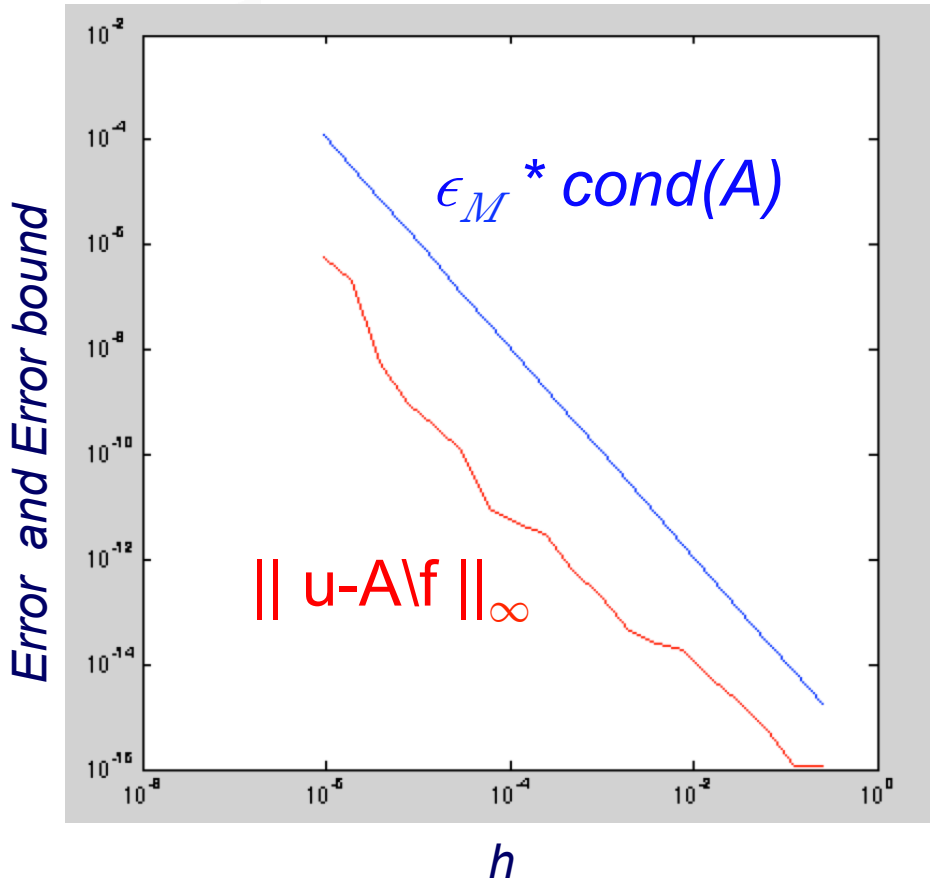
$$\begin{aligned}
||\Delta\mathbf{x}|| &\leq ||A^{-1}||\,||\Delta\mathbf{b}|| \\
||\mathbf{b}|| &\leq ||A||\,||\mathbf{x}|| \\
\frac{1}{||\mathbf{x}||} &\leq ||A||\,\frac{1}{||\mathbf{b}||} \\
\frac{\Delta\mathbf{x}}{||\mathbf{x}||} &\leq ||A||\,\frac{\Delta\mathbf{x}}{||\mathbf{b}||} \\
&\leq ||A||\,||A^{-1}||\,\frac{\Delta\mathbf{b}}{||\mathbf{b}||} \\
&= \mathrm{cond}(A)\,\frac{\Delta\mathbf{b}}{||\mathbf{b}||}.
\end{aligned}$$

- Key point: If cond(A)=10^k, then expected relative error is $\approx 10^k\epsilon_M$, meaning that you will lose k digits (of 16, if $\epsilon_M \approx 10^{-16}$.

Illustration of Impact of cond(A)

```
%% Check the error in solving Au=f vs eps*cond(A).

%% Test problem is finite difference solution to -u" = f
%% on [0,1] with u(0)=u(1)=0.

for k=2:20; n = (2^k)-1; h=1/(n+1);

  e = ones(n,1);
  A = spdiags([-e 2*e -e],-1:1, n,n)/(h*h);
  x=1:n; x=h*x';
  ue=1+sin(pi*(8*x.*x));

  f=A*ue;
  u=A\f;

  hk(k)=h; ck(k)=cond(A);
  ek(k)=max(abs(u-ue))/max(ue);
end;
loglog(hk,ek,'r-',hk,eps*ck,'b-');
axis square
```

*Here, we see that ϵ_M * cond(A) bounds the error in the solution to Au=f, as expected.*



Error Bounds, continued

- Similar result holds for relative change in matrix: if $(\boldsymbol{A} + \boldsymbol{E})\hat{\boldsymbol{x}} = \boldsymbol{b}$, then

$$\frac{\|\Delta \boldsymbol{x}\|}{\|\hat{\boldsymbol{x}}\|} \leq \text{cond}(\boldsymbol{A}) \frac{\|\boldsymbol{E}\|}{\|\boldsymbol{A}\|}$$

- If input data are accurate to machine precision, then bound for relative error in solution $\boldsymbol{x}$ becomes

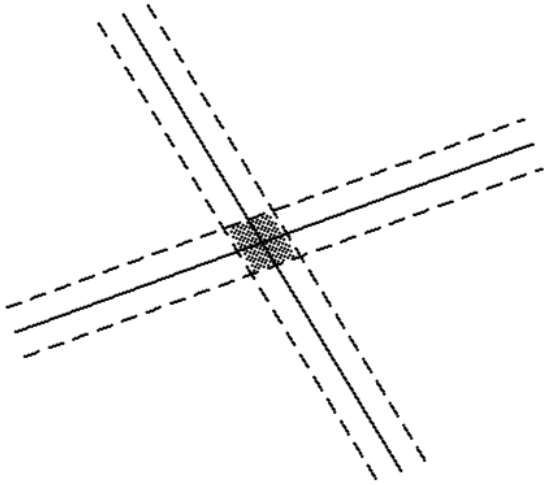
$$\frac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \text{cond}(\boldsymbol{A})\, \epsilon_{\text{mach}}$$

- Computed solution loses about $\log_{10}(\text{cond}(\boldsymbol{A}))$ decimal digits of accuracy relative to accuracy of input

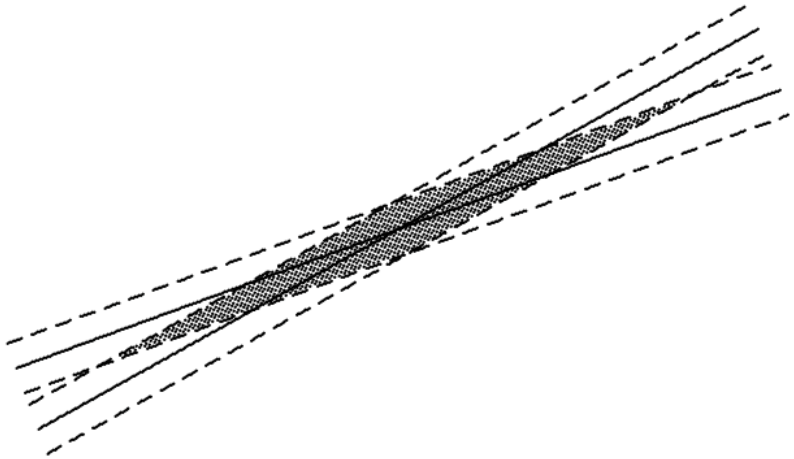
Example

Error Bounds – Illustration

- In two dimensions, uncertainty in intersection point of two lines depends on whether lines are nearly parallel

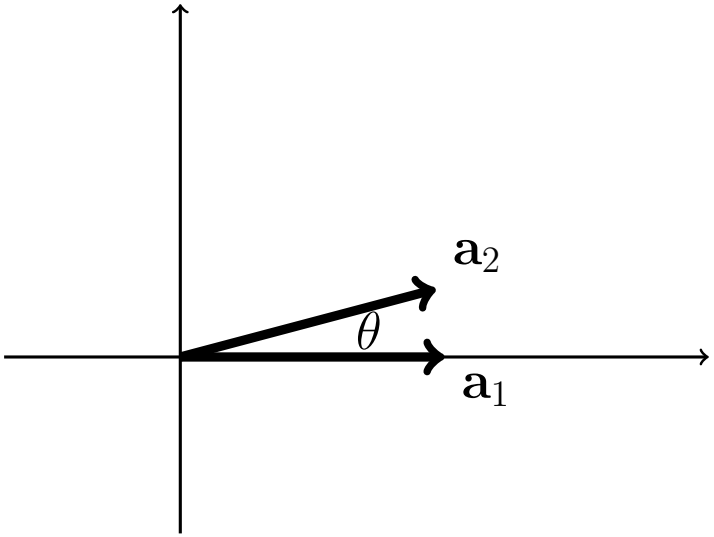


well-conditioned



ill-conditioned

A Nearly Singular Example



$$A = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 \end{bmatrix} = \begin{bmatrix} 1 & c \\ 0 & s \end{bmatrix}$$

$$c = \cos\theta, \qquad s = \sin\theta.$$

- Clearly, as $\theta \longrightarrow 0$ the matrix becomes singular.
- Can show that

$$\begin{aligned} \text{cond} &= \sqrt{\frac{1+|c|}{1-|c|}} \\ &\approx \frac{2}{\theta} \end{aligned}$$

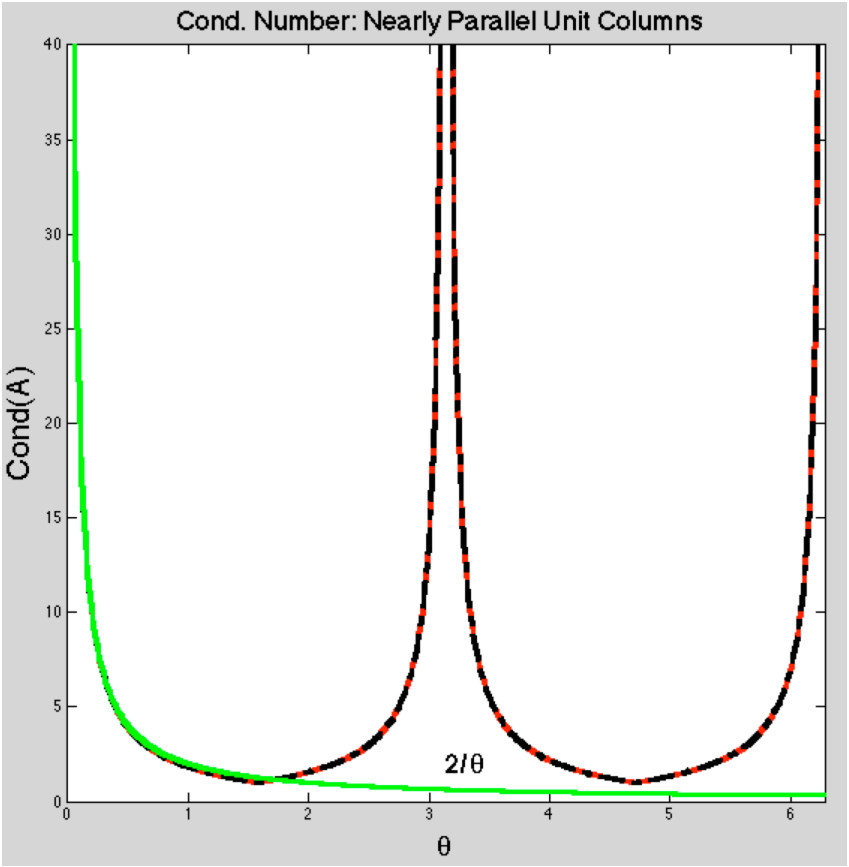
for small θ (by Taylor series!) *matlab demo.*

Matlab Demo cr2.m

This example plots cond(A) as a function of θ, as well as the estimates from the preceding slide.

- The computed value of cond(A) given by matlab exactly matches [(1+|cos θ |) / (1-|cos θ |)]$^{1/2}$
- The more interesting result is cond(A) ~ 2 / θ, which is very accurate for small angles.

```
%% Note - eigenvalues of A'*A are evals of C=A'*A =
%%
%%      1 c
%%      c 1
%%
%% (1-lam)*(1-lam) - c^2 , which is z^2 - c^2 with roots
%%
%%    z=c and z=-c
%%
%%    1-lam = c --> lam = 1 - c
%%
%%    1-lam = -c --> lam = 1+c
%%
%%    K2 = 1+c / 1 - c
%%
%%      ~ 2 / (1/2 theta^2) for small theta ~ 4 / theta^2
%%
%%    Therefore:      K(A) = sqrt(K2) ~ 2/theta
%%

format compact

jj=0; for j=.01:.01:(2*pi); cj=cos(j);sj=sin(j); jj=jj+1;
    R=[ cj -sj ; sj cj ];
    a1 = [ 1 ; 0 ]; a2 = R*a1; A = [ a1 a2 ];

    C(jj) = cond(A);
    t(jj)=j; aj = abs(cj); z(jj)=sqrt( (1+aj)/(1-aj) );
end;
plot(t,C,'r-',t,z,'k-.',t,2./abs(t),'g-','LineWidth',3);
axis([0 2*pi 0 40]);text(pi,2,'2/\theta','FontSize',18) axis square;
xlabel('\theta','FontSize',18);ylabel('Cond(A)','FontSize',20)
title('Cond. Number: Nearly Parallel Unit Columns','FontSize',18)
```



Error Bounds – Caveats

- Normwise analysis bounds relative error in *largest* components of solution; relative error in smaller components can be much larger
  - Componentwise error bounds can be obtained, but somewhat more complicated
- Conditioning of system is affected by relative scaling of rows or columns
  - Ill-conditioning can result from poor scaling as well as near singularity
  - Rescaling can help the former, but not the latter

Residual

- *Residual vector* of approximate solution $\hat{x}$ to linear system $Ax = b$ is defined by

$$r = b - A\hat{x}$$

- In theory, if A is nonsingular, then $\|\hat{x} - x\| = 0$ if, and only if, $\|r\| = 0$, but they are not necessarily small simultaneously

- Since

$$\frac{\|\Delta x\|}{\|\hat{x}\|} \leq \text{cond}(A) \frac{\|r\|}{\|A\| \cdot \|\hat{x}\|}$$

small relative residual implies small relative error in approximate solution *only if* A is well-conditioned

Residual, continued

- If computed solution $\hat{x}$ exactly satisfies

$$(A + E)\hat{x} = b$$

then

$$\frac{\|r\|}{\|A\| \, \|\hat{x}\|} \leq \frac{\|E\|}{\|A\|}$$

so large *relative residual* implies large backward error in matrix, and algorithm used to compute solution is unstable

- Stable algorithm yields small relative residual regardless of conditioning of nonsingular system
- Small residual is easy to obtain, but does not necessarily imply computed solution is accurate

Scaling Linear Systems

- In principle, solution to linear system is unaffected by diagonal scaling of matrix and right-hand-side vector
- In practice, scaling affects both conditioning of matrix and selection of pivots in Gaussian elimination, which in turn affect numerical accuracy in finite-precision arithmetic
- It is usually best if all entries (or uncertainties in entries) of matrix have about same size
- Sometimes it may be obvious how to accomplish this by choice of measurement units for variables, but there is no foolproof method for doing so in general
- Scaling can introduce rounding errors if not done carefully

Example: Scaling

- Linear system

$$\begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \epsilon \end{bmatrix}$$

  has condition number $1/\epsilon$, so is ill-conditioned if ϵ is small

- If second row is multiplied by $1/\epsilon$, then system becomes perfectly well-conditioned
- Apparent ill-conditioning was due purely to poor scaling
- In general, it is usually much less obvious how to correct poor scaling

- Sherman Morrison Formula

Solving Modified Problems

- If right-hand side of linear system changes but matrix does not, then LU factorization need not be repeated to solve new system
- Only forward- and back-substitution need be repeated for new right-hand side
- This is substantial savings in work, since additional triangular solutions cost only $\mathcal{O}(n^2)$ work, in contrast to $\mathcal{O}(n^3)$ cost of factorization

Sherman-Morrison Formula

- Sometimes refactorization can be avoided even when matrix *does* change
- *Sherman-Morrison formula* gives inverse of matrix resulting from rank-one change to matrix whose inverse is already known

$$(\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{u}(1 - \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1}$$

  where $\boldsymbol{u}$ and $\boldsymbol{v}$ are n-vectors

- Evaluation of formula requires $\mathcal{O}(n^2)$ work (for matrix-vector multiplications) rather than $\mathcal{O}(n^3)$ work required for inversion

Rank-One Updating of Solution

- To solve linear system $(\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)\boldsymbol{x} = \boldsymbol{b}$ with new matrix, use Sherman-Morrison formula to obtain

$$\begin{aligned} \boldsymbol{x} &= (\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)^{-1}\boldsymbol{b} \\ &= \boldsymbol{A}^{-1}\boldsymbol{b} + \boldsymbol{A}^{-1}\boldsymbol{u}(1 - \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{b} \end{aligned}$$

  which can be implemented by following steps
  - Solve $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{u}$ for $\boldsymbol{z}$, so $\boldsymbol{z} = \boldsymbol{A}^{-1}\boldsymbol{u}$
  - Solve $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$ for $\boldsymbol{y}$, so $\boldsymbol{y} = \boldsymbol{A}^{-1}\boldsymbol{b}$
  - Compute $\boldsymbol{x} = \boldsymbol{y} + ((\boldsymbol{v}^T\boldsymbol{y})/(1 - \boldsymbol{v}^T\boldsymbol{z}))\boldsymbol{z}$

- If $\boldsymbol{A}$ is already factored, procedure requires only triangular solutions and inner products, so only $\mathcal{O}(n^2)$ work and no explicit inverses

Example: Rank-One Updating of Solution

- Consider rank-one modification

$$\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -1 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}$$

  (with $3, 2$ entry changed) of system whose LU factorization was computed in earlier example

- One way to choose update vectors is

$$\boldsymbol{u} = \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{v} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

  so matrix of modified system is $\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T$

Original Matrix

$$\begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix}$$

Example, continued

- Using LU factorization of $\boldsymbol{A}$ to solve $\boldsymbol{Az} = \boldsymbol{u}$ and $\boldsymbol{Ay} = \boldsymbol{b}$,

$$\boldsymbol{z} = \begin{bmatrix} -3/2 \\ 1/2 \\ -1/2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{y} = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}$$

- Final step computes updated solution

Q: Under what circumstances could the denominator be zero ?

$$\boldsymbol{x} = \boldsymbol{y} + \frac{\boldsymbol{v}^T\boldsymbol{y}}{1 - \boldsymbol{v}^T\boldsymbol{z}}\,\boldsymbol{z} = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix} + \frac{2}{1 - 1/2}\begin{bmatrix} -3/2 \\ 1/2 \\ -1/2 \end{bmatrix} = \begin{bmatrix} -7 \\ 4 \\ 0 \end{bmatrix}$$

- We have thus computed solution to modified system without factoring modified matrix

Sherman Morrison

[1] Solve $A\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$:

$A \longrightarrow LU$ ($O(n^3)$ work)

Solve $L\tilde{\mathbf{y}} = \tilde{\mathbf{b}}$,

Solve $U\tilde{\mathbf{x}} = \tilde{\mathbf{y}}$ ($O(n^2)$ work).

[2] New problem:

$\left(A - \mathbf{u}\mathbf{v}^T\right)\mathbf{x} = \mathbf{b}$. (different $\mathbf{x}$ and $\mathbf{b}$)

Key Idea:

- $\left(A - \mathbf{u}\mathbf{v}^T\right)\mathbf{x}$ differs from $A\mathbf{x}$ by only a small amount of information.
- Rewrite as: $A\mathbf{x} + \mathbf{u}\gamma = \mathbf{b}$

  $\gamma := -\mathbf{v}^T\mathbf{x} \quad \longleftrightarrow \quad \mathbf{v}^T\mathbf{x} + \gamma = 0$

Sherman Morrison

Extended system:

$$\begin{aligned} A\mathbf{x} + \gamma\mathbf{u} &= \mathbf{b} \\ \mathbf{v}^T\mathbf{x} + \gamma &= 0 \end{aligned}$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\mathbf{x} + \gamma\mathbf{u} &= \mathbf{b} \\ \mathbf{v}^T\mathbf{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \mathbf{u} \\ \mathbf{v}^T & 1 \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix}$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\mathbf{x} + \gamma\mathbf{u} &= \mathbf{b} \\ \mathbf{v}^T\mathbf{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \mathbf{u} \\ \mathbf{v}^T & 1 \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix}$$

Eliminate for γ:

$$\begin{bmatrix} A & \mathbf{u} \\ 0 & 1 - \mathbf{v}^T A^{-1}\mathbf{u} \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{v}^T A^{-1}\mathbf{b} \end{pmatrix}$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\mathbf{x} + \gamma\mathbf{u} &= \mathbf{b} \\ \mathbf{v}^T\mathbf{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \mathbf{u} \\ \mathbf{v}^T & 1 \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix}$$

Eliminate for γ:

$$\begin{bmatrix} A & \mathbf{u} \\ 0 & 1 - \mathbf{v}^T A^{-1}\mathbf{u} \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{v}^T A^{-1}\mathbf{b} \end{pmatrix}$$

$$\gamma = -\left(1 - \mathbf{v}^T A^{-1}\mathbf{u}\right)^{-1} \mathbf{v}^T A^{-1}\mathbf{b}$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\mathbf{x} + \gamma\mathbf{u} &= \mathbf{b} \\ \mathbf{v}^T\mathbf{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \mathbf{u} \\ \mathbf{v}^T & 1 \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix}$$

Eliminate for γ:

$$\begin{bmatrix} A & \mathbf{u} \\ 0 & 1 - \mathbf{v}^T A^{-1}\mathbf{u} \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{v}^T A^{-1}\mathbf{b} \end{pmatrix}$$

$$\gamma = -\left(1 - \mathbf{v}^T A^{-1}\mathbf{u}\right)^{-1} \mathbf{v}^T A^{-1}\mathbf{b}$$

$$\mathbf{x} = A^{-1}\left(\mathbf{b} - \mathbf{u}\gamma\right) = A^{-1}\left[\mathbf{b} + \mathbf{u}\left(1 - \mathbf{v}^T A^{-1}\mathbf{u}\right)^{-1} \mathbf{v}^T A^{-1}\mathbf{b}\right]$$

Sherman Morrison

Extended system:

$$\begin{aligned} A\mathbf{x} + \gamma\mathbf{u} &= \mathbf{b} \\ \mathbf{v}^T\mathbf{x} + \gamma &= 0 \end{aligned}$$

In matrix form:

$$\begin{bmatrix} A & \mathbf{u} \\ \mathbf{v}^T & 1 \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix}$$

Eliminate for γ:

$$\begin{bmatrix} A & \mathbf{u} \\ 0 & 1 - \mathbf{v}^T A^{-1}\mathbf{u} \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{v}^T A^{-1}\mathbf{b} \end{pmatrix}$$

$$\gamma = -\left(1 - \mathbf{v}^T A^{-1}\mathbf{u}\right)^{-1} \mathbf{v}^T A^{-1}\mathbf{b}$$

$$\mathbf{x} = A^{-1}\left(\mathbf{b} - \mathbf{u}\gamma\right) = A^{-1}\left[\mathbf{b} + \mathbf{u}\left(1 - \mathbf{v}^T A^{-1}\mathbf{u}\right)^{-1} \mathbf{v}^T A^{-1}\mathbf{b}\right]$$

$$\left(A - \mathbf{u}\mathbf{v}^T\right)^{-1} = A^{-1} + A^{-1}\mathbf{u}\left(1 - \mathbf{v}^T A^{-1}\mathbf{u}\right)^{-1} \mathbf{v}^T A^{-1}.$$

Sherman Morrison: Potential Singularity

- Consider the modified system: $\left(A - \mathbf{u}\mathbf{v}^T\right)\mathbf{x} = \mathbf{b}$.
- The solution is

$$\begin{aligned}\mathbf{x} &= \left(A - \mathbf{u}\mathbf{v}^T\right)^{-1}\mathbf{b} \\ &= \left[I + A^{-1}\mathbf{u}\left(1 - \mathbf{v}^T A^{-1}\mathbf{u}\right)^{-1}\mathbf{v}^T A^{-1}\right]A^{-1}\mathbf{b}.\end{aligned}$$

- If $1 - \mathbf{v}^T A^{-1}\mathbf{u} = 0$, failure.
- Why?

Sherman Morrison: Potential Singularity

- Let $\tilde{A} := \left(A - \mathbf{u}\mathbf{v}^T\right)$ and consider,

$$\begin{aligned} \tilde{A}\, A^{-1} &= \left(A - \mathbf{u}\mathbf{v}^T\right) A^{-1} \\ &= \left(I - \mathbf{u}\mathbf{v}^T A^{-1}\right). \end{aligned}$$

- Look at the product $\tilde{A}A^{-1}\mathbf{u}$,

$$\begin{aligned} \tilde{A}\, A^{-1}\mathbf{u} &= \left(I - \mathbf{u}\mathbf{v}^T A^{-1}\right) \mathbf{u} \\ &= \mathbf{u} - \mathbf{u}\mathbf{v}^T A^{-1}\mathbf{u}. \end{aligned}$$

- If $\mathbf{v}^T A^{-1}\mathbf{u} = 1$, then

$$\tilde{A}\, A^{-1}\mathbf{u} = \mathbf{u} - \mathbf{u} = 0,$$

  which means that $\tilde{A}$ is singular since we assume that A^{-1} exists.

- Thus, an unfortunate choice of $\mathbf{u}$ and $\mathbf{v}$ can lead to a singular modified matrix and this singularity is indicated by $\mathbf{v}^T A^{-1}\mathbf{u} = 1$.

Computing $||A||_2$ and $\mathbf{cond}_2(A)$.

- Recall:

$$\begin{aligned}
\mathrm{cond}(A) &:= || A^{-1} || \cdot || A ||, \\
|| A || &:= \max_{\mathbf{x} \neq 0} \frac{|| A\mathbf{x} ||}{|| \mathbf{x} ||}, \\
|| \mathbf{x} ||_2 &= \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}} = \sqrt{\mathbf{x}^T \mathbf{x}}, \\
|| \mathbf{x} ||_2^2 &= \mathbf{x}^T \mathbf{x}.
\end{aligned}$$

- From now on, drop the subscript "$_2$".

$$\begin{aligned}
|| \mathbf{x} ||^2 &= \mathbf{x}^T \mathbf{x} \\
|| A\mathbf{x} ||^2 &= (A\mathbf{x})^T (A\mathbf{x}) = \mathbf{x}^T A^T A \, \mathbf{x}.
\end{aligned}$$

- Matrix norm:

$$\begin{aligned} || A ||^2 &= \max_{\mathbf{x}\neq 0} \frac{|| A\mathbf{x} ||^2}{|| \mathbf{x} ||^2}, \\ &= \max_{\mathbf{x}\neq 0} \frac{\mathbf{x}^T A^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}} \\ &= \lambda_{\max}(A^T A) \ =: \text{ spectral radius of } (A^T A). \end{aligned}$$

- The symmetric positive definite matrix $B := A^T A$ has positive eigenvalues.

- All symmetric matrices B have a complete set of orthonormal eigenvectors satisfying

$$B\mathbf{z}_j = \lambda_j \mathbf{z}_j, \quad \mathbf{z}_i^T \mathbf{z}_j = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} .$$

- **Note:** If $\lambda_i = \lambda_j$, $i \neq j$, then can have $\mathbf{z}_i^T \mathbf{z}_j \neq 0$, but we can orthogonalize $\mathbf{z}_i$ and $\mathbf{z}_j$ so that $\tilde{\mathbf{z}}_i^T \tilde{\mathbf{z}}_j = 0$ and

$$\begin{aligned} B\tilde{\mathbf{z}}_i &= \lambda_i \tilde{\mathbf{z}}_i \qquad \lambda_i = \lambda_j \\ B\tilde{\mathbf{z}}_j &= \lambda_j \tilde{\mathbf{z}}_j. \end{aligned}$$

- Assume eigenvalues are sorted with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.
- For any $\mathbf{x}$ we have: $\mathbf{x} = c_1\mathbf{z}_1 + c_2\mathbf{z}_2 + \cdots + c_n\mathbf{z}_n$.
- Let $||\,\mathbf{x}\,|| = 1$.

- Want to find $\displaystyle\max_{||\,\mathbf{x}\,||=1} \frac{\mathbf{x}^T B\mathbf{x}}{\mathbf{x}^T\mathbf{x}} = \max_{||\,\mathbf{x}\,||=1} \mathbf{x}^T B\mathbf{x}.$

- Note:
$$\mathbf{x}^T\mathbf{x} = \left(\sum_{i=1}^{n} c_i\mathbf{z}_i\right)^T \left(\sum_{j=1}^{n} c_j\mathbf{z}_j\right)$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i\, c_j \mathbf{z}_i^T\mathbf{z}_j$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i\, c_j \delta_{ij}$$
$$= \sum_{i=1}^{n} c_i^2 = 1.$$
$$\Longrightarrow\ c_1^2 = 1 - \sum_{i=2}^{n} c_i^2.$$

$$
\begin{aligned}
\mathbf{x}^T B\mathbf{x} &= \left(\sum_{i=1}^{n} c_i \mathbf{z}_i\right)^T \left(\sum_{j=1}^{n} c_j B\mathbf{z}_j\right) \\
&= \left(\sum_{i=1}^{n} c_i \mathbf{z}_i\right)^T \left(\sum_{j=1}^{n} c_j \lambda_j \mathbf{z}_j\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i\, \lambda_j c_j \mathbf{z}_i^T \mathbf{z}_j \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i\, \lambda_j c_j \delta_{ij} \\
&= \sum_{i=1}^{n} c_i^2 \lambda_i \;=\; c_1^2\lambda_1 + c_2^2\lambda_2 + \cdots + c_n^2\lambda_n \\
&= \lambda_1 \left[c_1^2 + c_2^2\beta_2 + \cdots + c_n^2\beta_n\right], \quad 0 < \beta_i := \frac{\lambda_i}{\lambda_1} \leq 1, \\
&= \lambda_1 \left[(1 - c_2^2 - \cdots - c_n^2) + c_2^2\beta_2 + \cdots + c_n^2\beta_n\right] \\
&= \lambda_1 \left[1 - (1-\beta_2)c_2^2 + (1-\beta_3)c_3^2 + \cdots + (1-\beta_n)c_n^2\right] \\
&= \lambda_1 \left[1 - \text{some positive (or zero) numbers}\right].
\end{aligned}
$$

- Expression is maximized when $c_2 = c_3 = \cdots = c_n = 0$, $\Longrightarrow c_1 = 1$.
- Maximum value $\mathbf{x}^T B\mathbf{x} = \lambda_{\max}(B) = \lambda_1$.
- Similarly, can show min $\mathbf{x}^T B\mathbf{x} = \lambda_{\min}(B) = \lambda_n$.

- So, $||A||^2 = \max_\lambda \lambda(A^TA) =$ spectral radius of A^TA.

- Now,
$$|| A^{-1} ||^2 = \max_{\mathbf{x}\neq 0} \frac{|| A^{-1}\mathbf{x} ||^2}{||\mathbf{x}||^2}.$$

- Let $\mathbf{x} = A\mathbf{y}$:
$$\begin{aligned} || A^{-1} ||^2 &= \max_{\mathbf{y}\neq 0} \frac{|| A^{-1}A\mathbf{y} ||^2}{||A\mathbf{y}||^2} = \max_{\mathbf{y}\neq 0} \frac{|| \mathbf{y} ||^2}{||A\mathbf{y}||^2} = \left(\min_{\mathbf{y}\neq 0} \frac{|| A\mathbf{y} ||^2}{||\mathbf{y}||^2} \right)^{-1} \\ &= \frac{1}{\lambda_{\min}(A^TA)}. \end{aligned}$$

- So, $\text{cond}_2(A) = || A^{-1} || \cdot || A ||$,
$$\text{cond}_2(A) = \sqrt{\frac{\lambda_{\max}(A^TA)}{\lambda_{\min}(A^TA)}}.$$

Special Types of Linear Systems

- Work and storage can often be saved in solving linear system if matrix has special properties
- Examples include
  - *Symmetric*: $\boldsymbol{A} = \boldsymbol{A}^T$, $a_{ij} = a_{ji}$ for all i, j
  - *Positive definite*: $\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq \boldsymbol{0}$
  - *Band*: $a_{ij} = 0$ for all $|i - j| > \beta$, where β is *bandwidth* of $\boldsymbol{A}$
  - *Sparse*: most entries of $\boldsymbol{A}$ are zero

Symmetric Positive Definite (SPD) Matrices

- Very common in optimization and physical processes
- Easiest example:
  - If B is invertible, then $A := B^T B$ is SPD.
- SPD systems of the form $A \underline{x} = \underline{b}$ can be solved using
  - (stable) Cholesky factorization $A = LL^T$, or
  - iteratively with the most robust iterative solver, conjugate gradient iteration (generally with preconditioning, known as preconditioned conjugate gradients, PCG).

Cholesky Factorization and SPD Matrices.

- A is SPD: $A = A^T$ and $\mathbf{x}^T A\mathbf{x} > 0$ for all $\mathbf{x} \neq 0$.
- Seek a symmetric factorization $A = \tilde{L}\tilde{L}^T$ (not LU).
  - L *not* lower triangular but not *unit* lower triangular.
  - That is, Lt_{ii} not necessarily 1.
- Alternatively, seek factorization $A = LDL^T$, where L *is* unit lower triangular and D is *diagonal*.

- Start with $LDL^T = A$.
- Clearly, $LU = A$ with $U = DL^T$.
  - Follows from uniqueness of LU factorization.
  - D is a *row scaling* of L^T and thus $D_{ii} = U_{ii}$.
  - A property of SPD matrices is that all pivots are positive.
  - (Another property is that you do not need to pivot.)

- Consider standard update step:

$$\begin{aligned} a_{ij} &= a_{ij} - \frac{a_{ik}\, a_{kj}}{a_{kk}} \\ &= a_{ij} - \frac{a_{ik}\, a_{jk}}{a_{kk}} \end{aligned}$$

- Usual multiplier column entries are $l_{ik} = a_{ik}/a_{kk}$.
- Usual pivot row entries are $u_{kj} = a_{kj} = a_{jk}$.
- So, if we factor $1/d_{kk} = 1/a_{kk}$ out of U, we have:

$$\begin{aligned} d_{kk}(a_{kj}/a_{kk}) &= d_{kk} l_{kj} \\ \longrightarrow U &= D(D^{-1}U) \\ &= DL^T. \end{aligned}$$

- For Cholesky, we have

$$A \;=\; LDL^T = L\sqrt{D}\sqrt{D}L^T = \tilde{L}\tilde{L}^T,$$

with $\tilde{L} = L\sqrt{D}$.

Symmetric Positive Definite Matrices

- If $\boldsymbol{A}$ is symmetric and positive definite, then LU factorization can be arranged so that $\boldsymbol{U} = \boldsymbol{L}^T$, which gives *Cholesky factorization*

$$\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{L}^T$$

  where $\boldsymbol{L}$ is lower triangular with positive diagonal entries
- Algorithm for computing it can be derived by equating corresponding entries of $\boldsymbol{A}$ and $\boldsymbol{L}\boldsymbol{L}^T$
- In 2×2 case, for example,

$$\begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{bmatrix}$$

  implies

$$l_{11} = \sqrt{a_{11}}, \quad l_{21} = a_{21}/l_{11}, \quad l_{22} = \sqrt{a_{22} - l_{21}^2}$$

Cholesky Factorization (Text)

Algorithm 2.7 Cholesky Factorization

```
for k = 1 to n                          { loop over columns }
    a_kk = sqrt(a_kk)
    for i = k + 1 to n
        a_ik = a_ik / a_kk              { scale current column }
    end
    for j = k + 1 to n                  { from each remaining column,
        for i = j to n                      subtract multiple
            a_ij = a_ij - a_ik * a_jk       of current column }
        end
    end
end
```

After a row scaling, this is just standard LU decomposition, exploiting symmetry in the LU factors and A. ($U=L^T$)

Cholesky Factorization

- One way to write resulting general algorithm, in which Cholesky factor L overwrites original matrix A, is

for $j = 1$ **to** n
  for $k = 1$ **to** $j - 1$
    for $i = j$ **to** n
      $a_{ij} = a_{ij} - a_{ik} \cdot a_{jk}$
    end
  end
  $a_{jj} = \sqrt{a_{jj}}$
  for $k = j + 1$ **to** n
    $a_{kj} = a_{kj}/a_{jj}$
  end
end

Cholesky Factorization, continued

- Features of Cholesky algorithm for symmetric positive definite matrices
  - All n square roots are of positive numbers, so algorithm is well defined
  - No pivoting is required to maintain numerical stability
  - Only lower triangle of $\boldsymbol{A}$ is accessed, and hence upper triangular portion need not be stored
  - Only $n^3/6$ multiplications and similar number of additions are required
- Thus, Cholesky factorization requires only about half work and ***half storage*** compared with LU factorization of general matrix by Gaussian elimination, and also avoids need for pivoting

Linear Algebra Very Short Summary

Main points:

- Conditioning of matrix cond(A) bounds our expected accuracy.
  - e.g., if cond(A) ~ 10^5 we expect at most 11 significant digits in $\underline{x}$.
  - Why?
  - We start with IEEE double precision – 16 digits. We lose 5 because condition (A) ~ 10^5, so we have 11 = 16-5.
- Stable algorithm (i.e., pivoting) important to realizing this bound.
  - Some systems don't need pivoting (e.g., SPD, diagonally dominant)
  - Unstable algorithms can sometimes be rescued with iterative refinement.
- Costs:
  - Full matrix → $O(n^2)$ storage, $O(n^3)$ work (wall-clock time)
  - Sparse or banded matrix, substantially less.

- The following slides present the book's derivation of the LU factorization process.

- I'll highlight a few of them that show the equivalence between the outer product approach and the elementary elimination matrix approach.

Example: Triangular Linear System

$$\begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$$

- Using back-substitution for this upper triangular system, last equation, $4x_3 = 8$, is solved directly to obtain $x_3 = 2$
- Next, x_3 is substituted into second equation to obtain $x_2 = 2$
- Finally, both x_3 and x_2 are substituted into first equation to obtain $x_1 = -1$

Elimination

- To transform general linear system into triangular form, we need to replace selected nonzero entries of matrix by zeros
- This can be accomplished by taking linear combinations of rows
- Consider 2-vector $\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$
- If $a_1 \neq 0$, then

$$\begin{bmatrix} 1 & 0 \\ -a_2/a_1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ 0 \end{bmatrix}$$

Elementary Elimination Matrices

- More generally, we can annihilate *all* entries below kth position in n-vector $\boldsymbol{a}$ by transformation

$$\boldsymbol{M}_k\boldsymbol{a} = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_n & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

  where $m_i = a_i/a_k$, $i = k+1, \ldots, n$

- Divisor a_k, called *pivot*, must be nonzero

Elementary Elimination Matrices, continued

- Matrix $\boldsymbol{M}_k$, called *elementary elimination matrix*, adds multiple of row k to each subsequent row, with *multipliers* m_i chosen so that result is zero
- $\boldsymbol{M}_k$ is unit lower triangular and nonsingular
- $\boldsymbol{M}_k = \boldsymbol{I} - \boldsymbol{m}_k \boldsymbol{e}_k^T$, where $\boldsymbol{m}_k = [0, \ldots, 0, m_{k+1}, \ldots, m_n]^T$ and $\boldsymbol{e}_k$ is kth column of identity matrix
- $\boldsymbol{M}_k^{-1} = \boldsymbol{I} + \boldsymbol{m}_k \boldsymbol{e}_k^T$, which means $\boldsymbol{M}_k^{-1} =: \boldsymbol{L}_k$ is same as $\boldsymbol{M}_k$ except signs of multipliers are reversed

Elementary Elimination Matrices, continued

- If $\boldsymbol{M}_j$, $j > k$, is another elementary elimination matrix, with vector of multipliers $\boldsymbol{m}_j$, then

$$\begin{aligned} \boldsymbol{M}_k \boldsymbol{M}_j &= \boldsymbol{I} - \boldsymbol{m}_k \boldsymbol{e}_k^T - \boldsymbol{m}_j \boldsymbol{e}_j^T + \boldsymbol{m}_k \boldsymbol{e}_k^T \boldsymbol{m}_j \boldsymbol{e}_j^T \\ &= \boldsymbol{I} - \boldsymbol{m}_k \boldsymbol{e}_k^T - \boldsymbol{m}_j \boldsymbol{e}_j^T \end{aligned}$$

  which means product is essentially "union," and similarly for product of inverses, $\boldsymbol{L}_k \boldsymbol{L}_j$

Comment on update step and $\underline{m}_k \underline{e}^T_k$

- Recall, $\underline{v} = C\,\underline{w} \in \text{span}\{C\}$.
- $\therefore\ V = (\underline{v}_1\,\underline{v}_2 \ldots \underline{v}_n) = C\,(\underline{w}_1\,\underline{w}_2 \ldots \underline{w}_n) \in \text{span}\{C\}$.

- If $C = \underline{c}$, i.e., C is a column vector and therefore of rank 1, then V is in span{C} and is of rank 1.

- All columns of V are multiples of $\underline{c}$.

- Thus, $W = \underline{c}\,\underline{r}^T$ is an n x n matrix of rank 1.
  - All columns are multiples of the first column and
  - All rows are multiples of the first row.

Elementary Elimination Matrices, continued

- Matrix $\boldsymbol{M}_k$, called *elementary elimination matrix*, adds multiple of row k to each subsequent row, with *multipliers* m_i chosen so that result is zero
- $\boldsymbol{M}_k$ is unit lower triangular and nonsingular
- $\boldsymbol{M}_k = \boldsymbol{I} - \boldsymbol{m}_k \boldsymbol{e}_k^T$, where $\boldsymbol{m}_k = [0, \ldots, 0, m_{k+1}, \ldots, m_n]^T$ and $\boldsymbol{e}_k$ is kth column of identity matrix
- $\boldsymbol{M}_k^{-1} = \boldsymbol{I} + \boldsymbol{m}_k \boldsymbol{e}_k^T$, which means $\boldsymbol{M}_k^{-1} =: \boldsymbol{L}_k$ is same as $\boldsymbol{M}_k$ except signs of multipliers are reversed

Example: Elementary Elimination Matrices

- For $\boldsymbol{a} = \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix}$,

$$\boldsymbol{M}_1\boldsymbol{a} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

and

$$\boldsymbol{M}_2\boldsymbol{a} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}$$

Example, continued

- Note that

$$\boldsymbol{L}_1 = \boldsymbol{M}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{L}_2 = \boldsymbol{M}_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/2 & 1 \end{bmatrix}$$

and

$$\boldsymbol{M}_1\boldsymbol{M}_2 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix}, \quad \boldsymbol{L}_1\boldsymbol{L}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1/2 & 1 \end{bmatrix}$$

Gaussian Elimination

- To reduce general linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ to upper triangular form, first choose $\boldsymbol{M}_1$, with a_{11} as pivot, to annihilate first column of $\boldsymbol{A}$ below first row
  - System becomes $\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}_1\boldsymbol{b}$, but solution is unchanged
- Next choose $\boldsymbol{M}_2$, using a_{22} as pivot, to annihilate second column of $\boldsymbol{M}_1\boldsymbol{A}$ below second row
  - System becomes $\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{b}$, but solution is still unchanged
- Process continues for each successive column until all subdiagonal entries have been zeroed

Gaussian Elimination

- To reduce general linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ to upper triangular form, first choose $\boldsymbol{M}_1$, with a_{11} as pivot, to annihilate first column of $\boldsymbol{A}$ below first row
  - System becomes $\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}_1\boldsymbol{b}$, but solution is unchanged
- Next choose $\boldsymbol{M}_2$, using a_{22} as pivot, to annihilate second column of $\boldsymbol{M}_1\boldsymbol{A}$ below second row
  - System becomes $\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{b}$, but solution is still unchanged
- *Technically, this should be a'$_{22}$, the 2-2 entry in A' := M_1A. Thus, we don't know all the pivots in advance.*

Gaussian Elimination, continued

- Resulting upper triangular linear system

$$\begin{aligned} \boldsymbol{M}_{n-1} \cdots \boldsymbol{M}_1 \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{M}_{n-1} \cdots \boldsymbol{M}_1 \boldsymbol{b} \\ \boldsymbol{M}\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{M}\boldsymbol{b} \end{aligned}$$

  can be solved by back-substitution to obtain solution to original linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

- Process just described is called *Gaussian elimination*

LU Factorization

- Product $\boldsymbol{L}_k\boldsymbol{L}_j$ is unit lower triangular if $k < j$, so

$$\boldsymbol{L} = \boldsymbol{M}^{-1} = \boldsymbol{M}_1^{-1} \cdots \boldsymbol{M}_{n-1}^{-1} = \boldsymbol{L}_1 \cdots \boldsymbol{L}_{n-1}$$

  is unit lower triangular

- By design, $\boldsymbol{U} = \boldsymbol{M}\boldsymbol{A}$ is upper triangular
- So we have

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

  with $\boldsymbol{L}$ unit lower triangular and $\boldsymbol{U}$ upper triangular

- Thus, Gaussian elimination produces *LU factorization* of matrix into triangular factors

LU Factorization, continued

- Having obtained LU factorization, $\boldsymbol{Ax} = \boldsymbol{b}$ becomes $\boldsymbol{LUx} = \boldsymbol{b}$, and can be solved by forward-substitution in lower triangular system $\boldsymbol{Ly} = \boldsymbol{b}$, followed by back-substitution in upper triangular system $\boldsymbol{Ux} = \boldsymbol{y}$
- Note that $\boldsymbol{y} = \boldsymbol{Mb}$ is same as transformed right-hand side in Gaussian elimination
- Gaussian elimination and LU factorization are two ways of expressing same solution process

Example: Gaussian Elimination

- Use Gaussian elimination to solve linear system

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \boldsymbol{b}$$

- To annihilate subdiagonal entries of first column of $\boldsymbol{A}$,

$$\boldsymbol{M}_1\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix},$$

$$\boldsymbol{M}_1\boldsymbol{b} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}$$

Example, continued

- To annihilate subdiagonal entry of second column of $\boldsymbol{M}_1\boldsymbol{A}$,

$$\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = \boldsymbol{U},$$

$$\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{b} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} = \boldsymbol{M}\boldsymbol{b}$$

Example, continued

- We have reduced original system to equivalent upper triangular system

$$\boldsymbol{U}\boldsymbol{x} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} = \boldsymbol{M}\boldsymbol{b}$$

which can now be solved by back-substitution to obtain

$$\boldsymbol{x} = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}$$

Example, continued

- To write out LU factorization explicitly,

$$\boldsymbol{L}_1\boldsymbol{L}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} = \boldsymbol{L}$$

so that

$$\boldsymbol{A} = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = \boldsymbol{L}\boldsymbol{U}$$