

# Scientific Computing: An Introductory Survey

## Chapter 2 – Systems of Linear Equations

Prof. Michael T. Heath

Department of Computer Science  
University of Illinois at Urbana-Champaign

Copyright © 2002. Reproduction permitted  
for noncommercial, educational use only.



### Systems of Linear Equations

- Given  $m \times n$  matrix  $A$  and  $m$ -vector  $b$ , find unknown  $n$ -vector  $x$  satisfying  $Ax = b$
- System of equations asks “Can  $b$  be expressed as linear combination of columns of  $A$ ?”
- If so, coefficients of linear combination are given by components of solution vector  $x$
- Solution may or may not exist, and may or may not be unique
- For now, we consider only *square* case,  $m = n$



### Existence and Uniqueness

- Existence and uniqueness of solution to  $Ax = b$  depend on whether  $A$  is singular or nonsingular
- Can also depend on  $b$ , but only in singular case
- If  $b \in \text{span}(A)$ , system is *consistent*

$A$	$b$	# solutions
nonsingular	arbitrary	one (unique)
singular	$b \in \text{span}(A)$	infinitely many
singular	$b \notin \text{span}(A)$	none



### Example: Nonsingularity

- $2 \times 2$  system

$$\begin{aligned} 2x_1 + 3x_2 &= b_1 \\ 5x_1 + 4x_2 &= b_2 \end{aligned}$$

or in matrix-vector notation

$$Ax = \begin{bmatrix} 2 & 3 \\ 5 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = b$$

is nonsingular regardless of value of  $b$

- For example, if  $b = [8 \ 13]^T$ , then  $x = [1 \ 2]^T$  is unique solution



## Outline

- Existence, Uniqueness, and Conditioning
- Solving Linear Systems
- Special Types of Linear Systems
- Software for Linear Systems



### Singularity and Nonsingularity

$n \times n$  matrix  $A$  is *nonsingular* if it has any of following equivalent properties

- Inverse of  $A$ , denoted by  $A^{-1}$ , exists
- $\det(A) \neq 0$
- $\text{rank}(A) = n$
- For any vector  $z \neq 0$ ,  $Az \neq 0$



### Geometric Interpretation

- In two dimensions, each equation determines straight line in plane
- Solution is intersection point of two lines
- If two straight lines are not parallel (nonsingular), then intersection point is unique
- If two straight lines are parallel (singular), then lines either do not intersect (no solution) or else coincide (any point along line is solution)
- In higher dimensions, each equation determines hyperplane; if matrix is nonsingular, intersection of hyperplanes is unique solution



### Example: Singularity

- $2 \times 2$  system

$$Ax = \begin{bmatrix} 2 & 3 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = b$$

is singular regardless of value of  $b$

- With  $b = [4 \ 7]^T$ , there is no solution
- With  $b = [4 \ 8]^T$ ,  $x = [\gamma \ (4 - 2\gamma)/3]^T$  is solution for any real number  $\gamma$ , so there are infinitely many solutions



## Vector Norms

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

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

for integer  $p > 0$  and  $n$ -vector  $\mathbf{x}$

- Important special cases
  - 1-norm:  $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$
  - 2-norm:  $\|\mathbf{x}\|_2 = \left( \sum_{i=1}^n |x_i|^2 \right)^{1/2}$
  - $\infty$ -norm:  $\|\mathbf{x}\|_\infty = \max_i |x_i|$



## Equivalence of Norms

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

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

- 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.



## Matrix Norms

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

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

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



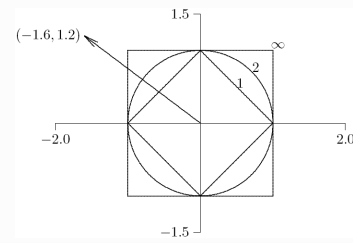
## Properties of Matrix Norms

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



## Example: Vector Norms

- Drawing shows unit sphere in two dimensions for each norm



- Norms have following values for vector shown
 
$$\|\mathbf{x}\|_1 = 2.8 \quad \|\mathbf{x}\|_2 = 2.0 \quad \|\mathbf{x}\|_\infty = 1.6$$

< interactive example >



## Properties of Vector Norms

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



## Matrix Norms

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

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

- Matrix norm corresponding to vector  $\infty$ -norm is maximum absolute *row* sum

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

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



## Condition Number

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

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$

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

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

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

- Large  $\text{cond}(\mathbf{A})$  means  $\mathbf{A}$  is **nearly singular**



## Properties of Condition Number

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

< interactive example >



## Computing Condition Number, continued

- From properties of norms, if  $Az = y$ , then

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

and bound is achieved for optimally chosen  $y$

- Efficient condition estimators heuristically pick  $y$  with large ratio  $\|z\|/\|y\|$ , yielding good estimate for  $\|A^{-1}\|$
- Good software packages for linear systems provide efficient and reliable condition estimator



## Error Bounds, continued

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

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

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

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

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



## 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



## 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  $\|A\|$  is easily computed as maximum absolute column sum (or row sum, depending on norm used)
- Estimating  $\|A^{-1}\|$  at low cost is more challenging



## Error Bounds

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

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

which leads to bound

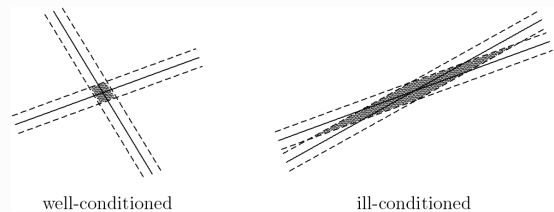
$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\Delta b\|}{\|b\|}$$

for possible relative change in solution  $x$  due to relative change in right-hand side  $b$  < interactive example >



## Error Bounds – Illustration

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



< interactive example >



## 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



## Example: Permutations

- Permutation matrix**  $P$  has one 1 in each row and column and zeros elsewhere, i.e., identity matrix with rows or columns permuted
- Note that  $P^{-1} = P^T$
- Premultiplying both sides of system by permutation matrix,  $PAx = Pb$ , reorders rows, but solution  $x$  is unchanged
- Postmultiplying  $A$  by permutation matrix,  $APx = b$ , reorders columns, which permutes components of original solution

$$x = (AP)^{-1}b = P^{-1}A^{-1}b = P^T(A^{-1}b)$$



## Triangular Linear Systems

- What type of linear system is easy to solve?
- If one equation in system involves only one component of solution (i.e., only one entry in that row of matrix is nonzero), then that component can be computed by division
- If another equation in system involves only one additional solution component, then by substituting one known component into it, we can solve for other component
- If this pattern continues, with only one new solution component per equation, then all components of solution can be computed in succession.
- System with this property is called **triangular**



## Forward-Substitution

- Forward-substitution** for lower triangular system  $Lx = b$

$$x_1 = b_1/\ell_{11}, \quad x_i = \left( b_i - \sum_{j=1}^{i-1} \ell_{ij}x_j \right) / \ell_{ii}, \quad i = 2, \dots, n$$

```

for j = 1 to n
  if  $\ell_{jj} = 0$  then stop
   $x_j = b_j/\ell_{jj}$ 
  for i = j + 1 to n
     $b_i = b_i - \ell_{ij}x_j$ 
  end
end
end
    
```



## Solving Linear Systems

- To solve linear system, transform it into one whose solution is same but easier to compute
- What type of transformation of linear system leaves solution unchanged?
- We can **premultiply** (from left) both sides of linear system  $Ax = b$  by any **nonsingular** matrix  $M$  without affecting solution
- Solution to  $MAx = Mb$  is given by

$$x = (MA)^{-1}Mb = A^{-1}M^{-1}Mb = A^{-1}b$$



## Example: Diagonal Scaling

- Row scaling: premultiplying both sides of system by nonsingular diagonal matrix  $D$ ,  $DAx = Db$ , multiplies each row of matrix and right-hand side by corresponding diagonal entry of  $D$ , but solution  $x$  is unchanged
- Column scaling: postmultiplying  $A$  by  $D$ ,  $ADx = b$ , multiplies each column of matrix by corresponding diagonal entry of  $D$ , which rescales original solution

$$x = (AD)^{-1}b = D^{-1}A^{-1}b$$



## Triangular Matrices

- Two specific triangular forms are of particular interest
  - lower triangular**: all entries *above* main diagonal are zero,  $a_{ij} = 0$  for  $i < j$
  - upper triangular**: all entries *below* main diagonal are zero,  $a_{ij} = 0$  for  $i > j$
- Successive substitution process described earlier is especially easy to formulate for lower or upper triangular systems
- Any triangular matrix can be permuted into upper or lower triangular form by suitable row permutation



## Back-Substitution

- Back-substitution** for upper triangular system  $Ux = b$

$$x_n = b_n/u_{nn}, \quad x_i = \left( b_i - \sum_{j=i+1}^n u_{ij}x_j \right) / u_{ii}, \quad i = n-1, \dots, 1$$

```

for j = n to 1
  if  $u_{jj} = 0$  then stop
   $x_j = b_j/u_{jj}$ 
  for i = 1 to j - 1
     $b_i = b_i - u_{ij}x_j$ 
  end
end
end
    
```



## 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  $x_1 = -1$



## Elementary Elimination Matrices

- More generally, we can annihilate *all* entries below  $k$ th position in  $n$ -vector  $a$  by transformation

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

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

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



## Elementary Elimination Matrices, continued

- If  $M_j, j > k$ , is another elementary elimination matrix, with vector of multipliers  $m_j$ , then

$$\begin{aligned} M_k M_j &= I - m_k e_k^T - m_j e_j^T + m_k e_k^T m_j e_j^T \\ &= I - m_k e_k^T - m_j e_j^T \end{aligned}$$

which means product is essentially “union,” and similarly for product of inverses,  $L_k L_j$



## Example, continued

- Note that

$$L_1 = M_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \quad L_2 = M_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/2 & 1 \end{bmatrix}$$

and

$$M_1 M_2 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix}, \quad L_1 L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1/2 & 1 \end{bmatrix}$$



## 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  $a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$
- 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, continued

- Matrix  $M_k$ , called *elementary elimination matrix*, adds multiple of row  $k$  to each subsequent row, with *multipliers*  $m_i$  chosen so that result is zero
- $M_k$  is unit lower triangular and nonsingular
- $M_k = I - m_k e_k^T$ , where  $m_k = [0, \dots, 0, m_{k+1}, \dots, m_n]^T$  and  $e_k$  is  $k$ th column of identity matrix
- $M_k^{-1} = I + m_k e_k^T$ , which means  $M_k^{-1} = L_k$  is same as  $M_k$  except signs of multipliers are reversed



## Example: Elementary Elimination Matrices

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

$$M_1 a = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

and

$$M_2 a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}$$



## Gaussian Elimination

- To reduce general linear system  $Ax = b$  to upper triangular form, first choose  $M_1$ , with  $a_{11}$  as pivot, to annihilate first column of  $A$  below first row
  - System becomes  $M_1 Ax = M_1 b$ , but solution is unchanged
- Next choose  $M_2$ , using  $a_{22}$  as pivot, to annihilate second column of  $M_1 A$  below second row
  - System becomes  $M_2 M_1 Ax = M_2 M_1 b$ , but solution is still unchanged
- Process continues for each successive column until all subdiagonal entries have been zeroed



## Gaussian Elimination, continued

- Resulting upper triangular linear system

$$\begin{aligned} M_{n-1} \cdots M_1 \mathbf{Ax} &= M_{n-1} \cdots M_1 \mathbf{b} \\ \mathbf{MAx} &= \mathbf{Mb} \end{aligned}$$

can be solved by back-substitution to obtain solution to original linear system  $\mathbf{Ax} = \mathbf{b}$

- Process just described is called *Gaussian elimination*



## LU Factorization, continued

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



## Example, continued

- To annihilate subdiagonal entry of second column of  $M_1\mathbf{A}$ ,

$$M_2 M_1 \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = \mathbf{U},$$

$$M_2 M_1 \mathbf{b} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} = \mathbf{Mb}$$



## Example, continued

- To write out LU factorization explicitly,

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

so that

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = \mathbf{LU}$$



## LU Factorization

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

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

is unit lower triangular

- By design,  $\mathbf{U} = \mathbf{MA}$  is upper triangular

- So we have

$$\mathbf{A} = \mathbf{LU}$$

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

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



## Example: Gaussian Elimination

- Use Gaussian elimination to solve linear system

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

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

$$M_1 \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix},$$

$$M_1 \mathbf{b} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}$$



## Example, continued

- We have reduced original system to equivalent upper triangular system

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

which can now be solved by back-substitution to obtain

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



## Row Interchanges

- Gaussian elimination breaks down if leading diagonal entry of remaining unreduced matrix is zero at any stage
- Easy fix: if diagonal entry in column  $k$  is zero, then interchange row  $k$  with some subsequent row having nonzero entry in column  $k$  and then proceed as usual
- If there is no nonzero on or below diagonal in column  $k$ , then there is nothing to do at this stage, so skip to next column
- Zero on diagonal causes resulting upper triangular matrix  $\mathbf{U}$  to be singular, but LU factorization can still be completed
- Subsequent back-substitution will fail, however, as it should for singular matrix



## Partial Pivoting

- In principle, any nonzero value will do as pivot, but in practice pivot should be chosen to minimize error propagation
- To avoid amplifying previous rounding errors when multiplying remaining portion of matrix by elementary elimination matrix, multipliers should not exceed 1 in magnitude
- This can be accomplished by choosing entry of largest magnitude on or below diagonal as pivot at each stage
- Such **partial pivoting** is essential in practice for numerically stable implementation of Gaussian elimination for general linear systems [< interactive example >](#)



## Complete Pivoting

- Complete pivoting** is more exhaustive strategy in which largest entry in entire remaining unreduced submatrix is permuted into diagonal pivot position
- Requires interchanging columns as well as rows, leading to factorization  $PAQ = LU$  with  $L$  unit lower triangular,  $U$  upper triangular, and  $P$  and  $Q$  permutations
- Numerical stability of complete pivoting is theoretically superior, but pivot search is more expensive than for partial pivoting
- Numerical stability of partial pivoting is more than adequate in practice, so it is almost always used in solving linear systems by Gaussian elimination



## Example: Small Pivots

- To illustrate effect of small pivots, consider 
$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}$$
 where  $\epsilon$  is positive number smaller than  $\epsilon_{\text{mach}}$
- If rows are not interchanged, then pivot is  $\epsilon$  and multiplier is  $-1/\epsilon$ , so 
$$M = \begin{bmatrix} 1 & 0 \\ -1/\epsilon & 1 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix},$$
 
$$U = \begin{bmatrix} \epsilon & 1 \\ 0 & 1 - 1/\epsilon \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}$$
 in floating-point arithmetic, but then 
$$LU = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} \neq A$$



## Pivoting, continued

- Although pivoting is generally required for stability of Gaussian elimination, pivoting is *not* required for some important classes of matrices
  - Diagonally dominant**
$$\sum_{i=1, i \neq j}^n |a_{ij}| < |a_{jj}|, \quad j = 1, \dots, n$$
  - Symmetric positive definite**
$$A = A^T \quad \text{and} \quad x^T A x > 0 \quad \text{for all} \quad x \neq 0$$



## LU Factorization with Partial Pivoting

- With partial pivoting, each  $M_k$  is preceded by permutation  $P_k$  to interchange rows to bring entry of largest magnitude into diagonal pivot position
- Still obtain  $MA = U$ , with  $U$  upper triangular, but now 
$$M = M_{n-1} P_{n-1} \cdots M_1 P_1$$
- $L = M^{-1}$  is still triangular in general sense, but not necessarily lower triangular
- Alternatively, we can write

$$PA = LU$$

where  $P = P_{n-1} \cdots P_1$  permutes rows of  $A$  into order determined by partial pivoting, and now  $L$  is lower triangular



## Example: Pivoting

- Need for pivoting has nothing to do with whether matrix is singular or nearly singular
- For example, 
$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
 is nonsingular yet has no LU factorization unless rows are interchanged, whereas 
$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$
 is singular yet has LU factorization



## Example, continued

- Using small pivot, and correspondingly large multiplier, has caused loss of information in transformed matrix
- If rows interchanged, then pivot is 1 and multiplier is  $-\epsilon$ , so 
$$M = \begin{bmatrix} 1 & 0 \\ -\epsilon & 1 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix},$$
 
$$U = \begin{bmatrix} 1 & 1 \\ 0 & 1 - \epsilon \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$
 in floating-point arithmetic
- Thus, 
$$LU = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix}$$
 which is correct after permutation



## Residual

- Residual  $r = b - A\hat{x}$  for solution  $\hat{x}$  computed using Gaussian elimination satisfies 
$$\frac{\|r\|}{\|A\| \|\hat{x}\|} \leq \frac{\|E\|}{\|A\|} \leq \rho n^2 \epsilon_{\text{mach}}$$
 where  $E$  is backward error in matrix  $A$  and **growth factor**  $\rho$  is ratio of largest entry of  $U$  to largest entry of  $A$
- Without pivoting,  $\rho$  can be arbitrarily large, so Gaussian elimination without pivoting is unstable
- With partial pivoting,  $\rho$  can still be as large as  $2^{n-1}$ , but such behavior is extremely rare



## Residual, continued

- There is little or no growth in practice, so

$$\frac{\|r\|}{\|A\| \|\hat{x}\|} \leq \frac{\|E\|}{\|A\|} \approx n \epsilon_{\text{mach}}$$

which means Gaussian elimination with partial pivoting yields small relative residual regardless of conditioning of system

- Thus, small relative residual does not necessarily imply computed solution is close to "true" solution unless system is well-conditioned
- Complete pivoting yields even smaller growth factor, but additional margin of stability usually is not worth extra cost

Michael T. Heath

Scientific Computing

57 / 88

## Example, continued

- Residual is as small as we can expect using 3-digit arithmetic, but exact solution is

$$x = [1.00 \quad 1.00]^T$$

so error is almost as large as solution

- Cause of this phenomenon is that matrix is nearly singular ( $\text{cond}(A) > 4000$ )
- Division that determines  $x_2$  is between two quantities that are both on order of rounding error, and hence result is essentially arbitrary
- When arbitrary value for  $x_2$  is substituted into first equation, value for  $x_1$  is computed so that first equation is satisfied, yielding small residual, but poor solution

Michael T. Heath

Scientific Computing

59 / 88

## 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  $PA = LU = \hat{L}\hat{U}$ , then  $\hat{L}^{-1}\hat{L} = \hat{U}U^{-1} = D$  is both lower and upper triangular, hence diagonal
- If both  $L$  and  $\hat{L}$  are unit lower triangular, then  $D$  must be identity matrix, so  $L = \hat{L}$  and  $U = \hat{U}$
- Uniqueness is made explicit in LDU factorization  $PA = LDU$ , with  $L$  unit lower triangular,  $U$  unit upper triangular, and  $D$  diagonal

Michael T. Heath

Scientific Computing

61 / 88

## Complexity of Solving Linear Systems

- LU factorization requires about  $n^3/3$  floating-point multiplications and similar number of additions
- Forward- and back-substitution for single right-hand-side vector together require about  $n^2$  multiplications and similar number of additions
- Can also solve linear system by matrix inversion:  
 $x = A^{-1}b$
- Computing  $A^{-1}$  is tantamount to solving  $n$  linear systems, requiring LU factorization of  $A$  followed by  $n$  forward- and back-substitutions, one for each column of identity matrix
- Operation count for inversion is about  $n^3$ , three times as expensive as LU factorization

Michael T. Heath

Scientific Computing

63 / 88

## Example: Small Residual

- Use 3-digit decimal arithmetic to solve

$$\begin{bmatrix} 0.641 & 0.242 \\ 0.321 & 0.121 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.883 \\ 0.442 \end{bmatrix}$$

- Gaussian elimination with partial pivoting yields triangular system

$$\begin{bmatrix} 0.641 & 0.242 \\ 0 & 0.000242 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.883 \\ -0.000383 \end{bmatrix}$$

- Back-substitution then gives solution

$$\hat{x} = [0.782 \quad 1.58]^T$$

- Exact residual for this solution is

$$r = b - A\hat{x} = \begin{bmatrix} -0.000622 \\ -0.000202 \end{bmatrix}$$

Michael T. Heath

Scientific Computing

58 / 88

## Implementation of Gaussian Elimination

- Gaussian elimination has general form of triple-nested loop

```

for _____
  for _____
    for _____
       $a_{ij} = a_{ij} - (a_{ik}/a_{kk})a_{kj}$ 
    end
  end
end

```

- Indices  $i$ ,  $j$ , and  $k$  of **for** loops can be taken in any order, for total of  $3! = 6$  different arrangements
- These variations have different memory access patterns, which may cause their performance to vary widely on different computers

Michael T. Heath

Scientific Computing

60 / 88

## Storage Management

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

Michael T. Heath

Scientific Computing

62 / 88

## Inversion vs. Factorization

- Even with many right-hand sides  $b$ , inversion never overcomes higher initial cost, since each matrix-vector multiplication  $A^{-1}b$  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  $A^{-1}B$  should be computed by LU factorization of  $A$ , followed by forward- and back-substitutions using each column of  $B$

Michael T. Heath

Scientific Computing

64 / 88

## Gauss-Jordan Elimination

- In Gauss-Jordan elimination, matrix is reduced to diagonal rather than triangular form
- Row combinations are used to annihilate entries above as well as below diagonal
- Elimination matrix used for given column vector  $a$  is of form

$$\begin{bmatrix} 1 & \cdots & 0 & -m_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & -m_{k-1} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & -m_n & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_{k-1} \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where  $m_i = a_i/a_k, i = 1, \dots, n$



## 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



## Rank-One Updating of Solution

- To solve linear system  $(A - uv^T)x = b$  with new matrix, use Sherman-Morrison formula to obtain

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

which can be implemented by following steps

- Solve  $Az = u$  for  $z$ , so  $z = A^{-1}u$
- Solve  $Ay = b$  for  $y$ , so  $y = A^{-1}b$
- Compute  $x = y + ((v^T y)/(1 - v^T z))z$
- If  $A$  is already factored, procedure requires only triangular solutions and inner products, so only  $\mathcal{O}(n^2)$  work and no explicit inverses



## Example, continued

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

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

- Final step computes updated solution

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

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



## Gauss-Jordan Elimination, continued

- Gauss-Jordan elimination requires about  $n^3/2$  multiplications and similar number of additions, 50% more expensive than LU factorization
- During elimination phase, same row operations are also applied to right-hand-side vector (or vectors) of system of linear equations
- Once matrix is in diagonal form, components of solution are computed by dividing each entry of transformed right-hand side by corresponding diagonal entry of matrix
- Latter requires only  $n$  divisions, but this is not enough cheaper to offset more costly elimination phase

< interactive example >



## 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

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

where  $u$  and  $v$  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



## 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

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

so matrix of modified system is  $A - uv^T$



## 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  $\epsilon$  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



## Iterative Refinement, continued

- Iterative refinement requires double storage, since both original matrix and its LU factorization are required
- Due to cancellation, residual usually must be computed with higher precision for iterative refinement to produce meaningful improvement
- For these reasons, iterative improvement is often impractical to use routinely, but it can still be useful in some circumstances
- For example, iterative refinement can sometimes stabilize otherwise unstable algorithm



## Symmetric Positive Definite Matrices

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

$$A = LL^T$$

where  $L$  is lower triangular with positive diagonal entries

- Algorithm for computing it can be derived by equating corresponding entries of  $A$  and  $LL^T$
- In  $2 \times 2$  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, 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  $A$  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

< interactive example >



## Iterative Refinement

- Given approximate solution  $x_0$  to linear system  $Ax = b$ , compute residual

$$r_0 = b - Ax_0$$

- Now solve linear system  $Az_0 = r_0$  and take

$$x_1 = x_0 + z_0$$

as new and "better" approximate solution, since

$$\begin{aligned} Ax_1 &= A(x_0 + z_0) = Ax_0 + Az_0 \\ &= (b - r_0) + r_0 = b \end{aligned}$$

- Process can be repeated to refine solution successively until convergence, potentially producing solution accurate to full machine precision



## Special Types of Linear Systems

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



## 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
      aij = aij - aik · ajk
    end
  end
  ajj = √ajj
  for k = j + 1 to n
    akj = akj / ajj
  end
end
    
```



## Symmetric Indefinite Systems

- For symmetric indefinite  $A$ , Cholesky factorization is not applicable, and some form of pivoting is generally required for numerical stability
- Factorization of form

$$PAP^T = LDL^T$$

with  $L$  unit lower triangular and  $D$  either tridiagonal or block diagonal with  $1 \times 1$  and  $2 \times 2$  diagonal blocks, can be computed stably using symmetric pivoting strategy

- In either case, cost is comparable to that of Cholesky factorization



## 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, continued

- LU factorization of  $A$  is then given by

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



## Iterative Methods for Linear Systems

- Gaussian elimination is direct method for solving linear system, producing exact solution in finite number of steps (in exact arithmetic)
- Iterative methods begin with initial guess for solution and successively improve it until desired accuracy attained
- In theory, it might take infinite number of iterations to converge to exact solution, but in practice iterations are terminated when residual is as small as desired
- For some types of problems, iterative methods have significant advantages over direct methods
- We will study specific iterative methods later when we consider solution of partial differential equations



## 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



## Tridiagonal Matrices

- Consider tridiagonal matrix

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

- Gaussian elimination without pivoting reduces to

```

d1 = b1
for i = 2 to n
    mi = ai/di-1
    di = bi - mi*ci-1
end
    
```



## General Band Matrices

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



## 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
- Both LINPACK and LAPACK are available from Netlib



## 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

