Outline

- Gaussian Elimination (without pivoting)
- Gaussian Elimination (with partial pivoting)

Gaussian Elimination and LU Factorization

- Gaussian elimination can be viewed as "triangular triangularization" of nonsingular $\mathbf{A} \in \mathbb{C}^{m\times m}$

$$\underbrace{\mathbf{L}_{m-1}\cdots\mathbf{L}_2\mathbf{L}_1}_{\mathbf{L}^{-1}}\mathbf{A} = \mathbf{U}$$

analogous to Householder QR factorization of matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$

$$\underbrace{\mathbf{Q}_n\cdots\mathbf{Q}_2\mathbf{Q}_1}_{\mathbf{Q}^*}\mathbf{A} = \mathbf{R}$$

- Example of LU factorization of 4×4 matrix $\mathbf{A}$

$$\xrightarrow{\mathbf{L}_1}\underbrace{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix}}_{\mathbf{L}_1\mathbf{A}} \xrightarrow{\mathbf{L}_2\mathbf{A}} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & 0 & \times & \times \\ & 0 & \times & \times \end{bmatrix}}_{\mathbf{L}_2\mathbf{L}_1\mathbf{A}} \xrightarrow{\mathbf{L}_3\mathbf{A}} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & & \times & \times \\ & & 0 & \times \end{bmatrix}}_{\mathbf{L}_3\mathbf{L}_2\mathbf{L}_1\mathbf{A}}$$

What is Matrices $\mathbf{L}_k$

- At step k, eliminate entries below a_{kk} : let x_k be kth column of $\mathbf{L}_{k-1}\cdots\mathbf{L}_2\mathbf{L}_1\mathbf{A}$,

$$x_k = [x_{1,k}, x_{2,k}, \cdots, x_{k,k}, x_{k+1,k} \cdots x_{m,k}]^T$$

$$\mathbf{L}_k x_k = [x_{1,k}, x_{2,k}, \cdots, x_{k,k}, 0 \cdots 0]^T$$

- The multipliers $l_{jk} = x_{jk}/x_{kk}$ appear in $\mathbf{L}_k$

$$\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & 0 & 1 & & & \\ & 0 & -l_{k+1,k} & 1 & & \\ & 0 & \vdots & & \ddots & \\ & & -l_{m,k} & & & 1 \end{bmatrix}$$

- Let $l_k = [0, \cdots, 0, l_{k+1,k}, \cdots, l_{m,k}]^T$ and $e_k = [\underbrace{0, \cdots, 0}_{k-1}, 1, \cdots, 0]^T$, then $\mathbf{L}_k = \mathbf{I} - l_k e_k^*$

Forming L

- Luckily, the **L** matrix contains the multipliers $l_{jk} = x_{jk}/x_{kk}$

$$\mathbf{L} = \mathbf{L}_1^{-1}\mathbf{L}_2^{-1}\cdots\mathbf{L}_{m-1}^{-1} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{m1} & l_{m2} & \cdots & l_{m,m-1} & 1 \end{bmatrix}$$

  and is said to be a unit lower triangular matrix
- First, $\mathbf{L}_k^{-1} = \mathbf{I} + l_k\mathbf{e}_k^*$, because $\mathbf{e}_k^* k_k = 0$ and $(\mathbf{I} - l_k\mathbf{e}_k^*)(\mathbf{I} + l_k\mathbf{e}_k^*) = \mathbf{I} - l_k\mathbf{e}_k^* l_k\mathbf{e}_k^* = \mathbf{I}$
- Second, $L_1^{-1}L_2^{-1}\cdots L_{k+1}^{-1} = \mathbf{I} + \sum_{j=1}^{k+1} l_j\mathbf{e}_j^*$, since (prove by induction) $(\mathbf{I} + \sum_{j=1}^{k+1} l_j\mathbf{e}_j^*)(\mathbf{I} + l_{k+}\mathbf{e}_{k+1}^*) = \mathbf{I} + \sum_{j=1}^{k+1} l_j\mathbf{e}_j^* + \sum_{j=1}^{k} l_j(\mathbf{e}_j^* l_{k+1}\mathbf{e}_{k+1}^*)$ where $\mathbf{e}_j^* l_{k+1} = 0$ for $j < k+1$
- In other words, **L** is "union" of $\mathbf{L}_1^{-1}, \mathbf{L}_2^{-1}, \cdots, \mathbf{L}_{m-1}^{-1}$

Gaussian Elimination without Pivoting

- Factorize $\mathbf{A} \in \mathbb{C}^{m\times m}$ into $\mathbf{A} = \mathbf{LU}$

> **Gaussian elimination without pivoting:**
> $\mathbf{U} = \mathbf{A}, \mathbf{L} = \mathbf{I}$;
> for $k = 1$ to $m-1$
>   for $j = k+1$ to m
> $l_{jk} = u_{jk}/u_{kk}$
> $u_{j,k:m} = u_{j,k:m} - l_{jk}u_{k,k:m}$

- Flop count $\sim \sum_{k=1}^{m} 2(m-k)(m-k) \sim 2\sum_{k=1}^{m} k^2 \sim 2m^3/3$
- In actually, **L** often overwrites lower-triangular part of **A** and **U** overwrites upper-triangular part of **A**
- Question: What if u_{kk} is 0? Answer: The algorithm would break.

Partial Pivoting

- At step k, we divide by u_{kk} , which would break if u_{kk} is 0 (or close to 0), which can happen even if **A** is nonsingular

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & x_{kk} & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & \times & \times \\ & x_{kk} & \times & \times & \times \\ & 0 & \times & \times & \times \\ & 0 & \times & \times & \times \\ & 0 & \times & \times & \times \end{bmatrix}$$

- However, any nonzero entry in kth column below diagonal can also be used as pivot

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & x_{ik} & \times & \times & \times \\ & \times & \times & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & \times & \times \\ & 0 & \times & \times & \times \\ & 0 & \times & \times & \times \\ & x_{ik} & \times & \times & \times \\ & 0 & \times & \times & \times \end{bmatrix}$$

  and we permute (interchange) row i with row k

- In general, we take nonzero entry with largest absolute value

More on Partial Pivoting

- kth step of Gaussian elimination of partial pivoting

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & x_{ik} & \times & \times \\ & \times & \times & \times \end{bmatrix}}_{Pivotselection} \xrightarrow{\mathbf{P}_k} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ & x_{kk} & \times & \times \\ & \times & \times & \times \\ & \times & \times & \times \end{bmatrix}}_{Rowinterchange} \xrightarrow{\mathbf{L}_k} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ & x_{kk} & \times & \times \\ & 0 & \times & \times \\ & 0 & \times & \times \end{bmatrix}}_{Elimination}$$

  and we interchange row i with row k
- In terms of matrices, it becomes $\underbrace{\mathbf{L}_{m-1}\mathbf{P}_{m-1}\cdots\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1}_{\mathbf{L}^{-1}\mathbf{P}}\mathbf{A} = \mathbf{U}$
- $\mathbf{P} = \mathbf{P}_{m-1}\cdots\mathbf{P}_2\mathbf{P}_1$ and $\mathbf{L} = (\mathbf{L}'_{m-1}\cdots\mathbf{L}'_2\mathbf{L}'_1)^{-1}$, where $\mathbf{L}'_k = \mathbf{P}_{m-1}\cdots\mathbf{P}_{k+1}\mathbf{L}_k\mathbf{P}_{k+1}^{-1}\cdots\mathbf{P}_{m-1}^{-1}$
- It is easy to verify that $\mathbf{L}_{m-1}\mathbf{P}_{m-1}\cdots\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1 = \mathbf{L}'_{m-1}\cdots\mathbf{L}'_2\mathbf{L}'_1(\mathbf{P}_{m-1}\cdots\mathbf{P}_2\mathbf{P}_1)$
- $\mathbf{L}'_k = \mathbf{I} - \mathbf{P}_{m-1}\cdots\mathbf{P}_{k+1}l_k\mathbf{e}_k^*$ and $\mathbf{L}$ is "union" of $(\mathbf{L}'_k)^{-1} \equiv \mathbf{I} + \mathbf{P}_{m-1}\cdots\mathbf{P}_{k+1}l_k\mathbf{e}_k^*$

Algorithm of Gaussian Elimination with Partial Pivoting

- Factorize $\mathbf{A} \in \mathbb{C}^{m\times m}$ into $\mathbf{PA} = \mathbf{LU}$

Gaussian elimination with partial pivoting:

$\mathbf{U} = \mathbf{A}, \mathbf{L} = \mathbf{I}, \mathbf{P} = \mathbf{I}$
for $k = 1$ to $m-1$
    $i \leftarrow \arg\max_{i\geq k} |u_{ik}|$
    $\mathbf{u}_{k,k:m} \leftrightarrow \mathbf{u}_{i,k:m}$
    $l_{k,1:k-1} \leftrightarrow l_{i,1:k-1}$
    $p_k \leftrightarrow p_i$
    for $j = k+1$ to m
        $l_{jk} = u_{jk}/u_{kk}$
        $u_{j,k:m} = u_{j,k:m} - l_{jk}u_{k,k:m}$

- Question: What if u_{kk} is 0?
- Flop count $\sim \sum_{k=1}^{m} 2(m-k)(m-k) \sim 2\sum_{k=1} k^2 \sim \frac{2}{3}m^3$, same as without pivoting

An Alternative Implementation

- In practice, L and U overwrite A and P is represented by a vector

Gaussian elimination with partial pivoting (alternative)

$\mathbf{p} = [1, 2, \cdots, m]$;
for $k = 1$ to $m-1$
 $i \leftarrow \arg\max_{i \geq k} |a_{ik}|$
 $\mathbf{a}_{k,1:m} \leftrightarrow \mathbf{a}_{i,1:m}$
 $p_k \leftrightarrow p_i$
 $a_{k+1:m,k} \leftarrow a_{k+1:m,k} / a_{k,k}$
 $\mathbf{A}_{k+1:m,k+1:m} \leftarrow \mathbf{A}_{k+1:m,k+1:m} - a_{k+1:m,k} * a_{k,k+1:m}$

- Using LU factorization to solve $Ax = b$:
  1. $\mathbf{PA} = \mathbf{LU}$; ($LU$ factorization with partial pivoting)
  2. $\mathbf{Ly} = \mathbf{Pb}$; (Forward substitution)
  3. $\mathbf{Ux} = \mathbf{y}$; (Back substitution)

Complete Pivoting

- More generally, we can use any nonzero entry
- In theory, any nonzero entry $(i,j), i \geq k, j \geq k$

$$\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & \times & x_{ij} & \times \\ & \times & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & \times \\ & \times & 0 & \times \\ & \times & x_{ij} & \times \\ & \times & 0 & \times \end{bmatrix}$$

  and we then permute row i with row k, column j with column k
- In matrix operations, it can be expressed as

$$\underbrace{\mathbf{L}_{m-1}\mathbf{P}_{m-1}\cdots\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1}_{\mathbf{L}^{-1}\mathbf{P}}\mathbf{A}\underbrace{\mathbf{Q}_1\mathbf{Q}_2\cdots\mathbf{Q}_{m-1}}_{\mathbf{Q}} = \mathbf{U}$$

- Therefore, $\mathbf{PAQ} = \mathbf{LU}$ where $\mathbf{P} = \mathbf{P}_{m-1}\cdots\mathbf{P}_2\mathbf{P}_1$ and $\mathbf{L} = (\mathbf{L}'_{m-1}\cdots\mathbf{L}'_2\mathbf{L}'_1)^{-1}$
- However, complete pivoting is typically not used in practice because it increases cost in search of pivot and complexity of implementation

Stability of LU Factorization

Stability of LU without Pivoting

- For $\mathbf{A} = \mathbf{LU}$ computed without pivoting

$$\tilde{\mathbf{L}}\tilde{\mathbf{U}} = \mathbf{A} + \delta\mathbf{A}, \quad \frac{\|\delta\mathbf{A}\|}{\|\mathbf{L}\|\,\|\mathbf{U}\|} = O(\varepsilon_{machine})$$

- This is close to backward stability, except that we have $\|\mathbf{L}\|\,\|\mathbf{U}\|$ instead of $\|\mathbf{A}\|$ in the denominator
- Instability of Gaussian elimination can happen only if one or both of the factors $\mathbf{L}$ and $\mathbf{U}$ is large relative to size of $\mathbf{A}$
- Unfortunately, $\|\mathbf{L}\|$ and $\|\mathbf{U}\|$ can be arbitrarily large (even for well-conditioned $\mathbf{A}$), e.g.,

$$\mathbf{A} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix} \begin{bmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{bmatrix}$$

- Therefore, the algorithm is unstable

Stability of LU Partial Pivoting

- With pivoting, all entries of **L** are in $[-1,1]$, so $\|L\| = O(1)$
- To measure growth in **U**, we introduce the growth factor $\rho = \frac{max_{i,j}|u_{ij}|}{max_{i,j}|a_{ij}|}$, and hence $\|\mathbf{U}\| = O(\rho\,\|\mathbf{A}\|)$
- We then have $\mathbf{PA} = \mathbf{LU}$

$$\tilde{\mathbf{L}}\tilde{\mathbf{U}} = \tilde{\mathbf{P}}\mathbf{A} + \delta\mathbf{A}, \quad \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} = O(\rho\varepsilon_{machine})$$

- If $|l_{ij}| < 1$ for each $i > j$(i.e., there is no tie for the pivoting), then $\tilde{\mathbf{P}} = \mathbf{P}$ for sufficiently small $\varepsilon_{machine}$
- If $\rho = O(1)$, then the algorithm is backward stable
- In fact, $\rho \leq 2^{m-1}$, so by definition ρ is a constant but can be very large

The Growth Factor

- ρ can indeed be as large as 2^{m-1}. Consider matrix

$$\begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & 0 \\ -1 & 1 & & & 0 \\ -1 & -1 & 1 & & 0 \\ -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4 \\ & & & 1 & 8 \\ & & & & 16 \end{bmatrix}$$

  where growth factor $\rho = 16 = 2^{m-1}$
- $\rho = 2^{m-1}$ is as large as ρ can get. It can be catastrophic in practice
- Theoretically, Gaussian elimination with partial pivoting is backward stable according to formal definition
- However, in the worst case, Gaussian elimination with partial pivoting may be unstable for practical values of m

The Growth Factor in Practice

- Good news: Large ρ occurs only for very skewed matrices. Experimentally, one rarely see very large ρ
- Probability of large ρ decreases exponentially in ρ
- "If you pick a billion matrices at random, you will almost certainly not find one for which Gaussian elimination is unstable"
- In practice, ρ is no larger than $O(\sqrt{m})$. However, this behavior is not fully understood yet
- In conclusion,
  - Gaussian elimination with partial pivoting is backward stable
  - In theory, its error may grow exponentially in m
  - In practice, it is stable for matrices of practical interests

Cholesky Factorization

Hermitian Positive-Definite Matrices

- Symmetric matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$ is symmetric positive definite (SPD) if $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for $\mathbf{x} \in \mathbb{R}^m \backslash \{0\}$
- Hermitian matrix $\mathbf{A} \in \mathbb{C}^{m \times m}$ is Hermitian positive definite (HPD) if $\mathbf{x}^*\mathbf{A}\mathbf{x} > 0$ for $\mathbf{x} \in \mathbb{C}^m \backslash \{0\}$
- If $\mathbf{A}$ is $m \times m$ HPD and $\mathbf{X} \in \mathbb{C}^{m \times n}$ has full column rank, then $\mathbf{X}^*\mathbf{A}\mathbf{X}$ is HPD
- Any principal submatrix (picking some rows and corresponding columns) of $\mathbf{A}$ is HPD and $a_{ii} > 0$
- HPD matrices have positive real eigenvalues and orthogonal eigenvectors
- Note: Most textbooks only talk about SPD or HPD matrices, but a positive-definite matrix does not need to be symmetric or Hermitian! A real matrix $\mathbf{A}$ is positive definite iff $\mathbf{A} + \mathbf{A}^T$ is SPD.

Cholesky Factorization

- Key idea: take advantage and preserve the properties of symmetry and positive-definiteness in factorization
- Eliminate below diagonal and to the right of diagonal

$$\mathbf{A} = \begin{bmatrix} a_{11} & \mathbf{w}^* \\ \mathbf{w} & \mathbf{K} \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ \frac{\mathbf{w}}{\alpha} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \alpha & \frac{\mathbf{w}^*}{\alpha} \\ 0 & \mathbf{K} - \frac{\mathbf{w}\mathbf{w}^*}{a_{11}} \end{bmatrix}$$

$$\begin{bmatrix} \alpha & 0 \\ \frac{\mathbf{w}}{\alpha} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \alpha & \frac{\mathbf{w}^*}{\alpha} \\ 0 & \mathbf{K} - \frac{\mathbf{w}\mathbf{w}^*}{a_{11}} \end{bmatrix} \begin{bmatrix} \alpha & \frac{\mathbf{w}^*}{\alpha} \\ 0 & \mathbf{I} \end{bmatrix} = \mathbf{R}_1^* \mathbf{A}_1 \mathbf{R}_1$$

where $\alpha = \sqrt{a_{11}}$, where $a_{11} > 0$

- $\mathbf{K} - \frac{\mathbf{w}\mathbf{w}^*}{a_{11}}$ is principal submatrix of HPD $\mathbf{A}_1 = \mathbf{R}_1^{-*}\mathbf{A}\mathbf{R}_1$ and therefore is HPD, with positive diagonal entries

Cholesky Factorization

- Apply recursively to obtain
$$\mathbf{A} = (\mathbf{R}_1^*\mathbf{R}_2^* \cdots \mathbf{R}_m^*)(\mathbf{R}_m \cdots \mathbf{R}_2\mathbf{R}_1) = \mathbf{R}^*\mathbf{R}, \quad \mathbf{r}_{jj} > 0$$
which is known as Cholesky factorization
- Question: Is $\mathbf{R}$ simply "union" of kth rows of $\mathbf{R}_k$ (or $\mathbf{R}^*$ "union" of kth columns of $\mathbf{R}_k^*$)? Yes. Hint: Write $\mathbf{R}_k^*$ in a form similar to $\mathbf{L}_k = \mathbf{I} + l_k\mathbf{e}_k^T$ in LU
- Existence and uniqueness: every HPD matrix has a unique Cholesky factorization
  - Exists because algorithm for Cholesky factorization always works for HPD matrices
  - Is unique since once $\alpha = \sqrt{a_{11}}$ is determined at each step, entire column $\frac{\mathbf{w}}{\alpha}$ is determined
  - Question: How to check whether a Hermitian matrix is positive definite? Answer: Run Cholesky factorization and it would succeeds iff the matrix is positive definite.

Algorithm of Cholesky Factorization

- Factorize Hermitian positive definite matrix $\mathbf{A} \in \mathbb{C}^{m \times m}$ into $\mathbf{A} = \mathbf{R}^*\mathbf{R}$

Algorithm: Cholesky factorization

$\mathbf{R} = \mathbf{A}$
for $k = 1$ to m
  for $j = k+1$ to m
    $r_{j,j:m} \leftarrow r_{j,j:m} - r_{k,j:m}\bar{r}_{kj}/r_{kk}$
  $r_{k,k:m} \leftarrow r_{k,k:m}/\sqrt{r_{kk}}$

- Operation count

$$\sum_{k=1}^{m}\sum_{j=k+1}^{m} 2(m-j) \sim 2\sum_{k=1}^{m}\sum_{j=1}^{k} j \sim \sum_{k=1}^{m} k^2 \sim \frac{m^3}{3}$$

Stability

Theorem

The computed Cholesky factor $\tilde{\mathbf{R}}$ satisfies

$$\tilde{\mathbf{R}}^*\tilde{\mathbf{R}} = \mathbf{A} + \delta\mathbf{A}, \quad \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} = O(\varepsilon_{machine}),$$

i.e. Cholesky factorization is backward stable

- Forward errors in $\tilde{\mathbf{R}}$ is $\left\|\tilde{\mathbf{R}} - \mathbf{R}\right\| / \|\mathbf{R}\| = O(\kappa(\mathbf{A})\varepsilon_{machine})$, which may be large for ill-conditioned $\mathbf{A}$
- Solve $\mathbf{Ax} = \mathbf{b}$ for positive definite $\mathbf{A}$
  - Factorize $\mathbf{A} = \mathbf{R}^*\mathbf{R}$; Solve $\mathbf{R}^*\mathbf{y} = \mathbf{b}$; Solve $\mathbf{Rx} = \mathbf{y}$
  - Operation count is $\sim m^3/3$
  - Algorithm is backward stable:

LDL^* Factorization

- Cholesky factorization is sometimes given by $\mathbf{A} = \mathbf{LDL}^*$ where $\mathbf{D}$ is diagonal matrix and L is unit lower triangular matrix
- It avoids computing square roots
- Analogously, LU factorization can also be written as LDU, where $\mathbf{U}$ is unit upper triangular
- Question: How is $\mathbf{R}$ in $\mathbf{A} = \mathbf{R}^*\mathbf{R}$ related to the $\mathbf{L}$ and $\mathbf{U}$ factors of $\mathbf{A} = \mathbf{LU}$?
  - $\mathbf{U} = \mathbf{DL}^* = \sqrt{\mathbf{D}}\mathbf{R}$, where $\sqrt{\mathbf{D}} \equiv diag(\sqrt{d_{11}}, \sqrt{d_{22}}, \cdots, \sqrt{d_{mm}})$
- Hermitian indefinite systems can be factorized with $\mathbf{PAP}^T = \mathbf{LDL}^*$, but $\mathbf{D}$ is block diagonal with 1×1 and 2×2 blocks. Its cost is similar to Cholesky factorization and is about 50 of Gaussian elimination.

Software for Linear Algebra

Software for Linear Algebra

- LAPACK: Linear Algebra PACKage (www.netlib.org/lapack/lug)
    - Standard library for solving linear systems and eigenvalue problems
    - Successor of LINPACK (www.netlib.org/linpack) and EISPACK (www.netlib.org/eispack)
    - Depends on BLAS (Basic Linear Algebra Subprograms)
    - Parallel extensions include ScaLAPACK and PLAPACK
    - Note: Uses Fortran conventions for matrix arrangements

Software for Linear Algebra

- MATLAB
  - Factorization $\mathbf{A}$: lu($\mathbf{A}$) and chol($\mathbf{A}$)
  - Solve $\mathbf{A}\mathbf{x} = \mathbf{b} : \mathbf{x} = \mathbf{A}\backslash\mathbf{b}$
    - Uses back/forward substitution for triangular matrices
    - Uses Cholesky factorization for positive-definite matrices
    - Uses LU factorization with column pivoting for nonsymmetric matrices
    - Uses Householder QR for least squares problems
    - Uses some special routines for matrices with special sparsity patterns
  - Uses LAPACK and other packages internally
- Serial and parallel solvers for sparse matrices (e.g., SuperLU, TAUCS)

Using LAPACK Routines in C Programs

- LAPACK was written in Fortran 77. Special attention is required when calling from C.
- Key differences between C and Fortran
  - Storage of matrices: column major (Fortran) versus row major (C/C++)
  - Argument passing for subroutines in C and Fortran: pass by reference (Fortran) and pass by value (C/C++)
- Smple example C code, example.c, for solving linear system using sgesv.
  - See class website for sample code.
  - To compile, issue command "cc -o example example.c -llapack -lblas"
- Hint: To find a function name, refer to LAPACK Users' Guide.
- To find out arguments for a given function, search on netlib.org