Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Float-point numbers in IEEE format
- Rounding, propagation of errors, and cancellation
- Truncation errors
- Matlab recap

Today

- Solving systems of linear equations
- LU decomposition
- Sparse matrices

Announcements

- Homework 2 is posted and due Tue, Oct 11 *before class*
- No office hour this week; send an email if you have questions

Linear systems of equations

It is said that 70-80% of computational mathematics research involves solving systems of m linear equations in n unknowns

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \,, \qquad i = 1, \ldots, m \,.$$

Linear systems arise directly from discrete models (e.g., in machine learning). Or through representing some abstract linear operator (such as a differential operator) in a finite basis as when numerically solving partial differential equations.

The common abstract way of writing systems of linear equations is

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \,,$$

with matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, right-hand side $\boldsymbol{b} \in \mathbb{R}^m$, and solution $\boldsymbol{x} \in \mathbb{R}^n$

The goal is to calculate solution $\boldsymbol{x}$ given data $\boldsymbol{A}, \boldsymbol{b}$ in a numerically stable and computationally efficient way.

The matrix inverse

- A square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is invertible or nonsingular if there exists a matrix inverse $\boldsymbol{A}^{-1} = \boldsymbol{B} \in \mathbb{R}^{n \times n}$ such that

$$\boldsymbol{A}\boldsymbol{B} = \boldsymbol{B}\boldsymbol{A} = \boldsymbol{I}\,,$$

  where $\boldsymbol{I}$ is the identity matrix.
- Matrix norm induced by a given vector norm

$$\|\boldsymbol{A}\| = \sup_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \implies \|\boldsymbol{A}\boldsymbol{x}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{x}\|$$

  with sub-multiplicativity: $\|\boldsymbol{A}\boldsymbol{B}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|$
- Special case of interest: The 2-norm or spectral norm: $\|\boldsymbol{A}\|_2 = \sigma_1$ (largest singular value)
- The Euclidean or Frobenius norm is *not* an induced norm

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$$

  but still is sub-multiplicative: $\|\boldsymbol{A}\boldsymbol{B}\|_F \leq \|\boldsymbol{A}\|_F \|\boldsymbol{B}\|_F$

Condition of solving system of linear equations

Recall that we derived the condition number $\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ of a matrix $\boldsymbol{A}$

Condition number of linear system $Ax=b$.

(1) Problem: Fix A, consider $b \mapsto A^{-1}b = x$, diff'able,

$\kappa_{abs} = \|f'(b)\| = \|A^{-1}\|$

$\kappa_{rel} = \dfrac{\|b\|}{\|A^{-1}b\|}\|A^{-1}\| \longrightarrow \kappa_{rel} = \dfrac{\|Ax\|}{\|x\|}\|A^{-1}\| \le \dfrac{\|A\|\|x\|}{\|x\|}\|A^{-1}\|$

$= \|A\|\|A^{-1}\| = \kappa(A)$

(x)

$\longrightarrow \dfrac{\|\delta x\|}{\|x\|} \le \kappa_{rel}(A)\dfrac{\|\delta b\|}{\|b\|}$

($\kappa_{rel}(A) = \kappa(A)$)

Condition of solving system of linear equations

Recall that we derived the condition number $\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ of a matrix $\boldsymbol{A}$

(2) Problem: Fix b, consider $A \mapsto A^{-1}b = x$

$f(A) = A^{-1}b$ diff'able, $\|f'(A)\| \le \|A^{-1}\|\,\|x\|$

$\kappa_{abs} = \|f'(A)\| \le \|A^{-1}\|\,\|x\|$

$\kappa_{rel} = \dfrac{\|A\|}{\|f(A)\|}\|f'(A)\| \le \dfrac{\|A\|}{\|x\|}\|A^{-1}\|\,\|x\|$

$= \|A^{-1}\|\,\|A\| = \kappa(A)$

Sketch for derivative:

$C \in \mathbb{R}^{n\times n}$

$\frac{\partial}{\partial t} f(A+tC) = \frac{\partial}{\partial t}(A+tC)^{-1}b =$

$\dots = A^{-1}CA^{-1}b$

$= A^{-1}Cx$

Again: $\dfrac{\|\delta x\|}{\|x\|} \le \kappa_{rel}\dfrac{\|\delta A\|}{\|A\|}$

So: Conditioning of linear system solves is governed by matrix condition number!

Condition of solving system of linear equations (cont'd)

Now consider the general perturbations of the data

$$(\boldsymbol{A} + \delta\boldsymbol{A})(\boldsymbol{x} + \delta\boldsymbol{x}) = \boldsymbol{b} + \delta\boldsymbol{b}$$

One obtains the condition (proof in Quarteroni et al., Sec. 3.1)

$$\frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{\kappa(\boldsymbol{A})}{1 - \kappa(\boldsymbol{A})\frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|}} \left(\frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} + \frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|} \right)$$

Important practical estimate: Roundoff error in the data, with rounding unit u (recall $\approx 10^{-16}$ for double precision), produces a relative error

$$\frac{\|\delta\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} \lesssim 2u\kappa(\boldsymbol{A})$$

$\implies$ it certainly makes *no* sense to try to solve systems with $\kappa(\boldsymbol{A}) > 10^{16}$

Numerical solution of linear systems

There are many numerical methods for solving a system of linear equations

The most appropriate method depends on the properties of $\boldsymbol{A}$

- General dense matrices, where the entries in $\boldsymbol{A}$ are mostly non-zero and nothing special is known $\rightsquigarrow$ we focus on Gaussian elimination today
- General sparse matrices, where only a small fraction of $a_{ij} \neq 0$ (sparse typically means that $\mathcal{O}(n)$ entries are non-zero in an $n \times n$ matrix)
- Symmetric and positive-definite matrices
- Special structured sparse matrices, often arising from specific physical properties of the underlying system

It is also important to consider how many times a linear system with the same or related matrix or right-hand side needs to be solved.

Gauss elimination and LU factorization

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

index of rhs

multiply first row by l_{i1} and subtract from 2nd row

$$l_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}}$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix}$$

eliminate using $a_{22}^{(2)}$

$$l_{32} = \frac{a_{32}^{(2)}}{a_{22}^{(2)}}$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \end{bmatrix}$$

Idea: Store multipliers l_{21}, l_{31}, l_{32} in matrix:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ l_{21} & u_{22} & u_{23} \\ l_{31} & l_{32} & u_{33} \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} U \\ L \end{bmatrix}$$

$$A = LU, \quad L = \begin{bmatrix} 1 & & \\ l_{21} & 1 & \\ l_{31} & l_{32} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & \cdots & u_{13} \\ 0 & & \vdots \\ 0 & 0 & \end{bmatrix}$$

Gauss elimination in Matlab

```
1: function A = mylu(A) % In-place LU factorization
2: % need square matrix
3: [n, m] = size(A);
4: assert(n == m);
5: for k=1:(n-1) % for variable x(k)
6:     % Assumed A(k, k) non-zero and then
7:     % calculate multipliers in column k
8:     A((k + 1):n, k) = A((k + 1):n, k)/A(k, k);
9:     for j = (k + 1):n
10:        % eliminate variable x(k)
11:        A((k + 1):n, j) = A((k + 1):n, j) - A((k + 1):
               n, k)*A(k, j);
12:    end
13: end
14: end
```

- Gaussian elimination is a general method for dense matrices and is commonly used
- Implementing Gaussian elimination efficiently is difficult and we will not discuss it
  $\rightsquigarrow$ course on HPC
- The LAPACK public-domain library is the main repository for excellent implementations of dense linear solvers
- Matlab (and numpy) use highly optimized variants of GEM by default, mostly based on LAPACK
- Matlab (and numpy) have specialized solvers for special cases of matrices, so always check help pages!

Problem?

```
>> A = [1 1 3; 2 2 2; 3 6 4]

A =

     1     1     3
     2     2     2
     3     6     4

>> mylu(A)

ans =

     1     1     3
     2     0    -4
     3   Inf   Inf
```

LU with pivoting

Zero diagonal entries (pivots) pose a problem $\rightsquigarrow$ pivoting by swapping rows

$\rightsquigarrow$ board

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix}$$

Gauss elimination

$$\begin{bmatrix} 1 & 1 & 3 \\ l_{21}=2 & 0 & -4 \\ l_{31}=3 & 3 & -5 \end{bmatrix}$$

Problem! Cannot divide by zero, pivot element is zero.

Swap rows 2 & 3

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

done – in general, use 3 as pivot.

$$LU = PA$$

Theorem: This always exists with appropriate perm. matrix P

P ... permutation matrix.

$$P_2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

```
>> A = [1 1 3; 2 2 2; 3 6 4];
>> P = [1 0 0; 0 0 1; 0 1 0] % swap row 2 and 3

P =

     1     0     0
     0     0     1
     0     1     0

>> mylu(P*A)

ans =

     1     1     3
     3     3    -5
     2     0    -4
```

- For any square (regular or singular) matrix $\boldsymbol{A}$, partial (row) pivoting ensures exists of

$$\boldsymbol{PA} = \boldsymbol{LU}$$

  where $\boldsymbol{P}$ is a permutation matrix

- **Q:** What else could pivoting be useful for?
  $\rightsquigarrow$ let's see what Matlab does

```
>> [L, U, P] = lu(A) % built-in lu
L =
    1.0000         0         0
    0.6667    1.0000         0
    0.3333    0.5000    1.0000
U =
    3.0000    6.0000    4.0000
         0   -2.0000   -0.6667
         0         0    2.0000
P =
     0     0     1
     0     1     0
     1     0     0
>> norm(L*U - P*A)
ans = 0
```

```
>> [L, U, P] = lu(A) % built-in lu
L =
    1.0000         0         0
    0.6667    1.0000         0
    0.3333    0.5000    1.0000
U =
    3.0000    6.0000    4.0000
         0   -2.0000   -0.6667
         0         0    2.0000
P =
     0     0     1
     0     1     0
     1     0     0
>> norm(L*U - P*A)
ans = 0
```

Reverses order of rows rather than just swapping 2 and 3.
Leads to entries of L with magnitude ≤ 1

```
>> A = [1e-20 1; 1 1]

A =

    1.0000e-20    1.0000e+00
    1.0000e+00    1.0000e+00

>> LUmat = mylu(A);
L = [1 0; LUmat(2, 1) 1];
U = LUmat; U(2, 1) = 0;
L*U

ans =

    1.0000e-20    1.0000e+00
    1.0000e+00             0
```

```
>> [L, U, P] = lu(A)
L =
   1.0000e+00             0
   1.0000e-20   1.0000e+00
U =
     1     1
     0     1
P =
     0     1
     1     0
>> P'*L*U
ans =
   1.0000e-20   1.0000e+00
   1.0000e+00   1.0000e+00
>> A
A =
   1.0000e-20   1.0000e+00
   1.0000e+00   1.0000e+00
```

Instability of LU decomposition *without* pivoting

If $\boldsymbol{A}$ has an $\boldsymbol{LU}$ factorization, then the computed $\tilde{\boldsymbol{L}}$ and $\tilde{\boldsymbol{U}}$ obtained in floating-point arithmetic with Gaussian elimination satisfy $\tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}} = \boldsymbol{A} + \delta\boldsymbol{A}$ with the bound

$$\frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{L}\|\|\boldsymbol{U}\|} \in \mathcal{O}(u)\,,$$

where u is the roundoff unit.

- Notice that we would have liked to bound $\|\delta\boldsymbol{A}\|/\|\boldsymbol{A}\|$ but we got $\|\delta\boldsymbol{A}\|/(\|\boldsymbol{L}\|\|\boldsymbol{U}\|)$
- Thus, for matrices with $\|\boldsymbol{L}\|\|\boldsymbol{U}\| \approx \|\boldsymbol{A}\|$, the algorithm will show stable behavior
- However, if $\|\boldsymbol{L}\|\|\boldsymbol{U}\| \not\approx \|\boldsymbol{A}\|$, then we can get an unstable result

$\rightsquigarrow$ Gaussian elimination is *not* stable in general

Example $\rightsquigarrow$ board

$$\boldsymbol{A} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{L} = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \boldsymbol{U} = \begin{bmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{bmatrix}$$

Compare $\|\boldsymbol{L}\|_\infty\|\boldsymbol{U}\|_\infty$ and $\|\boldsymbol{A}\|_\infty$

LU with row pivoting to maximum element

```
1: function [A, P] = myplu(A) % In-place LU factorization
2: [n, m] = size(A); P = eye(n);
3: for k=1:(n-1) % for variable x(k)
4:     [~, selI] = max(abs(A(k:n, k))); % select pivot
5:     c = A(k, k:end); d = P(k, :);
6:     A(k, k:end) = A(selI + (k-1), k:end); P(k, :) = P(
          selI + (k-1), :);
7:     A(selI + (k-1), k:end) = c; P(selI + k-1, :) = d;
8:     % calculate multipliers in column k
9:     A((k + 1):n, k) = A((k + 1):n, k)/A(k, k);
10:    for j = (k + 1):n % eliminate variable x(k)
11:        A((k + 1):n, j) = A((k + 1):n, j) - A((k + 1):
              n, k)*A(k, j);
12:    end
13: end
14: end
```

```
>> A = [1e-20 1; 1 1]
[LUmat, P] = myplu(A);
L = [1 0; LUmat(2, 1) 1];
U = LUmat; U(2, 1) = 0;
P'*L*U

A =

    1.0000e-20   1.0000e+00
    1.0000e+00   1.0000e+00


ans =

    1.0000e-20   1.0000e+00
    1.0000e+00   1.0000e+00
```

Stability of Gaussian elimination with pivoting

For $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$, introduce the growth factor

$$\rho = \frac{\max_{i,j} |u_{ij}|}{\max_{i,j} |a_{ij}|}$$

where u_{ij} is the i,j-th element of $\boldsymbol{U}$

Consider the factorization $\boldsymbol{P}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ with partial row pivoting w.r.t. taking the maximum element for a matrix $\boldsymbol{A}$ of dimension $n \times n$. Gaussian elimination gives $\tilde{\boldsymbol{P}}, \tilde{\boldsymbol{L}}, \tilde{\boldsymbol{U}}$ that satisfy

$$\tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}} = \tilde{\boldsymbol{P}}\boldsymbol{A} + \delta\boldsymbol{A}\,, \qquad \frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|} \in \mathcal{O}(\rho u)\,,$$

where u is the roundoff unit. If all off-diagonal entries of $\boldsymbol{L}$ are < 1, implying that there are no ties in the selection of pivots in exact arithmetic, then $\boldsymbol{P} = \tilde{\boldsymbol{P}}$ for sufficiently small u.

This means that Gaussian elimination with partial pivoting is backward stable if ρ holds uniformly for matrices with $n \times n$.

- For any square (regular or singular) matrix $\boldsymbol{A}$, partial (row) pivoting ensures exists of

$$\boldsymbol{PA} = \boldsymbol{LU}$$

  where $\boldsymbol{P}$ is a permutation matrix

- Furthermore, pivoting (w.r.t. $\max|a_{ij}|$) leads to a backward stable algorithm. **However**, the growth factor ρ can be huge and grow with the dimension of $\boldsymbol{A}$! Fortunately, large factors ρ "never seem to appear in real applications." (Trefethen & Bau, Chapter 22)

- There also is full pivoting (rows + columns)

$$\boldsymbol{PAQ} = \boldsymbol{LU}$$

  to further increases stability but it usually is not worth it in practice (higher costs to search for pivoting element over rows and columns but little improvement in terms of stability)

Solving linear systems

- Once an LU factorization is available, solving a linear system is cheap:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{L}\boldsymbol{U}\boldsymbol{x} = \boldsymbol{L}(\boldsymbol{U}\boldsymbol{x}) = \boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$$

- Solve for $\boldsymbol{y}$ using *forward substitution*
- Solve for $\boldsymbol{x}$ by using *backward substitution* $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$

What is *forward/backward substitution*? $\rightsquigarrow$ board

$$\begin{bmatrix} l_{11} & 0 & \cdots & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ \vdots & & & & \vdots \\ l_{n1} & \cdots & \cdots & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

Forward/backward substitution

Forward substitution requires

$$\frac{n(n-1)}{2} \text{ multiplications/additions,}$$
$$n \text{ divisons.}$$

Overall: $\sim n^2$ floating point operations (flops) $\rightsquigarrow$ costs scale as $\mathcal{O}(n^2)$

Similarly, backward substitution has costs that scale as $\mathcal{O}(n^2)$

We count flops to estimate the computational time/effort. Besides floating point operations, computer memory access has a significant influence on the efficiency of numerical methods.

- If row pivoting is used, the same process works by also permuting the right-hand side $\boldsymbol{b}$

$$\boldsymbol{PAx} = \boldsymbol{LUx} = \boldsymbol{Ly} = \boldsymbol{Pb}$$

or formally (never implement inverse for solving linear systems of equations)

$$\boldsymbol{x} = (\boldsymbol{LU})^{-1}\boldsymbol{Pb} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{Pb}$$

- Because $\boldsymbol{P}$ is orthonormal, we have $\boldsymbol{P}^{-1} = \boldsymbol{P}^{T}$ and thus

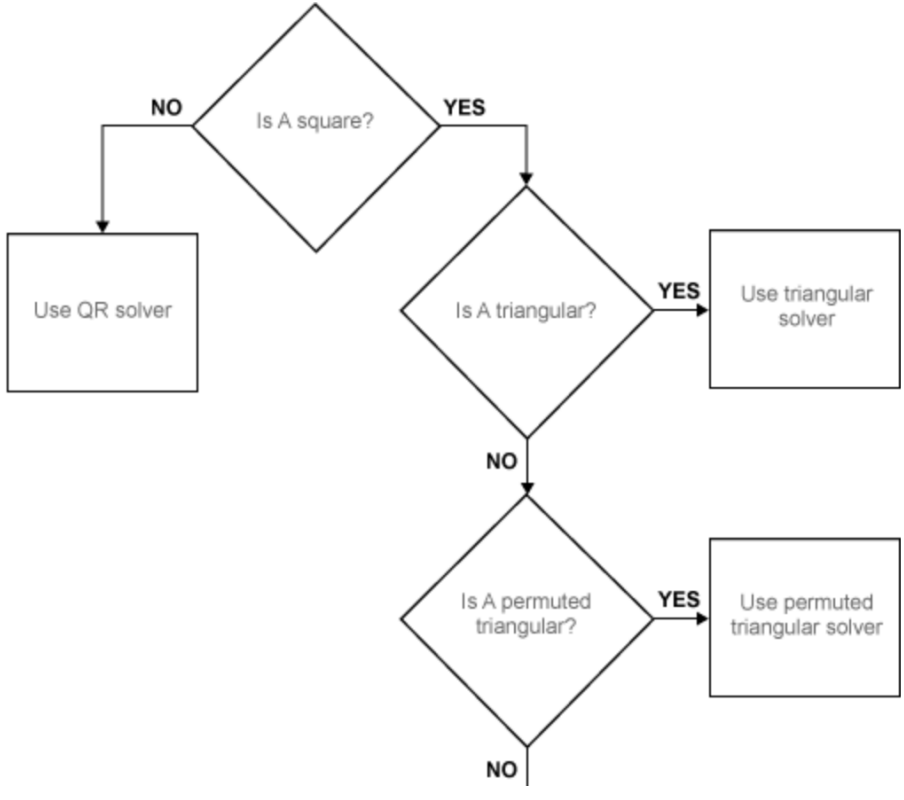
$$\boldsymbol{A} = \boldsymbol{P}^{-1}\boldsymbol{LU} = (\boldsymbol{P}^{T}\boldsymbol{L})\boldsymbol{U} = \tilde{\boldsymbol{L}}\boldsymbol{U}\,,$$

with $\tilde{\boldsymbol{L}}$ a row permutation of a unit lower triangular matrix

In Matlab, the backslash operator solves linear systems (see help `mldivide`)

```
A = [1 2 3; 4 5 6; 7 8 0];
b = [2; 1; -1];
x = A\b; x'
[L, U] = lu(A)
y = L\b; x = U\y; x'
ans =
   -2.5556e+00   2.1111e+00   1.1111e-01
L =
    1.4286e-01   1.0000e+00            0
    5.7143e-01   5.0000e-01   1.0000e+00
    1.0000e+00            0            0
U =
    7.0000e+00   8.0000e+00            0
              0   8.5714e-01   3.0000e+00
              0            0   4.5000e+00
ans =
   -2.5556e+00   2.1111e+00   1.1111e-01
```

Is the permuted triangular matrix $\tilde{\boldsymbol{L}}$ (which we get from `[L, U] = lu(A)`) detected as such? Yes!



Costs

For forward [backward] substitution at step k there are $\approx k$ $[(n-k)]$ multiplications and subtractions plus a few divisions. The total over all n steps is

$$\sum_{k=1}^{n} k \in \mathcal{O}(n^2)$$

$\rightsquigarrow$ the number of floating-point operations (FLOPs) scales as $\mathcal{O}(n^2)$

For Gaussian elimination, at step k, there are $\approx (n-k)^2$ operations. Thus, the total scales as

$$\sum_{k=1}^{n} (n-k)^2 \in \mathcal{O}(n^3)$$

Summary:

- Directly applying Gaussian elimination (=LU + fwd/bwd) scales as $\mathcal{O}(n^3)$
- Computing LU decomposition scales as $\mathcal{O}(n^3)$
- Forward/backward substitution scales as $\mathcal{O}(n^2)$
- LU + forward/backward scales as $\mathcal{O}(n^3)$ $\rightsquigarrow$ can *reuse LU* for other b

Choleski factorization

A matrix is symmetric positive definite (spd), if $A = A^T$ and for all $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} \neq 0$, the inner product $\langle A\boldsymbol{x}, \boldsymbol{x} \rangle > 0$.

For spd matrices, we can compute the factorization:

$$A = LDL^T,$$

where L is a lower triangular matrix with 1's on the diagonal, and D is a positive diagonal matrix.

The Choleski factorization is obtained by multiplying the square root of D (which exists!) with L:

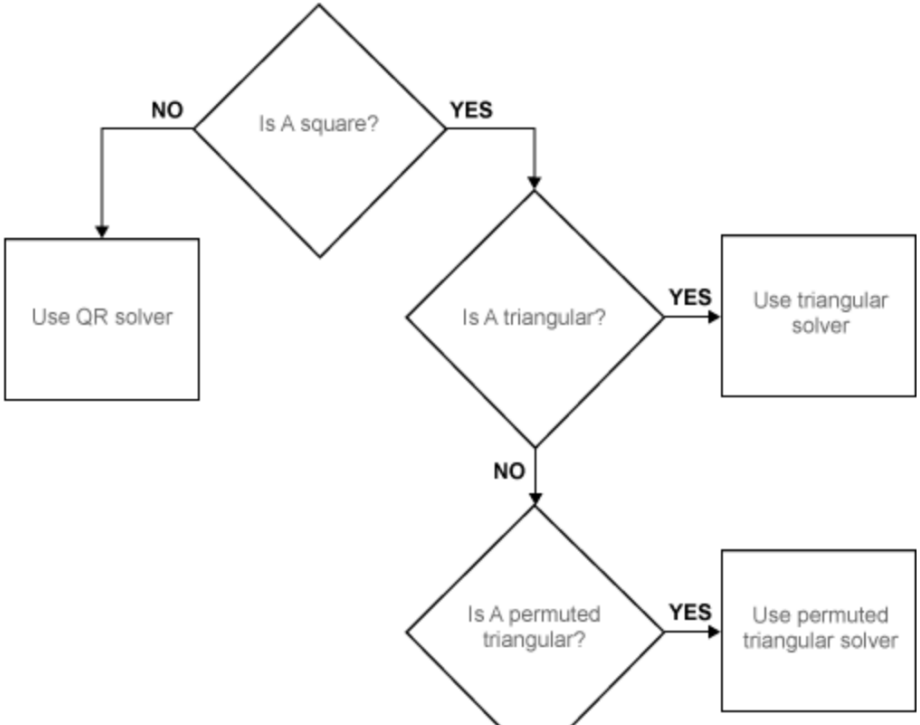
$$A = \bar{L}\bar{L}^T.$$

Algorithms for Choleski factorization are about twice as fast as Gaussian elimination but also scale as $\mathcal{O}(n^3)$.

```
1: >> A = randn(1000, 1000)*diag(linspace(1, 10, 1000))*
      randn(1000, 1000); A = A'*A;
2: >> tic; chol(A); toc
3: Elapsed time is 0.004863 seconds.
4: >> tic; lu(A); toc
5: Elapsed time is 0.010114 seconds.
```

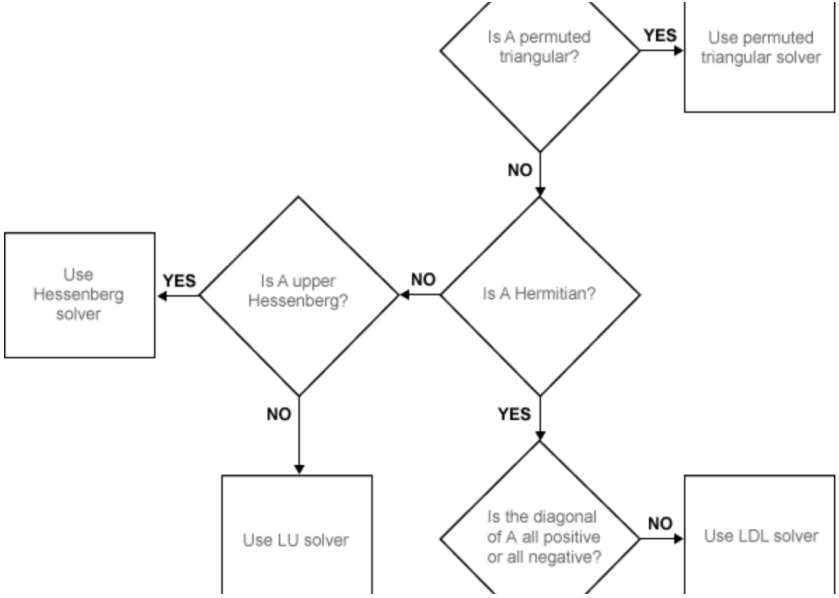
Special matrices in Matlab

Matlab tests for special matrices automatically and chooses a good decomposition/solver



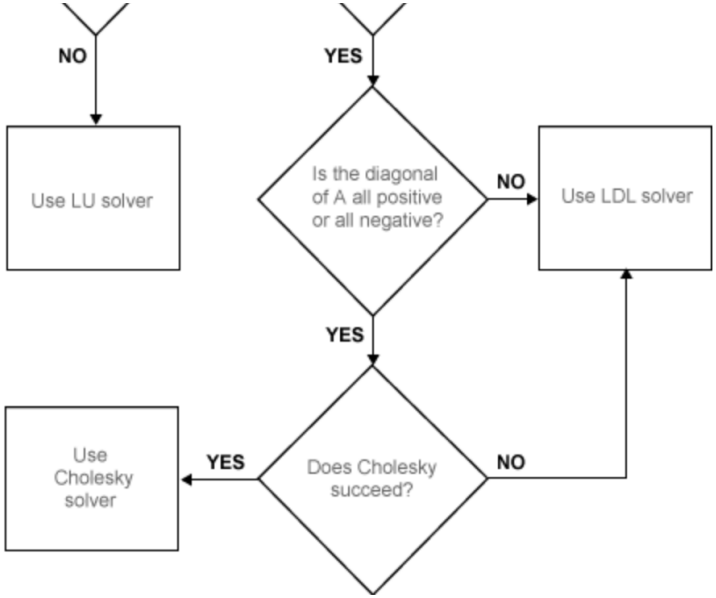
Special matrices in Matlab

Matlab tests for special matrices automatically and chooses a good decomposition/solver



Special matrices in Matlab

Matlab tests for special matrices automatically and chooses a good decomposition/solver



Conclusions/summary

- The condition of solving a linear system $\boldsymbol{Ax} = \boldsymbol{b}$ is determined by the condition number of the matrix $\boldsymbol{A}$

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\| \geq 1$$

- Gaussian elimination can be used to solve general square linear systems and produces a factorization, if it exists

$$\boldsymbol{A} = \boldsymbol{LU}$$

- Partial pivoting is sufficient for existence and stability of the LU decomposition

$$\boldsymbol{PA} = \boldsymbol{LU}\,, \qquad \boldsymbol{A} = \tilde{\boldsymbol{L}}\boldsymbol{U}$$

- The Cholesky factorization $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$ exists if $\boldsymbol{A}$ is spd and then it is the better choice (stabler, cheaper) than LU

- Rely on the highly optimized routines in Matlab (LAPACK) and other software packages than implementing these algorithms yourself $\rightsquigarrow$ take the course on HPC next spring to learn more about the efficient *implementation* of these algorithms