Module 2: Square Linear Systems

Preliminary: Floating-Point Numbers

Absolute and Relative Errors

In numerical analysis, we use an **algorithm** to *approximate* some quantity of interest.

- We estimate of the accuracy of the computed value via an **absolute error** or a **relative error**:

$$\begin{aligned} \mathrm{e}_{\mathrm{abs}} &= A_{\mathrm{approx}} - A_{\mathrm{exact}} && \text{(absolute error)} \\ \mathrm{e}_{\mathrm{rel}} &= \frac{A_{\mathrm{approx}} - A_{\mathrm{exact}}}{A_{\mathrm{exact}}} = \frac{A_{\mathrm{approx}}}{A_{\mathrm{exact}}} - 1\,, && \text{(relative error)} \end{aligned}$$

  where A_{exact} is the exact, analytical answer and A_{approx} is the approximate, numerical answer.

- If $\mathrm{e}_{\mathrm{abs}}$ or $\mathrm{e}_{\mathrm{rel}}$ is small, we say that the approximate answer is **accurate**.

Example: Stirling's Formula

Stirling's formula provides a "good" approximation to $n!$ for large n:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n. \qquad (\star)$$

- Assume that the exact value of $n!$ is found by `factorial`.
- Estimate $n!$ using $(\star)$.
- Show the accuracy of this approximation for various values of n.

Try in MATLAB:

```
n = ...;
err_abs = sqrt(2*pi*n)*(n/exp(1))^n - factorial(n);
err_rel = err_abs/factorial(n);
disp(err_abs)
disp(err_rel)
```

Limitations of Digital Representations

A digital computer uses a finite number of bits to represent a real number and so it cannot represent all real numbers.

- The represented numbers cannot be arbitrarily large or small;
- There must be gaps between them.

So for all operations involving real numbers, it uses a subset of $\mathbb{R}$ called the **floating-point numbers**, $\mathbb{F}$.

Floating-Point Numbers

A *floating-point number* is written in the form $\pm(1+F)2^E$ where

- E, the *exponent*, is an integer;
- F, the *mantissa*, is a number $F = \sum_{i=1}^{d} b_i 2^{-i}$, with $b_i = 0$ or $b_i = 1$.

Note that F can be rewritten as

$$F = 2^{-d} \underbrace{\sum_{k=0}^{d-1} b_{d-k} 2^k}_{=:M},$$

where M is an integer in $\mathbb{N}[0, 2^d - 1]$.

Consequently, there are 2^d evenly-spaced numbers between 2^E and 2^{E+1} in the floating-point number system.

Floating-Point Numbers – IEEE 754 Standard

- MATLAB, by default, uses *double precision* floating-point numbers, stored in memory in 64 bits (or 8 bytes):

$$\pm \underbrace{1.\mathtt{xxxxxxxx} \cdots \mathtt{xxxxxxxx}_{(2)}}_{\text{mantissa (base 2): 52+1 bits}} \times 2^{\overbrace{\mathtt{xxxx}\cdots\mathtt{xxxx}_{(2)} - 1023}}.$$

$$\underbrace{\phantom{\mathtt{xxxx}\cdots\mathtt{xxxx}_{(2)} - 1023}}_{\text{exponent: 11 bits}}$$

- Predefined variables:
  - `eps` = the distance from 1.0 to the next largest double-precision number:

$$\texttt{eps} = 2^{-52} \approx 2.2204 \times 10^{-16}.$$

  - `realmin` = the smallest positive floating-point number that is stroed to full accuracy; the actual smallest is `realmin/2^52`.
  - `realmax` = the largest positive floating-point number

Machine Epsilon and Relative Errors

The IEEE standard guarantees that the *relative representation error* and the *relative computational error* have sizes smaller than $\boxed{\text{eps}}$, the *machine epsilon*:

- **Representation**: The floating-point representation, $\hat{x} \in \mathbb{F}$, of $x \in \mathbb{R}$ satisfies
$$\hat{x} = x(1+\epsilon_1), \qquad \text{for some } |\epsilon_1| \leqslant \frac{1}{2}\boxed{\text{eps}}.$$
- **Arithmetic**: The floating-point representation, $\hat{x} \oplus \hat{y}$, of the result of $\hat{x} + \hat{y}$ with $\hat{x}, \hat{y} \in \mathbb{F}$ satisfies
$$\hat{x} \oplus \hat{y} = (\hat{x} + \hat{y})(1+\epsilon_2), \qquad \text{for some } |\epsilon_2| \leqslant \frac{1}{2}\boxed{\text{eps}}.$$

  Similarly with $\ominus, \otimes, \oplus$ corresponding to $-, \times, \div$, respectively.

Round-Off Errors

Computers CANNOT usually

- represent a number correctly;
- add, subtract, multiply, or divide correctly!!

Run the following and examine the answers:

```
format long
1.234567890123456789O
123456789012345678 90
(1 + eps) - 1
(1 + .5*eps) - 1
(1 + .51*eps) - 1
n = input(' n = '); ( n^(1/3) )^3 - n
```

Catastrophic Cancellation

In finite precision storage, two numbers that are close to each other are indistinguishable. So subtraction of two nearly equal numbers on a computer can result in loss of many significant digits.

Catastrophic Cancellation

Consider two real numbers stored with 10 digits of precision:

$$\begin{aligned} e &= 2.7182818284, \\ b &= 2.7182818272. \end{aligned}$$

- Suppose the actual numbers e and b have additional digits that are not stored.
- The stored numbers are good approximations of the true values.
- However, if we compute $e - b$ based on the stored numbers, we obtain $0.0000000012 = 1.2 \times 10^{-9}$, a number with only two significant digits.

Example 1: Cancellation for Large Values of x

Question

Compute $f(x) = e^x(\cosh x - \sinh x)$ at $x = 1, 10, 100$, and 1000.

Numerically:

```
format long
x = input(' x = ');
y = exp(x) * ( cosh(x) - sinh(x) );
disp([x, y])
```

Example 2: Cancellation for Small Values of x

Question

Compute $f(x) = \dfrac{\sqrt{1+x}-1}{x}$ at $x = 10^{-12}$.

Numerically:

```
x = 1e-12;
fx = (sqrt(1+x) - 1)/x;
disp( fx )
```

To Avoid Such Cancellations . . .

- Unfortunately, there is no universal way to avoid loss of precision.
- One way to avoid catastrophic cancellation is to remove the source of cancellation by simplifying the given expression before computing numerically.
- For Example 1, rewrite the given expression recalling that

$$\cosh x = (e^x + e^{-x})/2 \qquad \sinh x = (e^x - e^{-x})/2.$$

- For Example 2, try again after rewriting $f(x)$ as

$$f(x) = \frac{\sqrt{1+x} - 1}{x} \cdot \frac{\sqrt{1+x} + 1}{\sqrt{1+x} + 1} = \frac{1}{\sqrt{1+x} + 1}.$$

- Do you now have an improved accuracy?

Preliminary: Conditioning

Problems and Conditioning

- A mathematical *problem* can be viewed as a function $f : X \to Y$ from a data/input space X to a solution/output space Y.
- We are interested in changes in $f(x)$ caused by small perturbations of x.
- A *well-conditioned* problem is one with the property that all small perturbations of x lead to only small changes in $f(x)$

Condition Number

Let $f : \mathbb{R} \to \mathbb{R}$ and $\hat{x} = x(1+\epsilon)$ be the representation of $x \in \mathbb{R}$.

- The ratio of the relative error in f due to the change in x to the relative error in x simplifies to

$$\frac{\left|f(x) - f(x(1+\epsilon))\right|}{\left|\epsilon f(x)\right|}.$$

- In the limit of small error (ideal computer), we obtain

$$\kappa_f(x) := \lim_{\epsilon \to 0} \frac{\left|f(x) - f(x(1+\epsilon))\right|}{\left|\epsilon f(x)\right|}$$
$$= \left|\lim_{\epsilon \to 0} \frac{f(x+\epsilon x) - f(x)}{\epsilon x} \cdot \frac{x}{f(x)}\right| = \left|\frac{x f'(x)}{f(x)}\right|, \quad (\star)$$

which is called the **(relative) condition number**.

Example: Conditioning of Subtraction

Consider $f(x) = x - c$ where c is some constant. Using the formula ($\star$), we find that the associated condition number is

$$\kappa(x) = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x}{x-c}\right|.$$

- It is large when $x \approx c$.

Example: Conditioning of Multiplication

The condition number of $f(x) = cx$ is

$$\kappa(x) = \left| \frac{x f'(x)}{f(x)} \right| = \left| \frac{x \cdot c}{cx} \right| = 1.$$

- No magnification of error.

Example: Conditioning of Function Evaluation

The condition number of $f(x) = \cos(x)$ is

$$\kappa(x) = \left| \frac{x f'(x)}{f(x)} \right| = \left| \frac{-x \sin x}{\cos x} \right| = \left| x \tan x \right| .$$

- The condition number is large when $x = (n + 1/2)\pi$, where $n \in \mathbb{Z}$.

Example: Conditioning of Root-Finding

Let $r = f(a; b, c)$ be a root of $ax^2 + bx + c = 0$. Instead of direct differentiation, use implicit differentiation

$$r^2 + 2ar\frac{dr}{da} + b\frac{dr}{da} = 0.$$

Solve for the derivative,

$$f'(a) = \frac{dr}{da} = -\frac{r^2}{2ar + b} = -\frac{r^2}{\pm\sqrt{b^2 - 4ac}},$$

then compute the condition number using the formula ($\star$) to get

$$\kappa(a) = \left|\frac{a f'(a)}{f(a)}\right| = \left|\frac{ar^2}{\pm r\sqrt{b^2 - 4ac}}\right| = \left|\frac{ar}{\sqrt{b^2 - 4ac}}\right|.$$

- Conditioning is poor for small discriminant, *i.e.*, near repeated roots.

Preliminary: Stability

Algorithms

- Recall that we defined a *problem* as a function $f : X \to Y$.
- An *algorithm* can be viewed as another map $\tilde{f} : X \to Y$ between the same two spaces, which involves errors arising in
  - representing the actual input x as $\hat{x}$;
  - implementing the function f numerically on a computer.

Analysis – General Framework

The relative error of our interest is

$$\left|\frac{\tilde{f}(\hat{x}) - f(x)}{f(x)}\right| \leqslant \left|\frac{\tilde{f}(\hat{x}) - f(\hat{x})}{f(x)}\right| + \left|\frac{f(\hat{x}) - f(x)}{f(x)}\right|$$

$$\lessapprox \underbrace{\left|\frac{\tilde{f}(\hat{x}) - f(\hat{x})}{f(\hat{x})}\right|}_{\text{numerical error}} + \underbrace{\left|\frac{f(\hat{x}) - f(x)}{f(x)}\right|}_{\text{perturbation error}} \leqslant (\hat{\kappa}_{\text{num}} + \kappa_f)\,\boxed{\text{eps}}.$$

where $\kappa = \kappa_f$ be the (relative) condition number of the exact problem f and

$$\hat{\kappa}_{\text{num}} = \max \left|\frac{\tilde{f}(\hat{x}) - f(\hat{x})}{f(\hat{x})}\right| \Bigg/ \left|\frac{\hat{x} - x}{x}\right|.$$

Example: Root-Finding Revisited

Consider again solving the quadratic problem $ar^2 + br + c = 0$.

- Taking $a = c = 1$ and $b = -(10^6 + 10^{-6})$, the roots can be computed exactly by hand: $r_1 = 10^6$ and $r_2 = 10^{-6}$.
- If numerically computed in MATLAB using the quadratic equation formula, r_1 is correct but r_2 has only 5 correct digits.
- Fix it using $r_2 = (c/a)/r_1$.

Introduction to Square Linear Systems

Polynomial Interpolation

Formal Statement

Given a set of n data points $\{(x_j, y_j) \mid j \in \mathbb{N}[1, n]\}$ with distinct x_j's, not necessarily sorted, find a polynomial of degree $n - 1$,

$$p(x) = c_1 + c_2 x + c_3 x^2 + \cdots + c_n x^{n-1}, \qquad (\star)$$

which interpolates the given points, *i.e.*,

$$p(x_j) = y_j, \quad \text{for } j = 1, 2, \ldots, n\,.$$

- The goal is to determine the coefficients $c_1, c_2, \ldots, c_n$.
- Note that the total number of data point is 1 larger than the degree of the interpolating polynomial.

Why Do We Care?

- to find the values between the discrete data points;
- to approximate a (complicated) function by a polynomial, which makes such computations as differentiation or integration easier.

Interpolation to Linear System

Writing out the n *interpolating conditions* $p(x_j) = y_j$:

Equations

$$\left\{\begin{array}{ccccccccc} c_1 & + & c_2 x_1 & + & \cdots & + & c_n x_1^{n-1} & = & y_1 \\ c_1 & + & c_2 x_2 & + & \cdots & + & c_n x_2^{n-1} & = & y_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ c_1 & + & c_2 x_n & + & \cdots & + & c_n x_n^{n-1} & = & y_n \end{array}\right\}$$

Matrix equation

$$\rightarrow \underbrace{\begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}}_{\mathbf{c}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\mathbf{y}}$$

- This is a linear system of n equations with n unknowns.
- The matrix V is called a **Vandermonde matrix**.

Example: Fitting Population Data

U.S. Census data are collected every 10 years.

Year	Population (millions)
1980	226.546
1990	248.710
2000	281.422
2010	308.746
2020	332.639

Question. How do we estimate population in other years?

- Interpolate available data to compute population in intervening years.

Example: Fitting Population Data

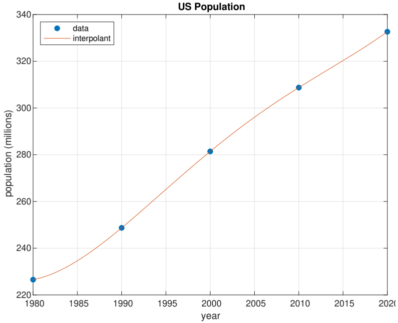
- Input data.
- Match up notation (optional).
- Note the shift in Line 7.
- Construct the Vandermonde matrix V by *broadcasting*.
- Solve the system using the backslash ($\backslash$) operator.

```
1  year = (1980:10:2020)';
2  pop = [226.546;
3         248.710;
4         281.422;
5         308.746;
6         332.639];
7  x = year - 1980;
8  y = pop;
9  n = length(x);
10 V = x.^(0:n-1);
11 c = V \ y;
```

Post-Processing

```
xx = linspace(0, 40, 100)';
yy = polyval(flip(c), xx);
clf
plot(1980+x, y, '.', 1980+xx, yy)
title('US Population'),
xlabel('year'), ylabel('population (millions)')
legend('data', 'interpolant', 'location', 'northwest')
```

- Use the `polyval` function to evaluate the polynomial.
- MATLAB expects coefficients to be in descending order. (`flip`)



Overview

Let $A \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^n$. Then the equation $A\mathbf{x} = \mathbf{b}$ has the following possibilities:

- If A is invertible (or nonsingular), then $A\mathbf{x} = \mathbf{b}$ has a unique solution $\mathbf{x} = A^{-1}\mathbf{b}$, or
- If A is not invertible (or singular), then $A\mathbf{x} = \mathbf{b}$ has either no solution or infinitely many solutions.

The Backslash Operator " \ "

To solve for $\mathbf{x}$ in MATLAB, we use the backslash symbol " \ ":

```
>> x = A \ b
```

This produces the solution without explicitly forming the inverse of A.

Warning: Even though $\mathbf{x} = A^{-1}\mathbf{b}$ analytically, don't use `x = inv(A)*b`!

Triangular Systems

Systems involving triangular matrices are easy to solve.

- A matrix $U \in \mathbb{R}^{n\times n}$ is **upper triangular** if all entries below main diagonal are zero:

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}.$$

- A matrix $L \in \mathbb{R}^{n\times n}$ is **lower triangular** if all entries above main diagonal are zero:

$$L = \begin{bmatrix} \ell_{11} & 0 & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & \ell_{nn} \end{bmatrix}.$$

Example: Upper Triangular Systems

Solve the following 4×4 system

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

General Results

- **Backward Substitution.** To solve a general $n \times n$ upper triangular system $U\mathbf{x} = \mathbf{y}$:

$$\begin{cases} x_n = \dfrac{b_n}{u_{nn}} \quad \text{and} \\ x_i = \dfrac{1}{u_{ii}} \left(b_i - \displaystyle\sum_{j=i+1}^{n} u_{ij} x_j \right) \end{cases}$$

  for $i = n-1, n-2, \ldots, 1$.

- **Forward Elimination.** To solve a general $n \times n$ lower triangular system $L\mathbf{x} = \mathbf{y}$:

$$\begin{cases} x_1 = \dfrac{b_1}{\ell_{11}} \quad \text{and} \\ x_i = \dfrac{1}{\ell_{ii}} \left(b_i - \displaystyle\sum_{j=1}^{i-1} \ell_{ij} x_j \right) \end{cases}$$

  for $i = 2, 3, \ldots, n$.

Implementation: Backward Substitution

```
function x = backsub(U,b)
% BACKSUB x = backsub(U,b)
% Solve an upper triangular linear system.
% Input:
%   U    upper triangular square matrix (n by n)
%   b    right-hand side vector (n by 1)
% Output:
%   x    solution of Ux=b (n by 1 vector)
    n = length(U);
    x = zeros(n,1); % preallocate
    for i = n:-1:1
        x(i) = ( b(i) - U(i,i+1:n)*x(i+1:n) ) / U(i,i);
    end
end
```

Implementation: Forward Elimination

Exercise. Complete the code below.

```
function x = forelim(U,b)
% FORELIM x = forelim(L,b)
% Solve a lower triangular linear system.
% Input:
%   L     lower triangular square matrix (n by n)
%   b     right-hand side vector (n by 1)
% Output:
%   x     solution of Lx=b (n by 1 vector)






end
```

Does It Always Work?

Theorem 1 (Singularity of Triangular Matrix)

A triangular matrix is singular if and only if at least one of its diagonal elements is zero.

LU Factorization

General Method: Gaussian Elimination

- *Gaussian elimination* is an algorithm for solving a general system of linear equations that involves a sequence of row operations performed on the associated matrix of coefficients.
- This is also known as the method of row reduction.
- There are three variations to this method:
  - G.E. without pivoting
  - G.E. with partial pivoting (that is, row pivoting)
  - G.E. with full pivoting (that is, row and column pivoting)

G.E. Without Pivoting: Example

Key Example

Solve the following system of equations.

$$\begin{cases} 2x_1 + 2x_2 + \ \ x_3 = \ \ 6 \\ -4x_1 + 6x_2 + \ \ x_3 = -8 \\ 5x_1 - 5x_2 + 3x_3 = \ \ 4 \end{cases} \xrightarrow{\text{matrix equation}} \underbrace{\begin{bmatrix} 2 & 2 & 1 \\ -4 & 6 & 1 \\ 5 & -5 & 3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 6 \\ -8 \\ 4 \end{bmatrix}}_{\mathbf{b}}$$

Step 1: Write the corresponding *augmented matrix* and row-reduce to an echelon form.

$$\left[\begin{array}{ccc|c} 2 & 2 & 1 & 6 \\ -4 & 6 & 1 & -8 \\ 5 & -5 & 3 & 4 \end{array}\right] \to \left[\begin{array}{ccc|c} 2 & 2 & 1 & 6 \\ 0 & 10 & 3 & 4 \\ 0 & 10 & -0.5 & 11 \end{array}\right] \to \left[\begin{array}{ccc|c} 2 & 2 & 1 & 6 \\ 0 & 10 & 3 & 4 \\ 0 & 0 & 3.5 & -7 \end{array}\right].$$

Step 2: Solve for x_3, then x_2, and then x_1 via *backward substitution*.

$$\mathbf{x} = (3, 1, -2)^{\mathrm{T}}.$$

G.E. without Pivoting: General Procedure

As shown in the example, G.E. without pivoting involves two steps:

1. **Row reduction:** Transform $A\mathbf{x} = \mathbf{b}$ to $U\mathbf{x} = \boldsymbol{\beta}$ where

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ \mathbf{0} & & & u_{nn} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}.$$

2. **Backward substitution:** Solve $U\mathbf{x} = \boldsymbol{\beta}$ for $\mathbf{x}$ by

$$\begin{cases} x_n = \dfrac{\beta_n}{u_{nn}} \quad \text{and} \\ x_i = \dfrac{1}{u_{ii}} \left(\beta_i - \displaystyle\sum_{j=i+1}^{n} u_{ij} x_j \right), \quad \text{for } i = n-1, n-2, \ldots, 1\,. \end{cases}$$

G.E. without Pivoting: MATLAB Implementation

```
1 function x = GEnp(A, b)
2   % Step 1: Row reduction to upper tri. system
3   S = [A, b];            % augmented matrix
4   n = size(A, 1);
5   for j = 1:n-1
6       for i = j+1:n
7           mult = -S(i,j)/S(j,j);
8           S(i,:) = S(i,:) + mult*S(j,:);
9       end
10  end
11  % Step 2: Backward substitution
12  U = S(:,1:end-1);
13  beta = S(:,end);
14  x = backsub(U, beta);
15 end
```

Exercise. Rewrite Lines 6–9 without using a loop. (Think *vectorized*!)

G.E. with Partial Pivoting: Procedure

In this variation of G.E., reduction to echelon form is done slightly differently.

- On the augmented matrix $[A \mid \mathbf{b}]$,

Key Process (partial pivoting)

1. Find the entry in the first column with the largest absolute value. This entry is called the *pivot*.
2. Perform a row interchange, if necessary, so that the pivot is on the first diagonal position.
3. Use elementary row operations to reduce the remaining entries in the first column to zero.

- Once done, ignore the first row and first column and repeat the **Key Process** on the remaining submatrix.
- Continue this until the matrix is in a row-echelon form.

G.E. with Partial Pivoting: Example

Let's solve the example on p. 41 again, now using G.E. with partial pivoting.

1st column:

$$\left[\begin{array}{ccc|c} 2 & 2 & 1 & 6 \\ -4 & 6 & 1 & -8 \\ 5 & -5 & 3 & 4 \end{array}\right] \xrightarrow{\text{pivot}} \left[\begin{array}{ccc|c} 5 & -5 & 3 & 4 \\ -4 & 6 & 1 & -8 \\ 2 & 2 & 1 & 6 \end{array}\right] \xrightarrow{\text{zero}} \left[\begin{array}{ccc|c} 5 & -5 & 3 & 4 \\ 0 & 2 & 3.4 & -4.8 \\ 0 & 4 & -0.2 & 4.4 \end{array}\right]$$

2nd column:

$$\left[\begin{array}{ccc|c} 5 & -5 & 3 & 4 \\ 0 & 2 & 3.4 & -4.8 \\ 0 & 4 & -0.2 & 4.4 \end{array}\right] \xrightarrow{\text{pivot}} \left[\begin{array}{ccc|c} 5 & -5 & 3 & 4 \\ 0 & 4 & -0.2 & 4.4 \\ 0 & 2 & 3.4 & -4.8 \end{array}\right] \xrightarrow{\text{zero}} \left[\begin{array}{ccc|c} 5 & -5 & 3 & 4 \\ 0 & 4 & -0.2 & 4.4 \\ 0 & 0 & 3.5 & -7 \end{array}\right]$$

Now that the last matrix is upper triangular, we work up from the third equation to the second to the first and obtain the same solution as before.

G.E. with Partial Pivoting: MATLAB Implementation

Exercise

Write a MATLAB function `GEpp.m` which carries out G.E. with partial pivoting.

- Modify `GEnp.m` on p. 43 to incorporate partial pivoting.
- The only part that needs to be changed is the for-loop starting at Line 5.
  - Right after `for j = 1:n-1`, find the index of the pivot element of the jth column of A below the diagonal.

```
[~, iM] = max(abs(A(j:end,j)));
iM = iM + j - 1;
```

  - If the pivot element is not on the diagonal, swap rows so that it is on the diagonal.

```
if j ~= iM
    S([j iM], :) = S([iM j], :)
end
```

Why Is Pivoting Necessary?

Example

Given $\epsilon \ll 1$, solve the system

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

using Gaussian elimination with and <u>without partial pivoting</u>.

Without pivoting: By $R_2 \to R_2 + (1/\epsilon)R_1$, we have

$$\begin{bmatrix} -\epsilon & 1 \\ 0 & -1+1/\epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1-\epsilon \\ 1/\epsilon - 1 \end{bmatrix} \implies \begin{cases} x_2 = 1, \\ x_1 = \dfrac{(1-\epsilon)-1}{-\epsilon}. \end{cases}$$

- In exact arithmetic, this yields the correct solution.
- In floating-point arithmetic, calculation of x_1 suffers from catastrophic cancellation.

Why Is Pivoting Necessary? (Cont')

Example

Given $\epsilon \ll 1$, solve the system

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

using Gaussian elimination with and without partial pivoting.

With partial pivoting: First, swap the rows $R_1 \leftrightarrow R_2$, and then do $R_2 \to R_2 + \epsilon R_1$ to obtain

$$\begin{bmatrix} 1 & -1 \\ 0 & 1-\epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1-\epsilon \end{bmatrix} \quad \Longrightarrow \quad \begin{cases} x_2 = 1, \\ x_1 = \dfrac{0-(-1)}{1}. \end{cases}$$

- Each of the arithmetic steps (to compute x_1, x_2) is well-conditioned.
- The solution is computed stably.

Emulation of Gaussian Elimination

In this section, we emulate row operations steps required in Gaussian elimination by matrix multiplications. **Two major operations.**

- Row interchange $R_i \leftrightarrow R_j$:

$$P(i,j)A, \quad \text{where } P(i,j) \text{ is an elementary permutation matrix.}$$

- Row replacement $R_i \to R_i + cR_j$:

$$(I + c\mathbf{e}_i\mathbf{e}_j^{\mathrm{T}})A$$

See Appendix for more details.

Key Example Revisited

Let's work out the key example from last time once again, now in matrix form $A\mathbf{x} = \mathbf{b}$.

$$\begin{bmatrix} 2 & 2 & 1 \\ -4 & 6 & 1 \\ 5 & -5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -8 \\ 4 \end{bmatrix}.$$

[Pivot] Switch R_1 and R_3 using $P(1,3)$:

$$\underbrace{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{P(1,3)} \left[\begin{bmatrix} 2 & 2 & 1 \\ -4 & 6 & 1 \\ 5 & -5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -8 \\ 4 \end{bmatrix} \right] \longrightarrow \begin{bmatrix} 5 & -5 & 3 \\ -4 & 6 & 1 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -8 \\ 6 \end{bmatrix}$$

[Zero] Do row operations $R_2 \to R_2 + (4/5)R_1$ and $R_3 \to R_3 - (2/5)R_1$:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 4/5 & 1 & 0 \\ -2/5 & 0 & 1 \end{bmatrix}}_{G_1} \left[\begin{bmatrix} 5 & -5 & 3 \\ -4 & 6 & 1 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -8 \\ 6 \end{bmatrix} \right]$$

$$\longrightarrow \begin{bmatrix} 5 & -5 & 3 \\ 0 & 2 & 3.4 \\ 0 & 4 & -0.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -4.8 \\ 4.4 \end{bmatrix}$$

Key Example Revisited (cont')

[Pivot] Switch R_2 and R_3 using $P(2,3)$:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{P(2,3)} \left[\begin{bmatrix} 5 & -5 & 3 \\ 0 & 2 & 3.4 \\ 0 & 4 & -0.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -4.8 \\ 4.4 \end{bmatrix} \right]$$

$$\longrightarrow \begin{bmatrix} 5 & -5 & 3 \\ 0 & 4 & -0.2 \\ 0 & 2 & 3.4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 4.4 \\ -4.8 \end{bmatrix}$$

[Zero] Do a row operation $R_3 \to R_3 - (1/2)R_2$:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/2 & 1 \end{bmatrix}}_{G_2} \left[\begin{bmatrix} 5 & -5 & 3 \\ 0 & 4 & -0.2 \\ 0 & 2 & 3.4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 4.4 \\ -4.8 \end{bmatrix} \right]$$

$$\longrightarrow \underbrace{\begin{bmatrix} 5 & -5 & 3 \\ 0 & 4 & -0.2 \\ 0 & 0 & 3.5 \end{bmatrix}}_{U} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 4.4 \\ -7 \end{bmatrix}$$

Analysis of Example

- The previous calculations can be summarized as

$$G_2 P(2,3) G_1 P(1,3) A = U \,. \qquad (\star)$$

- Using the noted properties of permutation matrices and GTMs, $(\star)$ can be written as

$$\begin{aligned} G_2 P(2,3) G_1 \underbrace{P(2,3)P(2,3)}_{=I} P(1,3) A = U \\ \longrightarrow G_2 \underbrace{P(2,3) G_1 P(2,3)}_{=:\widetilde{G}_1} \underbrace{P(2,3)P(1,3)}_{=:P} A = U \,. \end{aligned}$$

- The above can be summarized as $PA = LU$ where $L = (G_2 \widetilde{G}_1)^{-1}$ is a lower triangular matrix.

Generalization – PLU Factorization

For an arbitrary matrix $A \in \mathbb{R}^{n \times n}$, the partial pivoting and row operations are intermixed as

$$G_{n-1}P(n-1, r_{n-1}) \cdots G_2 P(2, r_2) G_1 P(1, r_1) A = U \,.$$

Going through the same calculations as above, it can always be written as

$$\left(\tilde{G}_{n-1} \cdots \tilde{G}_2 \tilde{G}_1\right) P(n-1, r_{n-1}) \cdots P(2, r_2) P(1, r_1) A = U \,,$$

which again leads to $PA = LU$:

$$\underbrace{P(n-1, r_{n-1}) \cdots P(2, r_2) P(1, r_1)}_{=:P} A = \underbrace{\left(\tilde{G}_{n-1} \cdots \tilde{G}_2 \tilde{G}_1\right)^{-1}}_{=:L} U \,.$$

This is called the **PLU factorization** of matrix A.

LU and PLU Factorization

If no pivoting is required, the previous procedure simplifies to

$$G_{n-1} \cdots G_2 G_1 A = U \,.$$

which leads to $A = LU$:

$$A = \underbrace{(G_{n-1} \cdots G_2 G_1)^{-1}}_{=:L} U \,.$$

This is called the **LU factorization** of matrix A.

Implementation of LU Factorization

```
function [L,U] = mylu(A)
% MYLU   LU factorization (demo only--not stable!).
% Input:
%   A    square matrix
% Output:
%   L,U  unit lower triangular and upper triangular such that
     LU=A
  n = length(A);
  L = eye(n);    % ones on diagonal
  % Gaussian elimination
  for j = 1:n-1
    for i = j+1:n
      L(i,j) = A(i,j) / A(j,j);    % row multiplier
      A(i,j:n) = A(i,j:n) - L(i,j)*A(j,j:n);
    end
  end
  U = triu(A);
end
```

Implementation of LU Factorization

Exercise. Write a MATLAB function `myplu` for PLU factorization by modifying the previous function `mylu.m`.

```
function [L,U,P] = myplu(A)
% MYPLU   PLU factorization (demo only--not stable!).
% Input:
%   A    square matrix
% Output:
%   P,L,U  permutation, unit lower triangular, and upper
   triangular such that LU=PA

% Your code here.




end
```

Solving a Square System Using PLU Factorization

Multiplying $A\mathbf{x} = \mathbf{b}$ on the left by P we obtain

$$\underbrace{PA}_{=LU}\mathbf{x} = \underbrace{P\mathbf{b}}_{=:\boldsymbol{\beta}} \quad \longrightarrow \quad LU\mathbf{x} = \boldsymbol{\beta}\,,$$

which can be solved in two steps:

- Define $U\mathbf{x} = \mathbf{y}$ and solve for $\mathbf{y}$ in the equation

$$L\mathbf{y} = \boldsymbol{\beta}\,. \qquad \text{(forward elimination)}$$

- Having calculated $\mathbf{y}$, solve for $\mathbf{x}$ in the equation

$$U\mathbf{x} = \mathbf{y}\,. \qquad \text{(backward substitution)}$$

Solving a Square System Using PLU Factorization

- Using the instructional codes (`backsub`, `forelim`, `myplu`):

```
[L,U,P] = myplu(A);
x = backsub( U, forelim(L, P*b) );
```

- Using MATLAB's built-in functions:

```
[L,U,P] = lu(A);
x = U \ (L \ (P*b));
```

  - The backslash is designed so that triangular systems are solved with the appropriate substitution.

- The most compact way:

```
x = A \ b;
```

  - The backslash does partial pivoting and triangular substitutions silently and automatically.

Analysis

Notation: Big-O and Asymptotic

Let f, g be positive functions defined on $\mathbb{N}$.

- $f(n) = O\left(g(n)\right)$ ("f is *big-O* of g") as $n \to \infty$ if

$$\frac{f(n)}{g(n)} \leqslant C, \quad \text{for all sufficiently large } n.$$

- $f(n) \sim g(n)$ ("f is *asymptotic* to g") as $n \to \infty$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

Timing Vector/Matrix Operations – FLOPS

- One way to measure the "efficiency" of a numerical algorithm is to count the number of floating-point arithmetic operations (FLOPS) necessary for its execution.
- The number is usually represented by $\sim cn^p$ where c and p are given explicitly.
- We are interested in this formula when n is large.

FLOPS for Major Operations

Vector/Matrix Operations

Let $x, y \in \mathbb{R}^n$ and $A, B \in \mathbb{R}^{n \times n}$. Then

- (vector-vector) $x^{\mathrm{T}} y$ requires $\sim 2n$ *flops*.
- (matrix-vector) Ax requires $\sim 2n^2$ *flops*.
- (matrix-matrix) AB requires $\sim 2n^3$ *flops*.

Cost of PLU Factorization

Note that we only need to count the number of *flops* required to zero out elements below the diagonal of each column.

- For each $i > j$, we replace R_i by $R_i + cR_j$ where $c = -a_{i,j}/a_{j,j}$. This requires approximately $2(n - j + 1)$ *flops*:
  - 1 division to form c
  - $n - j + 1$ multiplications to form cR_j
  - $n - j + 1$ additions to form $R_i + cR_j$
- Since $i \in \mathbb{N}[j + 1, n]$, the total number of *flops* needed to zero out all elements below the diagonal in the jth column is approximately $2(n - j + 1)(n - j)$.
- Summing up over $j \in \mathbb{N}[1, n - 1]$, we need about $(2/3)n^3$ *flops*:

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

Cost of Forward Elimination and Backward Substitution

Forward Elimination

- The calculation of $y_i = \beta_i - \sum_{j=1}^{i-1} \ell_{ij} y_j$ for $i > 1$ requires approximately $2i$ *flops*:
  - 1 subtraction
  - $i - 1$ multiplications
  - $i - 2$ additions
- Summing over all $i \in \mathbb{N}[2, n]$, we need about n^2 *flops*:

$$\sum_{i=2}^{n} 2i \sim 2\frac{n^2}{2} = n^2 .$$

Backward Substitution

- The cost of backward substitution is also approximately n^2 *flops*, which can be shown in the same manner.

Cost of G.E. with Partial Pivoting

Gaussian elimination with partial pivoting involves three steps:

- PLU factorization: $\sim (2/3)n^3$ *flops*
- Forward elimination: $\sim n^2$ *flops*
- Backward substitution: $\sim n^2$ *flops*

Summary

The total cost of Gaussian elimination with partial pivoting is approximately

$$\frac{2}{3}n^3 + n^2 + n^2 \sim \frac{2}{3}n^3$$

flops for large n.

Application: Solving Multiple Square Systems Simultaneously

To solve two systems $A\mathbf{x}_1 = \mathbf{b}_1$ and $A\mathbf{x}_2 = \mathbf{b}_2$.

Method 1.

- Use G.E. for both.
- It takes $\sim (4/3)n^3$ *flops*.

```
%% method 1
x1 = A \ b1;
x2 = A \ b2;
```

Method 2.

- Do it in two steps:
  1. Do PLU factorization $PA = LU$.
  2. Then solve $LU\mathbf{x}_1 = P\mathbf{b}_1$ and $LU\mathbf{x}_2 = P\mathbf{b}_2$.
- It takes $\sim (2/3)n^3$ *flops*.

```
%% method 2
[L, U, P] = lu(A);
x1 = U \ (L \ (P*b1));
x2 = U \ (L \ (P*b2));
```

```
%% compact implementation
X = A \ [b1, b2];
x1 = X(:, 1);
x2 = X(:, 2);
```

Further Analysis

Vector Norms

The "length" of a vector $\mathbf{v}$ can be measured by its **norm**.

Definition 2 (p-Norm of a Vector)

Let $p \in [1, \infty)$. The p-norm of $\mathbf{v} \in \mathbb{R}^m$ is denoted by $\|\mathbf{v}\|_p$ and is defined by

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{m} |v_i|^p \right)^{1/p} .$$

When $p = \infty$,

$$\|\mathbf{v}\|_\infty = \max_{1 \leqslant i \leqslant m} |v_i| .$$

The most commonly used p values are 1, 2, and ∞:

$$\|\mathbf{v}\|_1 = \sum_{i=1}^{m} |v_i| , \quad \|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{m} |v_i|^2}.$$

Vector Norms

In general, any function $\|\cdot\| : \mathbb{R}^m \to \mathbb{R}^+ \cup \{0\}$ is called a **vector norm** if it satisfies the following three properties:

1. $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = 0$.
2. $\|\alpha\mathbf{x}\| = |\alpha| \, \|\mathbf{x}\|$ for any constant α and any $\mathbf{x} \in \mathbb{R}^m$.
3. $\|\mathbf{x}+\mathbf{y}\| \leqslant \|\mathbf{x}\| + \|\mathbf{y}\|$ for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$. This is called the *triangle inequality*.

Unit Vectors

- A vector $\mathbf{u}$ is called a **unit vector** if $\|\mathbf{u}\| = 1$.
- Depending on the norm used, unit vectors will be different.
- For instance:

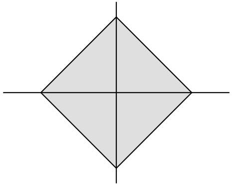


Figure 1: 1-norm

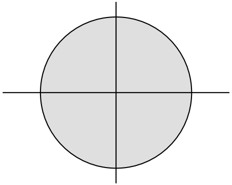


Figure 2: 2-norm

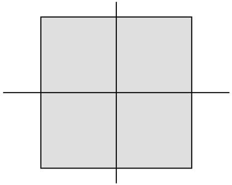


Figure 3: ∞-norm

Matrix Norms

The "size" of a matrix $A \in \mathbb{R}^{m \times n}$ can be measured by its **norm** as well. As above, we say that a function $\|\cdot\| : \mathbb{R}^{m \times n} \to \mathbb{R}^{+} \cup \{0\}$ is a **matrix norm** if it satisfies the following three properties:

1. $\|A\| = 0$ if and only if $A = 0$.
2. $\|\alpha A\| = |\alpha| \, \|A\|$ for any constant α and any $A \in \mathbb{R}^{m \times n}$.
3. $\|A + B\| \leqslant \|A\| + \|B\|$ for any $A, B \in \mathbb{R}^{m \times n}$. This is called the *triangle inequality*.

Matrix Norms (Cont')

- If, in addition to satisfying the three conditions, it satisfies

$$\|AB\| \leqslant \|A\| \, \|B\| \quad \text{for all } A \in \mathbb{R}^{m\times n} \text{ and all } B \in \mathbb{R}^{n\times p},$$

  it is said to be **consistent**.

- If, in addition to satisfying the three conditions, it satisfies

$$\|A\mathbf{x}\| \leqslant \|A\| \, \|\mathbf{x}\| \quad \text{for all } A \in \mathbb{R}^{m\times n} \text{ and all } \mathbf{x} \in \mathbb{R}^{n},$$

  then we say that it is **compatible** with a vector norm.

Induced Matrix Norms

Definition 3 (p-Norm of a Matrix)

Let $p \in [1, \infty]$. The p-norm of $A \in \mathbb{R}^{m \times n}$ is given by

$$\|A\|_p = \max_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|_p}{\|\mathbf{x}\|_p} = \max_{\|\mathbf{x}\|_p = 1} \|A\mathbf{x}\|_p .$$

- The definition of this particular matrix norm is **induced** from the vector p-norm.
- By construction, matrix p-norm is a compatible norm.
- Induced norms describe how the matrix stretches unit vectors with respect to the vector norm.

Induced Matrix Norms

The commonly used p-norms (for $p = 1, 2, \infty$) can also be calculated by

$$
\begin{aligned}
\|A\|_1 &= \max_{1 \leqslant j \leqslant n} \sum_{i=1}^{m} |a_{ij}|, \\
\|A\|_2 &= \sqrt{\lambda_{\max}(A^{\mathrm{T}}A)} = \sigma_{\max}(A), \\
\|A\|_\infty &= \max_{1 \leqslant i \leqslant m} \sum_{j=1}^{n} |a_{ij}|.
\end{aligned}
$$

In words,

- The 1-norm of A is the maximum of the 1-norms of all column vectors.
- The 2-norm of A is the square root of the largest eigenvalue of $A^{\mathrm{T}}A$.
- The ∞-norm of A is the maximum of the 1-norms of all row vectors.

Non-Induced Matrix Norm – Frobenius Norm

Definition 4 (Frobenius Norm of a Matrix)

The Frobenius norm of $A \in \mathbb{R}^{m\times n}$ is given by

$$\|A\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2}.$$

- This is not induced from a vector p-norm.
- However, both p-norm and the Frobenius norm are consistent and compatible.
- For compatibility of the Frobenius norm, the vector norm must be the 2-norm, that is, $\|A\mathbf{x}\|_2 \leqslant \|A\|_F \|\mathbf{x}\|_2$.

Norms in MATLAB

- Vector p-norms can be easily computed:

```
norm(v, 1)        % = sum(abs(v))
norm(v, 2)        % = sqrt(v'*v)  if v is a column
norm(v, 'inf')    % = max(abs(v))
```

- The same function `norm` is used to calculate matrix p-norms:

```
norm(A, 1)         % = max(sum(abs(A), 1))
norm(A, 2)         % = max(sqrt(eig(A'*A)))
norm(A, Inf)       % = max(sum(abs(A), 2))
```

- To calculate the Frobenius norm:

```
norm(A, 'fro')     % = sqrt(A(:)'*A(:))
                   % = norm(A(:), 2)
```

Conditioning of Solving Linear Systems: Overview

- Analyze how robust (or sensitive) the solutions of $A\mathbf{x} = \mathbf{b}$ are to perturbations of A and $\mathbf{b}$.
- For simplicity, consider separately the cases where

1. $\mathbf{b}$ changes to $\mathbf{b} + \delta\mathbf{b}$, while A remains unchanged, that is

$$A\mathbf{x} = \mathbf{b} \quad \longrightarrow \quad A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}.$$

2. A changes to $A + \delta A$, while $\mathbf{b}$ remains unchanged, that is

$$A\mathbf{x} = \mathbf{b} \quad \longrightarrow \quad (A + \delta A)(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b}.$$

Sensitivity to Perturbation of RHS

Case 1. $A\mathbf{x} = \mathbf{b} \;\rightarrow\; A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$

- Bound $\|\delta\mathbf{x}\|$ in terms of $\|\delta\mathbf{b}\|$:

$$\begin{aligned} A\mathbf{x} + A\delta\mathbf{x} &= \mathbf{b} + \delta\mathbf{b} \\ A\delta\mathbf{x} &= \delta\mathbf{b} \\ \delta\mathbf{x} &= A^{-1}\delta\mathbf{b} \end{aligned} \qquad \Longrightarrow \qquad \|\delta\mathbf{x}\| \leqslant \|A^{-1}\|\|\delta\mathbf{b}\|.$$

- Sensitivity in terms of relative errors:

$$\frac{\dfrac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|}}{\dfrac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}} = \frac{\|\delta\mathbf{x}\|\,\|\mathbf{b}\|}{\|\delta\mathbf{b}\|\,\|\mathbf{x}\|} \leqslant \frac{\|A^{-1}\|\,\|\delta\mathbf{b}\| \cdot \|A\|\,\|\mathbf{x}\|}{\|\delta\mathbf{b}\|\,\|\mathbf{x}\|} = \|A^{-1}\|\|A\|.$$

Sensitivity to Perturbation of Matrix

Case 2. $A\mathbf{x} = \mathbf{b} \;\rightarrow\; (A + \delta A)(\mathbf{x} + \delta \mathbf{x}) = \mathbf{b}$

- Bound $\|\delta \mathbf{x}\|$ now in terms of $\|\delta A\|$:

$$\begin{aligned} A\mathbf{x} + A\delta\mathbf{x} + (\delta A)\mathbf{x} + (\delta A)\delta\mathbf{x} &= \mathbf{b} \\ A\delta\mathbf{x} &= -(\delta A)\mathbf{x} - (\delta A)\delta\mathbf{x} \\ \delta\mathbf{x} &= -A^{-1}(\delta A)\mathbf{x} - A^{-1}(\delta A)\delta\mathbf{x} \end{aligned} \quad\Longrightarrow\quad \|\delta\mathbf{x}\| \lesssim \|A^{-1}\| \|\delta A\| \, \|\mathbf{x}\| \, .$$

(first-order truncation)

- Sensitivity in terms of relative errors:

$$\frac{\dfrac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|}}{\dfrac{\|\delta A\|}{\|A\|}} = \frac{\|\delta\mathbf{x}\| \, \|A\|}{\|\delta A\| \, \|\mathbf{x}\|} \lesssim \frac{\|A^{-1}\| \|\delta A\| \, \|\mathbf{x}\| \cdot \|A\|}{\|\delta A\| \, \|\mathbf{x}\|} = \|A^{-1}\| \|A\|.$$

Matrix Condition Number

- Motivated by the previous estimations, we define the **matrix condition number** by
$$\kappa(A) = \|A^{-1}\|\|A\|,$$
where the norms can be any p-norm or the Frobenius norm.
- A subscript on κ such as 1, 2, ∞, or F(robenius) is used if clarification is needed.

Matrix Condition Number (Cont')

- We can write

$$\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leqslant \kappa(A) \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|}, \qquad \frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leqslant \kappa(A) \frac{\|\delta A\|}{\|A\|},$$

  where the second inequality is true only in the limit of infinitesimal perturbations δA.

- The matrix condition number $\kappa(A)$ is equal to the condition number of solving a linear system of equation $A\mathbf{x} = \mathbf{b}$.
- The exponent of $\kappa(A)$ in scientific notation determines the approximate number of digits of accuracy that will be lost in calculation of $\mathbf{x}$.
- Since $1 = \|I\| = \|A^{-1}A\| \leqslant \|A^{-1}\|\|A\| = \kappa(A)$, a condition number of 1 is the best we can hope for.
- If $\kappa(A) > \boxed{\text{eps}}^{-1}$, then for computational purposes the matrix is singular.

Condition Numbers in MATLAB

- Use `cond` to calculate various condition numbers:

```
cond(A)              % the 2-norm; or  cond(A, 2)
cond(A, 1)           % the 1-norm
cond(A, Inf)         % the infinity-norm
cond(A, 'fro')       % the Frobenius norm
```

- A condition number estimator (in 1-norm)

```
condest(A)           % faster than cond
```

- The fastest method to estimate the condition number is to use `linsolve` function as below:

```
[x, inv_condest] = linsolve(A, b);
fast_condest = 1/inv_condest;
```

Appendix: Row and Column Operations

Notation: Unit Basis Vectors

Throughout this tutorial, suppose $n \in \mathbb{N}$ is fixed. Let I be the $n \times n$ identity matrix and denote by $\mathbf{e}_j$ its jth column, *i.e.*,

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = \left[\begin{array}{c|c|c|c} & & & \\ \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \\ & & & \end{array} \right].$$

That is,

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \cdots, \quad \mathbf{e}_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Notation: Concatenation

Let $A \in \mathbb{R}^{n \times n}$. We can view it as a concatenation of its rows or columns as visualized below.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \left[\begin{array}{c|c|c|c} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{array}\right] = \left[\begin{array}{c} \boldsymbol{\alpha}_1^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_2^{\mathrm{T}} \\ \hline \vdots \\ \hline \boldsymbol{\alpha}_n^{\mathrm{T}} \end{array}\right].$$

Row or Column Extraction

A row or a column of A can be extracted using columns of I.

Operation	Mathematics	MATLAB
extract the ith row of A	$\mathbf{e}_i^{\mathrm{T}} A$	`A(i,:)`
extract the jth column of A	$A\mathbf{e}_j$	`A(:,j)`
extract the (i,j) entry of A	$\mathbf{e}_i^{\mathrm{T}} A\mathbf{e}_j$	`A(i,j)`

Elementary Permutation Matrices

Definition 5 (Elementary Permutation Matrix)

For $i, j \in \mathbb{N}[1, n]$ distinct, denote by $P(i, j)$ the $n \times n$ matrix obtained by interchanging the ith and jth rows of the $n \times n$ identity matrix. Such matrices are called *elementary permutation matrices*.

Example. ($n = 4$)

$$P(1,2) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad P(1,3) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \cdots$$

Notable Properties.

- $P(i, j) = P(j, i)$
- $P(i, j)^2 = I$

Row or Column Interchange

Elementary permutation matrices are useful in interchanging rows or columns.

Operation	Mathematics	MATLAB
$\boldsymbol{\alpha}_i^{\mathrm{T}} \leftrightarrow \boldsymbol{\alpha}_j^{\mathrm{T}}$	$P(i,j)A$	`A([i,j],:)=A([j,i],:)`
$\mathbf{a}_i \leftrightarrow \mathbf{a}_j$	$AP(i,j)$	`A(:,[i,j])=A(:,[j,i])`

Permutation Matrices

Definition 6 (Permutation Matrix)

A *permutation matrix* $P \in \mathbb{R}^{n \times n}$ is a square matrix obtained from the same-sized identity matrix by re-ordering of rows.

Notable Properties.

- $P^{\mathrm{T}} = P^{-1}$
- A product of *elementary permutation matrices* is a permutation matrix.

Row and Column Operations. For any $A \in \mathbb{R}^{n \times n}$,

- PA permutes the rows of A.
- AP permutes the columns of A.

Row or Column Rearrangement

Question

Let $A \in \mathbb{R}^{6\times 6}$, and suppose that it is stored in MATLAB. Rearrange rows of A by moving 1st to 2nd, 2nd to 3rd, 3rd to 5th, 4th to 6th, 5th to 4th, and 6th to 1st, that is,

$$\begin{bmatrix} \boldsymbol{\alpha}_1^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_2^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_3^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_4^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_5^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_6^{\mathrm{T}} \end{bmatrix} \longrightarrow \begin{bmatrix} \boldsymbol{\alpha}_6^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_1^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_2^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_5^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_3^{\mathrm{T}} \\ \hline \boldsymbol{\alpha}_4^{\mathrm{T}} \end{bmatrix}$$

Row or Column Rearrangement

Solution.

- Mathematically: PA where

$$P = \left[\begin{array}{c} \mathbf{e}_6^{\mathrm{T}} \\ \hline \mathbf{e}_1^{\mathrm{T}} \\ \hline \mathbf{e}_2^{\mathrm{T}} \\ \hline \mathbf{e}_5^{\mathrm{T}} \\ \hline \mathbf{e}_3^{\mathrm{T}} \\ \hline \mathbf{e}_4^{\mathrm{T}} \end{array}\right] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

- MATLAB:

```
A = A([6 1 2 5 3 4], :)
% short for  A([1 2 3 4 5 6], :) = A([6 1 2 5 3 4], :)
```

Elementary Row Operation and GTM

Let $1 \leqslant j < i \leqslant n$.

- The row operation $R_i \to R_i + cR_j$ on $A \in \mathbb{R}^{n\times n}$, for some $c \in \mathbb{R}$, can be emulated by a matrix multiplication[1]

$$(I + c\,\mathbf{e}_i\mathbf{e}_j^{\mathrm{T}})A.$$

- In the context of Gaussian elimination, the operation of introducing zeros below the jth diagonal entry can be done via

$$\underbrace{\Big(I + \sum_{i=j+1}^{n} c_{i,j}\,\mathbf{e}_i\mathbf{e}_j^{\mathrm{T}}\Big)}_{=G_j} A, \quad 1 \leqslant j < n.$$

The matrix G_j is called a *Gaussian transformation matrix* (GTM).

[1]Many linear algebra texts refer to the matrix in parentheses as an *elementary matrix*.

Elementary Row Operation and GTM (cont')

- To emulate $(I + c\mathbf{e}_i\mathbf{e}_j^{\mathrm{T}})A$ in MATLAB:

```
A(i,:) = A(i,:) + c*A(j,:);
```

- To emulate

$$G_jA = (I + \sum_{i=j+1}^{n} c_{i,j}\mathbf{e}_i\mathbf{e}_j^{\mathrm{T}})A$$

  in MATLAB:

```
for i = j+1:n
    c = ....
    A(i,:) = A(i,:) + c*A(j,:);
end
```

  This can be done without using a loop.

Analytical Properties of GTM

- GTMs are *unit* lower triangular matrices.
- The product of GTMs is another unit lower triangular matrix.
- The inverse of a GTM is also a unit lower triangular matrix.