9. Scientific Computing
Applications of Scientific Computing

Science and engineering challenges.
- Fluid dynamics.
- Seismic surveys.
- Plasma dynamics.
- Ocean circulation.
- Electronics design.
- Pharmaceutical design.
- Human genome project.
- Vehicle crash simulation.
- Global climate simulation.
- Nuclear weapons simulation.
- Molecular dynamics simulation.

Commercial applications.
- Web search.
- Financial modeling.
- Computer graphics.
- Digital audio and video.
- Natural language processing.
- Architecture walk-throughs.
- Medical diagnostics (MRI, CAT).

Common features.
- Problems tend to be continuous instead of discrete.
- Algorithms must scale to handle huge problems.
Floating Point

IEEE 754 representation.
- Used by all modern computers.
- Scientific notation, but in binary.
- Single precision: float = 32 bits.
- Double precision: double = 64 bits.

Ex. Single precision representation of $-0.453125$.

```
<table>
<thead>
<tr>
<th>sign bit</th>
<th>exponent</th>
<th>significand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>125</td>
<td>1/2 + 1/4 + 1/16 = 0.8125</td>
</tr>
</tbody>
</table>
```

```
-1 \times 2^{125 - 127} \times 1.8125 = -0.453125
```
Floating Point

**Remark.** Most real numbers are not representable, including $\pi$ and $1/10$.

**Roundoff error.** When result of calculation is not representable.

**Consequence.** Non-intuitive behavior for uninitiated.

```java
if (0.1 + 0.2 == 0.3) { // NO }
if (0.1 + 0.3 == 0.4) { // YES }
```

**Financial computing.** Calculate 9% sales tax on a 50¢ phone call.

**Banker's rounding.** Round to nearest integer, to even integer if tie.

```java
double a1 = 1.14 * 75; // 85.49999999999999
double a2 = Math.round(a1); // 85  you lost 1¢
double b1 = 1.09 * 50; // 54.50000000000001
double b2 = Math.round(b1); // 55  SEC violation (!)
```
Catastrophic Cancellation

A simple function. \[ f(x) = \frac{1 - \cos x}{x^2} \]

Goal. Plot \( f(x) \) for \(-4 \cdot 10^{-8} \leq x \leq 4 \cdot 10^{-8}\).
Catastrophic Cancellation

A simple function. \[ f(x) = \frac{1 - \cos x}{x^2} \]

Goal. Plot \( f(x) \) for \(-4 \cdot 10^{-8} \leq x \leq 4 \cdot 10^{-8}\).

IEEE 754 double precision answer
Catastrophic Cancellation

Ex. Evaluate $f_l(x)$ for $x = 1.1e-8$.

- $\text{Math.cos}(x) = 0.9999999999999998888897769753748434595763683319091796875$.
  - Nearest floating point value agrees with exact answer to 16 decimal places.

- $(1.0 - \text{Math.cos}(x)) = 1.1102e-16$
  - Inaccurate estimate of exact answer ($6.05 \cdot 10^{-17}$)

- $(1.0 - \text{Math.cos}(x)) / (x \cdot x) = 0.9175$
  - 80% larger than exact answer (about 0.5)

Catastrophic cancellation. Devastating loss of precision when small numbers are computed from large numbers, which themselves are subject to roundoff error.
Numerical Catastrophes

Ariane 5 rocket. [June 4, 1996]
- 10 year, $7 billion ESA project exploded after launch.
- 64-bit float converted to 16 bit signed int.
- Unanticipated overflow.

Vancouver stock exchange. [November, 1983]
- Index undervalued by 44%.
- Recalculated index after each trade by adding change in price.
- 22 months of accumulated truncation error.

Patriot missile accident. [February 25, 1991]
- Failed to track scud; hit Army barracks, killed 28.
- Inaccuracy in measuring time in 1/20 of a second since using 24 bit binary floating point.
Gaussian Elimination
Linear System of Equations

Linear system of equations. N linear equations in N unknowns.

\[
\begin{align*}
0 x_0 + 1 x_1 + 1 x_2 &= 4 \\
2 x_0 + 4 x_1 - 2 x_2 &= 2 \\
0 x_0 + 3 x_1 + 15 x_2 &= 36
\end{align*}
\]

\[
A = \begin{bmatrix}
0 & 1 & 1 \\
2 & 4 & -2 \\
0 & 3 & 15
\end{bmatrix}, \quad b = \begin{bmatrix}
4 \\
2 \\
36
\end{bmatrix}
\]

matrix notation: find \( x \) such that \( Ax = b \)

Fundamental problems in science and engineering.

- Chemical equilibrium.
- Linear and nonlinear optimization.
- Kirchoff's current and voltage laws.
- Hooke's law for finite element methods.
- Leontief's model of economic equilibrium.
- Numerical solutions to differential equations.
- ...
Chemical Equilibrium

Ex. Combustion of propane.

\[ x_0 C_3H_8 + x_1 O_2 \Rightarrow x_2 CO_2 + x_3 H_2O \]

Stoichiometric constraints.

- Carbon: \(3x_0 = x_2\).
- Hydrogen: \(8x_0 = 2x_3\).
- Oxygen: \(2x_1 = 2x_2 + x_3\).
- Normalize: \(x_0 = 1\).

\[ C_3H_8 + 5O_2 \Rightarrow 3CO_2 + 4H_2O \]

Remark. Stoichiometric coefficients tend to be small integers; among first hints suggesting the atomic nature of matter.
Ex. Find current flowing in each branch of a circuit.

Kirchoff's current law.

- \[ 10 = 1x_0 + 25(x_0 - x_1) + 50(x_0 - x_2). \]
- \[ 0 = 25(x_1 - x_0) + 30x_1 + 1(x_1 - x_2). \]
- \[ 0 = 50(x_2 - x_0) + 1(x_2 - x_1) + 55x_2. \]

Solution. \( x_0 = 0.2449, \ x_1 = 0.1114, \ x_2 = 0.1166. \)
Upper Triangular System of Equations

Upper triangular system.  \( a_{ij} = 0 \) for \( i > j \).

\[
\begin{align*}
2 x_0 + 4 x_1 - 2 x_2 &= 2 \\
0 x_0 + 1 x_1 + 1 x_2 &= 4 \\
0 x_0 + 0 x_1 + 12 x_2 &= 24
\end{align*}
\]

Back substitution. Solve by examining equations in reverse order.

- Equation 2: \( x_2 = 24/12 = 2 \).
- Equation 1: \( x_1 = 4 - x_2 = 2 \).
- Equation 0: \( x_0 = (2 - 4x_1 + 2x_2) / 2 = -1 \).

```java
for (int i = N-1; i >= 0; i--) {
    double sum = 0.0;
    for (int j = i+1; j < N; j++)
        sum += A[i][j] * x[j];
    x[i] = (b[i] - sum) / A[i][i];
}
```

\[
x_i = \frac{1}{a_{ii}} \left[ b_i - \sum_{j=i+1}^{N-1} a_{ij} x_j \right]
\]
Gaussian Elimination

**Gaussian elimination.**
- Among oldest and most widely used solutions.
- Repeatedly apply **row operations** to make system upper triangular.
- Solve upper triangular system by back substitution.

**Elementary row operations.**
- Exchange row $p$ and row $q$.
- Add a multiple $\alpha$ of row $p$ to row $q$.

Key invariant. Row operations preserve solutions.
Gaussian Elimination: Row Operations

Elementary row operations.

\[
\begin{align*}
0 x_0 + 1 x_1 + 1 x_2 &= 4 \\
2 x_0 + 4 x_1 - 2 x_2 &= 2 \\
0 x_0 + 3 x_1 + 15 x_2 &= 36
\end{align*}
\]

\[\downarrow\]

(interchange row 0 and 1)

\[
\begin{align*}
2 x_0 + 4 x_1 - 2 x_2 &= 2 \\
0 x_0 + 1 x_1 + 1 x_2 &= 4 \\
0 x_0 + 3 x_1 + 15 x_2 &= 36
\end{align*}
\]

\[\downarrow\]

(subtract 3x row 1 from row 2)

\[
\begin{align*}
2 x_0 + 4 x_1 - 2 x_2 &= 2 \\
0 x_0 + 1 x_1 + 1 x_2 &= 4 \\
0 x_0 + 0 x_1 + 12 x_2 &= 24
\end{align*}
\]
**Gaussian Elimination: Forward Elimination**

**Forward elimination.** Apply row operations to make upper triangular.

**Pivot.** Zero out entries below pivot $a_{pp}$.

\[
\begin{align*}
    a_{ij} &= a_{ij} - \frac{a_{ip}}{a_{pp}} a_{pj} \\
    b_i &= b_i - \frac{a_{ip}}{a_{pp}} b_p
\end{align*}
\]

```cpp
for (int i = p + 1; i < N; i++) {
    double alpha = A[i][p] / A[p][p];
    b[i] -= alpha * b[p];
    for (int j = p; j < N; j++)
        A[i][j] -= alpha * A[p][j];
}
```
Gaussian Elimination: Forward Elimination

Forward elimination. Apply row operations to make upper triangular.

Pivot. Zero out entries below pivot $a_{pp}$.

\[
\begin{bmatrix}
* & * & * & * \\
* & * & * & * \\
* & * & * & * \\
* & * & * & *
\end{bmatrix} \Rightarrow
\begin{bmatrix}
0 & * & * & * \\
0 & * & * & * \\
0 & 0 & * & * \\
0 & 0 & 0 & *
\end{bmatrix} \Rightarrow
\begin{bmatrix}
0 & 0 & * & * \\
0 & 0 & 0 & * \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

```c
for (int p = 0; p < N; p++) {
    for (int i = p + 1; i < N; i++) {
        double alpha = A[i][p] / A[p][p];
        b[i] -= alpha * b[p];
        for (int j = p; j < N; j++)
            A[i][j] -= alpha * A[p][j];
    }
}
```
Gaussian Elimination Example

\[
\begin{align*}
1 x_0 & \quad + \quad 0 x_1 & \quad + \quad 1 x_2 & \quad + \quad 4 x_3 & = & \quad 1 \\
2 x_0 & \quad + \quad -1 x_1 & \quad + \quad 1 x_2 & \quad + \quad 7 x_3 & = & \quad 2 \\
-2 x_0 & \quad + \quad 1 x_1 & \quad + \quad 0 x_2 & \quad + \quad -6 x_3 & = & \quad 3 \\
1 x_0 & \quad + \quad 1 x_1 & \quad + \quad 1 x_2 & \quad + \quad 9 x_3 & = & \quad 4
\end{align*}
\]
Gaussian Elimination Example

\[
\begin{align*}
1x_0 + 0x_1 + 1x_2 + 4x_3 &= 1 \\
0x_0 + -1x_1 + -1x_2 + -1x_3 &= 0 \\
0x_0 + 1x_1 + 2x_2 + 2x_3 &= 5 \\
0x_0 + 1x_1 + 0x_2 + 5x_3 &= 3
\end{align*}
\]
Gaussian Elimination Example

\[
\begin{align*}
1x_0 + 0x_1 + 1x_2 + 4x_3 &= 1 \\
0x_0 + -1x_1 + -1x_2 + -1x_3 &= 0 \\
0x_0 + 0x_1 + 1x_2 + 1x_3 &= 5 \\
0x_0 + 0x_1 + -1x_2 + 4x_3 &= 3
\end{align*}
\]
Gaussian Elimination Example

<table>
<thead>
<tr>
<th>Term</th>
<th>Coefficient</th>
<th>Term</th>
<th>Coefficient</th>
<th>Term</th>
<th>Coefficient</th>
<th>Term</th>
<th>Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1x_0$</td>
<td>+</td>
<td>$0x_1$</td>
<td>+</td>
<td>$1x_2$</td>
<td>+</td>
<td>$4x_3$</td>
<td>=</td>
</tr>
<tr>
<td>$0x_0$</td>
<td>+</td>
<td>$-1x_1$</td>
<td>+</td>
<td>$-1x_2$</td>
<td>+</td>
<td>$-1x_3$</td>
<td>=</td>
</tr>
<tr>
<td>$0x_0$</td>
<td>+</td>
<td>$0x_1$</td>
<td>+</td>
<td>$1x_2$</td>
<td>+</td>
<td>$1x_3$</td>
<td>=</td>
</tr>
<tr>
<td>$0x_0$</td>
<td>+</td>
<td>$0x_1$</td>
<td>+</td>
<td>$0x_2$</td>
<td>+</td>
<td>$5x_3$</td>
<td>=</td>
</tr>
</tbody>
</table>
Gaussian Elimination Example

\[
\begin{align*}
1x_0 + 0x_1 + 1x_2 + 4x_3 &= 1 \\
0x_0 + -1x_1 + -1x_2 + -1x_3 &= 0 \\
0x_0 + 0x_1 + 1x_2 + 1x_3 &= 5 \\
0x_0 + 0x_1 + 0x_2 + 5x_3 &= 8
\end{align*}
\]

\[
\begin{align*}
x_3 &= 8/5 \\
x_2 &= 5 - x_3 = 17/5 \\
x_1 &= 0 - x_2 - x_3 = -25/5 \\
x_0 &= 1 - x_2 - 4x_3 = -44/5
\end{align*}
\]
Gaussian Elimination: Partial Pivoting

**Remark.** Previous code fails spectacularly if pivot $a_{pp} = 0.$

<table>
<thead>
<tr>
<th></th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_3$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation 1</td>
<td>$1x_0$</td>
<td>$1x_1$</td>
<td>$0x_3$</td>
<td>$1$</td>
</tr>
<tr>
<td>Equation 2</td>
<td>$2x_0$</td>
<td>$2x_1$</td>
<td>$-2x_3$</td>
<td>$-2$</td>
</tr>
<tr>
<td>Equation 3</td>
<td>$0x_0$</td>
<td>$3x_1$</td>
<td>$15x_3$</td>
<td>$33$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_3$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation 1</td>
<td>$1x_0$</td>
<td>$1x_1$</td>
<td>$0x_3$</td>
<td>$1$</td>
</tr>
<tr>
<td>Equation 2</td>
<td>$0x_0$</td>
<td>$0x_1$</td>
<td>$-2x_3$</td>
<td>$-4$</td>
</tr>
<tr>
<td>Equation 3</td>
<td>$0x_0$</td>
<td>$3x_1$</td>
<td>$15x_3$</td>
<td>$33$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_3$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation 1</td>
<td>$1x_0$</td>
<td>$1x_1$</td>
<td>$0x_3$</td>
<td>$1$</td>
</tr>
<tr>
<td>Equation 2</td>
<td>$0x_0$</td>
<td>$0x_1$</td>
<td>$-2x_3$</td>
<td>$-4$</td>
</tr>
<tr>
<td>Equation 3</td>
<td>$0x_0$</td>
<td>Nan $x_1$</td>
<td>Inf $x_3$</td>
<td>Inf</td>
</tr>
</tbody>
</table>
Partial pivoting. Swap row \( p \) with the row that has largest entry in column \( p \) among rows \( i \) below the diagonal.

```java
// find pivot row
int max = p;
for (int i = p + 1; i < N; i++)
    if (Math.abs(A[i][p]) > Math.abs(A[max][p]))
        max = i;

// swap rows \( p \) and \( \text{max} \)
double t = b[p]; b[p] = b[max]; b[max] = t;
```

Q. What if pivot \( a_{pp} = 0 \) while partial pivoting?
A. System has no solutions or infinitely many solutions.
public static double[] lsolve(double[][] A, double[] b) {
    int N = b.length;

    // Gaussian elimination
    for (int p = 0; p < N; p++) {
        // partial pivot
        int max = p;
        for (int i = p+1; i < N; i++)
            if (Math.abs(A[i][p]) > Math.abs(A[max][p]))
                max = i;
        double t = b[p]; b[p] = b[max]; b[max] = t;

        // zero out entries of A and b using pivot A[p][p]
        for (int i = p+1; i < N; i++) {
            double alpha = A[i][p] / A[p][p];
            b[i] -= alpha * b[p];
            for (int j = p; j < N; j++)
                A[i][j] -= alpha * A[p][j];
        }
    }

    // back substitution
    double[] x = new double[N];
    for (int i = N-1; i >= 0; i--) {
        double sum = 0.0;
        for (int j = i+1; j < N; j++)
            sum += A[i][j] * x[j];
        x[i] = (b[i] - sum) / A[i][i];
    }
    return x;
}
Stability and Conditioning
Numerically Unstable Algorithms

**Stability.** Algorithm $f_l(x)$ for computing $f(x)$ is **numerically stable** if $f_l(x) \approx f(x+\varepsilon)$ for some small perturbation $\varepsilon$.

> Nearly the right answer to nearly the right problem.

**Ex 1.** Numerically unstable way to compute $f(x) = \frac{1 - \cos x}{x^2}$

```java
public static double fl(double x) {
    return (1.0 - Math.cos(x)) / (x * x);
}
```

- $f_l(1.1e-8) = 0.9175$.
  - true answer $\approx 1/2$.

$$f(x) = \frac{2\sin^2(x/2)}{x^2}$$

a numerically stable formula
Numerically Unstable Algorithms

**Stability.** Algorithm $f_l(x)$ for computing $f(x)$ is **numerically stable** if $f_l(x) \approx f(x+\varepsilon)$ for some small perturbation $\varepsilon$.

Ex 2. Gaussian elimination (w/o partial pivoting) can fail spectacularly.

\[ a = 10^{-17} \]

\[
\begin{align*}
ax_0 + x_1 &= 1 \\
1x_0 + 2x_1 &= 3
\end{align*}
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$x_0$</th>
<th>$x_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>no pivoting</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>partial pivoting</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>exact</td>
<td>$\frac{1}{1-2a} \approx 1$</td>
<td>$\frac{1-3a}{1-2a} \approx 1$</td>
</tr>
</tbody>
</table>

**Theorem.** Partial pivoting improves numerical stability.
Ill-Conditioned Problems

**Conditioning.** Problem is well-conditioned if \( f(x) \approx f(x+\varepsilon) \) for all small perturbation \( \varepsilon \).

Solution varies gradually as problem varies.

**Ex.** Hilbert matrix.
- Tiny perturbation to \( H_n \) makes it singular.
- Cannot solve \( H_{12} \cdot x = b \) using floating point.

\[
H_4 = \begin{bmatrix}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\
\frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\
\frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\
\frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7}
\end{bmatrix}
\]

Hilbert matrix

**Matrix condition number.** [Turing, 1948] Widely-used concept for detecting ill-conditioned linear systems.
Numerically Solving an Initial Value ODE

Lorenz attractor.

- Idealized atmospheric model to describe turbulent flow.
- Convective rolls: warm fluid at bottom, rises to top, cools off, and falls down.

Solution. No closed form solution for $x(t)$, $y(t)$, $z(t)$.

Approach. Numerically solve ODE.

\[ \frac{dx}{dt} = -10(x+y) \]
\[ \frac{dy}{dt} = -xz + 28x - y \]
\[ \frac{dz}{dt} = xy - \frac{8}{3}z \]

$x = \text{fluid flow velocity}$
$y = \nabla \text{temperature between ascending and descending currents}$
$z = \text{distortion of vertical temperature profile from linearity}$

Edward Lorenz
Euler's Method

Euler's method. [to numerically solve initial value ODE]

- Choose $\Delta t$ sufficiently small.
- Approximate function at time $t$ by tangent line at $t$.
- Estimate value of function at time $t + \Delta t$ according to tangent line.
- Increment time to $t + \Delta t$.
- Repeat.

\[
\begin{align*}
  x_{t+\Delta t} &= x_t + \Delta t \frac{dx}{dt}(x_t, y_t, z_t) \\
  y_{t+\Delta t} &= y_t + \Delta t \frac{dy}{dt}(x_t, y_t, z_t) \\
  z_{t+\Delta t} &= z_t + \Delta t \frac{dz}{dt}(x_t, y_t, z_t)
\end{align*}
\]

Advanced methods. Use less computation to achieve desired accuracy.

- 4th order Runge-Kutta: evaluate slope four times per step.
- Variable time step: automatically adjust timescale $\Delta t$.
- See COS 323.
Lorenz Attractor: Java Implementation

```java
public class Lorenz {
    public static double dx(double x, double y, double z) {
        return -10*(x - y);
    }
    public static double dy(double x, double y, double z) {
        return -x*z + 28*x - y;
    }
    public static double dz(double x, double y, double z) {
        return x*y - 8*z/3;
    }
    public static void main(String[] args) {
        double x = 0.0, y = 20.0, z = 25.0;
        double dt = 0.001;
        StdDraw.setXscale(-25, 25);
        StdDraw.setYscale(0, 50);
        while (true) {
            double xnew = x + dt * dx(x, y, z);
            double ynew = y + dt * dy(x, y, z);
            double znew = z + dt * dz(x, y, z);
            x = xnew; y = ynew; z = znew;
            StdDraw.point(x, z);
        }
    }
}
```
The Lorenz Attractor

% java Lorenz
The Lorenz Attractor

% java Lorenz

(-25, 0)

(25, 50)
Butterfly Effect

Experiment.
- Initialize $y = 20.01$ instead of $y = 20$.
- Plot original trajectory in blue, perturbed one in magenta.
- What happens?

Ill-conditioning.
- Sensitive dependence on initial conditions.
- Property of system, not of numerical solution approach.

Predictability: Does the Flap of a Butterfly’s Wings in Brazil set off a Tornado in Texas?
- Title of 1972 talk by Edward Lorenz
Stability and Conditioning

Accuracy depends on both stability and conditioning.
- Danger: apply unstable algorithm to well-conditioned problem.
- Danger: apply stable algorithm to ill-conditioned problem.
- Safe: apply stable algorithm to well-conditioned problem.

Numerical analysis. Art and science of designing numerically stable algorithms for well-conditioned problems.

Lesson 1. Some algorithms are unsuitable for floating point solutions.
Lesson 2. Some problems are unsuitable to floating point solutions.