9. 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).

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.

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.

Example. 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>0 1 1 1 1 1 0 1</td>
<td>1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Ex. Single precision representation of \(-0.453125\).

\[-1 \times 2^{125} + 127 \times 1.8125 = -0.453125\]

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.

```
if (0.1 + 0.2 == 0.3) { /* false */ }
if (0.1 + 0.3 == 0.4) { /* true */ }
```

Financial computing. Calculate 9% sales tax on a 50¢ phone call.
Banker’s rounding. Round to nearest integer, to even integer if tie.

```
double a1 = 1.14 * 75; // 85.499999999999999 // you lost 1¢
double a2 = Math.round(a1); // 85

double b1 = 1.09 * 50; // 54.500000000000005

double b2 = Math.round(b1); // 55 ← SEC violation (!)
```
Remark. Most real numbers are not representable, including π and 1/10.

Roundoff error. When result of calculation is not representable.

Consequence. Non-intuitive behavior for uninitiated.

```
if (0.1 + 0.2 == 0.3) { /* false */ }
if (0.1 + 0.3 == 0.4) { /* true */ }
```

"Floating point numbers are like piles of sand; every time you move them around, you lose a little sand and pick up a little dirt." — Brian Kernighan and P. J. Plauger

Catastrophic Cancellation

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

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

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

Ex. Evaluate \( f(1) \) for \( x = 1.1 \cdot 10^{-8} \).
- \( \text{Math.cos}(x) = 0.999999999999988897769753758434595763683319091796875 \)
  nearest floating point value agrees with exact answer to 16 decimal places.
- \( (1.0 - \text{Math.cos}(x)) = 1.1102230555511829681055569371270188986135458105447765625 \cdot 10^{-16} \)
  inaccurate estimate of exact answer (6.05 \cdot 10^{-17})
- \( (1.0 - \text{Math.cos}(x)) / (x^2) = 0.9175 \)
  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.

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

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

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.**

\[
x_0 C_3H_8 + x_1 O_2 \Rightarrow x_2 CO_2 + x_3 H_2 O
\]

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

\[
C_3H_8 + 5O_2 \Rightarrow 3CO_2 + 4H_2 O
\]

**Remark.** Stoichiometric coefficients tend to be small integers; among first hints suggesting the atomic nature of matter.
Kirchoff’s Current Law

**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. \( x_0 = 0.2449, x_1 = 0.1114, x_2 = 0.1166 \).

---

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.

---

Upper Triangular System of Equations

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

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

**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: \( x_0 = (2 - 4x_1 + 2x_2) / 2 = -1 \).

```c
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];
```

---

Gaussian Elimination: Row Operations

**Elementary row operations.**
- \( 0x_0 + 1x_1 + 1x_2 = 4 \)
- \( 2x_0 + 4x_1 - 2x_2 = 2 \)
- \( 0x_0 + 3x_1 + 15x_2 = 36 \)

**(interchange row 0 and 1)**

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

**(subtract 3x row 1 from row 2)**

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

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

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

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

\[
\begin{bmatrix}
    a_{ij} & b_i \\
    & & \vdots \\
    a_{ij} & b_i \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
    a_{ij} & b_i \\
    & & \vdots \\
    0 & 0 \\
\end{bmatrix}
\]

\[
\begin{array}{c}
\text{for} \ (\text{int} \ p = 0; \ p < N; \ p++) \\
\text{for} \ (\text{int} \ i = p + 1; \ i < N; \ i++) \\
\quad \text{double alpha} = A[i][p] / A[p][p]; \\
\quad b[i] -= \alpha A[p][i]; \\
\quad \text{for} \ (\text{int} \ j = p; \ j < N; \ j++) \\
\quad \quad A[i][j] -= \alpha A[p][j];
\end{array}
\]

Gaussian Elimination Example

1 $x_0$ + 0 $x_1$ + 1 $x_2$ + 4 $x_3$ = 1
2 $x_0$ -1 $x_1$ + 1 $x_2$ + 7 $x_3$ = 2
-2 $x_0$ + 1 $x_1$ + 0 $x_2$ + -6 $x_3$ = 3
1 $x_0$ + 1 $x_1$ + 1 $x_2$ + 9 $x_3$ = 4
Gaussian Elimination Example

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

\[
x_3 = \frac{8}{5}
\]
\[
x_2 = \frac{17}{5}
\]
\[
x_1 = \frac{-25}{5}
\]
\[
x_0 = \frac{-44}{5}
\]

Remark. Previous code fails spectacularly if pivot \( a_{pp} = 0 \).

\begin{align*}
1 x_0 &+ 1 x_1 + 0 x_3 &= 1 \\
2 x_0 &+ 2 x_1 + -2 x_3 &= -2 \\
0 x_0 &+ 3 x_1 + 15 x_3 &= 33
\end{align*}

\begin{align*}
1 x_0 &+ 1 x_1 + 0 x_3 &= 1 \\
0 x_0 &+ 0 x_1 + -2 x_3 &= -4 \\
0 x_0 &+ \text{Nan} x_1 + \text{Inf} x_3 &= \text{Inf}
\end{align*}

Gaussian Elimination Example
Gaussian Elimination: Partial Pivoting

Partial pivoting. Swap row \( p \) with the row that has largest entry in column \( p \) among rows \( i \) below the diagonal.

```java
public static double[] lsolve(double[][] A, double[] b)
{
    int N = b.length;
    // Gaussian elimination
    for (int p = 0; p < N; p++)
    {
        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 max
        double t = b[p]; b[p] = b[max]; b[max] = t;
    }
    // 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;
}
```

Q. What if pivot \( a_{pp} = 0 \) while partial pivoting?
A. System has no solutions or infinitely many solutions.

Stability and Conditioning

Numerically-Unstable Algorithms

Stability. Algorithm \( fl(x) \) for computing \( f(x) \) is numerically stable if
\( fl(x) = f(x + \epsilon) \) for some small perturbation \( \epsilon \).

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);
}
```

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

Note. Numerically stable formula:
\[
    f(x) = \frac{2 \sin^2(x/2)}{x^2}
\]
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 2.** Gaussian elimination (w/o partial pivoting) can fail spectacularly.

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

\[
\begin{array}{c|cc}
\text{Algorithm} & x_0 & x_1 \\
\hline
\text{no pivoting} & 0.0 & 1.0 \\
\text{partial pivoting} & 1.0 & 1.0 \\
\text{exact} & \frac{1}{10^{-8}} & \frac{1}{10^{-8}} \\
\end{array}
\]

**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 1.** arcos() and tan() functions.

- $\text{arccos}(0.99999991) \approx 0.000425$  $\tan(1.57078) \approx 6.12490 \times 10^5$
- $\text{arccos}(0.99999992) \approx 0.000400$  $\tan(1.57079) \approx 1.58058 \times 10^4$

**Consequence.** The following formula for computing the great circle distance between $(x_1, y_1)$ and $(x_2, y_2)$ is inaccurate for nearby points.

\[
d = 60 \arccos(\sin x_1 \sin x_2 + \cos x_1 \cos x_2 \cos(y_1 - y_2))
\]

very close to 1 when two points are close

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.

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

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

**Matrix condition number.** [Turing, 1948] Widely-used concept for detecting ill-conditioned linear systems.

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

**Approach.** Numerically solve ODE.
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.

$$x_{n+1} = x_n + \Delta t \cdot \frac{dx}{dt}(x_n, y_n, z_n)$$
$$y_{n+1} = y_n + \Delta t \cdot \frac{dy}{dt}(x_n, y_n, z_n)$$
$$z_{n+1} = z_n + \Delta t \cdot \frac{dz}{dt}(x_n, y_n, z_n)$$

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, y);
        }
    }
}
```

The Lorenz Attractor

```bash
% 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 a 1972 talk by Edward Lorenz
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 computation.  
Lesson 2. Some problems are unsuitable to floating-point computation.