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.
- Most real numbers are not representable, including \( \pi \) and 1/10.

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>0</td>
<td>11110</td>
<td>00000000000000000000000000000000</td>
</tr>
<tr>
<td>-1</td>
<td>125</td>
<td>1/2 + 1/4 + 1/16 = 0.8125</td>
</tr>
</tbody>
</table>

\( 1 \times 2^{127} \times 1.8125 = -0.453125 \)

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.

Roundoff error. When result of calculation is not representable.
Consequence. Non-intuitive behavior for uninitialized.

```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 (0)
```
Floating Point

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 \(-4 \cdot 10^{-8} \leq x \leq 4 \cdot 10^{-8}\).

Ex. Evaluate \( f_1(x) \) for \( x = 1.1 \cdot 10^{-8} \).
- \( \text{Math.cos}(x) = 0.999999999999999888977695374843459576368331901796875 \) 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 \( \cdot 10^{-17} \))
- \( (1.0 - \text{Math.cos}(x)) / (x^2) = 0.9175 \) 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.

\[
\begin{align*}
0 \times_0 + 1 \times_1 + 1 \times_2 &= 4 \\
2 \times_0 + 4 \times_1 - 2 \times_2 &= 2 \\
0 \times_0 + 3 \times_1 + 15 \times_2 &= 36 \\
\end{align*}
\]

Matrix notation: \[ \mathbf{A} \mathbf{x} = \mathbf{b} \]

**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.

\[ \text{C}_3\text{H}_8 + \text{O}_2 \rightarrow \text{CO}_2 + \text{H}_2\text{O} \]

**Stoichiometric constraints.**
- Carbon: \[ 3 \times_0 = \times_2 \]
- Hydrogen: \[ 8 \times_0 = 2 \times_3 \]
- Oxygen: \[ 2 \times_1 = 2 \times_2 + \times_3 \]
- Normalize: \[ \times_0 = 1 \]

\[ \text{C}_3\text{H}_8 + 5\text{O}_2 \rightarrow 3\text{CO}_2 + 4\text{H}_2\text{O} \]

**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 Diagram]

**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 $q$.

![Elementary Row Operations Diagram]

**Key invariant.** Row operations preserve solutions.

---

**Upper Triangular System of Equations**

**Upper triangular system.** $a_{ij} = 0$ for $i > j$.

- $2x_0 + 4x_1 - 2x_2 = 2$
- $0x_0 + 1x_1 + 1x_2 = 4$
- $0x_0 + 0x_1 + 12x_2 = 24$

**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)**

- $2x_0 + 4x_1 - 2x_2 = 2$
- $0x_0 + 1x_1 + 1x_2 = 4$
- $0x_0 + 3x_1 + 15x_2 = 36$

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

- $2x_0 + 4x_1 - 2x_2 = 2$
- $0x_0 + 1x_1 + 1x_2 = 4$
- $0x_0 + 0x_1 + 12x_2 = 24$
Gaussian Elimination: Forward Elimination

Forward elimination. Apply row operations to make upper triangular.

Pivot. Zero out entries below pivot \(a_{pp}\).

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

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

for (int \(i = p + 1; i < N; i++\)) {
    \(A[i][j] = alpha \times A[p][j]\);
}

Gaussian Elimination Example

\[
\begin{align*}
1x_0 + 0x_1 + 1x_2 + 4x_3 &= 1 \\
2x_0 + -1x_1 + 1x_2 + 7x_3 &= 2 \\
-2x_0 + 1x_1 + 0x_2 + -6x_3 &= 3 \\
1x_0 + 1x_1 + 1x_2 + 9x_3 &= 4
\end{align*}
\]
Remark. Previous code fails spectacularly if pivot $a_{pp} = 0.$
For computing \( f(x) \) numerically stable if \( f(x) = f(x+\varepsilon) \) 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 f1(double x) {
    return (1.0 - Math.cos(x)) / (x*x);
}
```

\( f(1.1e-8) = 0.9175 \).

\( f(x) = \frac{2 \sin^2(x/2)}{x^2} \)
a stable formula
Numerically Unstable Algorithms

**Stability.** Algorithm \( f_1(x) \) for computing \( f(x) \) is **numerically stable** if \( f_1(x) \approx f(x+\varepsilon) \) for **some** small perturbation \( \varepsilon \).

*Nearly the right answer to nearly the right problem.*

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

\[
\begin{array}{c|c|c}
\text{Algorithm} & \text{x}_0 & \text{x}_1 \\
\hline
\text{no pivoting} & 0.0 & 1.0 \\
\text{partial pivoting} & 1.0 & 1.0 \\
\text{exact} & \frac{1}{17} & \frac{1}{17} \\
\end{array}
\]

**Theorem.** Partial pivoting improves numerical stability.

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

\[
\begin{align*}
\frac{dx}{dt} &= -10(x+y) \\
\frac{dy}{dt} &= -xz + 28x - y \\
\frac{dz}{dt} &= xy - \frac{3}{2}z
\end{align*}
\]

- \( x \): fluid flow velocity
- \( y \): temperature between ascending and descending currents
- \( z \): distortion of vertical temperature profile from linearity

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

**Approach.** Numerically solve ODE.

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 \( H_{12} \) makes it singular.
- Cannot solve \( H_{12}x = b \) using floating point.

**Matrix condition number.** [Turing, 1948] Concept useful for detecting ill-conditioned linear systems.

Numerically Solving an Initial Value 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.**

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

**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);
        }
    }
}
```

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
```

The Lorenz Attractor

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.