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`.

sign bit	exponent	significand
1	0 1 1 1 1 1 0 1	1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-1	125	1/2 + 1/4 + 1/16 = 0.8125

bias phantom bit

$$-1 \times 2^{125 - 127} \times 1.8125 = -0.453125$$

Floating Point

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  */  }
```

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

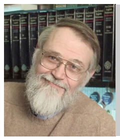
```
double a1 = 1.14 * 75;        // 85.49999999999999
double a2 = Math.round(a1);   // 85   ← you lost 1¢

double b1 = 1.09 * 50;        // 54.50000000000001
double b2 = Math.round(b1);   // 55   ← SEC violation (!)
```

Floating Point

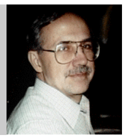
Remark. Most real numbers are not representable, including π and 1/10.

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  */  }
```



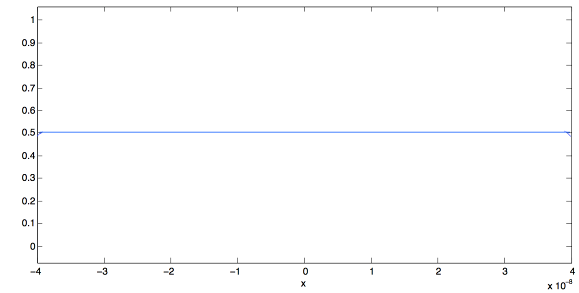
“ Floating point numbers are like piles of sand; every time you move them around, you lose a little sand and pick up a little dirt. ” — Brian Kernighan and P. J. Plauger



Catastrophic Cancellation

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

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

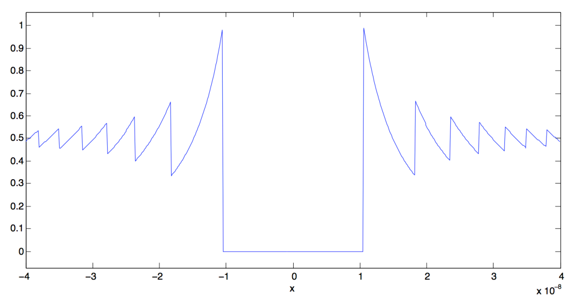


Exact answer

Catastrophic Cancellation

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

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



IEEE 754 double precision answer

Catastrophic Cancellation

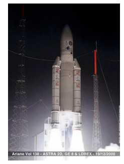
```
public static double fl(double x) {
   return (1.0 - Math.cos(x)) / (x* x);
}
```

Ex. Evaluate `fl(x)` for `x = 1.1e-8`.

- `Math.cos(x) = 0.99999999999999988897769753748434595763683319091796875.`
  nearest floating point value agrees with exact answer to 16 decimal places.
- `(1.0 - Math.cos(x)) = 1.1102e-16`
  inaccurate estimate of exact answer ($6.05 \cdot 10^{-17}$)
- `(1.0 - Math.cos(x)) / (x*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



Copyright, Arianespace

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.

$$\begin{array}{rcrcrcr} 0\,x_0 & + & 1\,x_1 & + & 1\,x_2 & = & 4 \\ 2\,x_0 & + & 4\,x_1 & - & 2\,x_2 & = & 2 \\ 0\,x_0 & + & 3\,x_1 & + & 15\,x_2 & = & 36 \end{array}$$

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

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: $x_0 = 1$.

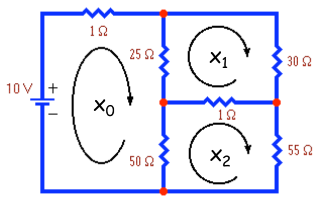
conservation of mass

$$C_3H_8 + 5O_2 \Rightarrow 3CO_2 + 4H_2O$$

Remark. Stoichiometric coefficients tend to be small integers; among first hints suggesting the atomic nature of matter.

Kirchoff's Current Law

Ex. Find current flowing in each branch of a circuit.



Kirchoff's current law.

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

conservation of electrical charge

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 $i > j$.

$$\begin{aligned} 2\,x_0 + 4\,x_1 - 2\,x_2 &= 2 \\ 0\,x_0 + 1\,x_1 + 1\,x_2 &= 4 \\ 0\,x_0 + 0\,x_1 + 12\,x_2 &= 24 \end{aligned}$$

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

```
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];
}
```

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

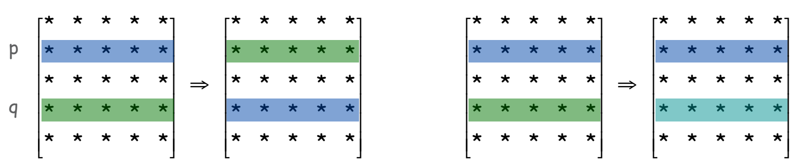
Gaussian Elimination

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 α of row p to row q.



Key invariant. Row operations preserve solutions.

Gaussian Elimination: Row Operations

Elementary row operations.

$$\begin{aligned} 0\,x_0 + 1\,x_1 + 1\,x_2 &= 4 \\ 2\,x_0 + 4\,x_1 - 2\,x_2 &= 2 \\ 0\,x_0 + 3\,x_1 + 15\,x_2 &= 36 \end{aligned}$$

(interchange row 0 and 1)

$$\begin{aligned} 2\,x_0 + 4\,x_1 - 2\,x_2 &= 2 \\ 0\,x_0 + 1\,x_1 + 1\,x_2 &= 4 \\ 0\,x_0 + 3\,x_1 + 15\,x_2 &= 36 \end{aligned}$$

(subtract 3x row 1 from row 2)

$$\begin{aligned} 2\,x_0 + 4\,x_1 - 2\,x_2 &= 2 \\ 0\,x_0 + 1\,x_1 + 1\,x_2 &= 4 \\ 0\,x_0 + 0\,x_1 + 12\,x_2 &= 24 \end{aligned}$$

Gaussian Elimination: Forward Elimination

Forward elimination. Apply row operations to make upper triangular.

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

$$a_{ij} = a_{ij} - \frac{a_{ip}}{a_{pp}} a_{pj}$$

$$b_i = b_i - \frac{a_{ip}}{a_{pp}} b_p$$

$$\begin{array}{cc} & \begin{matrix} & & p & & & \end{matrix} \\ \begin{matrix} \\ \\ p \\ \\ \\ \\ \end{matrix} & \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \end{bmatrix} \end{array} \Rightarrow \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \end{bmatrix}$$

```
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}.

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \Rightarrow \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{bmatrix} \Rightarrow \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{bmatrix} \Rightarrow \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix} \Rightarrow \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{bmatrix}$$

```
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

$1\ x_0$	+	$0\ x_1$	+	$1\ x_2$	+	$4\ x_3$	=	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

$1\ x_0$	+	$0\ x_1$	+	$1\ x_2$	+	$4\ x_3$	=	1
$0\ x_0$	+	$-1\ x_1$	+	$-1\ x_2$	+	$-1\ x_3$	=	0
$0\ x_0$	+	$1\ x_1$	+	$2\ x_2$	+	$2\ x_3$	=	5
$0\ x_0$	+	$1\ x_1$	+	$0\ x_2$	+	$5\ x_3$	=	3

Gaussian Elimination Example

$1 x_0$	+	$0 x_1$	+	$1 x_2$	+	$4 x_3$	=	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

Gaussian Elimination Example

$1 x_0$	+	$0 x_1$	+	$1 x_2$	+	$4 x_3$	=	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
$0 x_0$	+	$0 x_1$	+	$0 x_2$	+	$5 x_3$	=	8

Gaussian Elimination Example

$1 x_0$	+	$0 x_1$	+	$1 x_2$	+	$4 x_3$	=	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
$0 x_0$	+	$0 x_1$	+	$0 x_2$	+	$5 x_3$	=	8

$$
\begin{aligned}
x_3 & & &= 8/5 \\
x_2 &= 5 - x_3 & &= 17/5 \\
x_1 &= 0 - x_2 - x_3 & &= -25/5 \\
x_0 &= 1 - x_2 - 4x_3 & &= -44/5
\end{aligned}
$$

Gaussian Elimination: Partial Pivoting

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

$1 x_0$	+	$1 x_1$	+	$0 x_3$	=	1
$2 x_0$	+	$2 x_1$	+	$-2 x_3$	=	-2
$0 x_0$	+	$3 x_1$	+	$15 x_3$	=	33

$1 x_0$	+	$1 x_1$	+	$0 x_3$	=	1
$0 x_0$	+	$0 x_1$	+	$-2 x_3$	=	-4
$0 x_0$	+	$3 x_1$	+	$15 x_3$	=	33

$1 x_0$	+	$1 x_1$	+	$0 x_3$	=	1
$0 x_0$	+	$0 x_1$	+	$-2 x_3$	=	-4
$0 x_0$	+	Nan x_1	+	Inf x_3	=	Inf

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.

```
// 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 max
double[] T = A[p]; A[p] = A[max]; A[max] = T;
double   t = b[p]; b[p] = b[max]; b[max] = t;
```

$$
\begin{array}{cc}
 & \begin{array}{cccccc} & & p & & & \end{array} \\
\begin{array}{c} \\ \\ p \\ \\ max \\ \\ \end{array} &
\begin{bmatrix}
* & * & * & * & * & * \\
0 & * & * & * & * & * \\
0 & 0 & 0 & * & * & * \\
0 & 0 & 3 & * & * & * \\
0 & 0 & 9 & * & * & * \\
0 & 0 & 2 & * & * & *
\end{bmatrix}
\end{array}
$$

Q. What if pivot $a_{pp} = 0$ while partial pivoting?

A. System has no solutions or infinitely many solutions.

Gaussian Elimination with Partial Pivoting

```
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 = A[p]; A[p] = A[max]; A[max] = T;
      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;
}
```

~ $N^3/3$ additions,
~ $N^3/3$ multiplications

~ $N^2/2$ additions,
~ $N^2/2$ multiplications

Stability and Conditioning

Numerically-Unstable Algorithms

Stability. Algorithm `fl(x)` for computing f(x) is numerically stable if `fl(x)` ≈ f(x+ε) for some small perturbation ε.

Nearly the right answer to nearly the right problem.

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

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

- `fl(1.1e-8) = 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 `fl(x)` for computing f(x) is numerically stable if `fl(x)` ≈ f(x+ε) for some small perturbation ε.

Nearly the right answer to nearly the right problem.

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

$a = 10^{-17}$

$$\begin{aligned} a\,x_0 &+ 1\,x_1 = 1 \\ 1\,x_0 &+ 2\,x_1 = 3 \end{aligned}$$

Algorithm	x_0	x_1
no pivoting	0.0	1.0
partial pivoting	1.0	1.0
exact	$\frac{1}{1-2a} \approx 1$	$\frac{1-3a}{1-2a} \approx 1$

Theorem. Partial pivoting improves numerical stability.

Ill-Conditioned Problems

Conditioning. Problem is well-conditioned if f(x) ≈ f(x+ε) for all small perturbation ε.

Solution varies gradually as problem varies.

Ex 1. arccos() and tan() functions.

- arccos(.99999991) ≈ 0.000425 $\tan(1.57078) \approx 6.12490 \times 10^5$
- arccos(.99999992) ≈ 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.

$$d = 60 \arccos(\underbrace{\sin x_1 \sin x_2 + \cos x_1 \cos x_2 \cos(y_1 - y_2)}_{\text{very close to 1 when two points are close}})$$

Ill-Conditioned Problems

Conditioning. Problem is well-conditioned if f(x) ≈ f(x+ε) for all small perturbation ε.

Solution varies gradually as problem varies.

Ex 2. Hilbert matrix.

- Tiny perturbation to H_n makes it singular.
- Cannot solve $H_{12}\,x = b$ 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.

$$\begin{aligned} \frac{dx}{dt} &= -10\,(x + y) \\ \frac{dy}{dt} &= -xz + 28x - y \\ \frac{dz}{dt} &= xy - \frac{8}{3}z \end{aligned}$$

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.

Euler's Method

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

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

$$\begin{aligned} x_{t+\Delta t} &= x_t + \Delta t \, \tfrac{dx}{dt}(x_t, y_t, z_t) \\ y_{t+\Delta t} &= y_t + \Delta t \, \tfrac{dy}{dt}(x_t, y_t, z_t) \\ z_{t+\Delta t} &= z_t + \Delta t \, \tfrac{dz}{dt}(x_t, y_t, z_t) \end{aligned}$$

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 Δt.
- See COS 323.

Lorenz Attractor: Java Implementation

```
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);
      }
   }
}
```

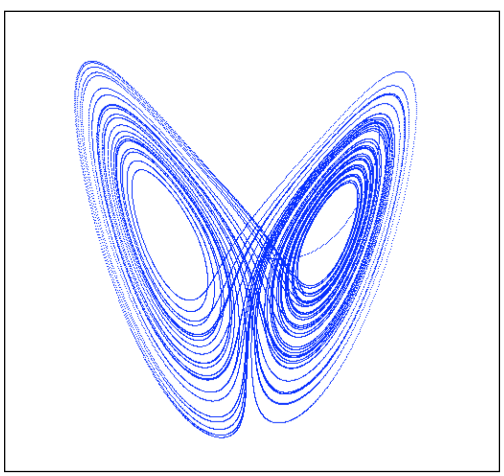
Euler's method

plot x vs. z

The Lorenz Attractor

```
% java Lorenz
```

(25, 50)



(-25, 0)

Butterfly Effect

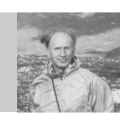
Experiment.

- 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



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 computation.
Lesson 2. Some problems are unsuitable to floating-point computation.