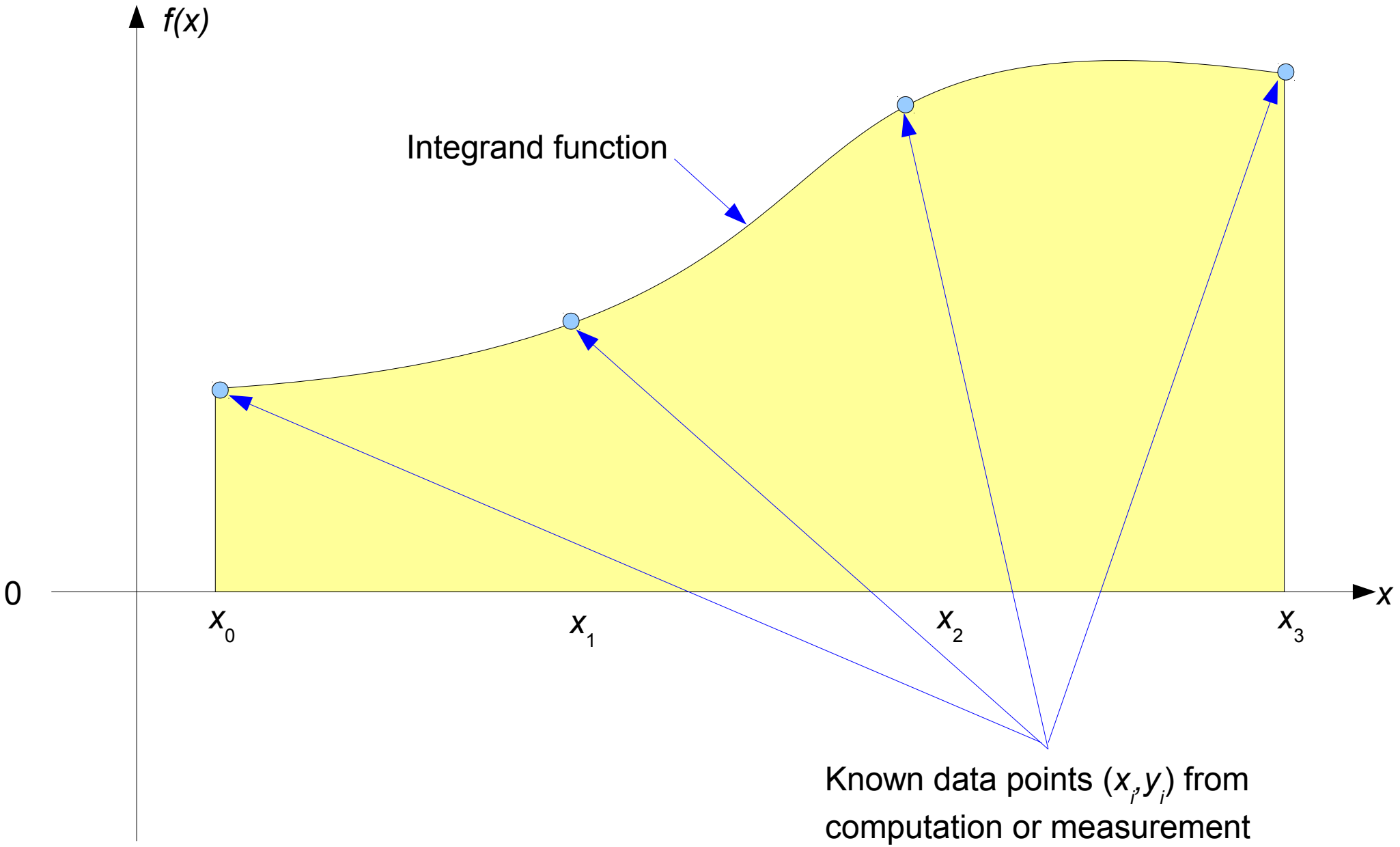
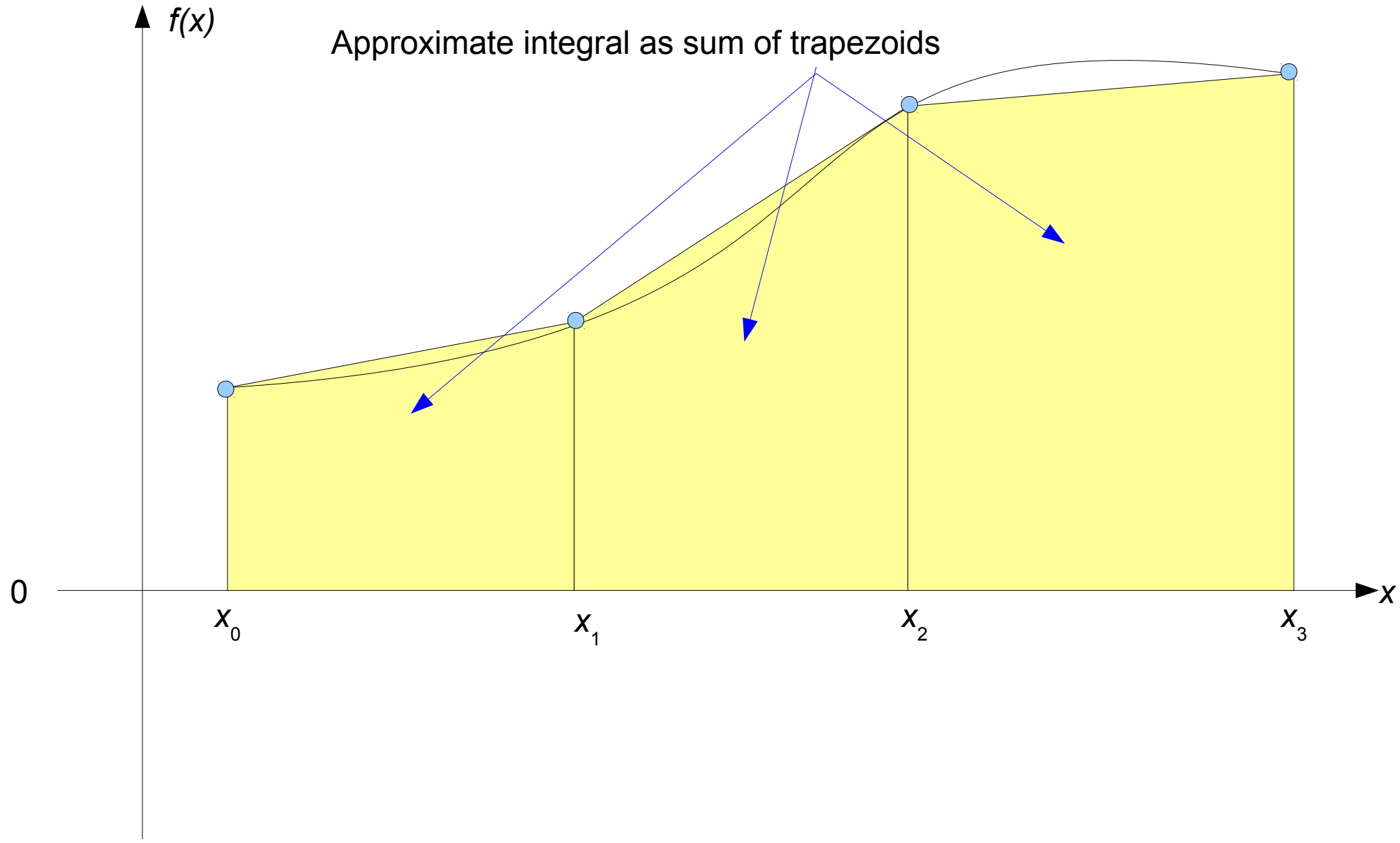
Class Progress

Basics of Linux, gnuplot, C
Visualization of numerical data
Roots of nonlinear equations
(Midterm 1)
Solutions of systems of linear equations
Solutions of systems of nonlinear equations
Monte Carlo simulation
Interpolation of sparse data points
Numerical integration
(Midterm 2)
Solutions of ordinary differential equations

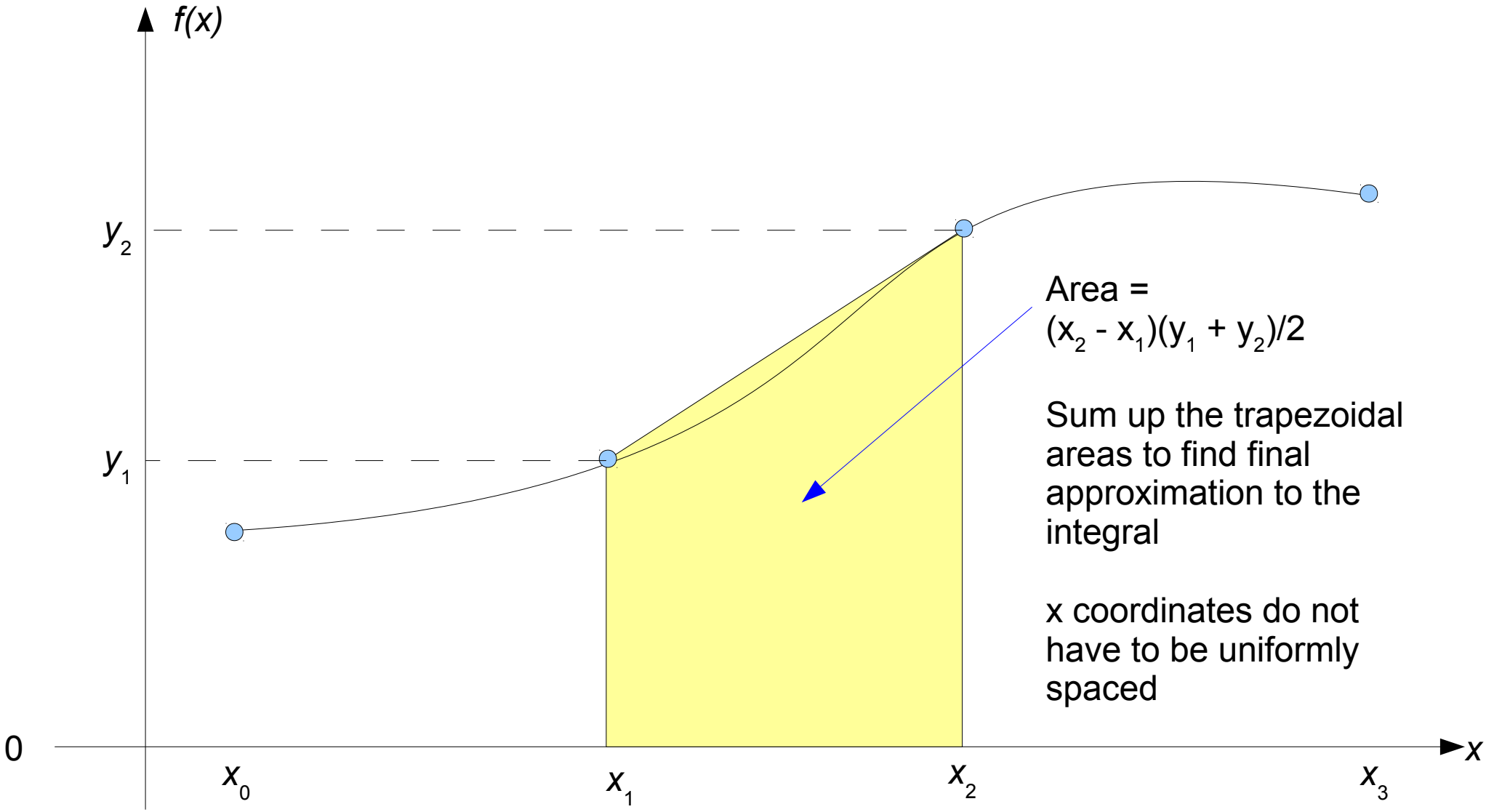
Integration of a Function from Known Data Points



Integration of a Function with Trapezoidal Rule



Sum the Area of Each Trapezoid

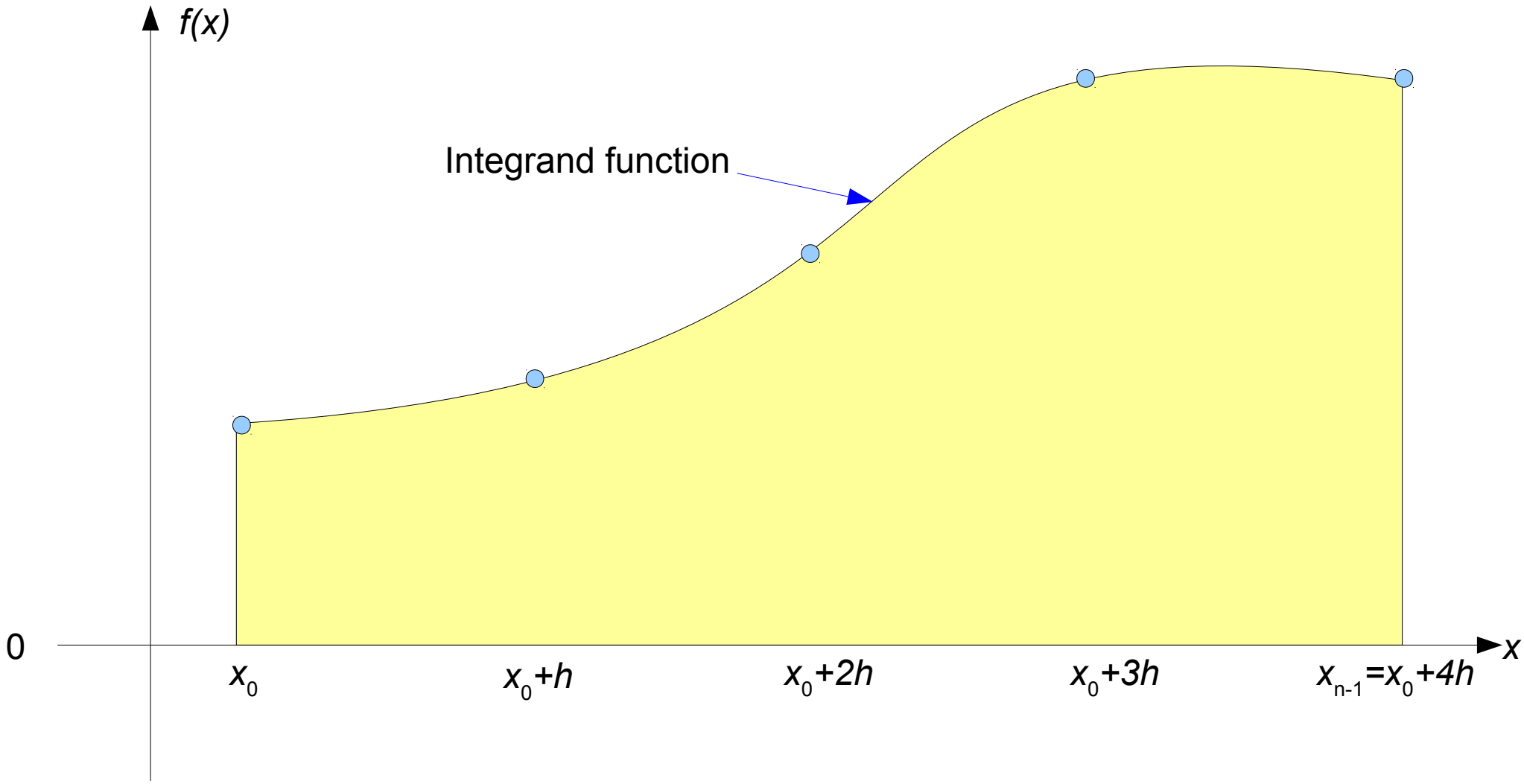


Integration C Code for Tabulated Data Points

```
int trapezoidal_data(int n,struct point2d *sample) {
  double integral;
  int i;

  integral = 0.0;
  i = 1;
  while (i < n) {
    integral += 0.5 * (sample[i].y + sample[i-1].y) * (sample[i].x -
sample[i-1].x);
    i++;
  }
  printf("%.12g\n",integral);
  return(0);
}
```

Trapezoidal Rule with Uniformly Spaced *x* Values



Define $h = x_{i+1} - x_i$ as the step size

Four steps shown in this example

Integration of a Function with Trapezoidal Rule

$$\text{Each trapezoidal area now} = \frac{h}{2}\left(y_i + y_{i+1}\right)$$

So sum up the areas of all the trapezoids:

$$\begin{aligned}
&\frac{h}{2}y_0 + \frac{h}{2}y_1 \\
&\qquad\qquad \frac{h}{2}y_1 + \frac{h}{2}y_2 \\
&\qquad\qquad\qquad\qquad \frac{h}{2}y_2 + \frac{h}{2}y_3 \\
&\qquad\qquad\qquad\qquad\qquad\qquad \cdots\cdots \\
+ &\qquad\qquad\qquad\qquad\qquad\qquad \frac{h}{2}y_{n-2} + \frac{h}{2}y_{n-1} \\
\hline
&\frac{h}{2}y_0 + h y_1 + h y_2 + \cdots + h y_{n-2} + \frac{h}{2}y_{n-1}
\end{aligned}$$

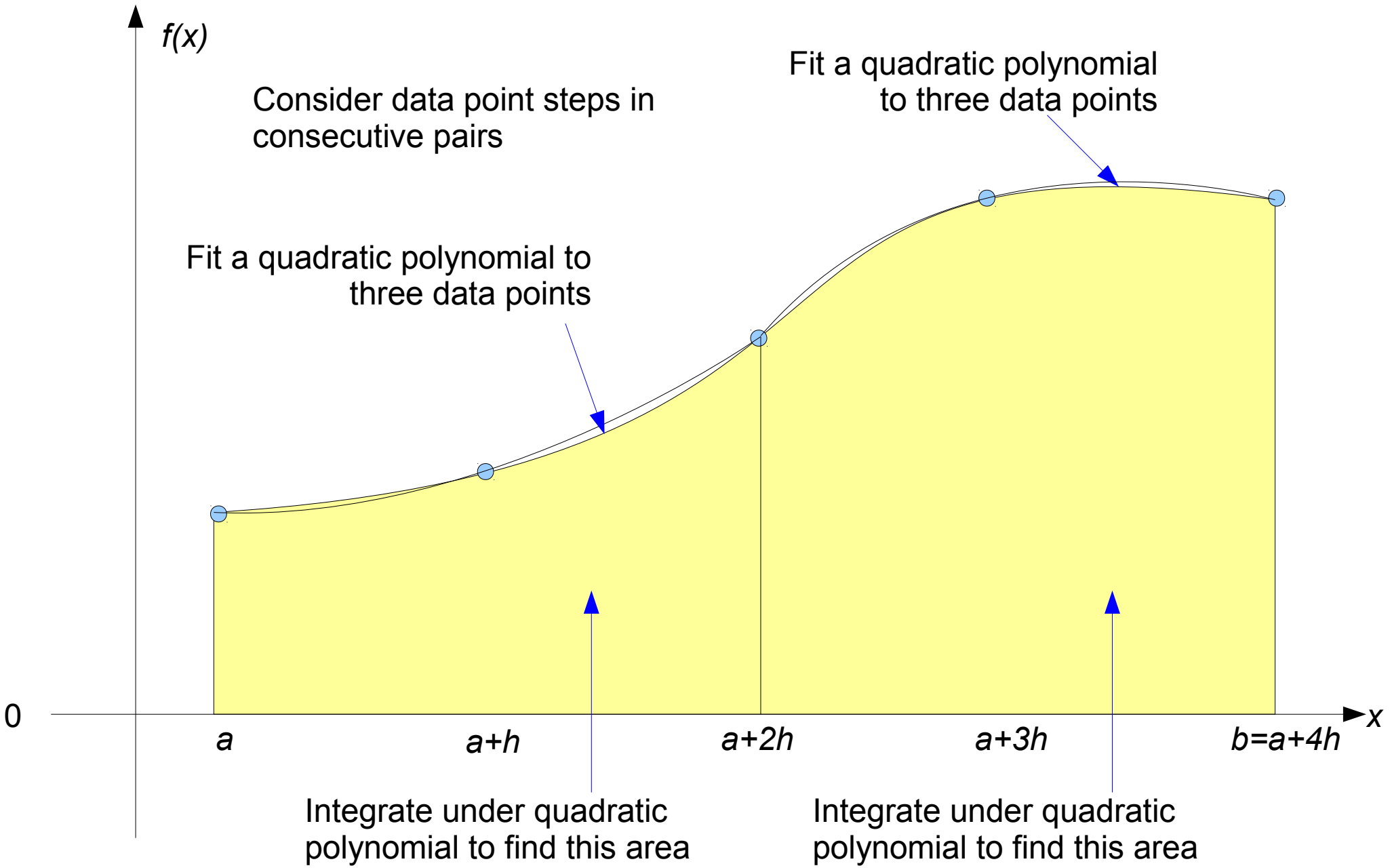
Integration of a Function with Trapezoidal Rule

For n uniformly spaced data points:

$$\int_{x_0}^{x_{n-1}} f(x)\,dx \approx h\cdot\left[\frac{(y_0 + y_{n-1})}{2} + \sum_{i=1}^{n-2} y_i\right]$$

$$\text{Error term} = -\left(\frac{1}{12}\right)(x_{n-1} - x_0)\,h^2\,f''(\xi)$$

Simpson's Rule with Uniformly Spaced *x* Values



Integration of a Function with Simpson's 1/3 Rule

Fit second order Lagrange interpolation polynomial to first three data points:

$$f(x)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}\cdot y_0+\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\cdot y_1+\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\cdot y_2$$

Change x axis so that x_0=0 x_1=h x_2=2h and rewrite Lagrange interpolation polynomial for uniformly spaced data points:

$$f(x)=\frac{(x-h)(x-2h)}{2h^2}\cdot y_0-\frac{x(x-2h)}{h^2}\cdot y_1+\frac{x(x-h)}{2h^2}\cdot y_2$$

Integrate under quadratic polynomial from 0 to 2h:

$$\int_0^{2h} f(x)\,dx \approx \left(\frac{h}{3}\right)\cdot\left[y_0+4y_1+y_2\right]$$

Integration of a Function with Simpson's 1/3 Rule

So sum up the areas under all the quadratic polynomials:

$$\frac{h}{3}y_0+\frac{4h}{3}y_1+\frac{h}{3}y_2$$

$$\frac{h}{3}y_2+\frac{4h}{3}y_3+\frac{h}{3}y_4$$

$$\frac{h}{3}y_4+\frac{4h}{3}y_5+\frac{h}{3}y_6$$

.....

$$\frac{h}{3}y_{n-3}+\frac{4h}{3}y_{n-2}+\frac{h}{3}y_{n-1}$$

+

$$\frac{h}{3}y_0+\frac{4h}{3}y_1+\frac{2h}{3}y_2+\frac{4h}{3}y_3+\frac{2h}{3}y_4+\cdots+\frac{2h}{3}y_{n-3}+\frac{4h}{3}y_{n-2}+\frac{h}{3}y_{n-1}$$

Integration of a Function with Simpson's 1/3 Rule

So for n uniformly spaced data points:

$$\int_{x_0}^{x_{n-1}} f(x)\,dx \approx \left(\frac{h}{3}\right)\cdot\left[\left(y_0 + y_{n-1}\right) + 4\cdot\sum_{i=1,\,i\ \text{odd}}^{n-2} y_i + 2\cdot\sum_{i=2,\,i\ \text{even}}^{n-3} y_i\right]$$

Applicable for only an even number of steps, so *n*-1 must be an even number, or the number of function points must be odd

$$\text{Error term} = -\left(\frac{1}{180}\right)\left(x_{n-1} - x_0\right)h^4 f^{(4)}(\xi)$$

Integration C Code for C Function

```
#include <stdio.h>
#include <stdlib.h>
#include "integrate.h"

int trapezoidal(double (*func)(double x),double xstart,double
xstop,double xinc) {
  double x,integral;

  x = xstart;
  integral = 0.5 * xinc * (*func)(x);
  x = x + xinc;
  xstop = xstop - (xinc * 0.5);
  while (((xinc > 0.0) && (x < xstop)) || ((xinc < 0.0) && (x >
xstop))) {
    integral += xinc * (*func)(x);
    x = x + xinc;
  }
  integral += 0.5 * xinc * (*func)(x);
  printf("%.8g\n",integral);
  return(0);
}
```

Note '-' sign to stop loop before the last desired point!

Integration C Code for C Function

```
int simpson(double (*func)(double x),double xstart,double xstop,double xinc) {
  double x,integral;
  int state;

  x = xstart;
  integral = xinc * (*func)(x) / 3.0;
  x = x + xinc;
  xstop = xstop - (xinc * 0.5);
  state = 0;
  while (((xinc > 0.0) && (x < xstop)) || ((xinc < 0.0) && (x > xstop))) {
    if (state == 0) {
      integral += 4.0 * xinc * (*func)(x) / 3.0;
      state = 1;
    }
    else {
      integral += 2.0 * xinc * (*func)(x) / 3.0;
      state = 0;
    }
    x = x + xinc;
  }
  if (state != 1) {
    fprintf(stderr,"Warning: Odd number of integration segments\n");
  }
  integral += xinc * (*func)(x) / 3.0;
  printf("%.8g\n",integral);
  return(0);
}
```

Integration C Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "integrate.h"
#include "constants.h"

double mu,sigma;

double gaussian(double x) {
  return(exp(-(x - mu) * (x - mu) / (2.0 * sigma * sigma)) / (sqrt(2.0 * PI) * sigma));
}

int main(int argc,char *argv[]) {
  double xstart,xstop,xinc;

  if (argc != 6) {
    fprintf(stderr,"%s <mu> <sigma> <xstart> <xstop> <xinc>\n",argv[0]);
    exit(1);
  }
  mu = atof(argv[1]);
  sigma = atof(argv[2]);
  xstart = atof(argv[3]);
  xstop = atof(argv[4]);
  xinc = atof(argv[5]);
  printf("Trapezoidal rule:\n");
  trapezoidal(gaussian,xstart,xstop,xinc);
  printf("Simpson's rule:\n");
  simpson(gaussian,xstart,xstop,xinc);
  exit(0);
}
```

Rotational Inertia

For a given object, define

$$I = \int r^2 \, dm$$

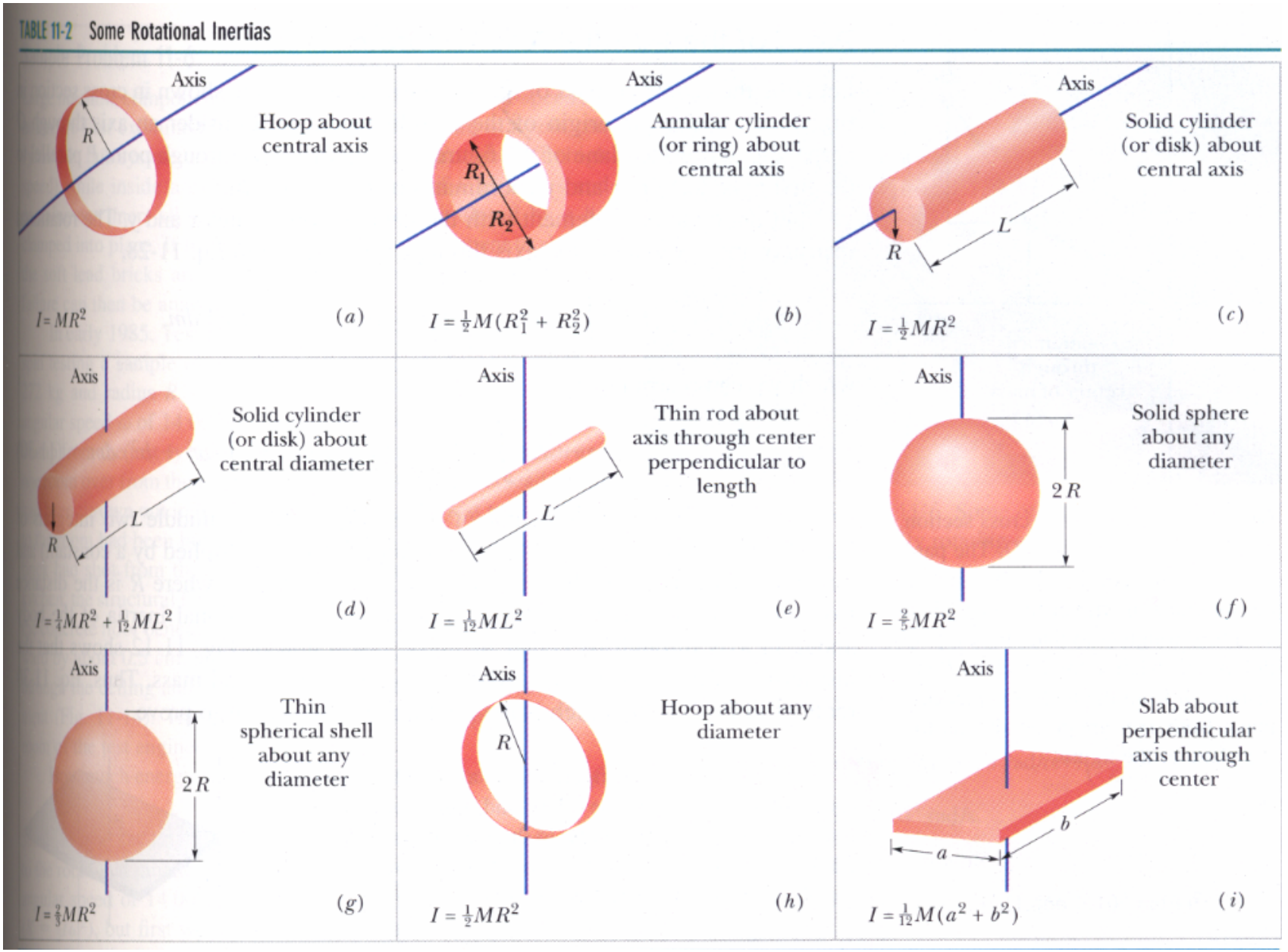
As the rotational inertia of an object about a particular axis of rotation

Then simple rotational equivalent of the Newtonian translational relationships can be derived, such as

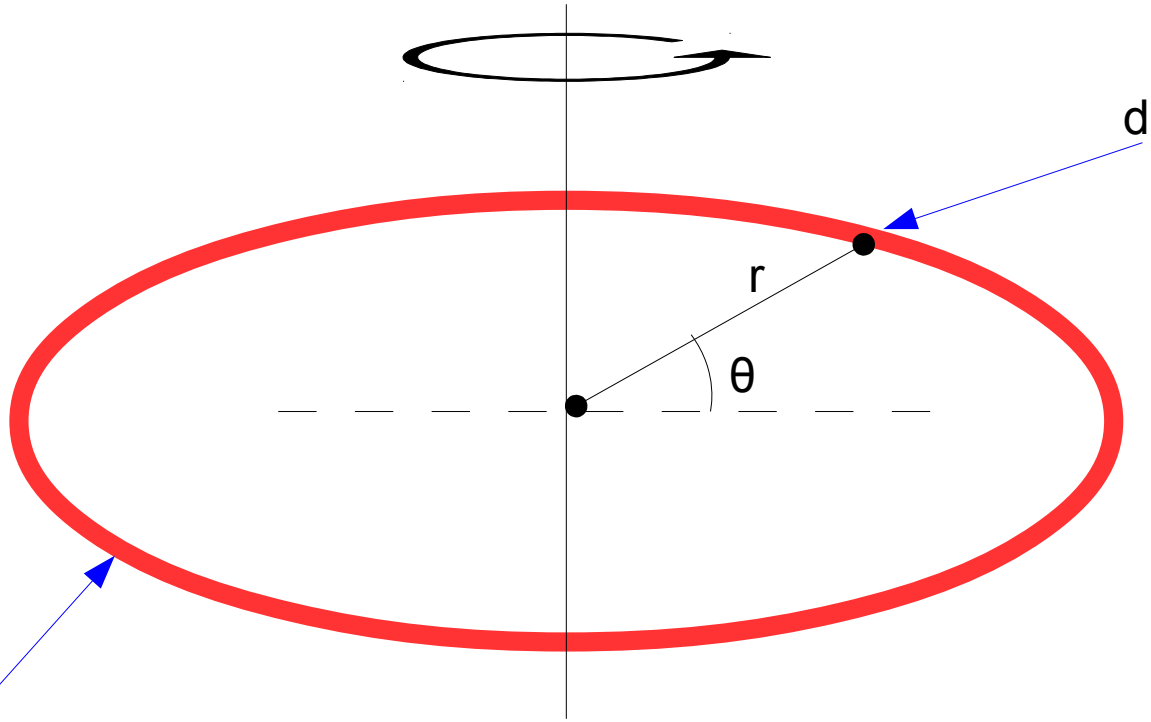
$$\text{Torque} \quad \tau = I\,\alpha$$

$$\text{Kinetic energy} \quad K = \frac{1}{2} I\,\omega^2$$

Sample Rotational Inertia Integrals



Simple Rotational Inertial Problem



Circular hoop
Total mass = M
Linear mass density = μ

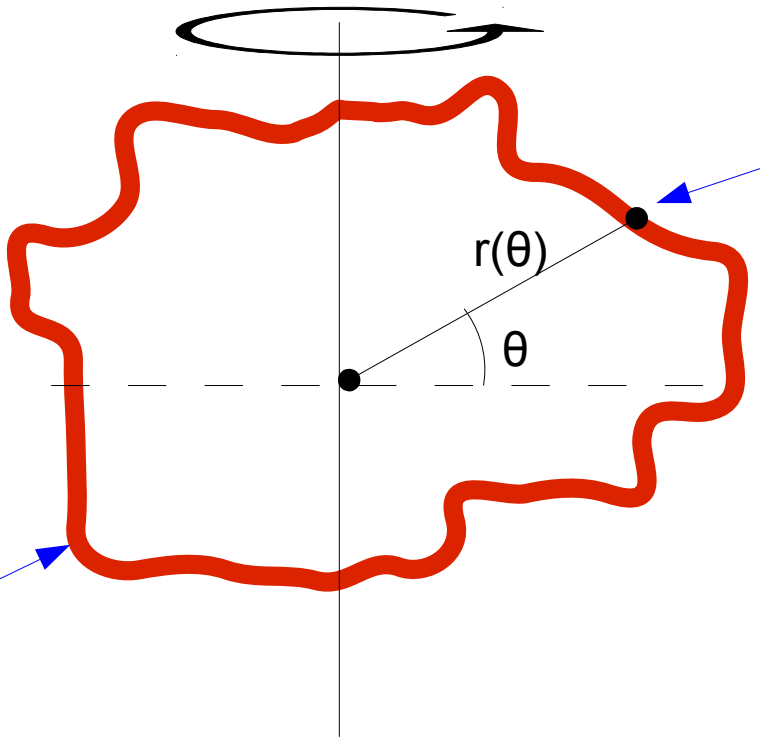
$$I = \int r^2\, dm = \int_0^{2\pi} r^2 \cdot (\mu\, r\, d\,\theta)$$

Let $r = R$, a constant

$$I = R^3 \mu \int_0^{2\pi} d\,\theta = 2\pi R^3 \mu$$

$$= (2\pi R \mu) \cdot R^2 = M\, R^2$$

Not So Simple Rotational Inertial Problem



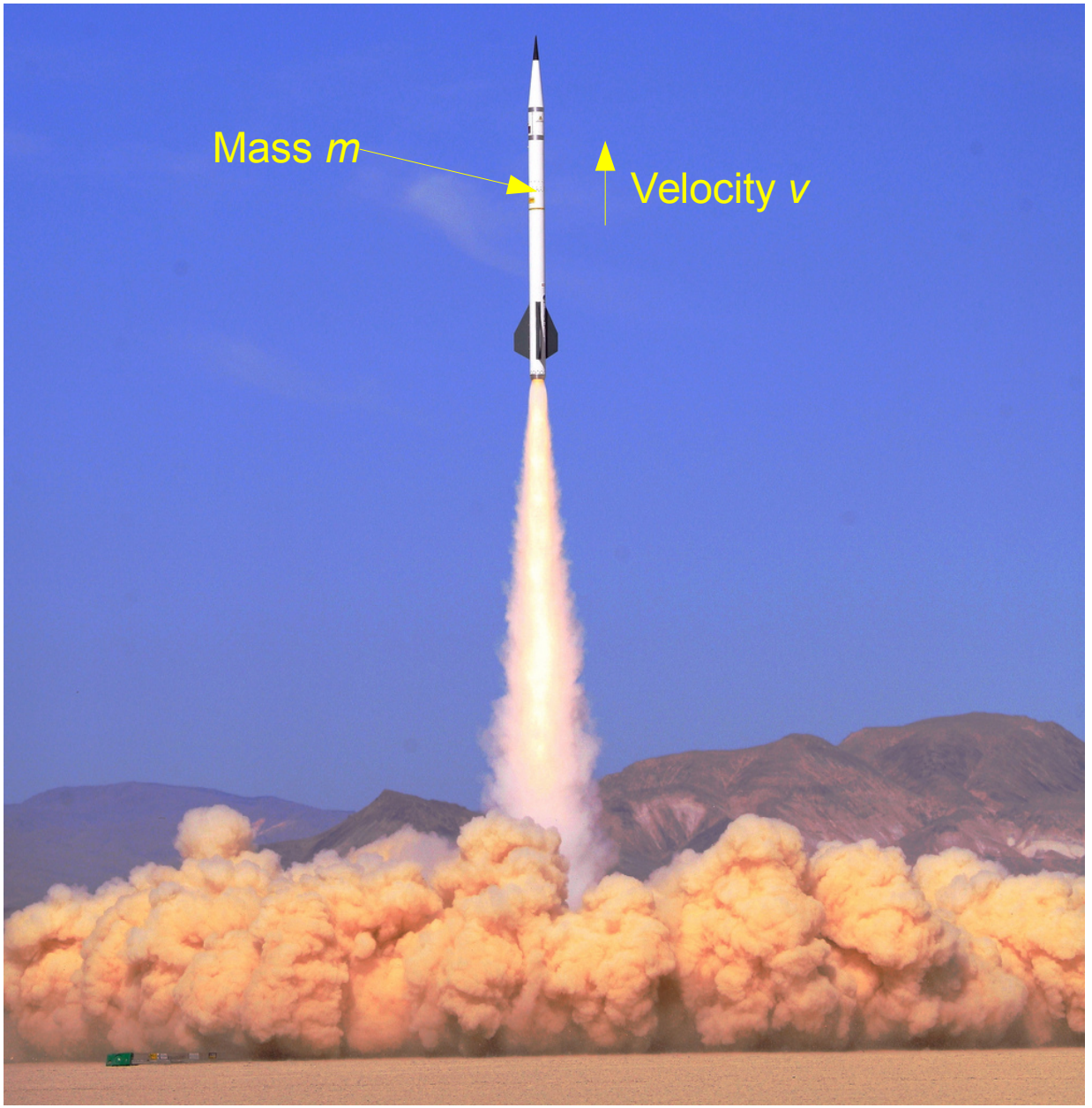
Not so circular hoop
Total mass = M
Linear mass density = μ

$$I = \int r^2(\theta)\, dm = \int_0^{2\pi} r^2(\theta)\cdot(\mu\, r(\theta)\, d\theta)$$

$$= \mu \int_0^{2\pi} r^3(\theta)\, d\theta$$

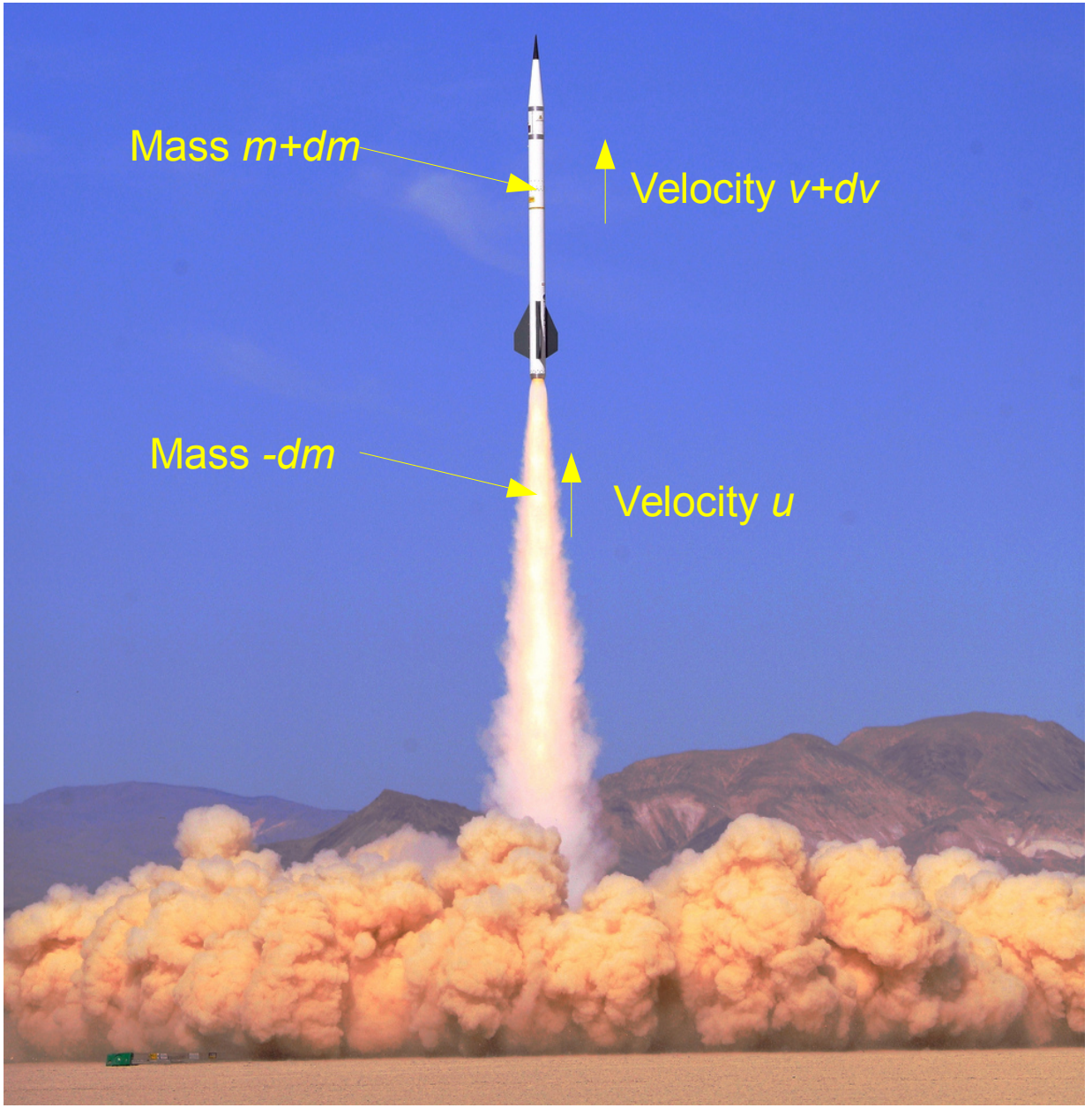
Rocket Propulsion

"Initial" state:



Rocket Propulsion

"Final" state, after small mass of fuel consumed and propelled out the nozzle is *-dm*:



Rocket Propulsion

Conservation of linear momentum:

$$P_i = P_f$$

$$mv = -dm \cdot u + (m + dm)(v + dv)$$

Look at velocity of rocket relative to the exhaust in final state v_{rel}

$$v + dv = v_{rel} + u$$

or

$$u = v + dv - v_{rel}$$

Substituting for u :

$$-dm \cdot v_{rel} = m\,dv$$

Divide by dt to see thrust and acceleration of rocket:

$$-\frac{dm}{dt} v_{rel} = T = m\frac{dv}{dt}$$

Rocket Propulsion

Or integrate dv to find increase in rocket velocity over time:

$$dv = -v_{rel}\frac{dm}{m}$$

$$\int_{v_i}^{v_f} dv = -\int_{m_i}^{m_f} v_{rel}\cdot\frac{dm}{m}$$

If v_{rel} is constant

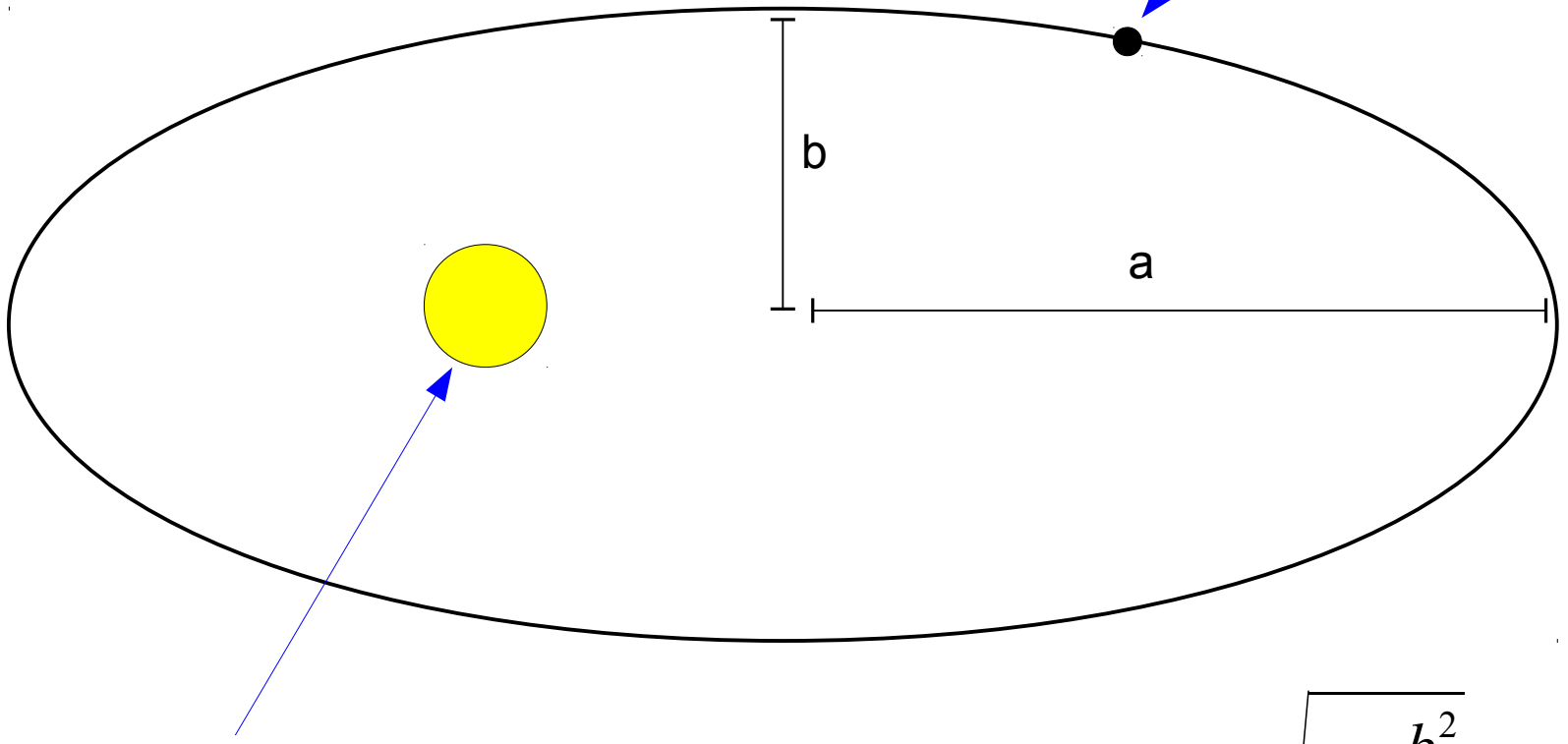
$$v_f - v_i = v_{rel}\ln\left(\frac{m_i}{m_f}\right)$$

But in general

$$v_f - v_i = -\int_{m_i}^{m_f}\frac{v_{rel}(m)}{m}dm$$

Circumference of Elliptical Orbits

a = “Semi-major axis”
b = “Semi-minor axis”



$$\text{Eccentricity } \epsilon = \sqrt{1 - \frac{b^2}{a^2}}$$

Circumference of Elliptical Orbits

The circumference of an elliptical orbit $C = 4aE(\epsilon)$

where a is the length of the semi-major axis, ϵ is the eccentricity and where the function E is the complete elliptic integral of the second kind:

$$E(\epsilon) = \int_0^{\frac{\pi}{2}} \sqrt{1-\epsilon^2 \sin^2(\theta)}\, d\theta$$

This integral cannot be evaluated analytically.

Note: for a circular orbit, $a = b =$ radius, $\epsilon = 0$ and

$$E(\epsilon) = \int_0^{\frac{\pi}{2}} d\theta = \frac{\pi}{2}$$

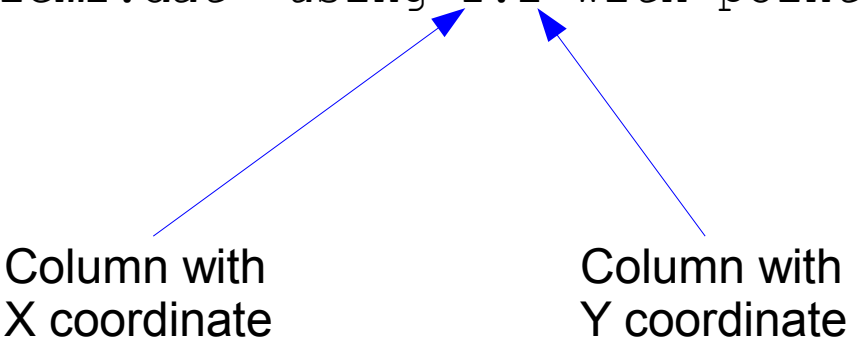
so $C = 4a\frac{\pi}{2} = 2\pi a$ as expected

Plotting Vectors with gnuplot

We have used:

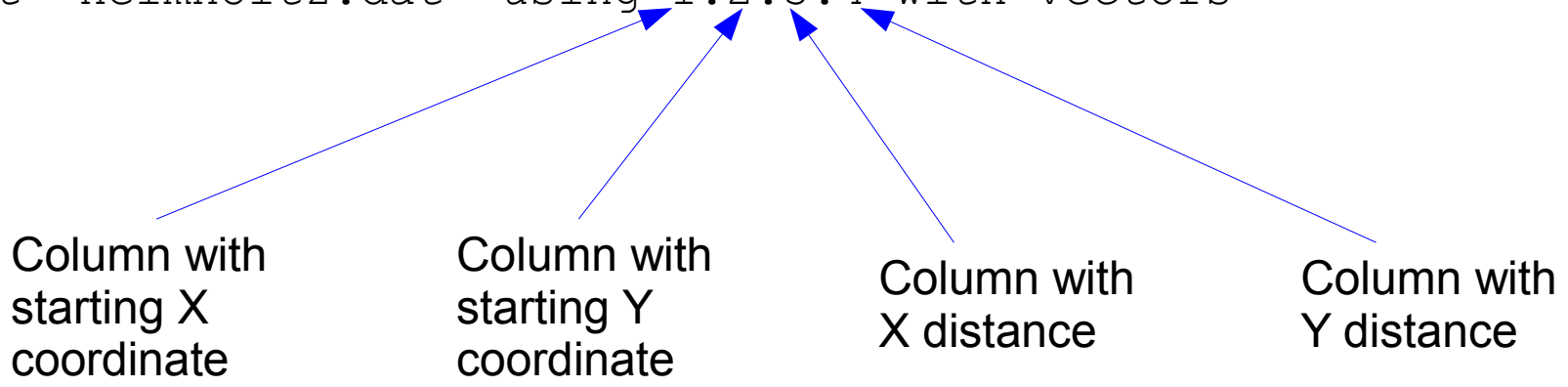
```
plot 'problem1.dat' using 1:2 with lines

plot 'problem2.dat' using 1:2 with points
```

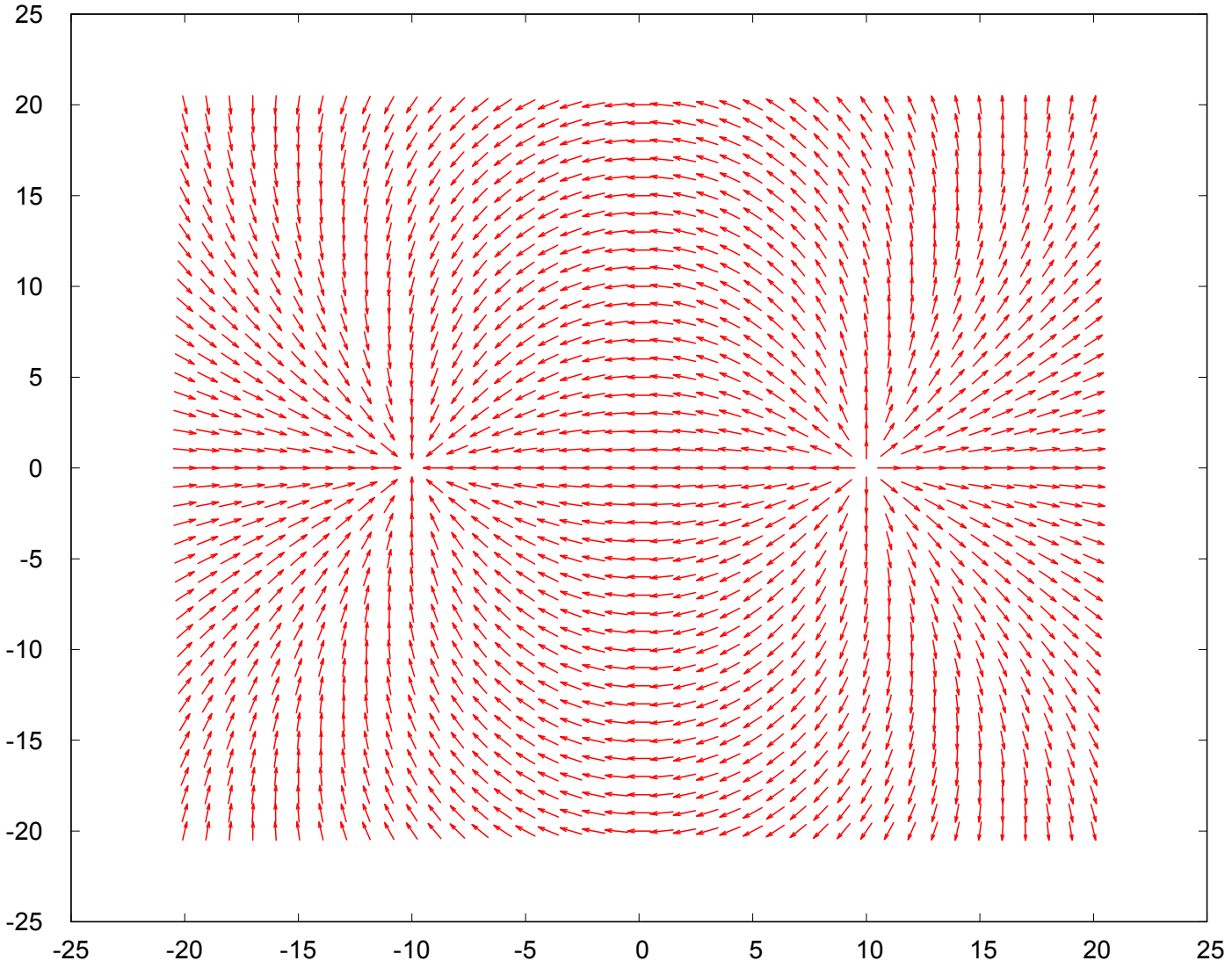


Another plot mode:

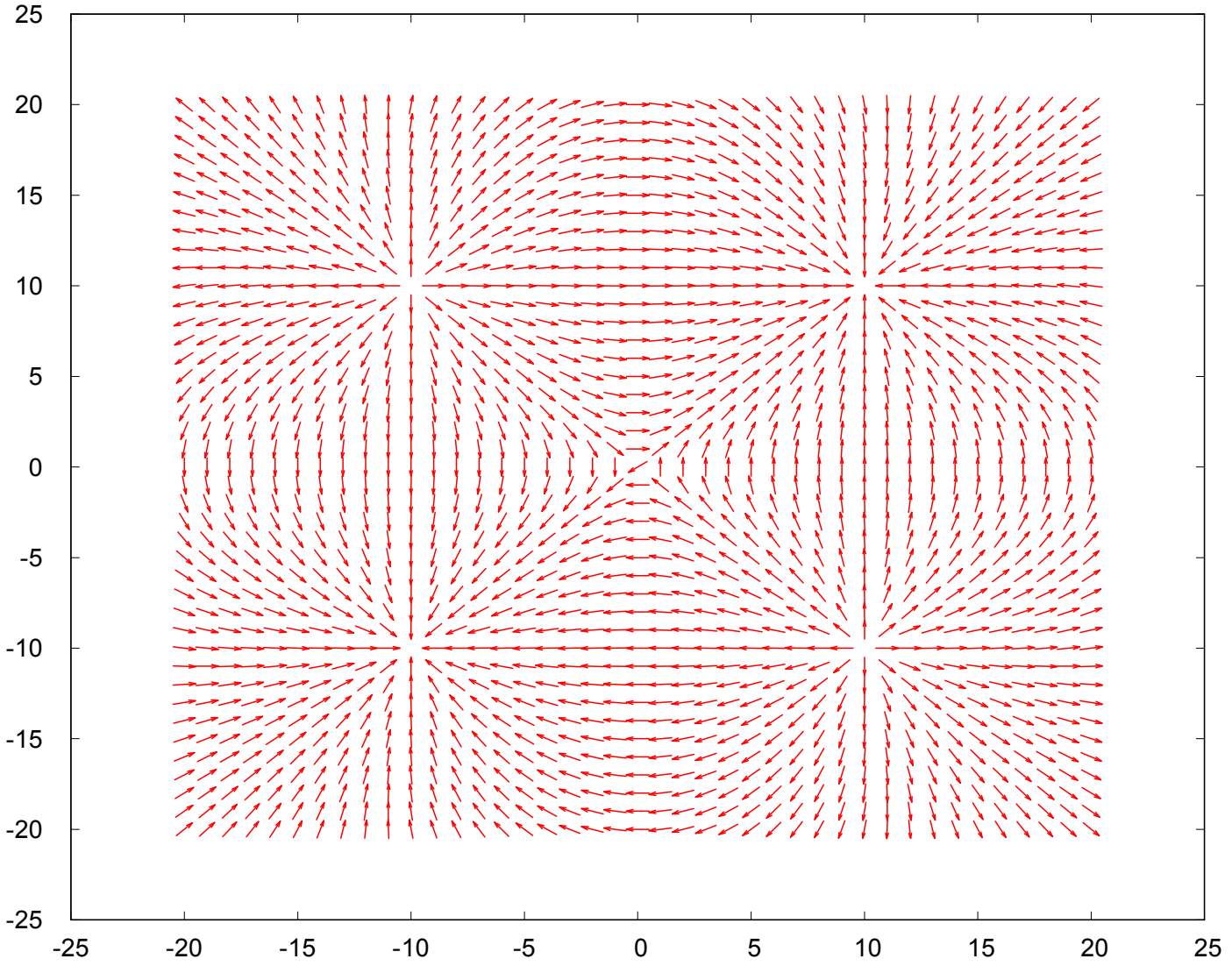
```
plot 'helmholtz.dat' using 1:2:3:4 with vectors
```



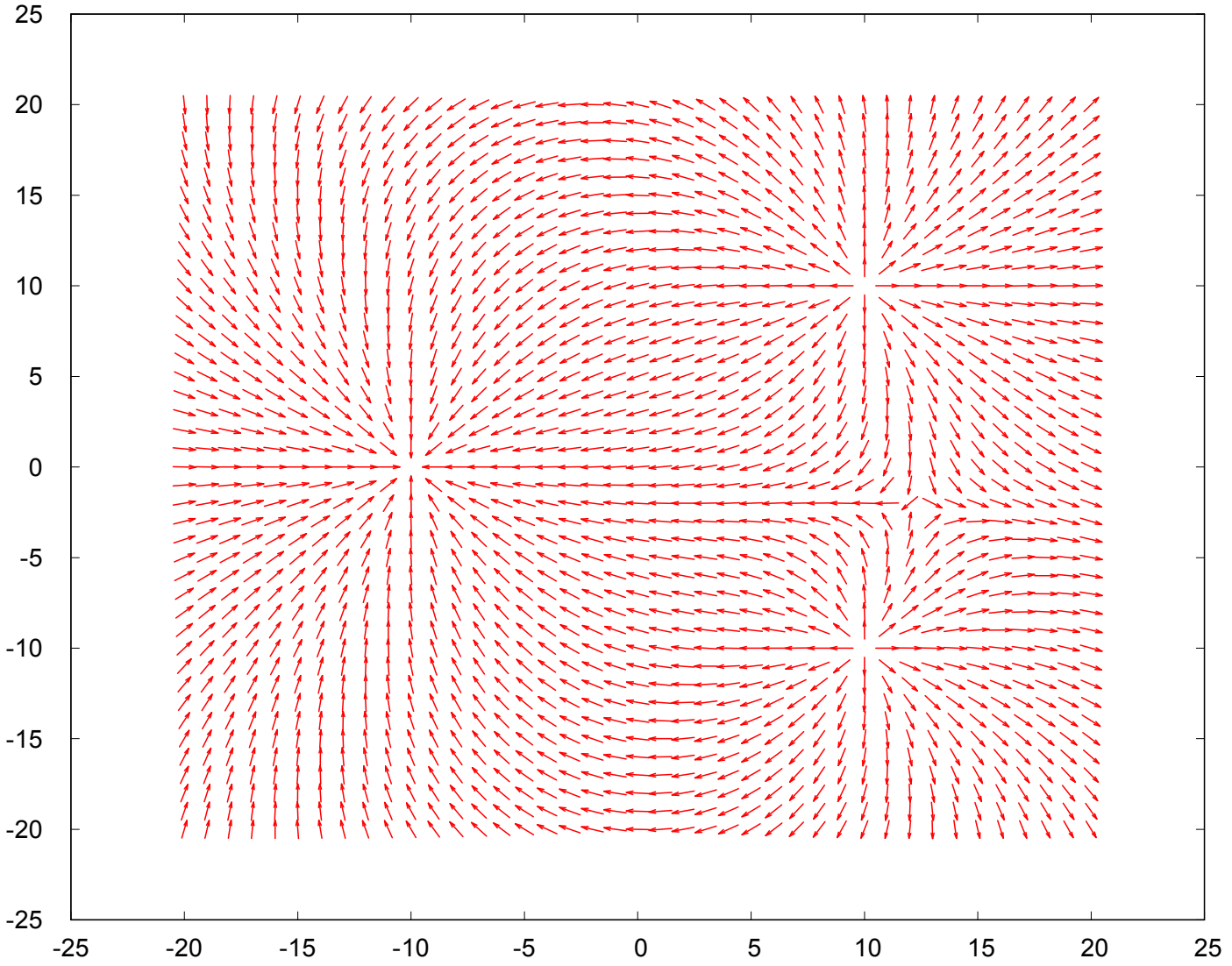
Electric Field Vectors for Dipole



Electric Field Vectors for Quadrapole



Electric Field Vectors for Example Arbitrary Charges



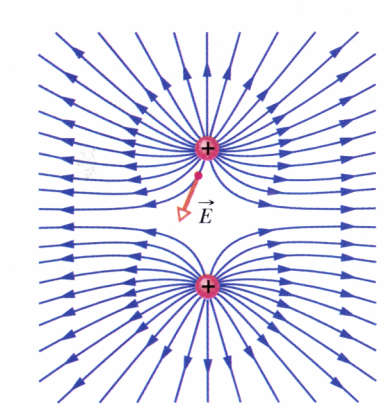
C Code for Mapping Electric Field Vectors

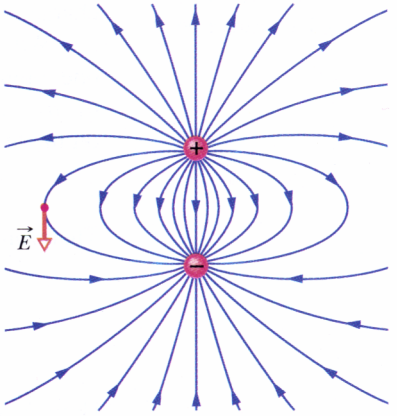
```
xstop = xstop + xinc * 0.5;
ystop = ystop + yinc * 0.5;
rtest = 1.0e-6 * (xinc*xinc + yinc*yinc);
x = xstart;
while (((xinc > 0.0) && (x < xstop)) || ((xinc < 0.0) && (x > xstop))) {
  y = ystart;
  while (((yinc > 0.0) && (y < ystop)) || ((yinc < 0.0) && (y > ystop))) {
    ex = 0.0;
    ey = 0.0;
    i = 0;
    while (i < n) {
      rsq = (x-chg[i].x)*(x-chg[i].x) + (y-chg[i].y)*(y-chg[i].y);
      if (rsq < rtest) break;
      r = sqrt(rsq);
      ex += chg[i].q * (x-chg[i].x) / (r * rsq);
      ey += chg[i].q * (y-chg[i].y) / (r * rsq);
      i++;
    }
    if (i >= n) {
      emag = sqrt(ex*ex + ey*ey);
      vx = xinc * ex / emag;
      vy = yinc * ey / emag;
      printf("%.8g %.8g %.8g %.8g\n",x-0.5*vx,y-0.5*vy,vx,vy);
    }
    y = y + yinc;
  }
  x = x + xinc;
}
```

Loop over n charges. Structure array chg[] holds each charge q_i and x_i and y_i coordinates

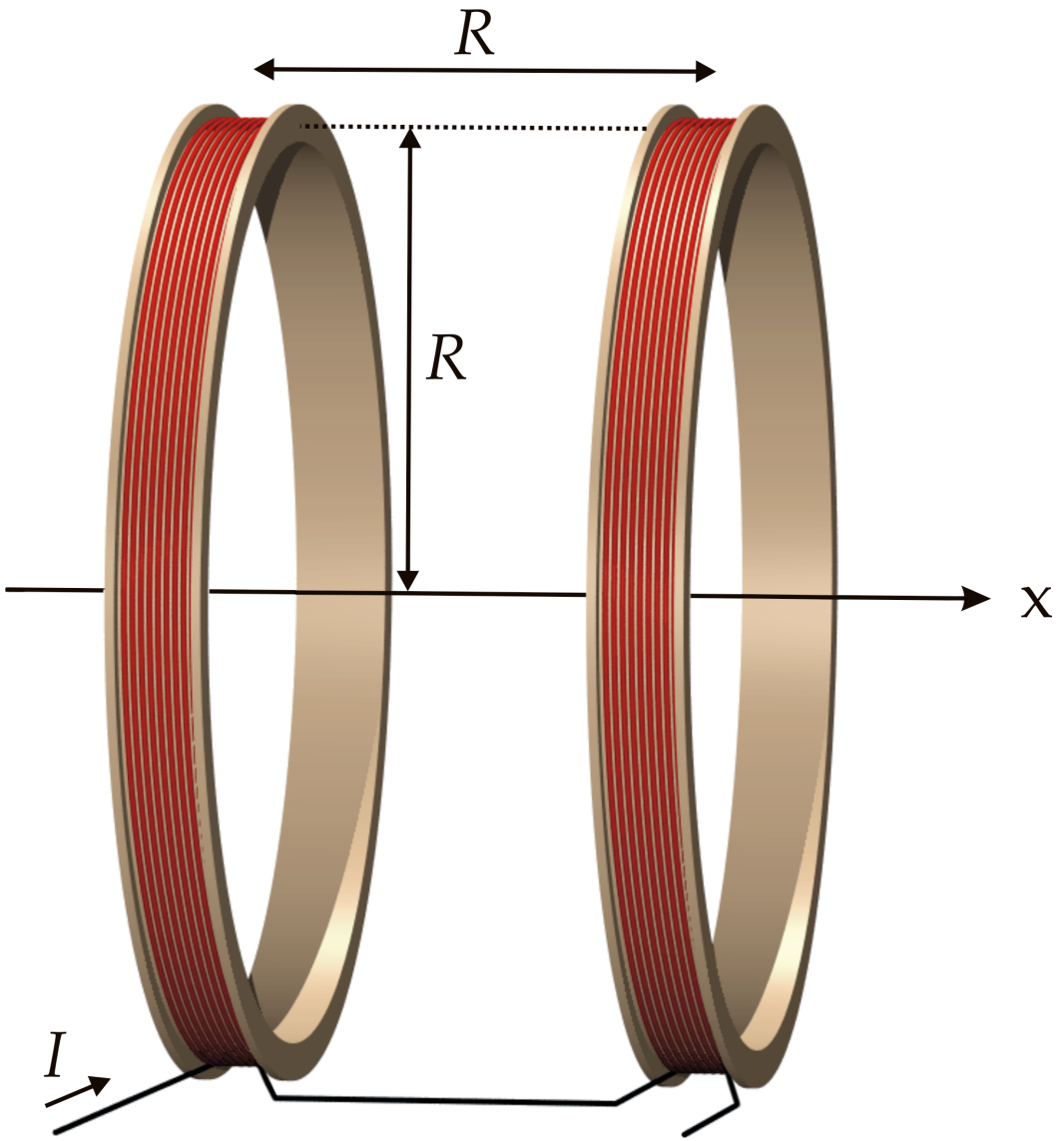
Classical Electric Field Lines

Note that the vectors plotted by this method only record the direction of the E field vector at an array of sampled points. These are not quite equivalent to the classical field lines, which convey both the direction of the field and its magnitude from the density of drawn lines.





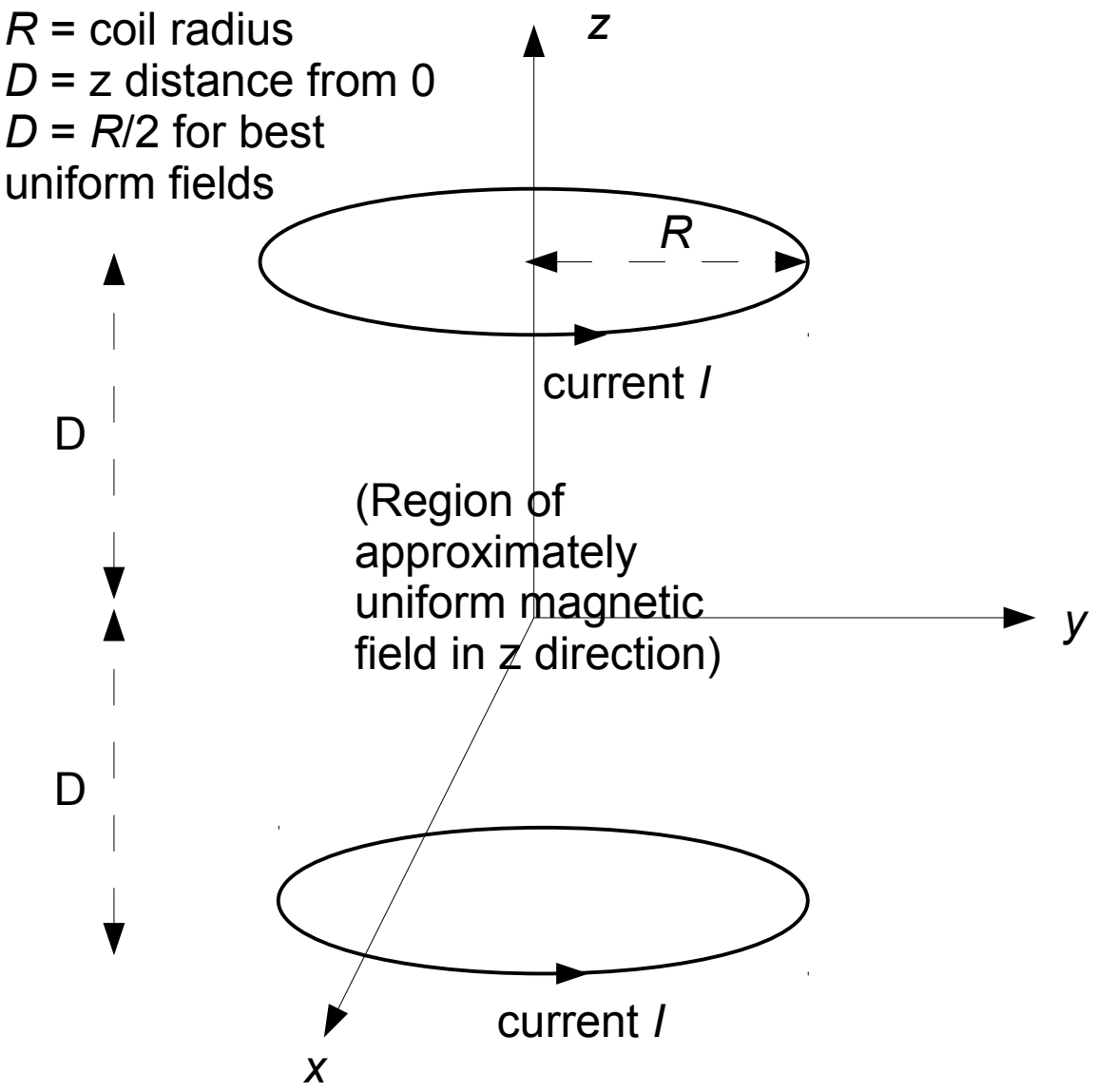
Helmholtz Coil for Uniform Magnetic Field



Magnetic Field of a Helmholtz Coil

Coordinates of two conductors follow the parameterized equations

$$x = R\cos(\theta)$$
$$y = R\sin(\theta)$$
$$z = \pm D = \pm\frac{R}{2}$$
$$\text{for } -\pi \leqslant \theta \leqslant \pi$$



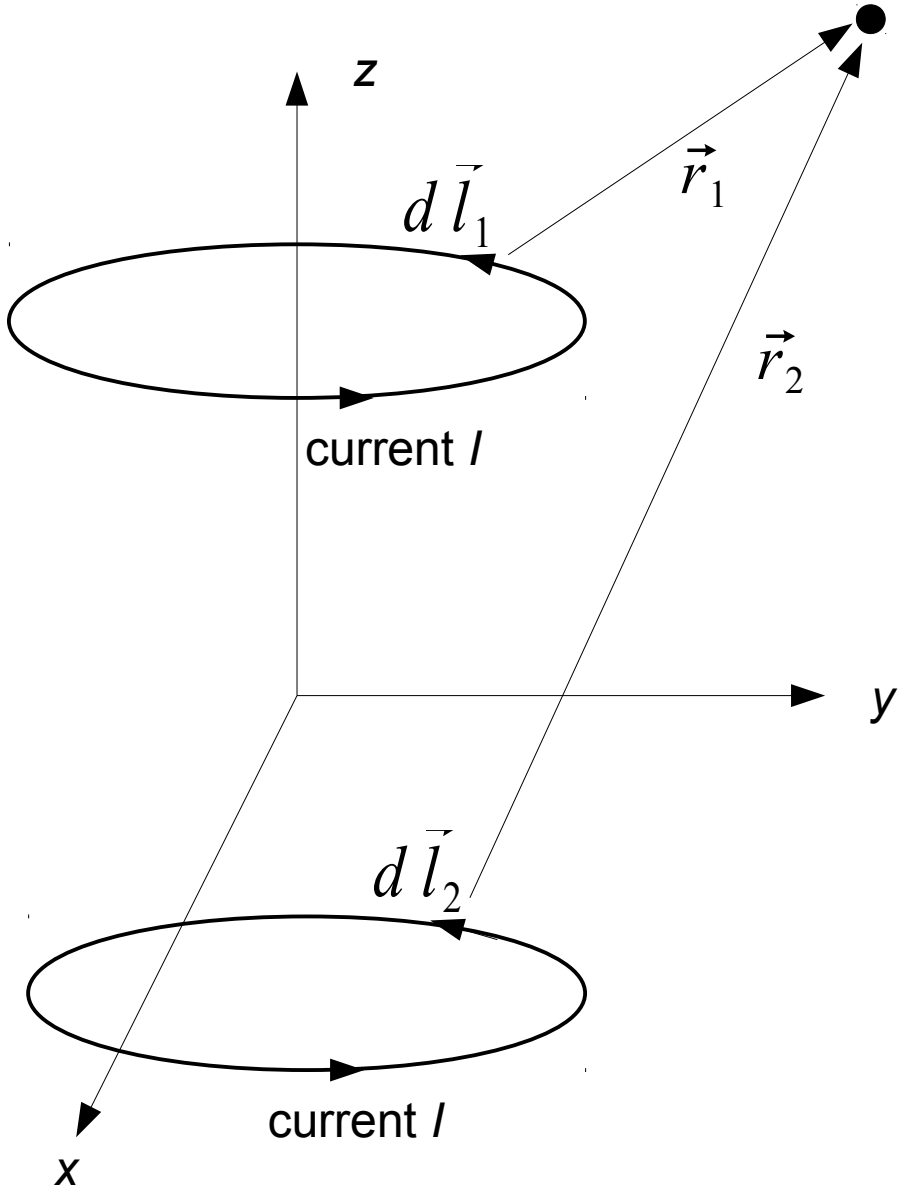
Finding Magnetic Field with Biot-Savart Law

$$d\vec{B}_i = \frac{\mu_0}{4\pi}\frac{I\, d\vec{l}_i \times \vec{r}_i}{r_i^3}$$

$$\vec{B}(\vec{r}) = \int d\vec{B}$$

$$d\vec{B} = d\vec{B}_1 + d\vec{B}_2$$

probe at (x_0, y_0, z_0)



Review of Vector Cross Product

if $\vec{C} = \vec{A} \times \vec{B}$

then

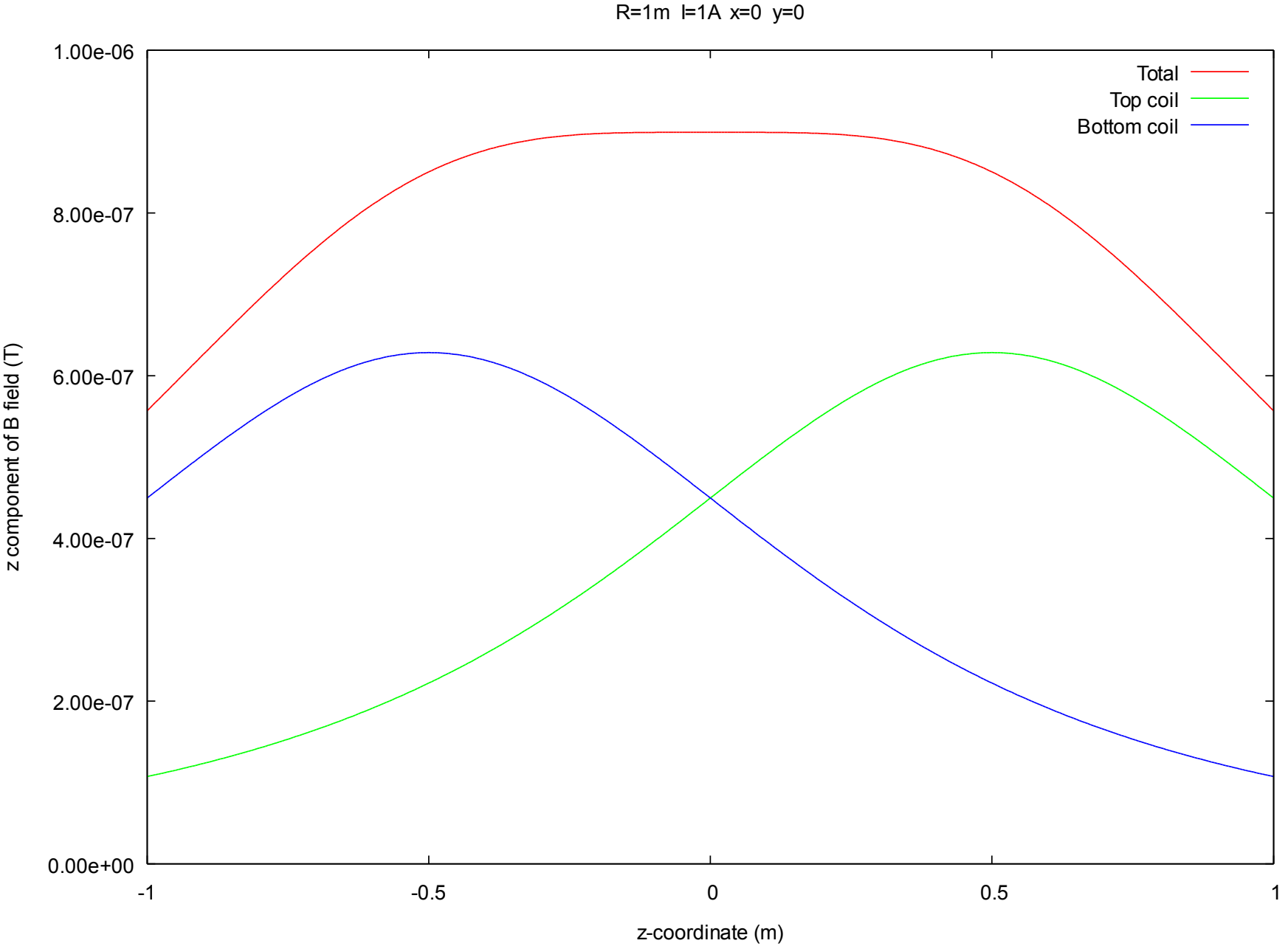
$$C_x = A_y B_z - A_z B_y$$
$$C_y = A_z B_x - A_x B_z$$
$$C_z = A_x B_y - A_y B_x$$

When computing

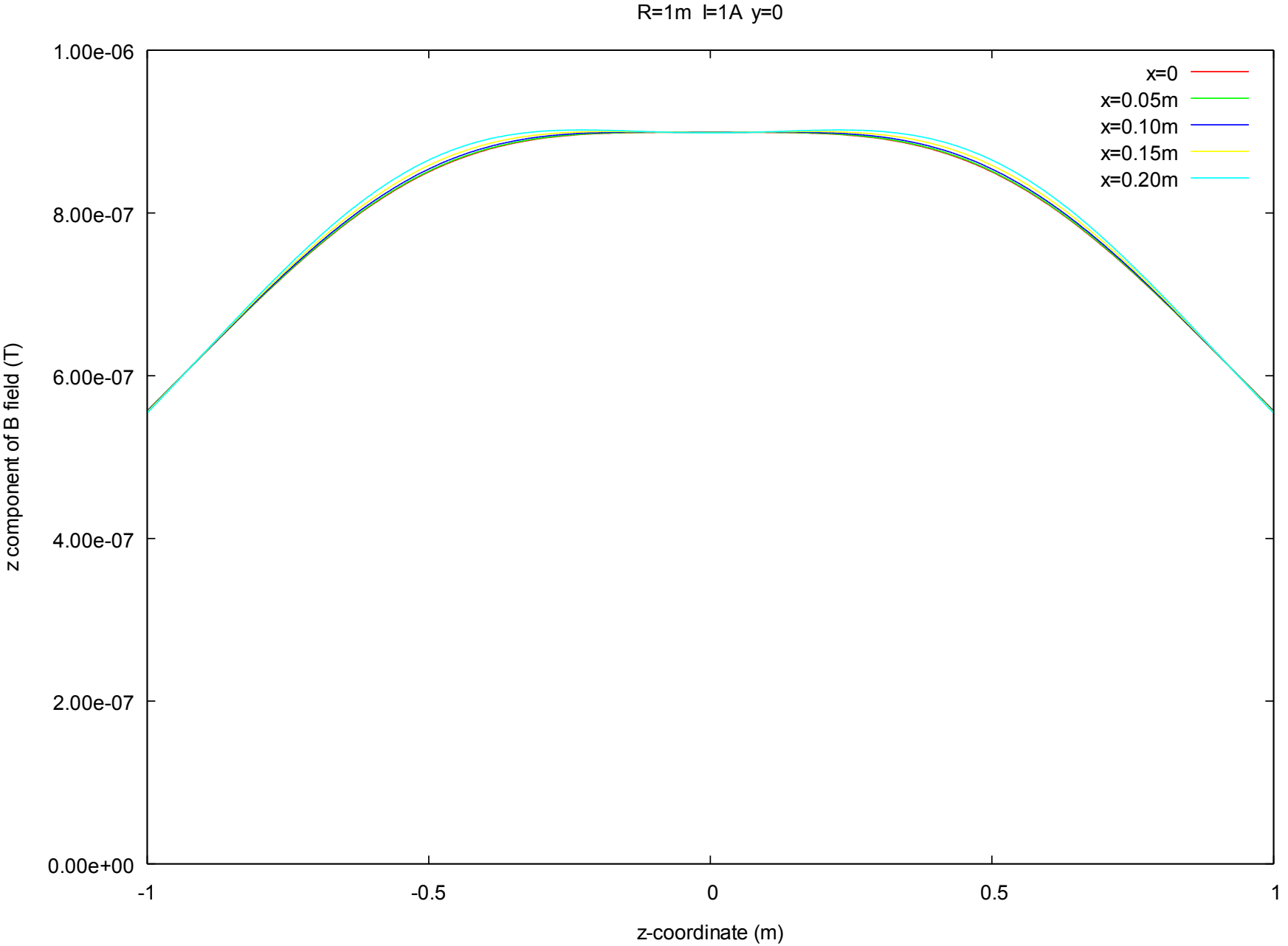
$$d\vec{l} \times \vec{r}$$

for this geometry, look for several components and terms that are zero!

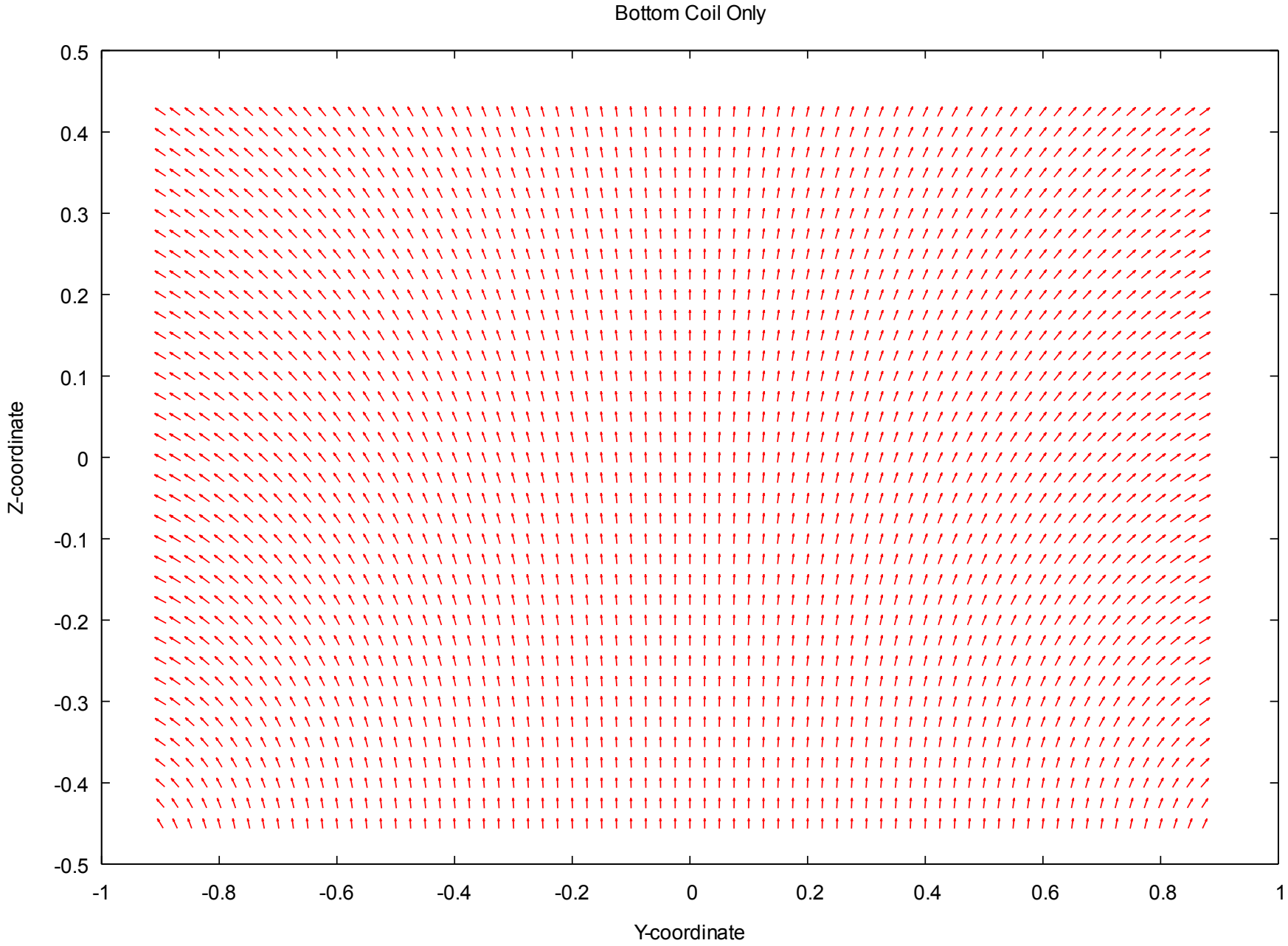
Magnetic Field Along z-Axis



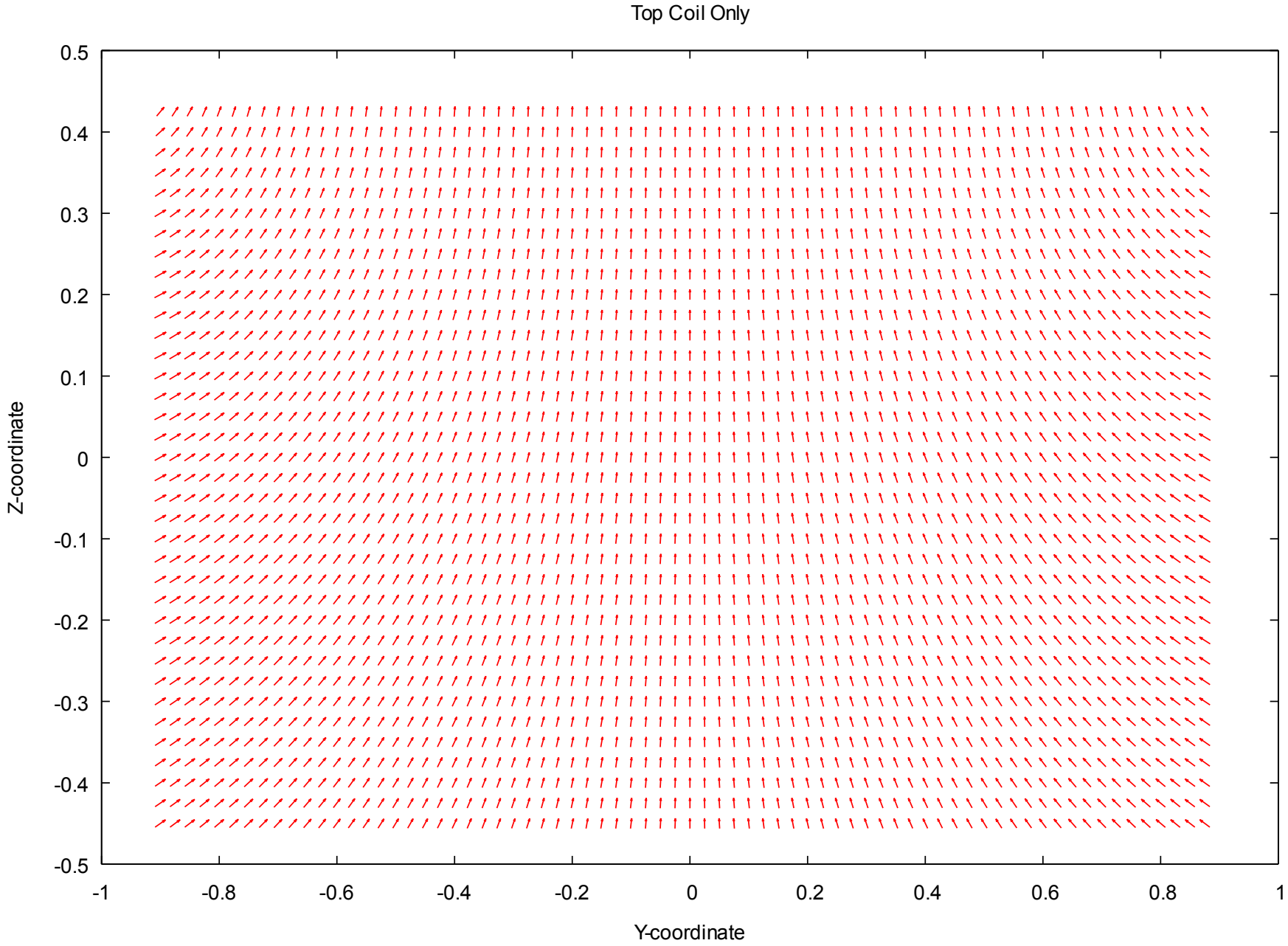
Magnetic Field Off Of z-Axis



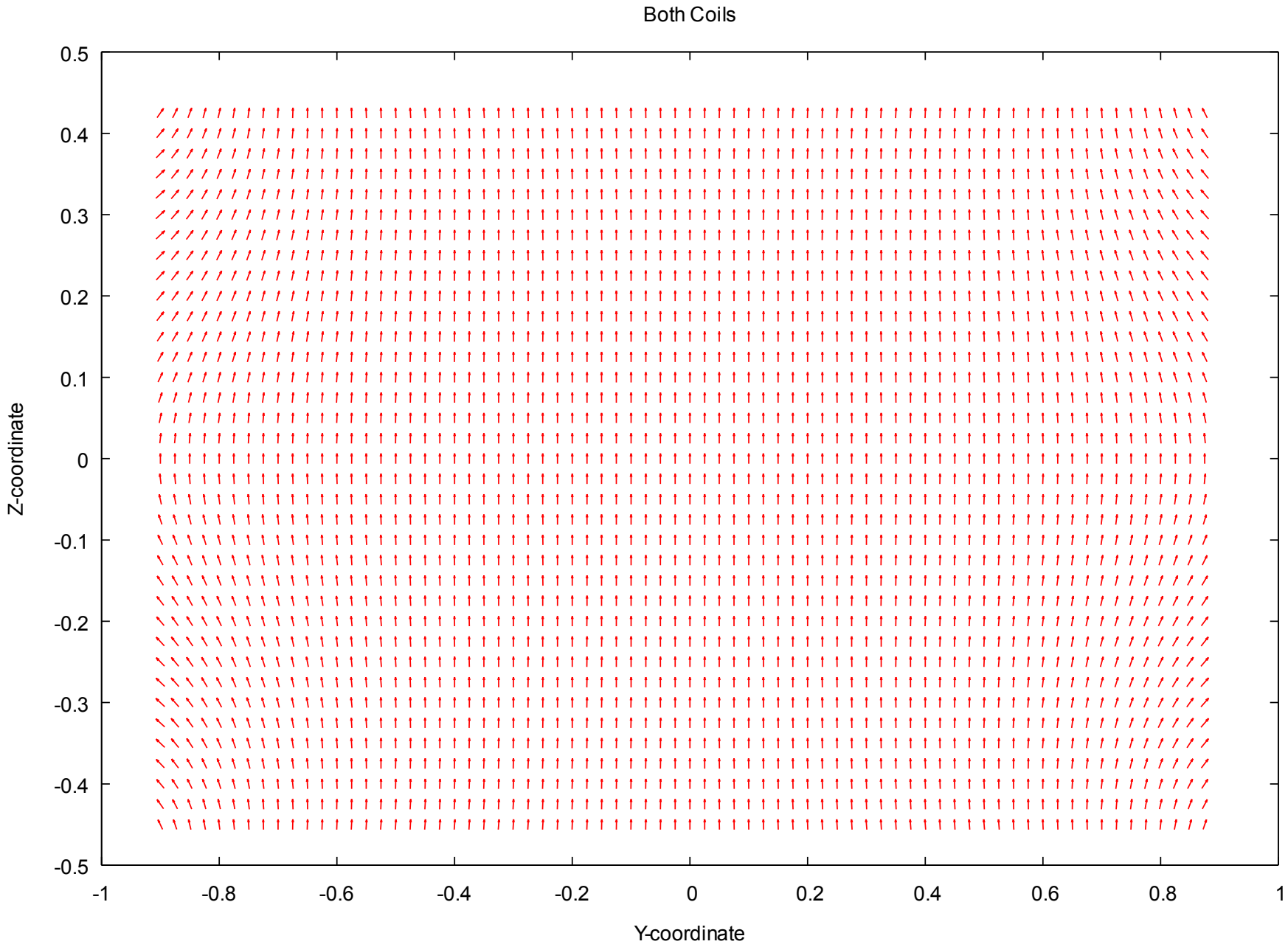
Vector Plot of Magnetic Field in Y-Z Plane



Vector Plot of Magnetic Field in Y-Z Plane



Vector Plot of Magnetic Field in Y-Z Plane



Magnetic Field of a Real Solenoid

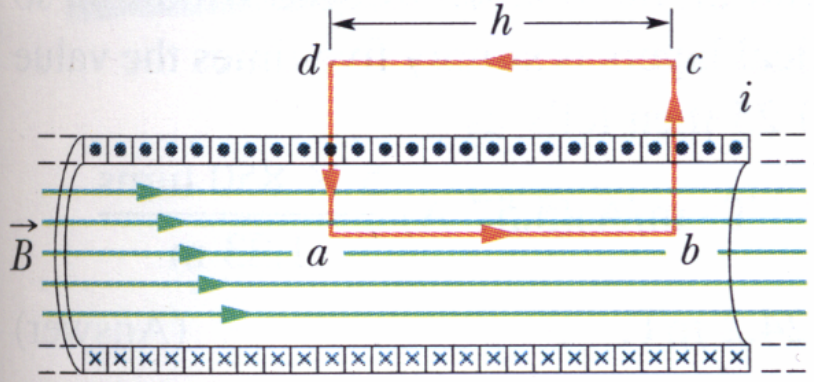


Fig. 30-19 Application of Ampere's law to a section of a long ideal solenoid carrying a current i. The Amperian loop is the rectangle $abcd$.

$$B = \mu_0 i n$$

where n = turns per unit length

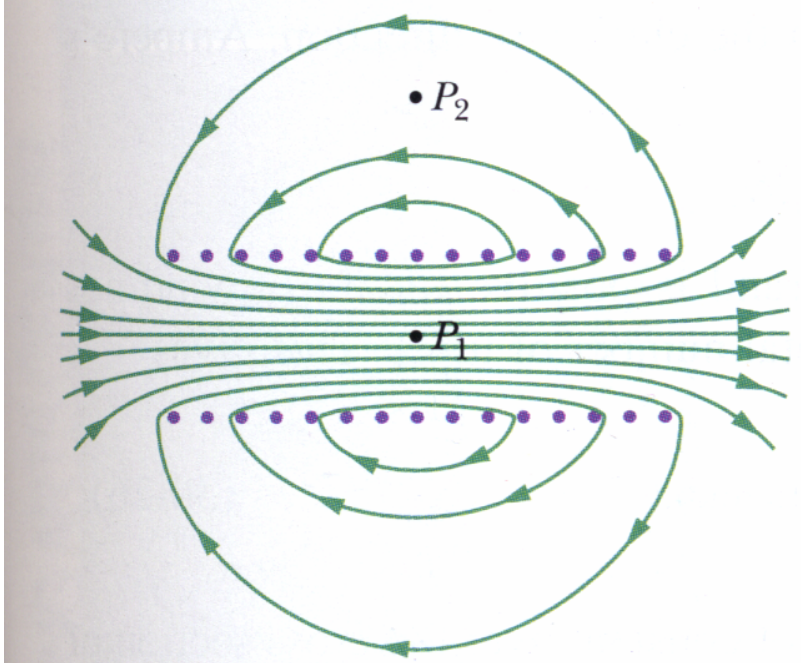


Fig. 30-18 Magnetic field lines for a real solenoid of finite length. The field is strong and uniform at interior points such as P_1 but relatively weak at external points such as P_2.

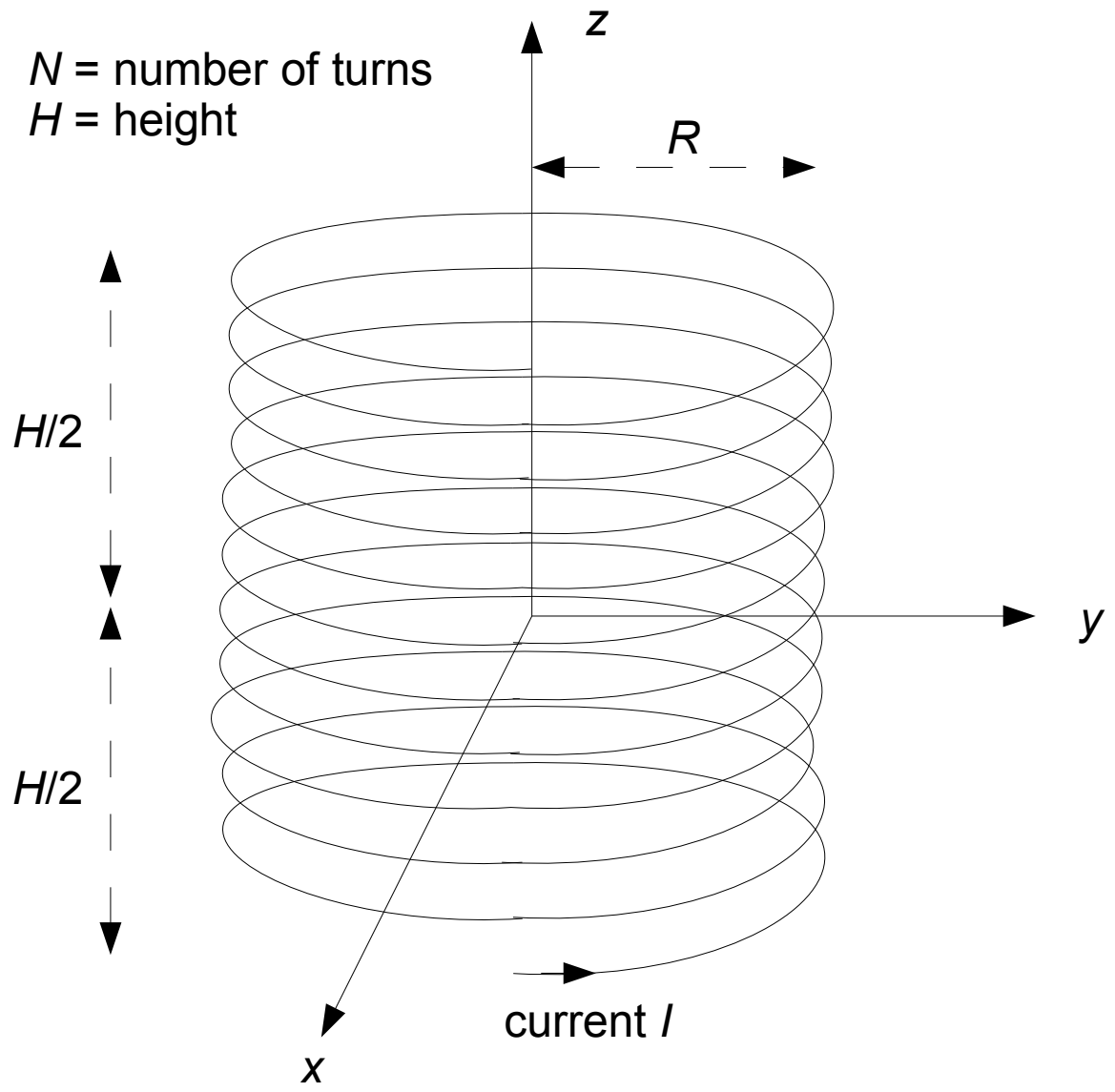
Magnetic Field of a Real Solenoid

Coordinates of solenoid conductor follow the parameterized equations

$$x = R\cos(\theta)$$
$$y = R\sin(\theta)$$
$$z = a\,\theta$$

for $-N\,\pi \leqslant \theta \leqslant N\,\pi$

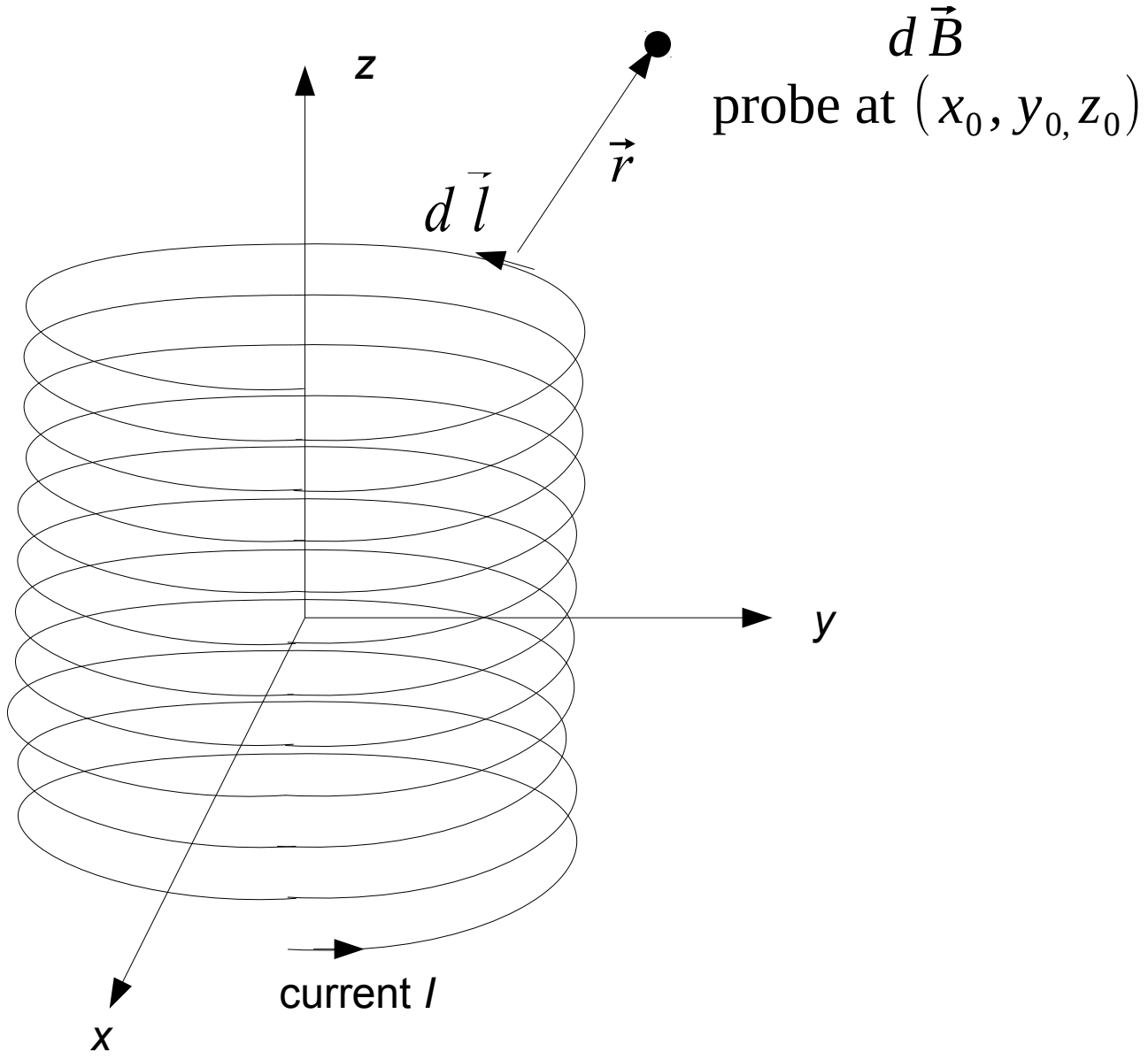
and where $a = \dfrac{H}{2\,\pi\,N}$



Finding Magnetic Field with Biot-Savart Law

$$d\vec{B} = \frac{\mu_0}{4\pi}\frac{I\, d\vec{l}\times\vec{r}}{r^3}$$

$$\vec{B}(\vec{r}) = \int d\vec{B}$$



Beware of Random Walks of Roundoff Error!

Example: edit integration lab program to try to compute pi:

$$\pi = \int_0^1 4\sqrt{1-x^2}\,dx$$

Beware of Random Walks of Roundoff Error!

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "integrate.h"

double func(double x) {
  return(4.0 * sqrt(1.0 - x*x));
}

int main(int argc,char *argv[]) {
  double xinc;

  if (argc != 2) {
    fprintf(stderr,"%s <xinc>\n",argv[0]);
    exit(1);
  }
  xinc = atof(argv[1]);
  printf("Trapezoidal rule:\n");
  trapezoidal(func,0.0,1.0,xinc);
  printf("Simpson's rule:\n");
  simpson(func,0.0,1.0,xinc);
  exit(0);}
```

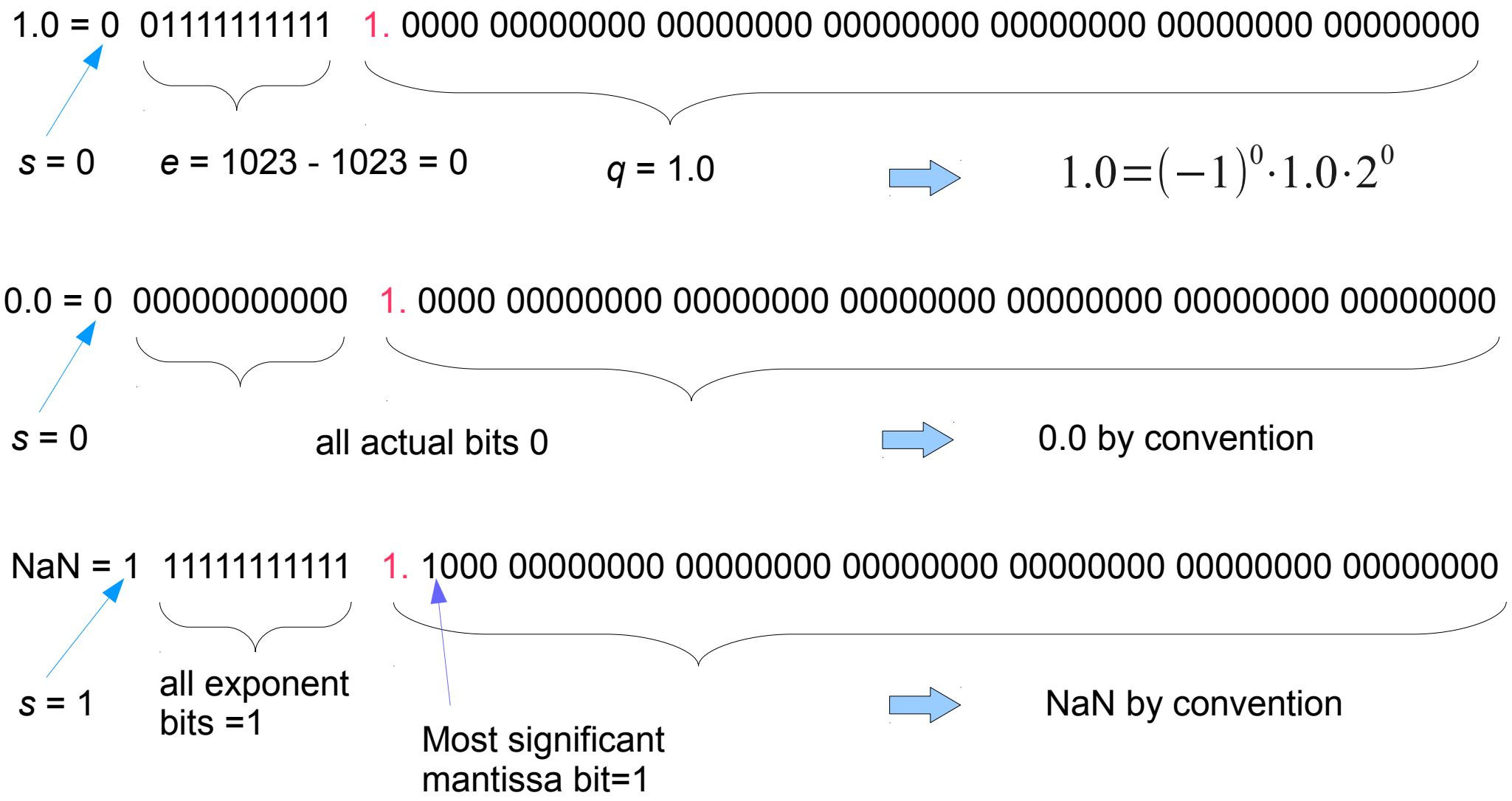
Beware of Random Walks of Roundoff Error!

```
$ ./pi_integrate 0.01
Trapezoidal rule:
-nan
Simpson's rule:
-nan
$ ./pi_integrate 0.0025
Trapezoidal rule:
3.14144566781
Simpson's rule:
3.14153524166
```

Try two different step sizes through the integration routines

So what is this nan business???

NaN Binary Storage Convention



The man Page for sqrt()

```
SQRT(3)                    Linux Programmer's Manual                    SQRT(3)

NAME
       sqrt, sqrtf, sqrtl - square root function

SYNOPSIS
       #include <math.h>

       double sqrt(double x);
       float sqrtf(float x);
       long double sqrtl(long double x);

       Link with -lm.

DESCRIPTION
       The sqrt() function returns the nonnegative square root of x.

RETURN VALUE
       On success, these functions return the square root of x.
       If x is a NaN, a NaN is returned.
       If x is +0 (-0), +0 (-0) is returned.
       If x is positive infinity, positive infinity is returned.
       If x is less than -0, a domain error occurs, and a NaN is returned.
```