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 First Order Differential Equation

A general first order differential equation can be written

$$\frac{dy}{dx} = f(x, y)$$

$$\text{If } f(x, y) = f(x) \quad (\text{independent of } y)$$

Then the solution can be found with an integration

$$y = \int f(x)\, dx$$

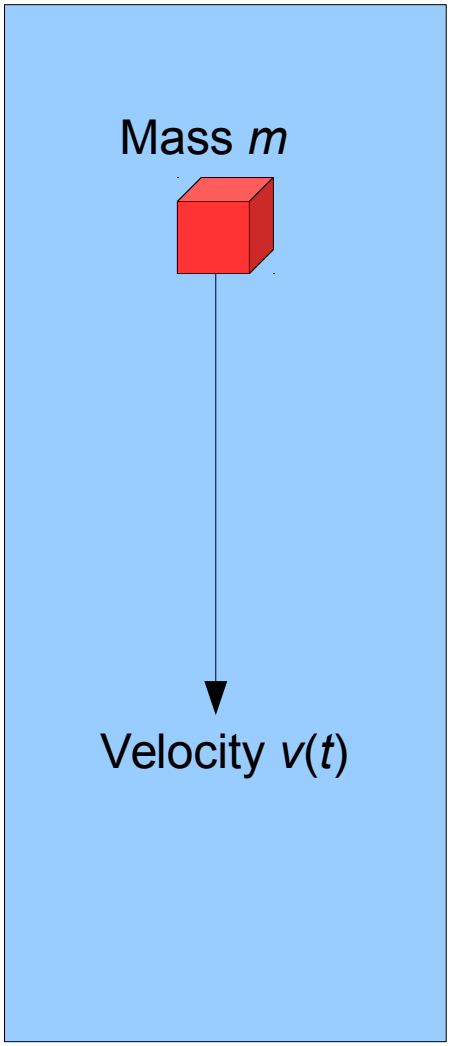
But in the general case above finding the solution is still called “integration” of the differential equation

Example of First Order Differential Equation

An object falls in a uniform gravitational field through a fluid and reaches a terminal velocity due to drag forces

$$v(t) = \text{Downward velocity}$$

$$\frac{dv(t)}{dt} = \text{Downward acceleration} = mg - F_{drag}$$



If the drag forces through the fluid are modeled with a simple linear relationship

$$F_{drag} = \alpha v$$

$$\frac{dv}{dt} = mg - \alpha v = f(v)$$

Then this simple differential equation can be easily solved, and $v(t)$ will have an exponential trajectory. But in general more complete models of drag forces will be some nonlinear function of v to take turbulence into account, and a numerical solution will be necessary.

Vector Field for First Order Differential Equation

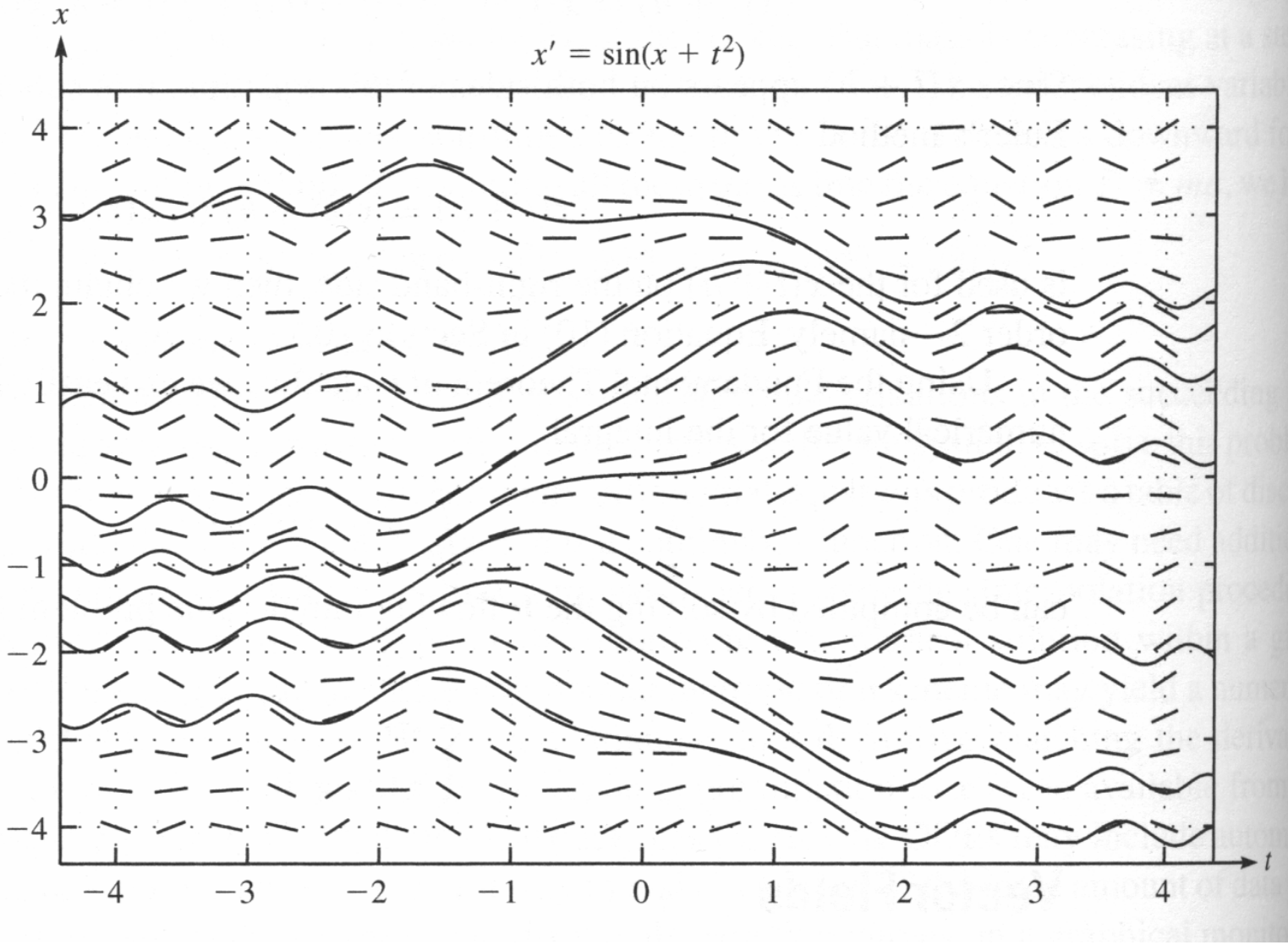


FIGURE 10.1
Vector field and some solution curves for $x' = \sin(x + t^2)$

Vector Field for First Order Differential Equation

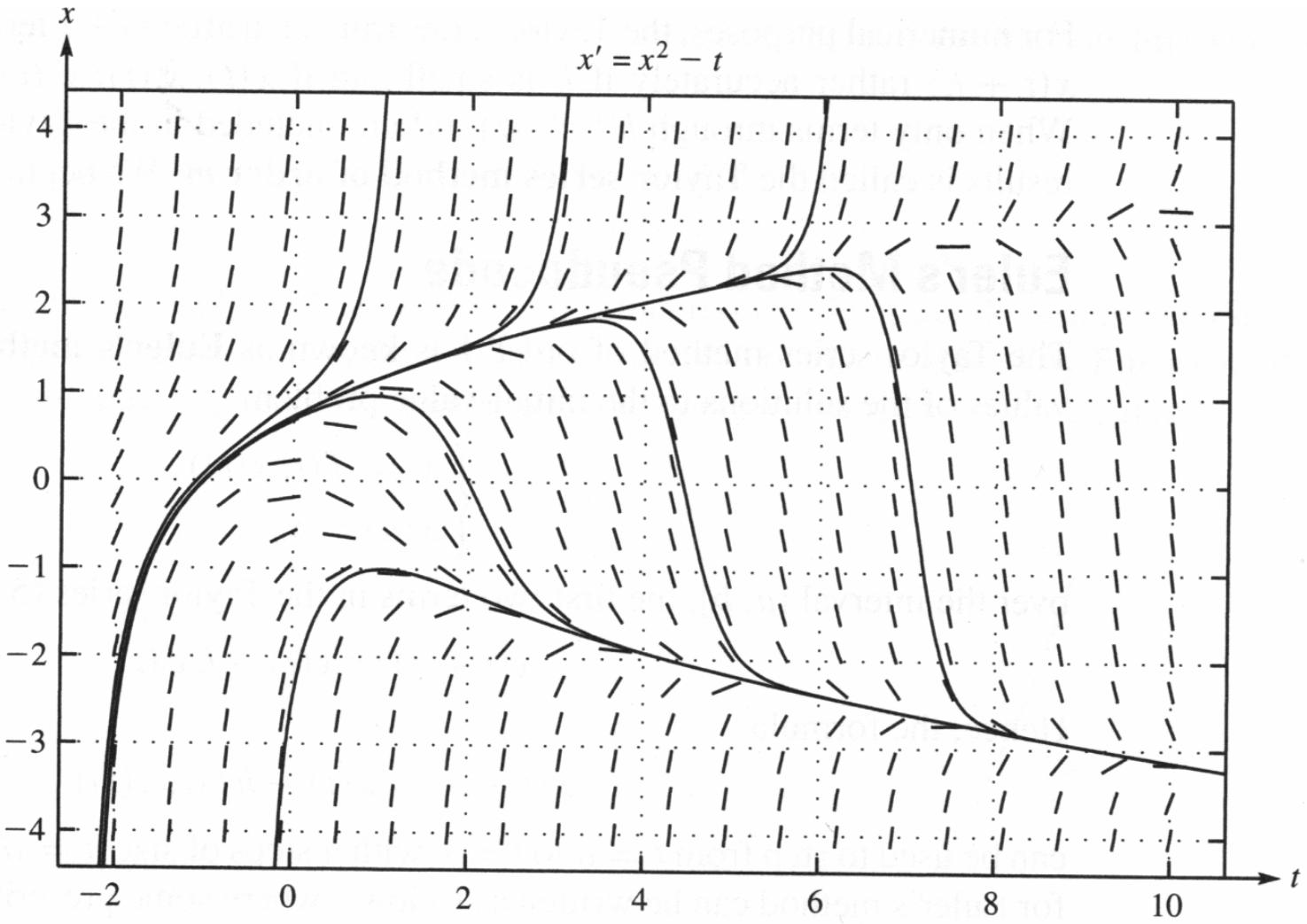
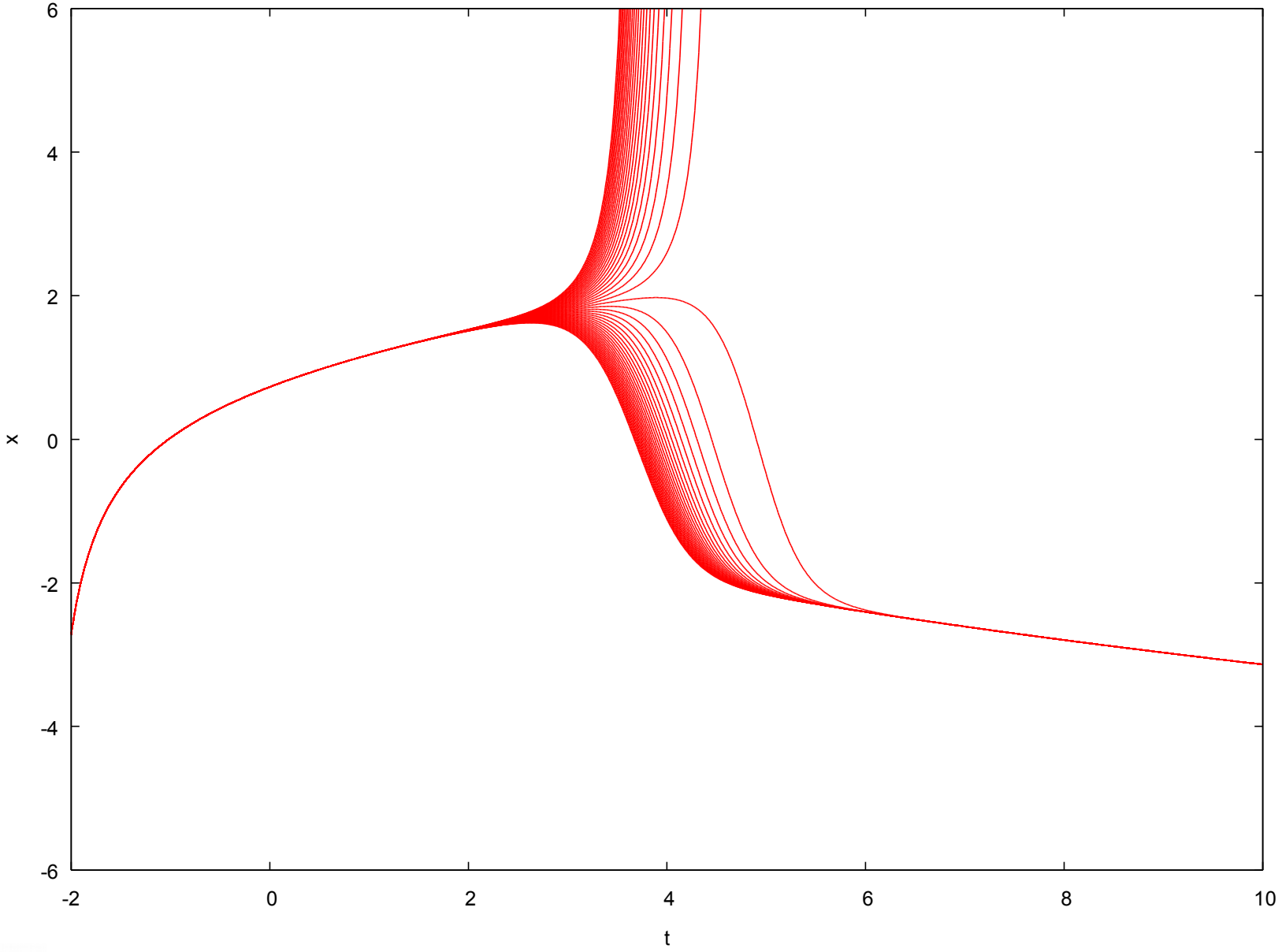


FIGURE 10.2
Vector field and some solution curves for $x' = x^2 - t$

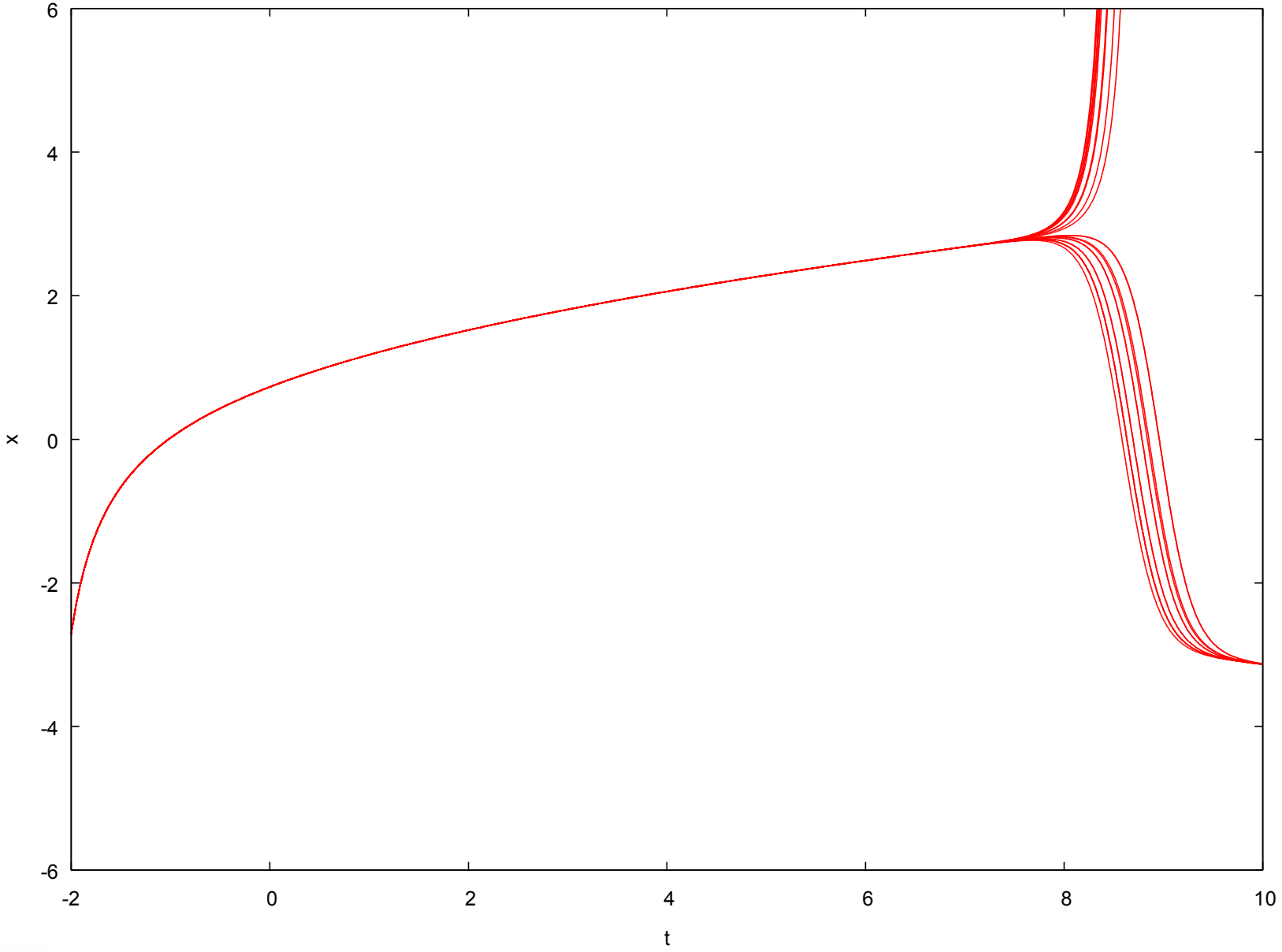
Solutions of $x' = x^2 - t$

Each solution for -2<t<10, sweep initial x from -2.7190 to -2.7185 with increment 1e-5

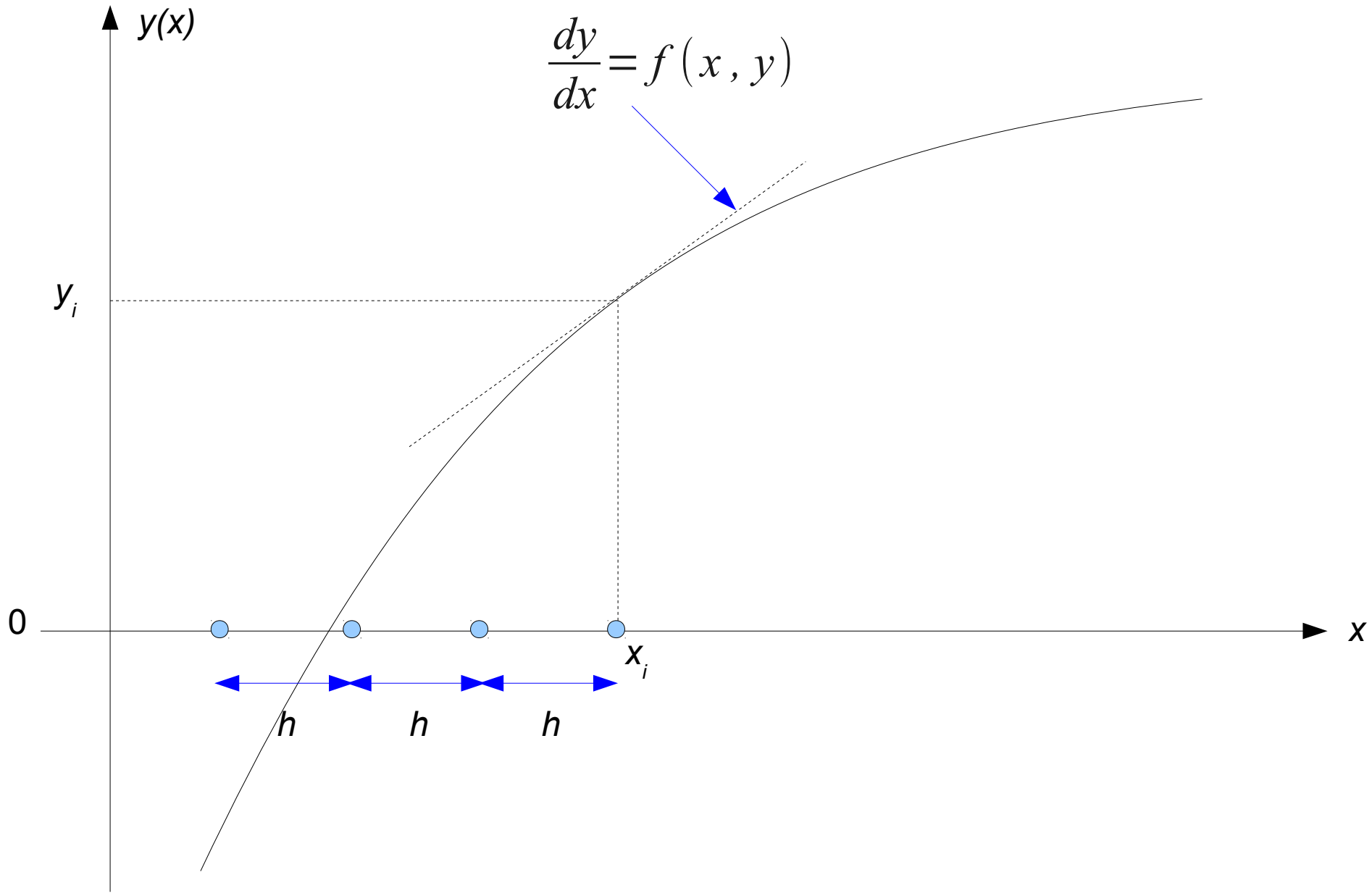


Solutions of $x' = x^2 - t$

sweep initial x from -2.718728347039740 to -2.718728347039720 with increment 1e-15 !!



Numerical Integration of First Order DE



Return to Taylor's Theorem for y(x+h)

$$y(x+h)=\sum_{k=0}^{n}\frac{y^{(k)}(x)}{k!}h^k \;+\; E_{n+1}$$

The error term $E_{n+1}=\frac{y^{(n+1)}(\xi)}{(n+1)!}h^{n+1}$ where ξ is between x and $x+h$

Examples:

$$y(x+h) \;=\; y(x) \;+\; y'(x)h \;+\; E_2$$

$$y(x+h) \;=\; y(x) \;+\; y'(x)h \;+\; \frac{y''(x)}{2}h^2 \;+\; E_3$$

$$y(x+h) \;=\; y(x) \;+\; y'(x)h \;+\; \frac{y''(x)}{2}h^2 \;+\; \frac{y'''(x)}{6}h^3 \;+\; E_4$$

Direct Taylor Series Algorithms

$$y_{i+1} = y_i + h \cdot f(x_i, y_i)$$ (Euler algorithm)

$$y_{i+1} = y_i + h \cdot f(x_i, y_i) + h^2 \cdot \frac{f'(x_i, y_i)}{2}$$

$$y_{i+1} = y_i + h \cdot f(x_i, y_i) + h^2 \cdot \frac{f'(x_i, y_i)}{2} + h^3 \cdot \frac{f''(x_i, y_i)}{6}$$

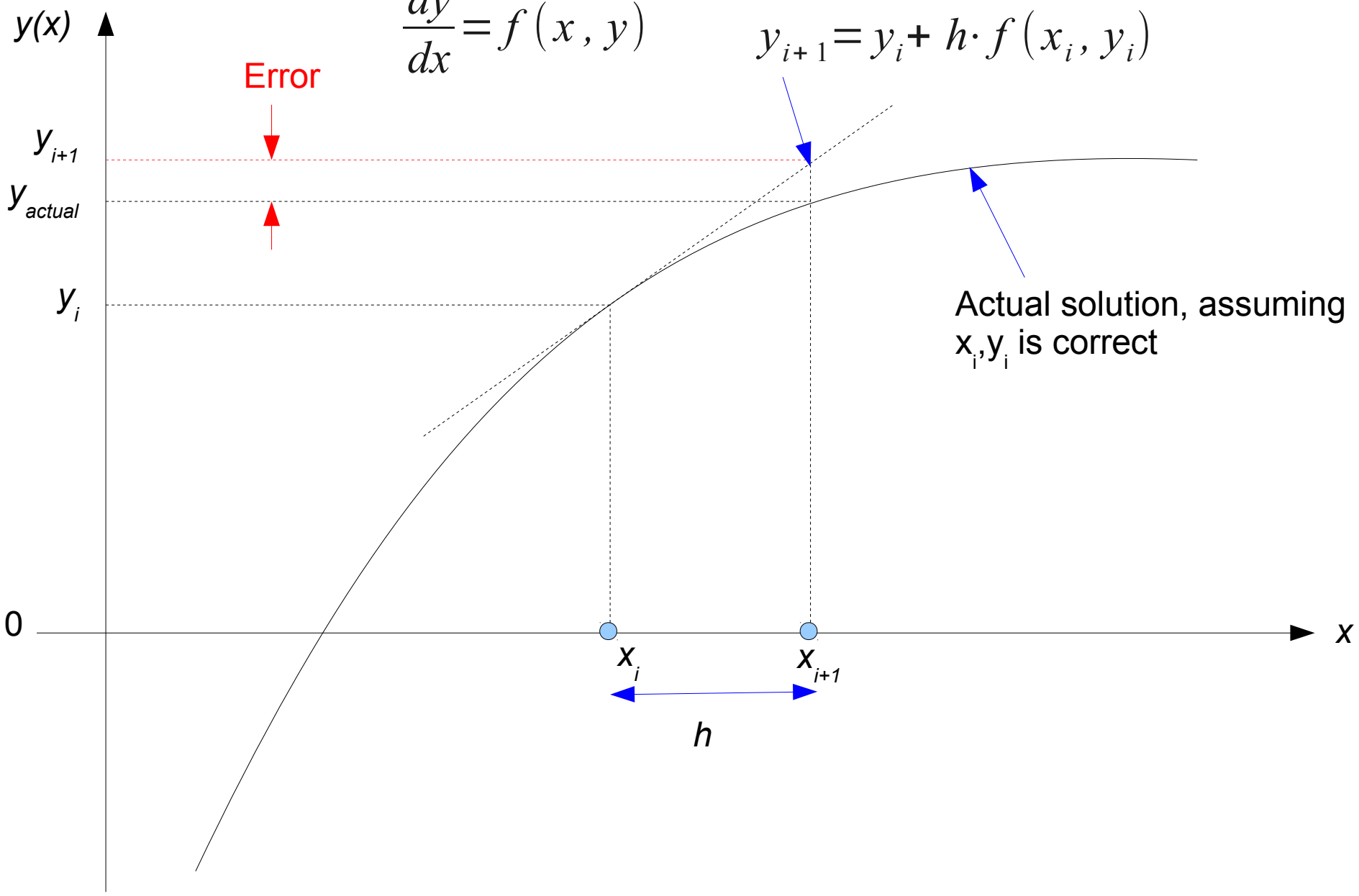
.
.
.

Requires more complex implicit differentiation of $f(x,y)$ for smaller error terms, so not a practical method for arbitrary differential equations.

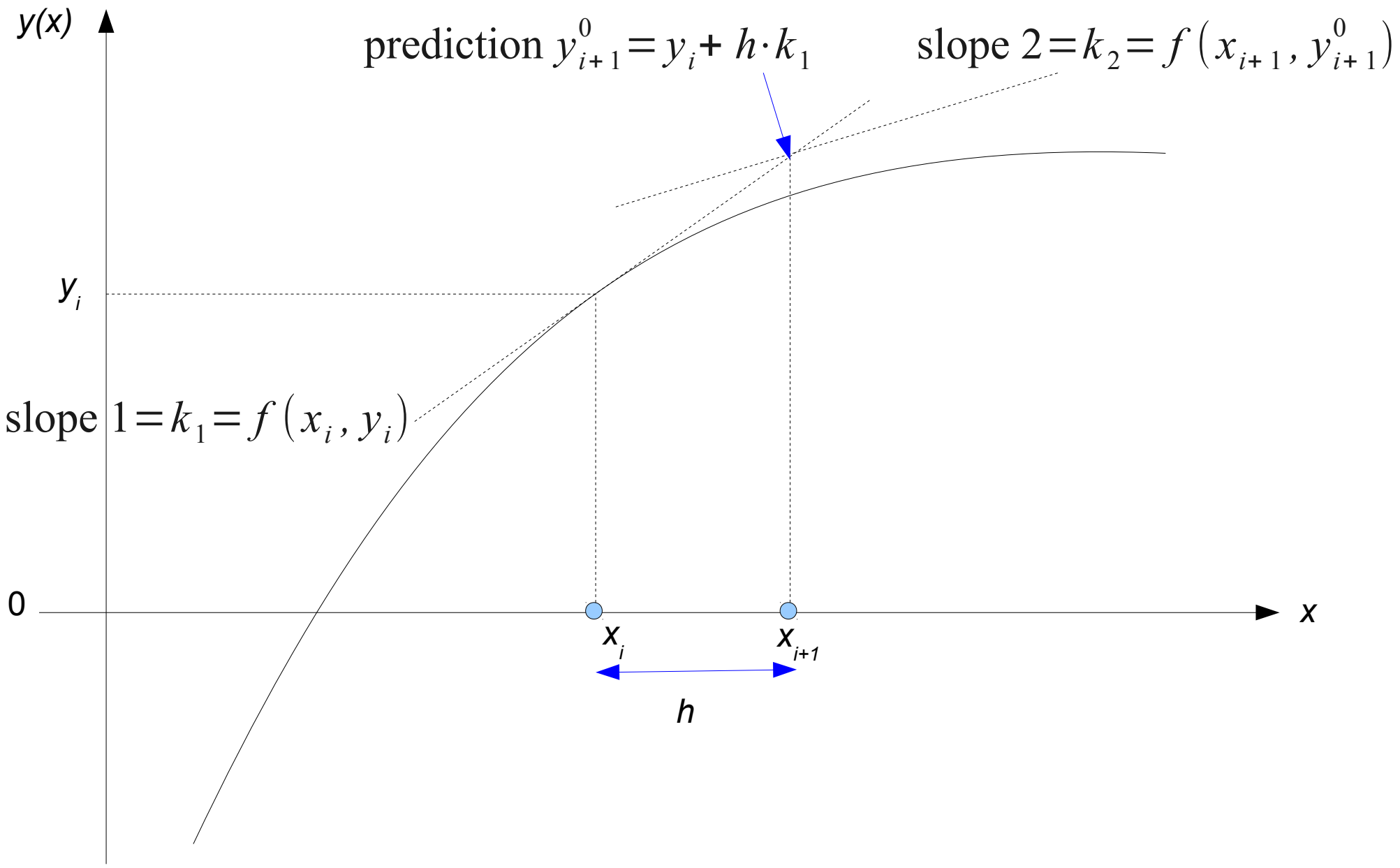
Error in Euler Algorithm

$$\frac{dy}{dx} = f(x, y)$$

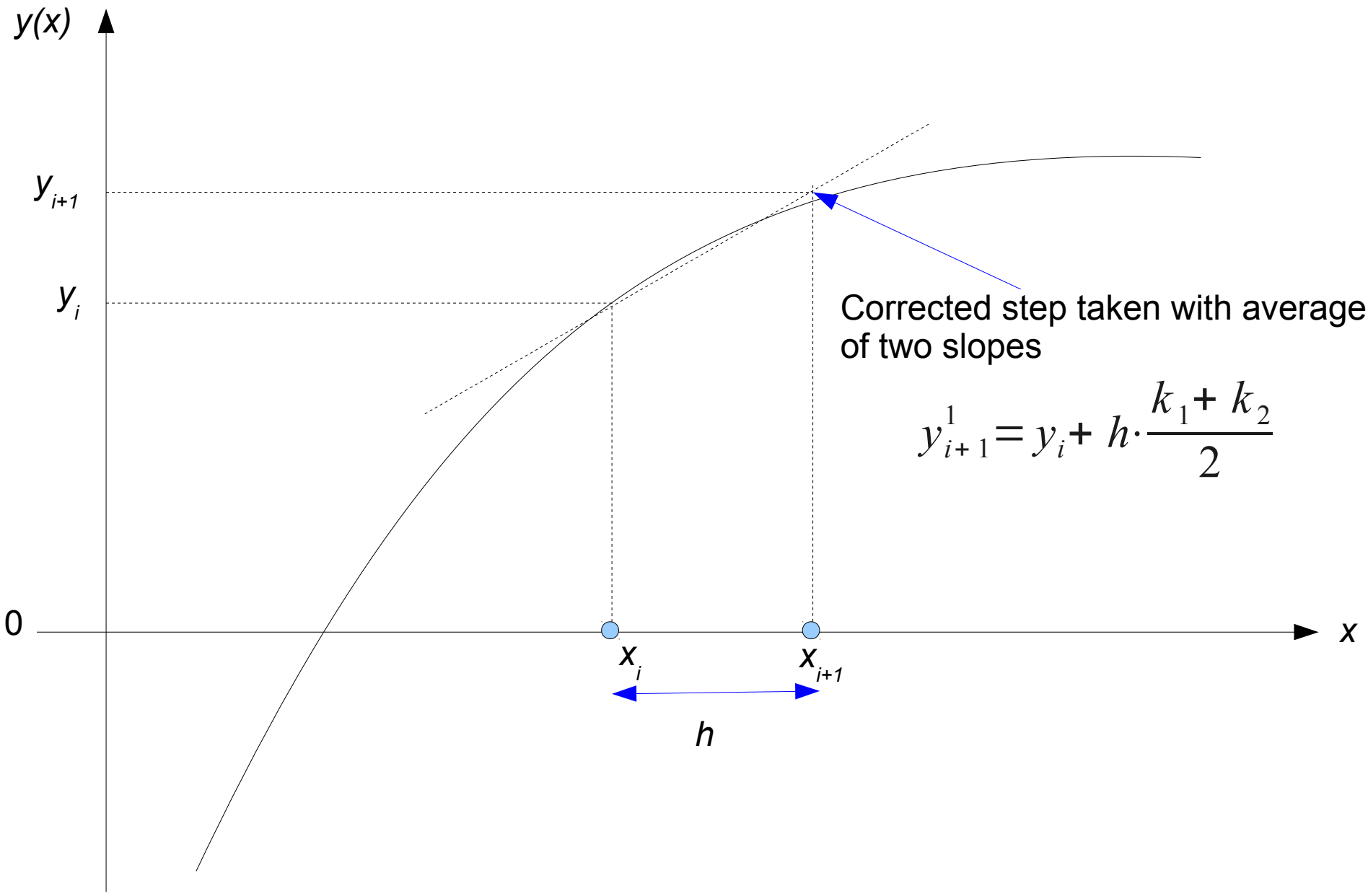
$$y_{i+1} = y_i + h \cdot f(x_i, y_i)$$



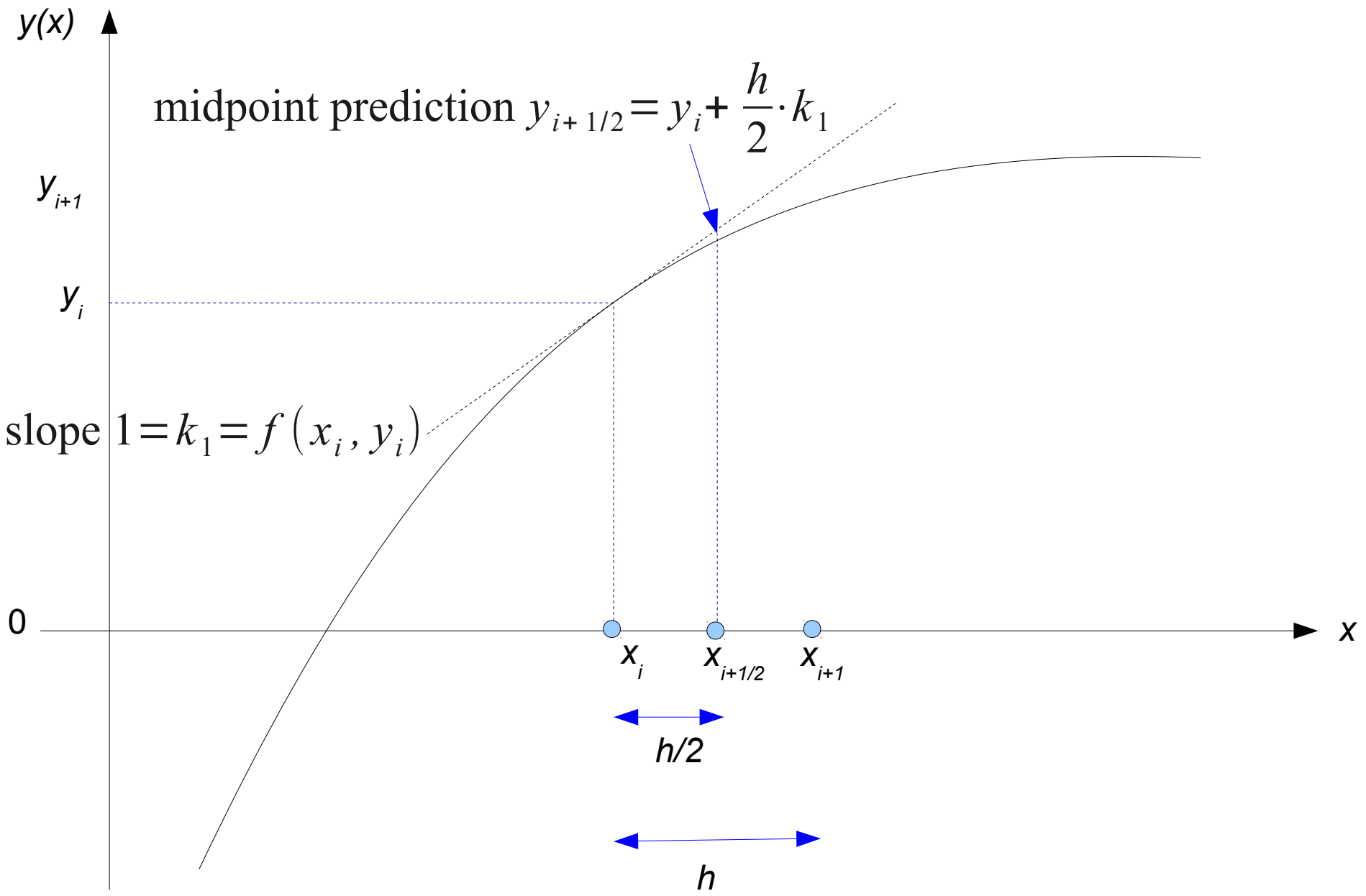
Heun's Method for Reducing Euler Algorithm Error



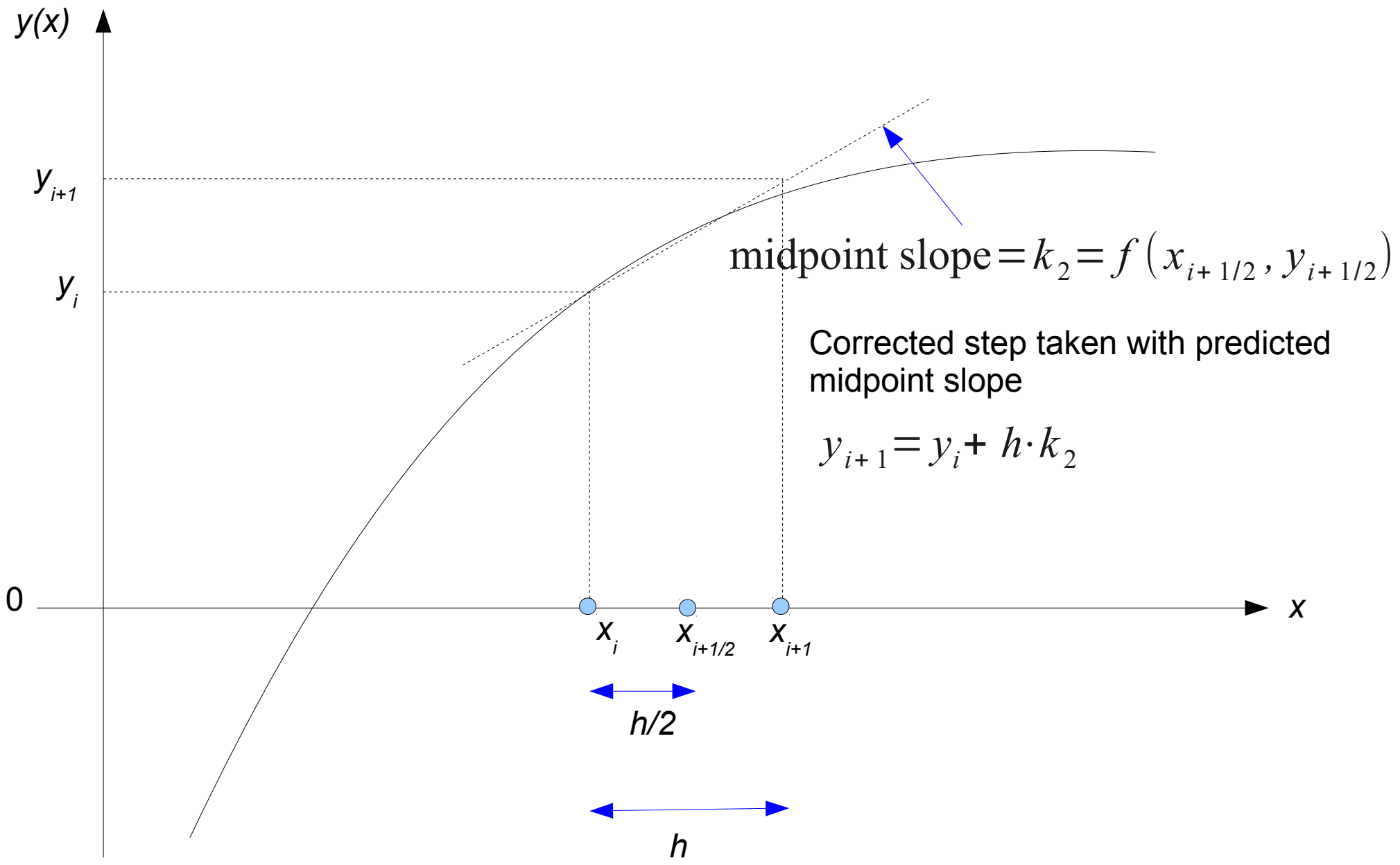
Heun's Method for Reducing Euler Algorithm Error



'Modified Euler Algorithm' with Reduced Error



'Modified Euler Algorithm' with Reduced Error



Second Order Runge-Kutta Algorithm

$$y(x+h) = y(x) + y'(x)h + \frac{y''(x)}{2}h^2 + E_3$$

Assume Taylor series terms through the second order can be made equivalent to

$$y(x+h) = y(x) + (a_1 k_1 + a_2 k_2)h$$

where $k_1 = f(x, y)$

and $k_2 = f(x + p_1 h \ , \ y + q_{11} k_1 h)$

Solve for a_1, a_2, p_1, q_{11}. This will generate one less equation than the number of coefficients to solve for, so an arbitrary choice is needed. Both Heun's method and the Modified Euler algorithm are special cases of the second order Runge-Kutta algorithm

Fourth Order Runge-Kutta Algorithm

$$y(x+h) = y(x) + y'(x)h + \frac{y''(x)}{2}h^2 + \frac{y'''(x)}{6}h^3 + \frac{y''''(x)}{24}h^4 + E_5$$

Assume Taylor series terms through the fourth order can be made equivalent to

$$y(x+h) = y(x) + (a_1 k_1 + a_2 k_2 + a_3 k_3 + a_4 k_4)h$$

result is $$y_{i+1} = y_i + \frac{h}{6}\cdot[k_1 + 2k_2 + 2k_3 + k_4]$$

where

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}\cdot k_1\right)$$

$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}\cdot k_2\right)$$

$$k_4 = f(x_i + h, y_i + h\cdot k_3)$$

System of First Order Ordinary Differential Eqns

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, y_3, \cdots, y_n)$$
$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, y_3, \cdots, y_n)$$
$$\frac{dy_3}{dx} = f_3(x, y_1, y_2, y_3, \cdots, y_n)$$
$$\vdots$$
$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, y_3, \cdots, y_n)$$

Note: Subscript on y variables above mean an index from among several state variables describing the state of the system, not an index for the iterative estimate.

Solving systems of First order ODEs is numerically much easier than, for example, systems of nonlinear static equations. The iterative Runge-Kutta or predictor-corrector algorithms may be applied to systems by simply taking a step for all y_i variables one at a time sequentially for each step of the independent variable.

Why look at solving systems of first order ODEs? Physics problems often involve vector quantities with simultaneous equations for the vector components. Also, a single ODE of higher order *n* can be transformed into a system of *n* first order ODEs!

Transforming Higher Order ODE into System of First Order ODEs

$$a_n \frac{d^n y}{dx^n} + \cdots + a_2 \frac{d^2 y}{dx^2} + a_1 \frac{dy}{dx} = g(x)$$

$$\text{Let} \quad y_1 = y \quad y_2 = \frac{dy_1}{dx} \quad y_3 = \frac{dy_2}{dx} \quad \cdots$$

Then this original high order ODE will be equivalent to the system:

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, y_3, \cdots, y_n)$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, y_3, \cdots, y_n)$$

$$\frac{dy_3}{dx} = f_3(x, y_1, y_2, y_3, \cdots, y_n)$$

$$\vdots$$

$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, y_3, \cdots, y_n)$$

Solving System of First Order ODEs

Notation: $y_{j,i+1} = y_{j,i} + \Delta y_j$

Increment of y_j needed for next step

i is iteration index stepping through independent variable *x* with step size *h*

j is index for state variables, 0 through *n*-1, where *n* is the order of the original differential equation

```
for j = 0 to n-1 do
  y_j ← initial condition
end for
for i = 1 to end do
  for j = 0 to n-1 do
    calculate Δy_j from all current y_j values
  end for
  for j = 0 to n-1 do
    y_j ← y_j + Δy_j
  end for
  output x, all y_j
end for
```

Be careful to not disturb any of the current y_j values until all the increments for the next step have been calculated!

C Code for Runge-Kutta Algorithms

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

int rk2(int order,void (*func)(int n,double x,double y[],double yp[]),double
xstart,double xstop,double xinc,double y[]) {
  double x,xhalfinc,xnext,*ymid;
  double *k1,*k2;   /*Runge-Kutta slope values*/
  int j;
  ymid = (double *) malloc(order * sizeof(double));
  k1 = (double *) malloc(order * sizeof(double));
  k2 = (double *) malloc(order * sizeof(double));
  xstop = xstop + (xinc * 0.5);
  xhalfinc = xinc * 0.5;
  x = xstart;
  while (1) {
    printf("%.8g ",x);   /*output independent variables*/
    for (j = 0; j < order; j++) printf("%.8g ",y[j]);   /*output state value*/
    putchar('\n');
    xnext = x + xinc;
    if (((xinc > 0.0) && (xnext > xstop)) || ((xinc < 0.0) && (xnext < xstop))) break;
    (*func)(order,x,y,k1);   /*calculate RK2 (Improved polygon method) algorithm step*/
    for (j = 0; j < order; j++) ymid[j] = y[j] + (xhalfinc * k1[j]);
    (*func)(order,x + xhalfinc,ymid,k2);
    for (j = 0; j < order; j++) y[j] += xinc * k2[j];
    x = xnext;
  }
  free(ymid);
  free(k1);
  free(k2);
  return(0);
}
```

C Code for Runge-Kutta Algorithms

```
int rk4(int order,void (*func)(int n,double x,double y[],double yp[]),double xstart,double xstop,double
xinc,double y[]) {
  double x,xhalfinc,xnext,*ymid;
  double *k1,*k2,*k3,*k4;   /*Runge-Kutta slope values*/
  int j;

  ymid = (double *) malloc(order * sizeof(double));
  k1 = (double *) malloc(order * sizeof(double));
  k2 = (double *) malloc(order * sizeof(double));
  k3 = (double *) malloc(order * sizeof(double));
  k4 = (double *) malloc(order * sizeof(double));
  xstop = xstop + (xinc * 0.5);
  xhalfinc = xinc * 0.5;
  x = xstart;
  while (1) {
    printf("%.8g ",x);   /*output independent variables*/
    for (j = 0; j < order; j++) printf("%.8g ",y[j]);   /*output state value*/
    putchar('\n');
    xnext = x + xinc;
    if (((xinc > 0.0) && (xnext > xstop)) || ((xinc < 0.0) && (xnext < xstop))) break;
    (*func)(order,x,y,k1);   /*calculate RK4 algorithm step*/
    for (j = 0; j < order; j++) ymid[j] = y[j] + (xhalfinc * k1[j]);
    (*func)(order,x + xhalfinc,ymid,k2);
    for (j = 0; j < order; j++) ymid[j] = y[j] + (xhalfinc * k2[j]);
    (*func)(order,x + xhalfinc,ymid,k3);
    for (j = 0; j < order; j++) ymid[j] = y[j] + (xinc * k3[j]);
    (*func)(order,x + xinc,ymid,k4);
    for (j = 0; j < order; j++) y[j] += xinc * (k1[j] + (2.0 * k2[j]) + (2.0 * k3[j]) + k4[j]) / 6.0;
    x = xnext;
  }
  free(ymid);
  free(k1);
  free(k2);
  free(k3);
  free(k4);
  return(0);
}
```

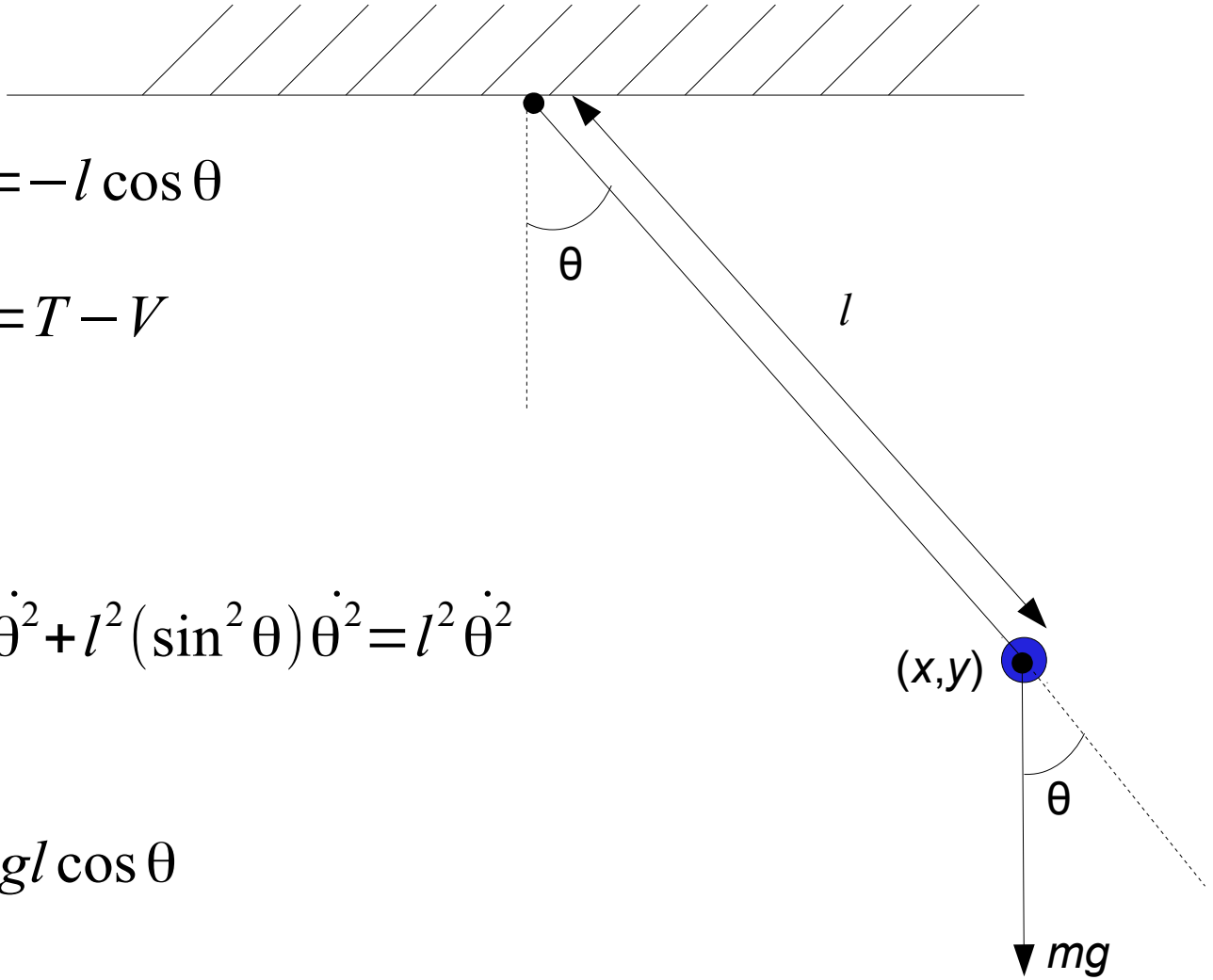
Fourth Order Adams-Bashforth-Moulton Algorithm

Predict $$y_{i+1} = y_i + \frac{h}{24} \cdot \left(55y'_i - 59y'_{i-1} + 37y'_{i-2} - 9y'_{i-3} \right)$$

Evaluate $$y'_{i+1} = f\left(x + h, y_{i+1} \right)$$

Correct $$y_{i+1} = y_i + \frac{h}{24} \cdot \left(9y'_{i+1} + 19y'_i - 5y'_{i-1} + y'_{i-2} \right)$$

Lagrangian Derivation of Single Pendulum



$$x = l\sin\theta \qquad y = -l\cos\theta$$

Lagrangian $L = T - V$

$$T = \frac{m|\vec{v}|^2}{2}$$

$$|\vec{v}|^2 = \dot{x}^2 + \dot{y}^2$$

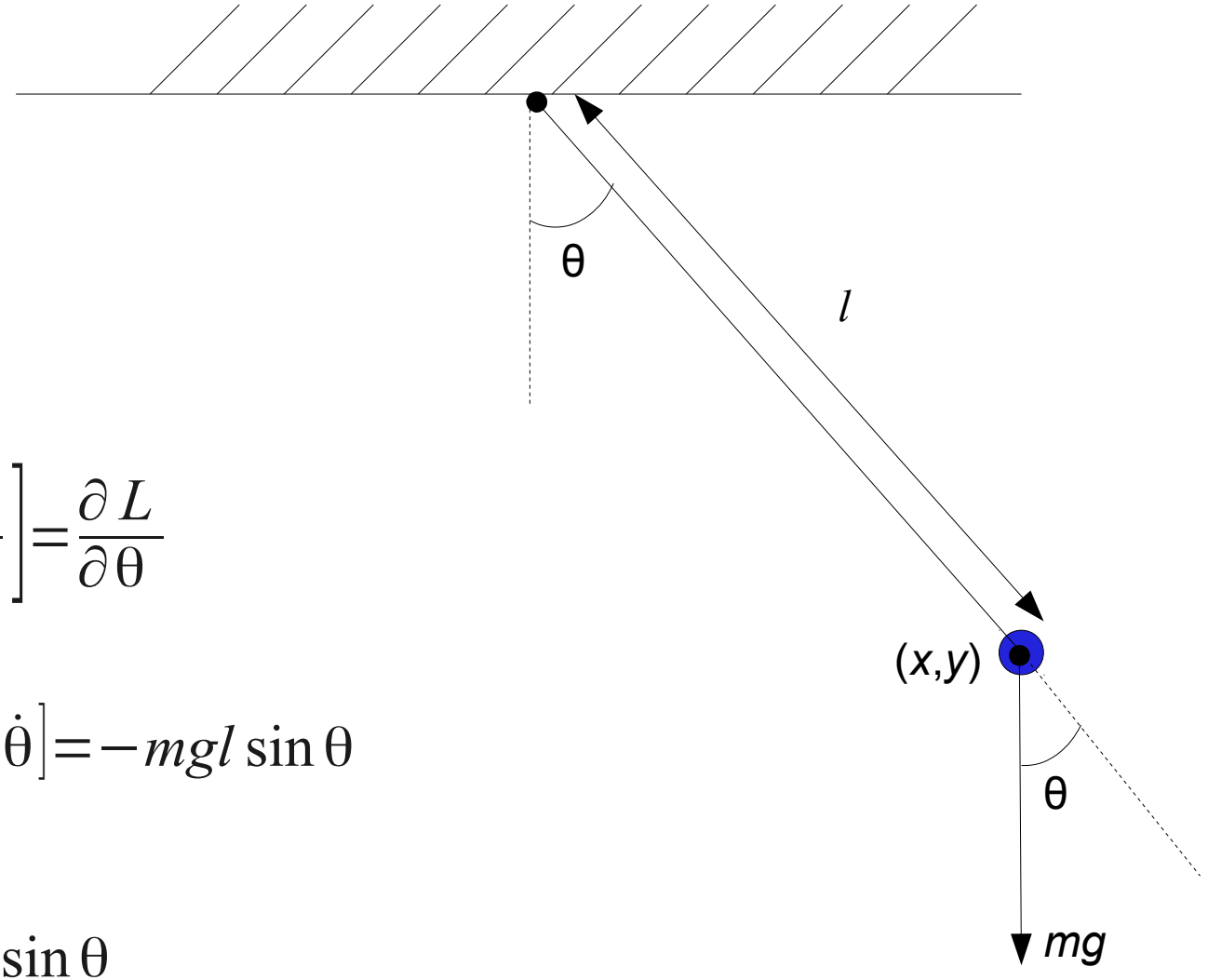
$$= l^2(\cos^2\theta)\dot{\theta}^2 + l^2(\sin^2\theta)\dot{\theta}^2 = l^2\dot{\theta}^2$$

$$T = \frac{m l^2}{2}\dot{\theta}^2$$

$$V = mgy = -mgl\cos\theta$$

$$L = \frac{m l^2}{2}\dot{\theta}^2 + mgl\cos\theta$$

Lagrangian Derivation of Single Pendulum



$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial L}{\partial \dot{\theta}}\right]=\frac{\partial L}{\partial \theta}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[ml^2\dot{\theta}\right]=-mgl\sin\theta$$

$$\ddot{\theta}=-\frac{g}{l}\sin\theta$$

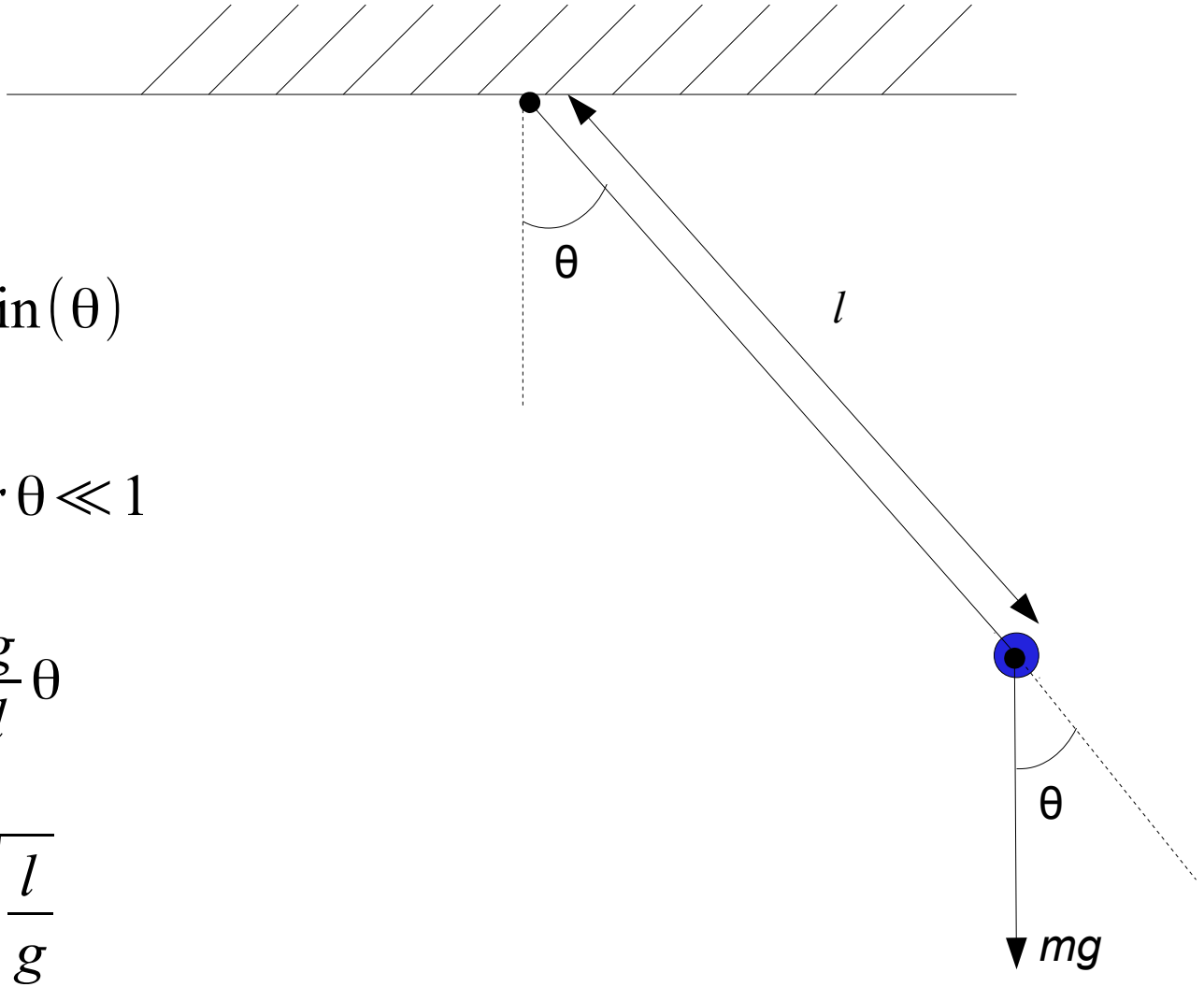
Approximate Analytic Solution of Pendulum

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta)$$

$$\sin(\theta) \approx \theta \; for \; \theta \ll 1$$

$$\frac{d^2\theta}{dt^2} \approx -\frac{g}{l}\theta$$

$$T_0 = 2\pi\sqrt{\frac{l}{g}}$$



Differential equation is linear only for deflection angle θ << 1

Analytic Solution of Large Angle Pendulum

Either a numerical evaluation of an elliptical integral must be performed, or for simply the period, a numerical evaluation of an infinite series:

$$\frac{T}{T_0} = 1 + \sum_{n=1}^{\infty} \left(\frac{(2n)!}{2^{2n}(n!)^2} \right)^2 \sin^{2n}\left(\frac{\theta_0}{2} \right)$$

Or simply solve the nonlinear differential equation of motion numerically

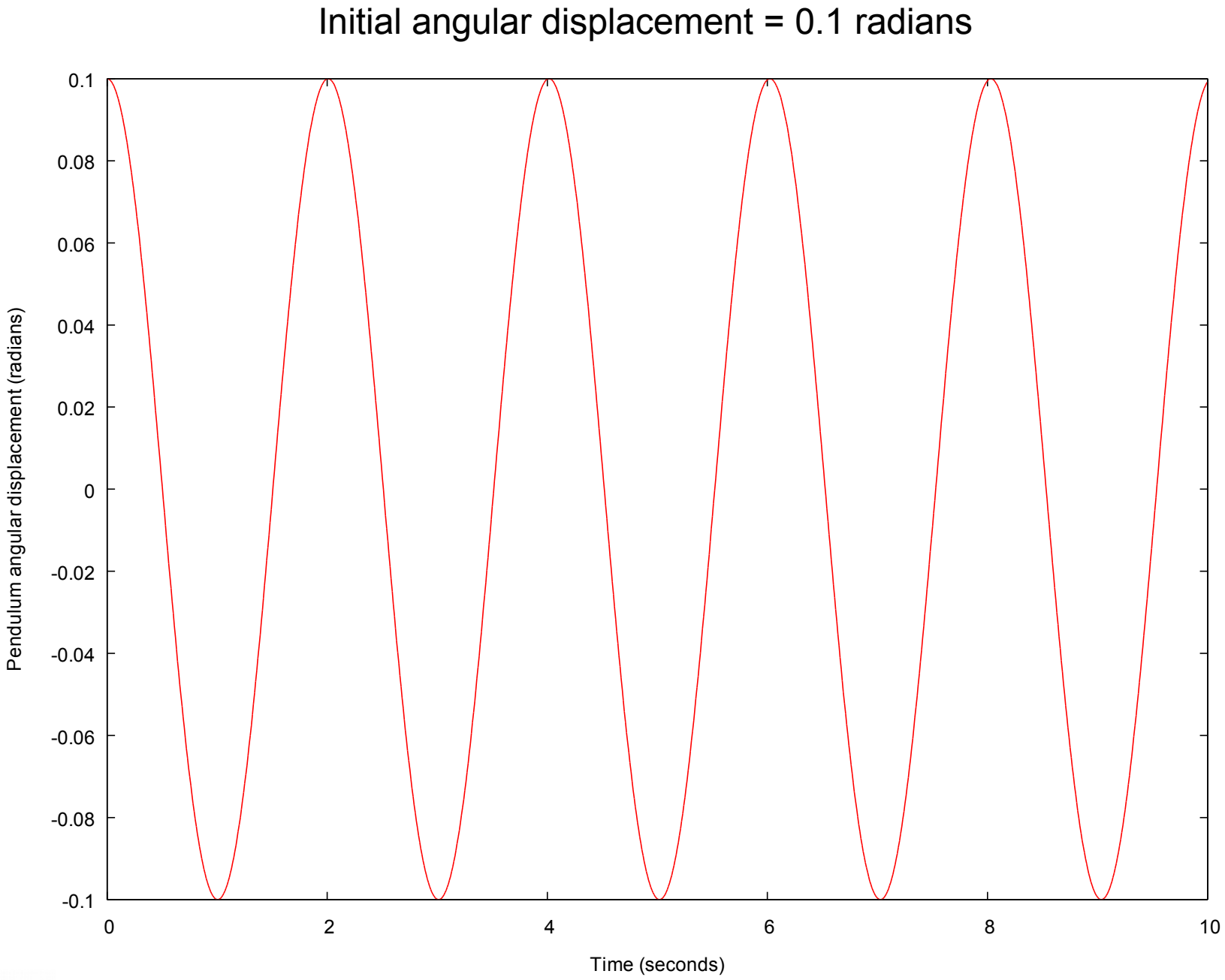
Transform to System of First Order ODEs

Original 2nd order ODE is $$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta)$$

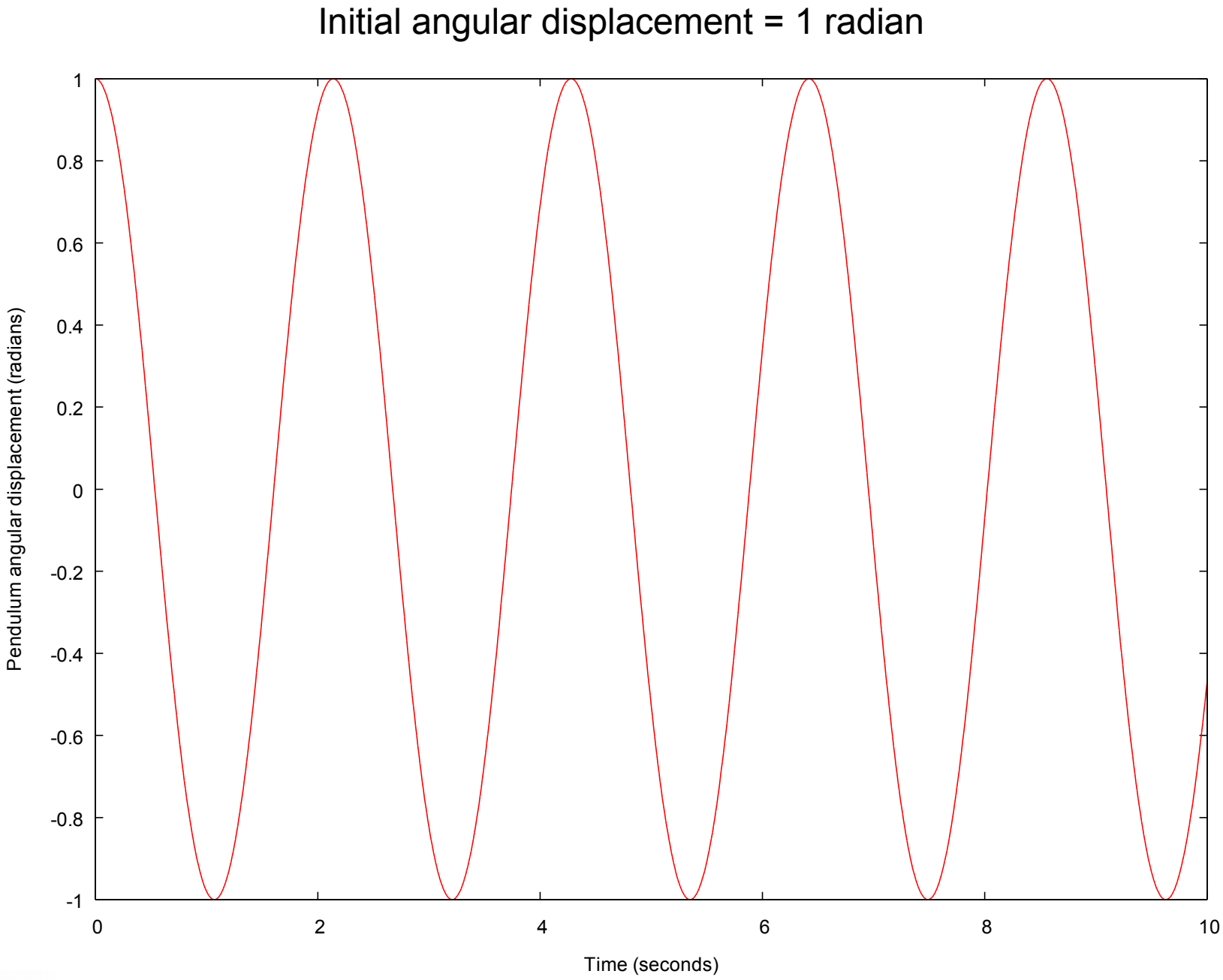
Let $y_1 = \theta$ $y_2 = \frac{dy_1}{dt}$

So system becomes
$$\frac{dy_1}{dt} = y_2$$
$$\frac{dy_2}{dt} = -\frac{g}{l}\sin(y_1)$$

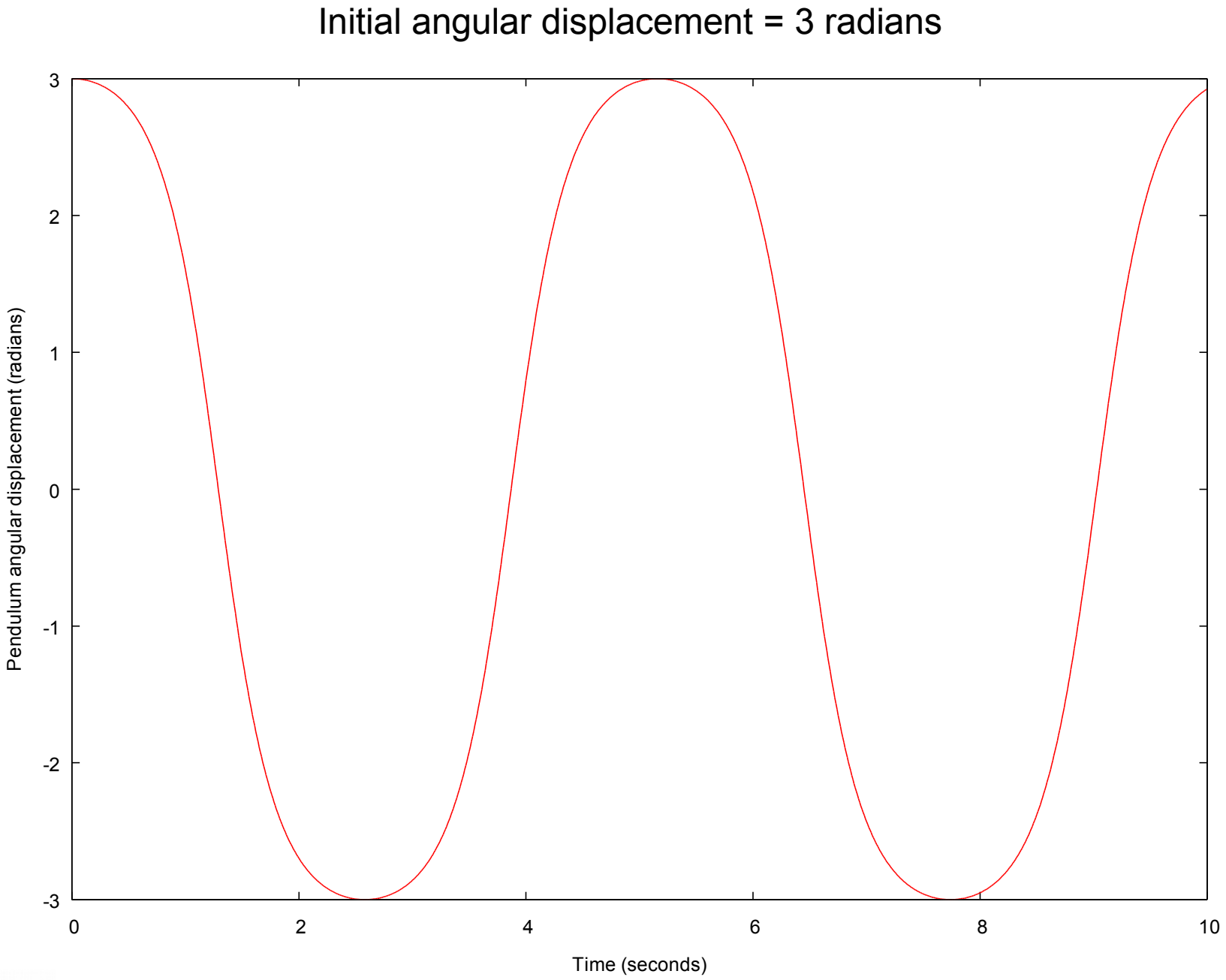
Solution of Pendulum Equations of Motion



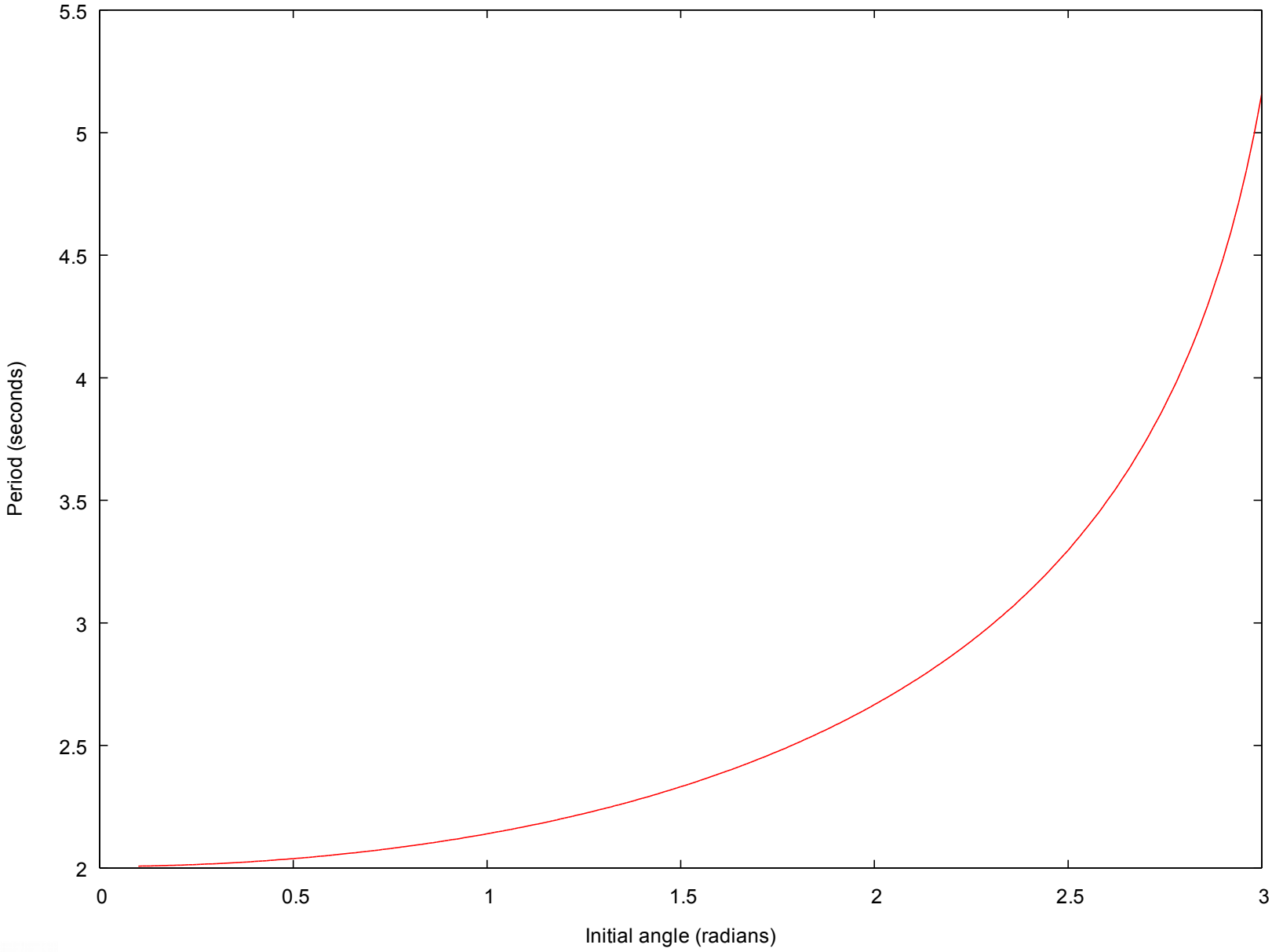
Solution of Pendulum Equations of Motion



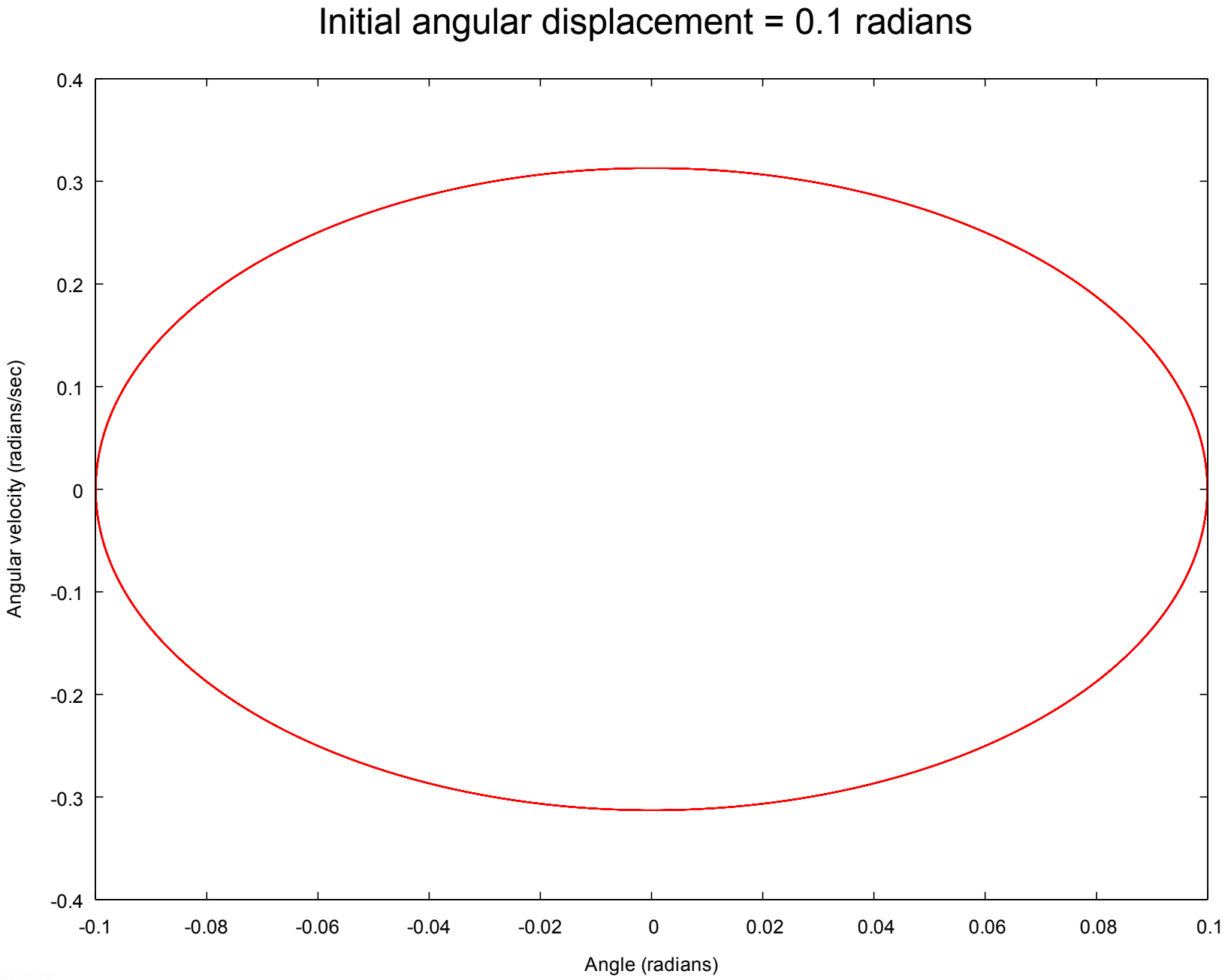
Solution of Pendulum Equations of Motion



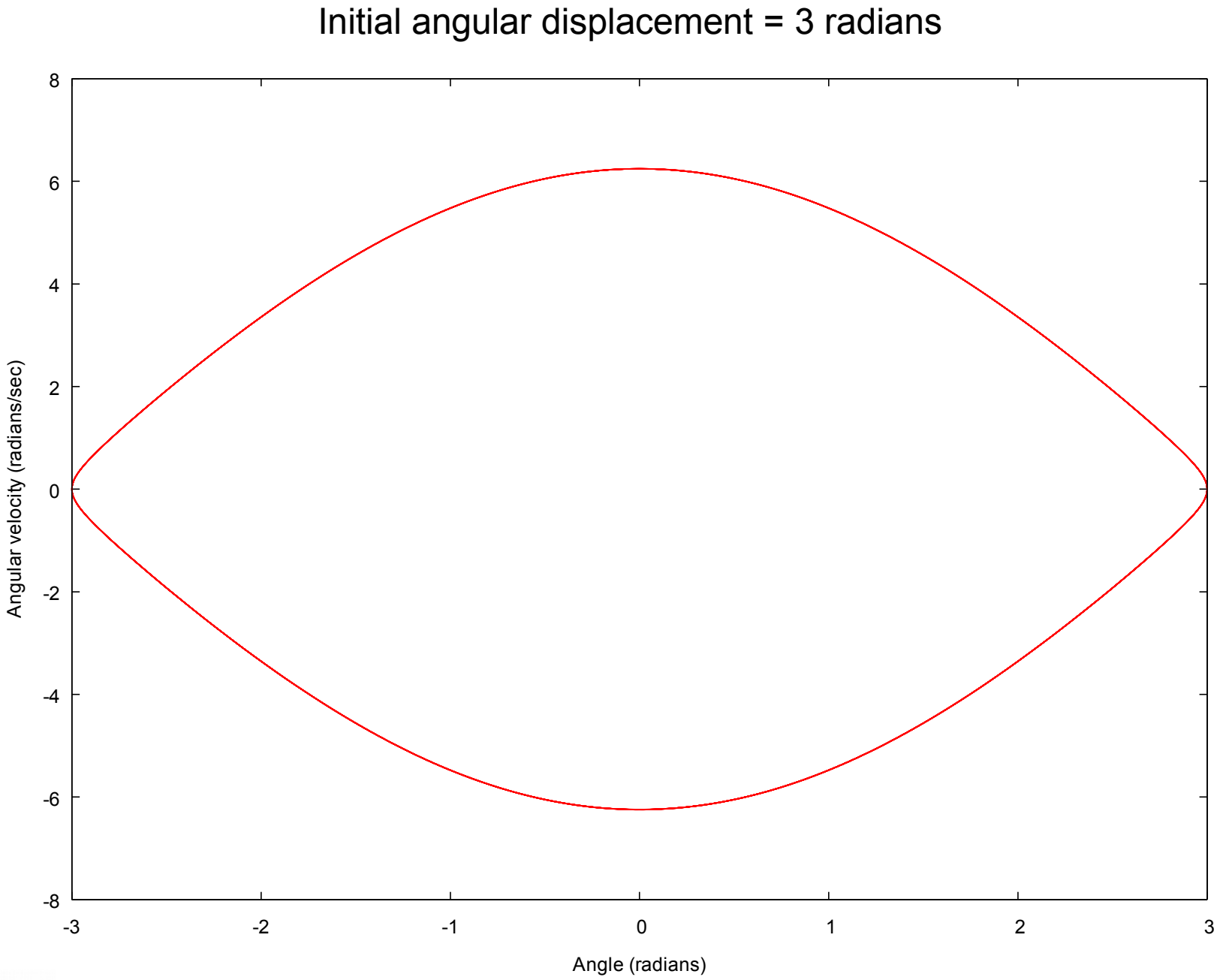
Dependence of Period on Initial Angle



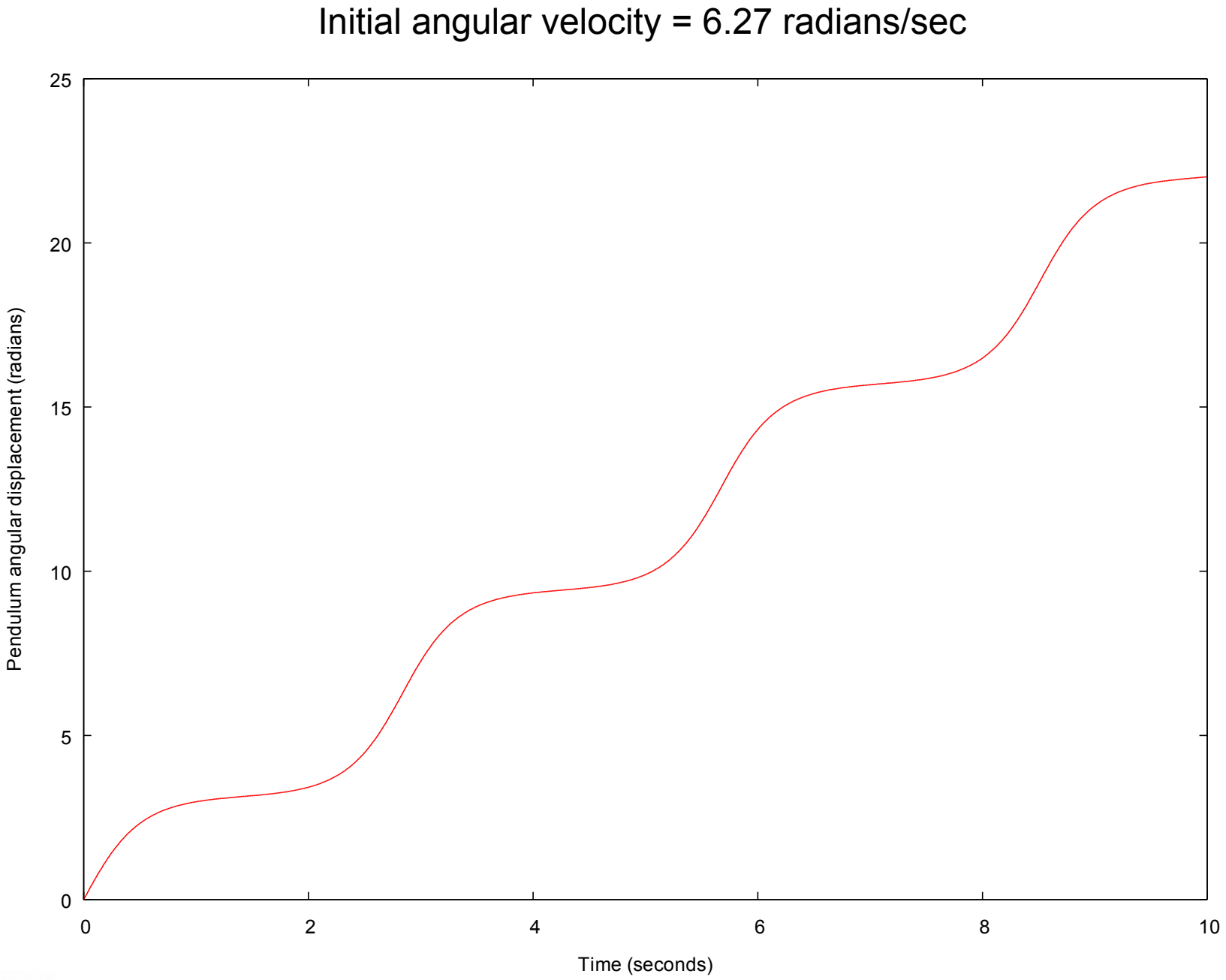
Periodic Pendulum Motion in State-Space



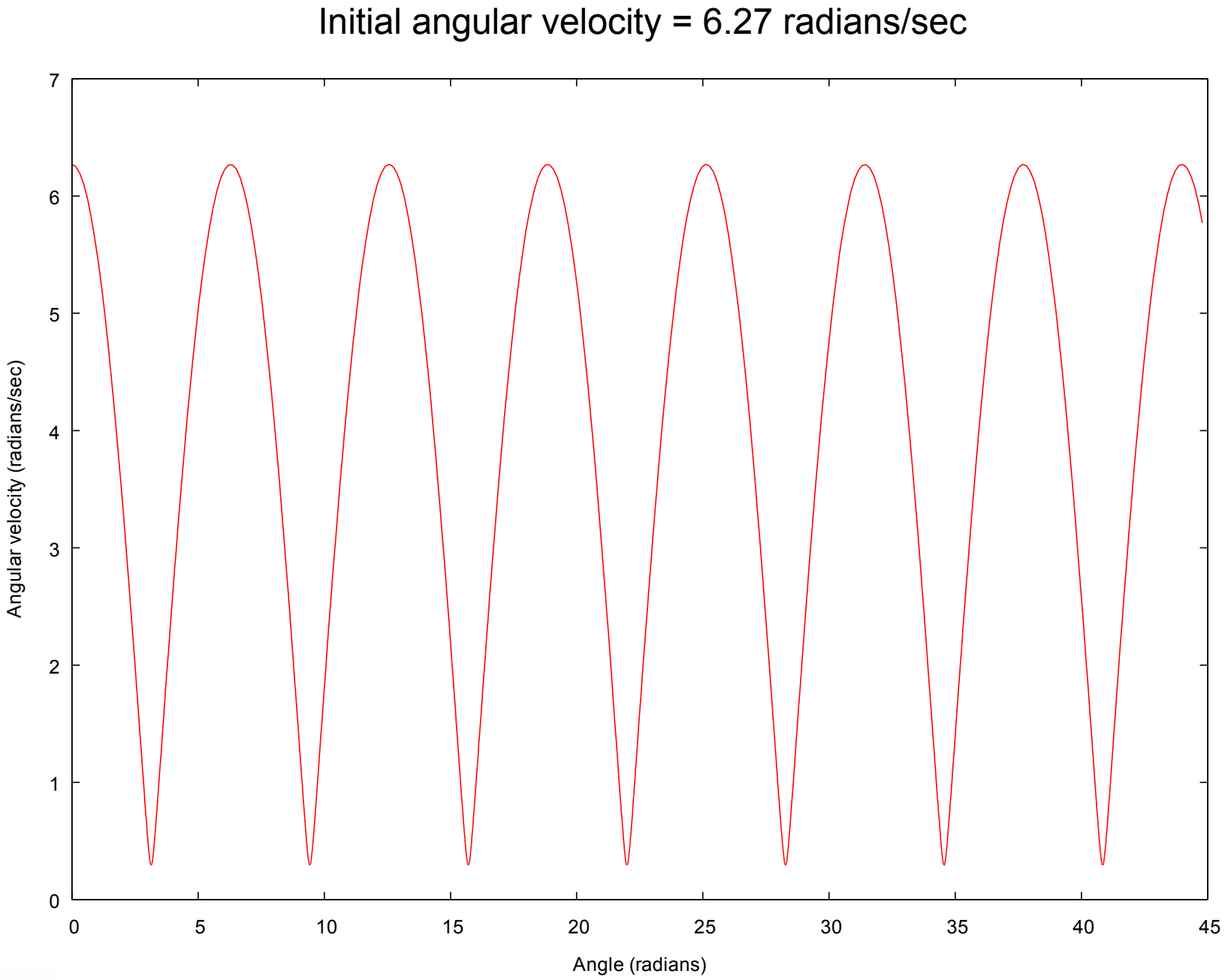
Periodic Pendulum Motion in State-Space



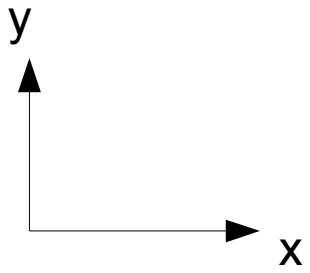
Solution of Pendulum Equations of Motion



Periodic Pendulum Motion in State-Space

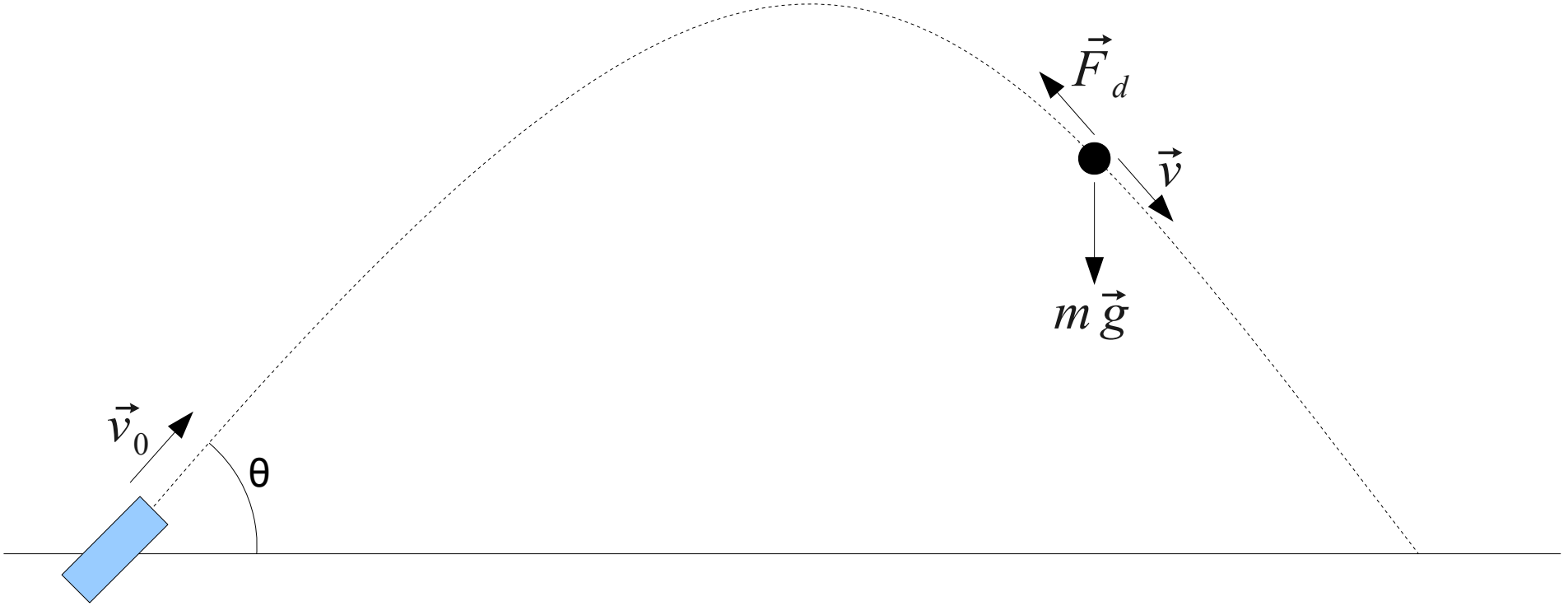


Ballistics With Air Drag



$$|\vec{F}_d| = \frac{C_d \rho A}{2} |\vec{v}|^2 \text{ and } \vec{F}_d \text{ directed opposite } \vec{v}$$

$$\text{Let } \frac{C_d \rho A}{2} = \alpha \quad \text{then} \quad \vec{F}_d = -\alpha |\vec{v}| \vec{v}$$



Ballistics With Air Drag

Ballistic object has position vector $\vec{r}$ velocity vector $\vec{v}$ and mass m

Force on object is $\vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt} = m\frac{d^2\vec{r}}{dt^2}$

$$m\frac{d^2\vec{r}}{dt^2} = -\alpha|\vec{v}|\vec{v} - m\vec{g}$$

$$\frac{d^2\vec{r}}{dt^2} = -\frac{\alpha|\vec{v}|\vec{v}}{m} - \vec{g}$$

If system is solved in two dimensions

$$\frac{d^2 r_x}{dt^2} = -\frac{\alpha|\vec{v}|v_x}{m} \qquad \frac{d^2 r_y}{dt^2} = -\frac{\alpha|\vec{v}|v_y}{m} - g$$

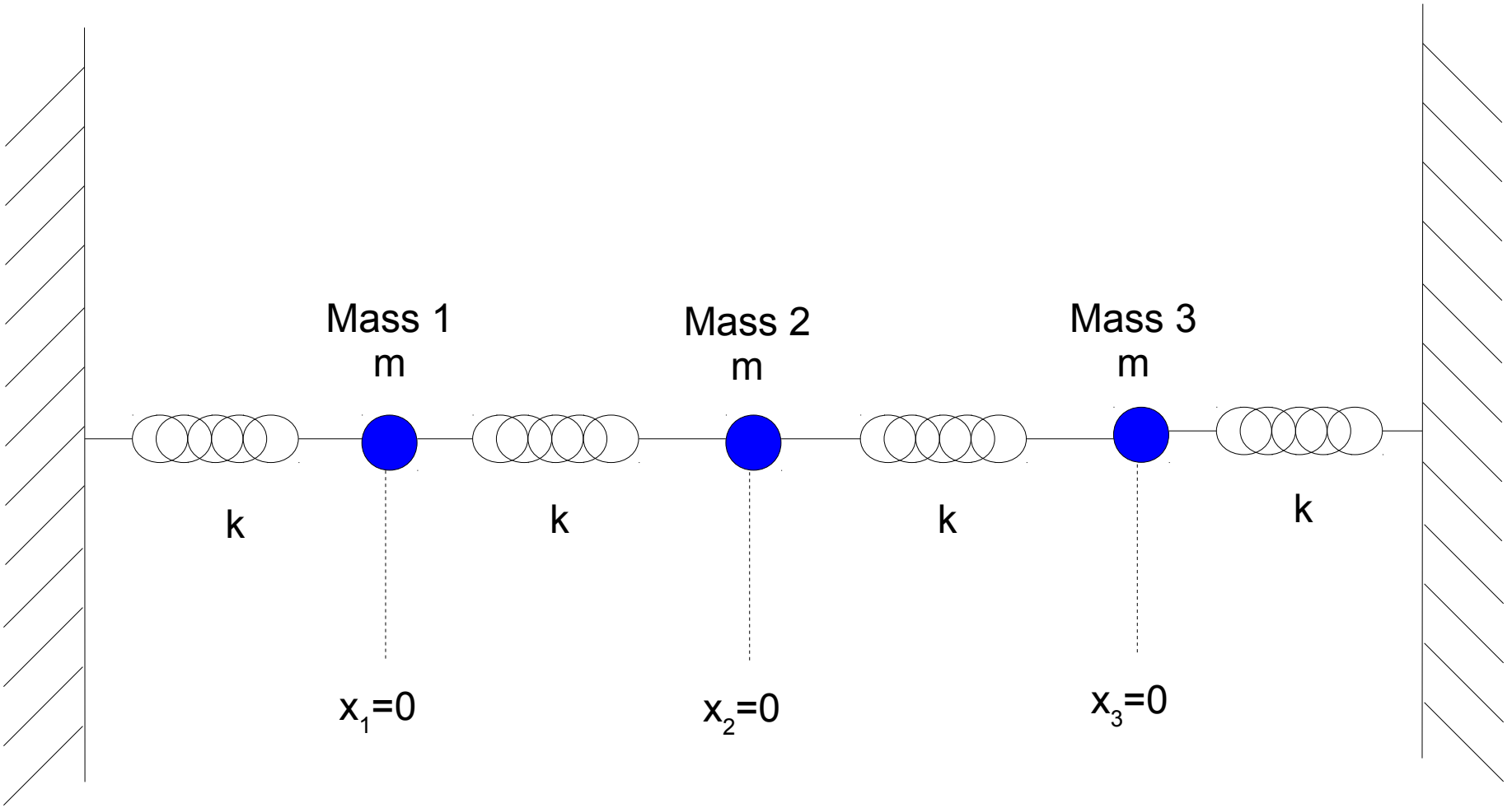
Transform to System of First Order ODEs

Let $y_1 = r_x \quad y_2 = r_y \quad y_3 = \frac{dy_1}{dt} = v_x \quad y_4 = \frac{dy_2}{dt} = v_y$

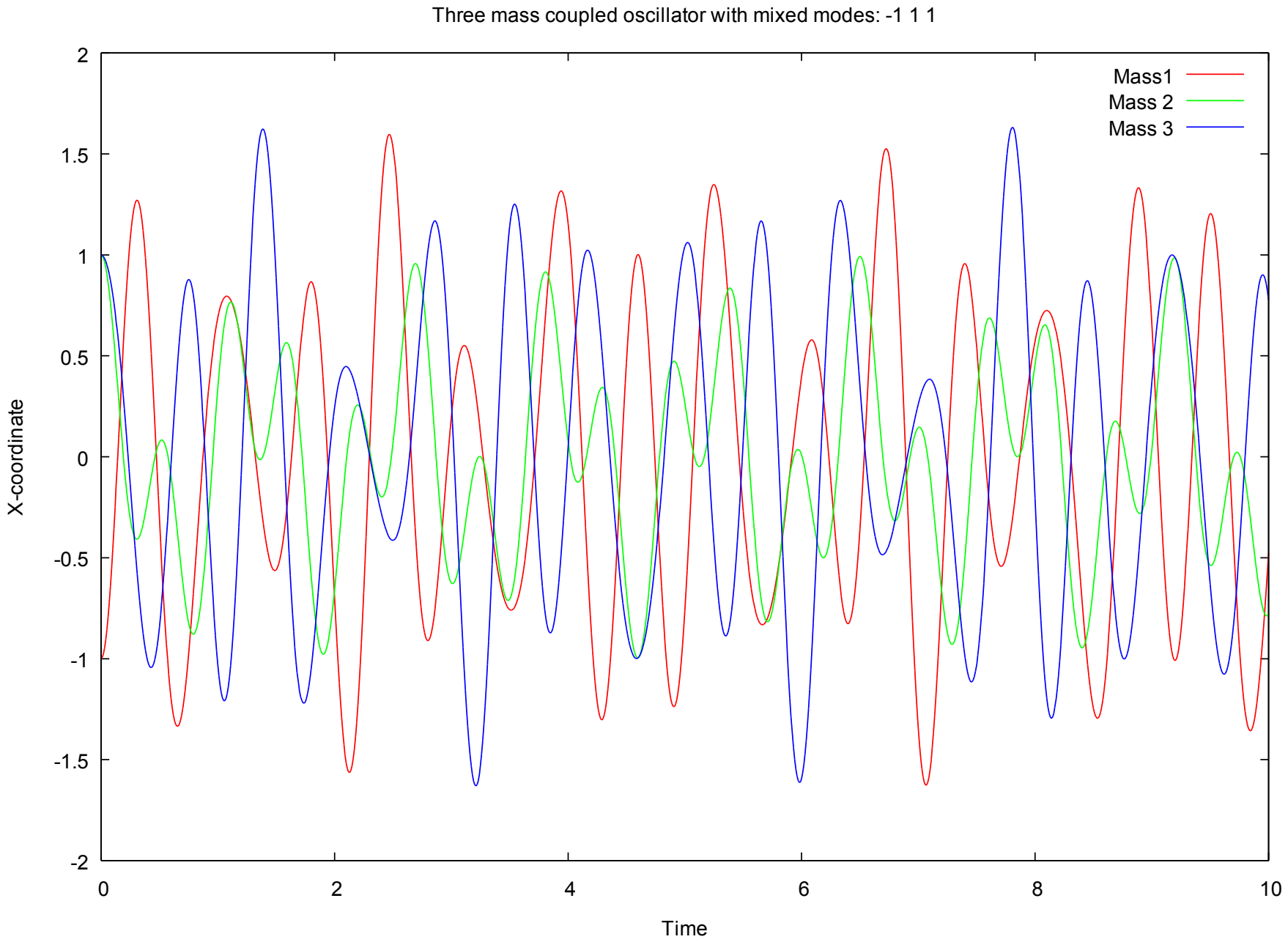
So system becomes

$$\begin{aligned}
\frac{dy_1}{dt} &= y_3 \\
\frac{dy_2}{dt} &= y_4 \\
\frac{dy_3}{dt} &= -\frac{\alpha |\vec{v}| x_x}{m} \\
\frac{dy_4}{dt} &= -\frac{\alpha |\vec{v}| v_y}{m} - g
\end{aligned}$$

Mass/Spring Coupled Oscillator

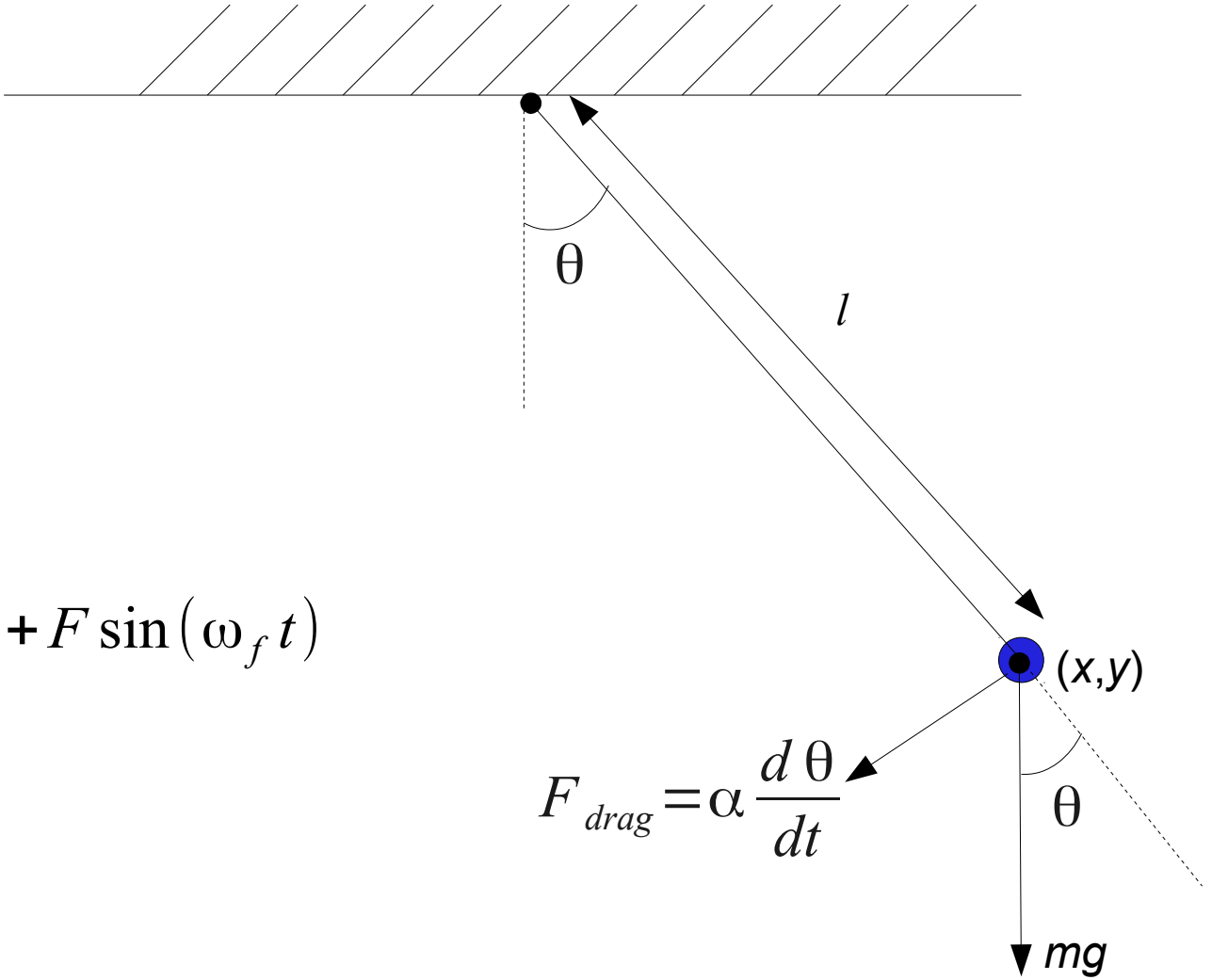


Complex But Not Chaotic Oscillator



Add Damping and Forcing to Single Pendulum

$$\ddot{\theta} = -\frac{g}{l}\sin\theta - \alpha\dot{\theta} + F\sin\left(\omega_f t\right)$$



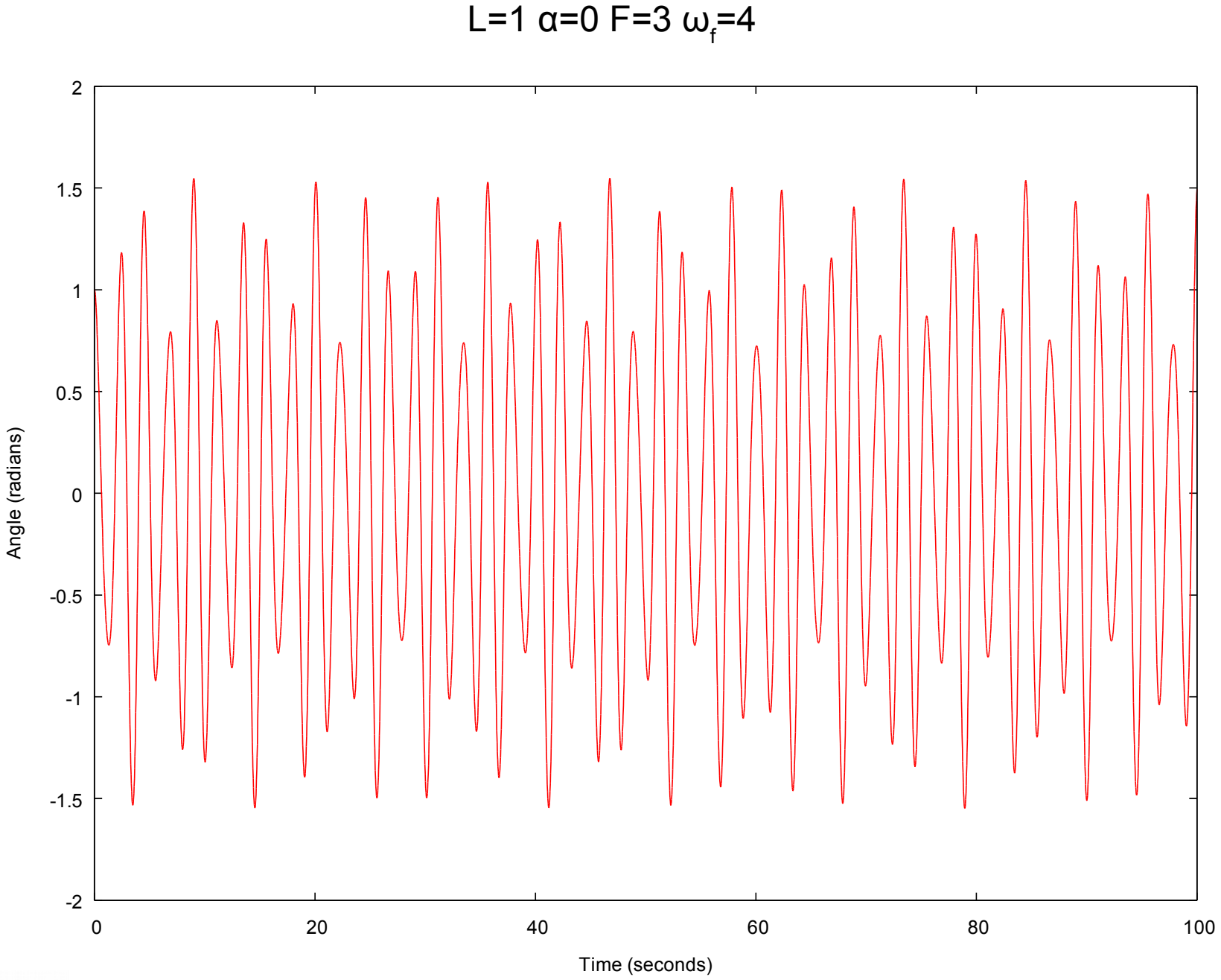
Transform to System of First Order ODEs

Original 2nd order ODE is $$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta) - \alpha\frac{d\theta}{dt} + F\sin(\omega_f t)$$

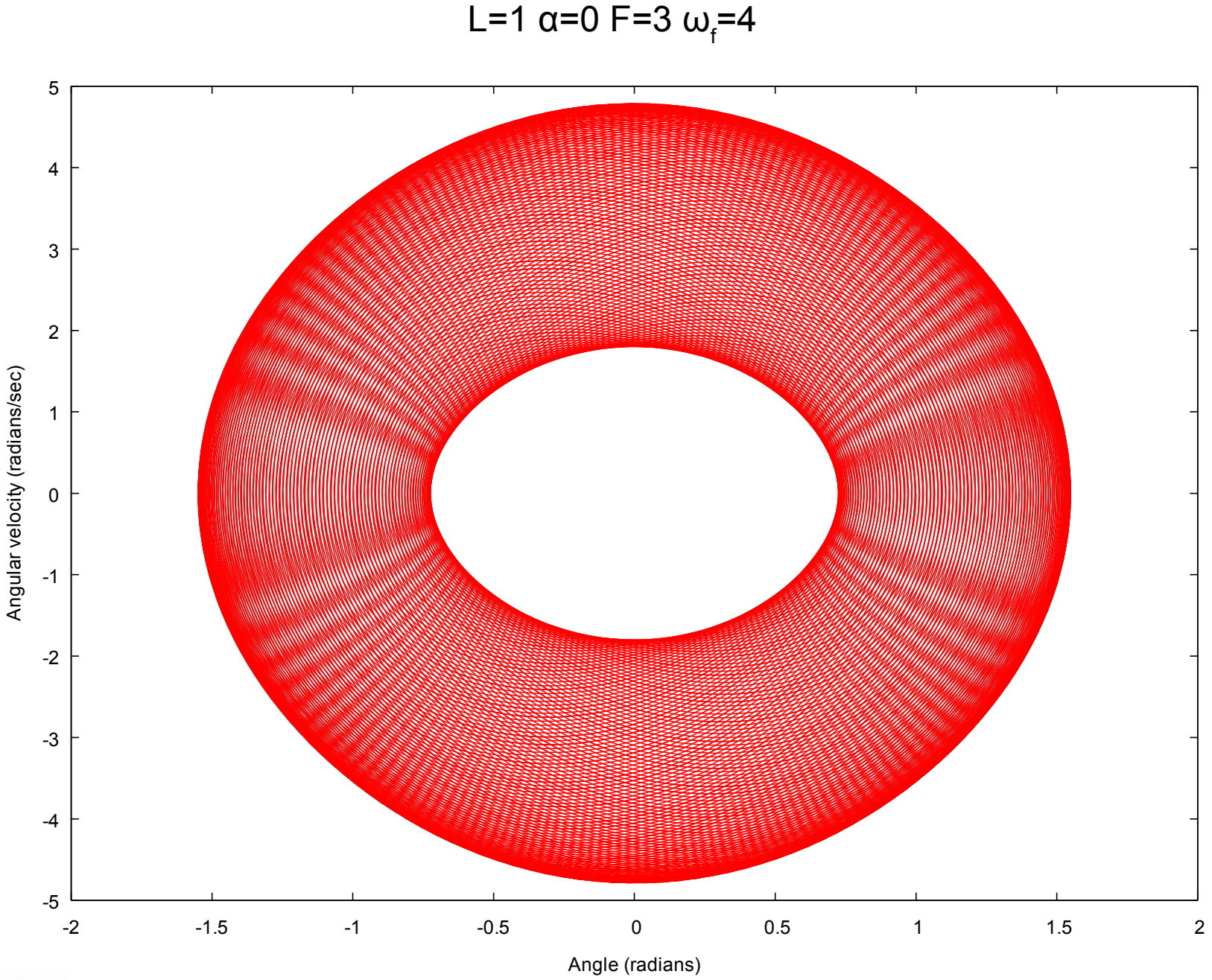
Let $y_1 = \theta$ $y_2 = \frac{dy_1}{dt}$

So system becomes
$$\frac{dy_1}{dt} = y_2$$
$$\frac{dy_2}{dt} = -\frac{g}{l}\sin(y_1) - \alpha y_2 + F\sin(\omega_f t)$$

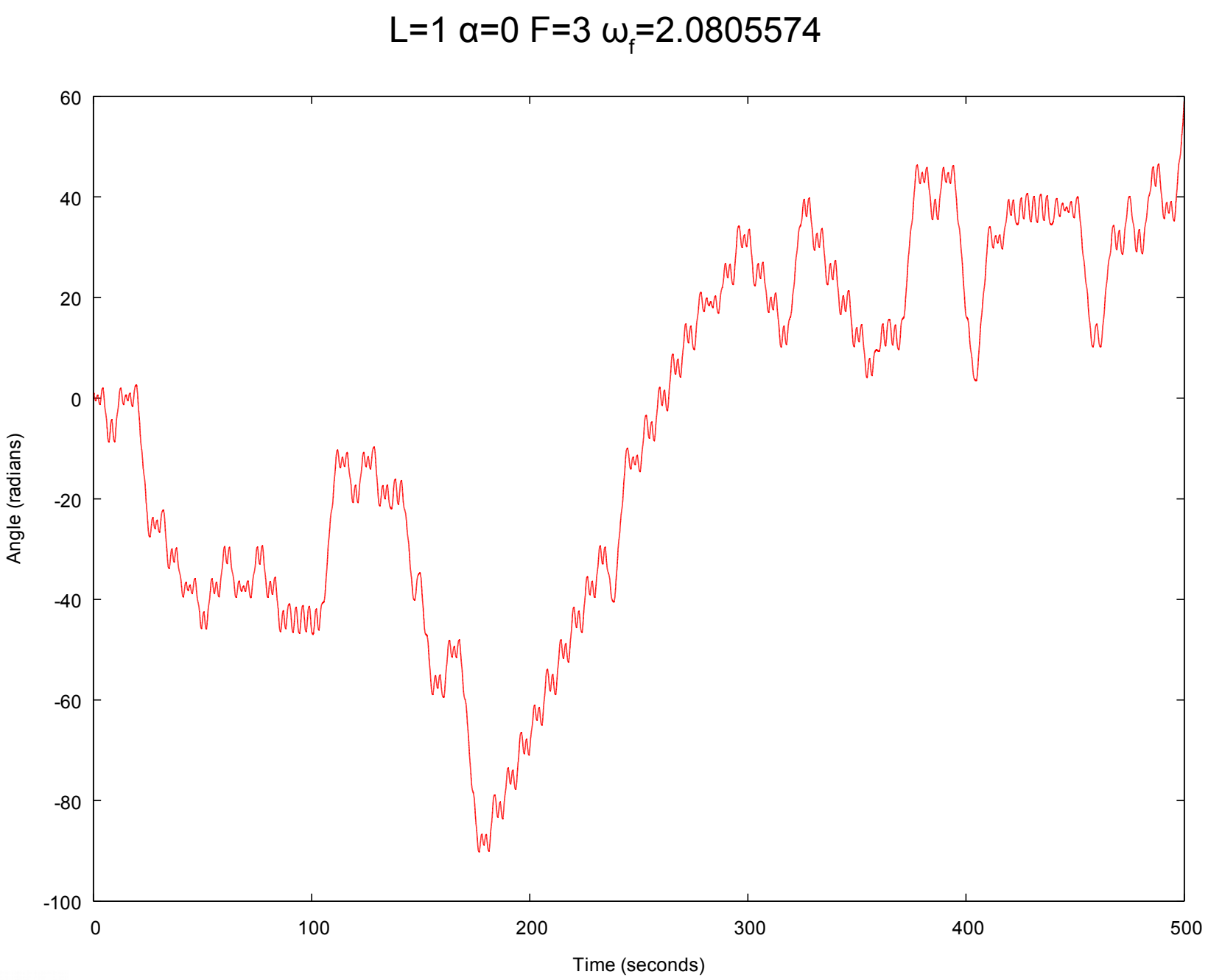
Example Quasi-Periodic Behavior



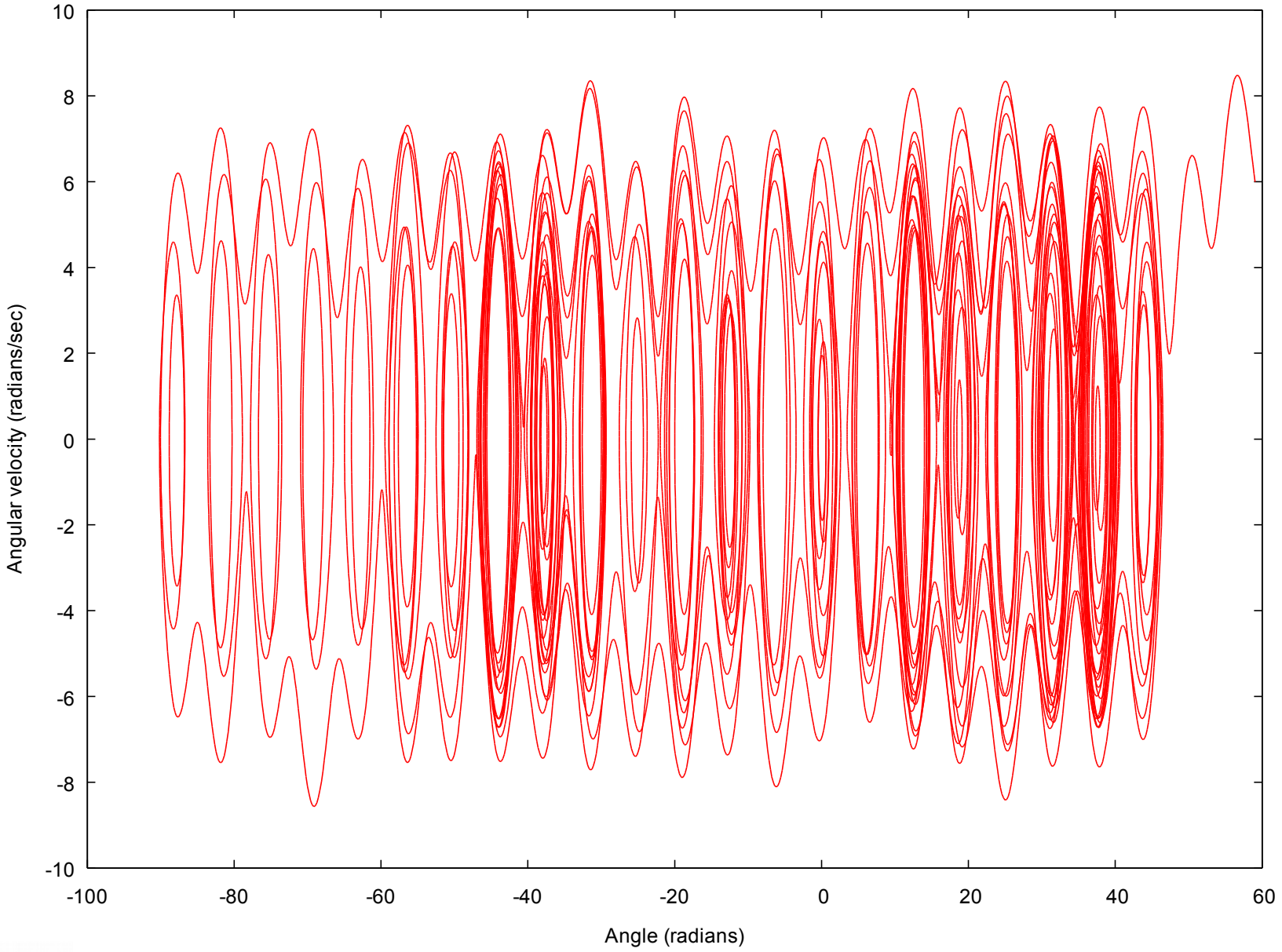
Quasi-Periodic Behavior in State-Space



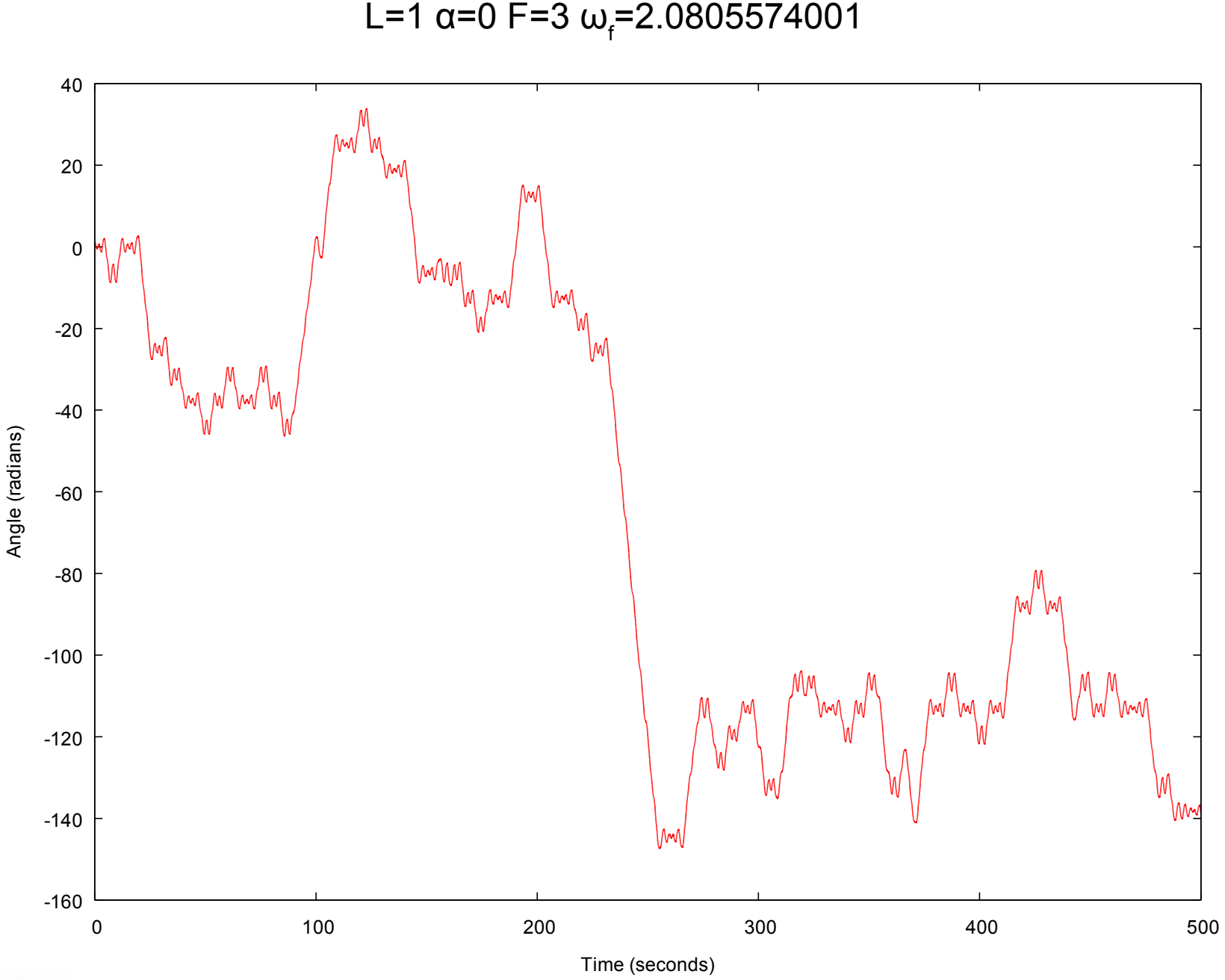
Example Chaotic Behavior



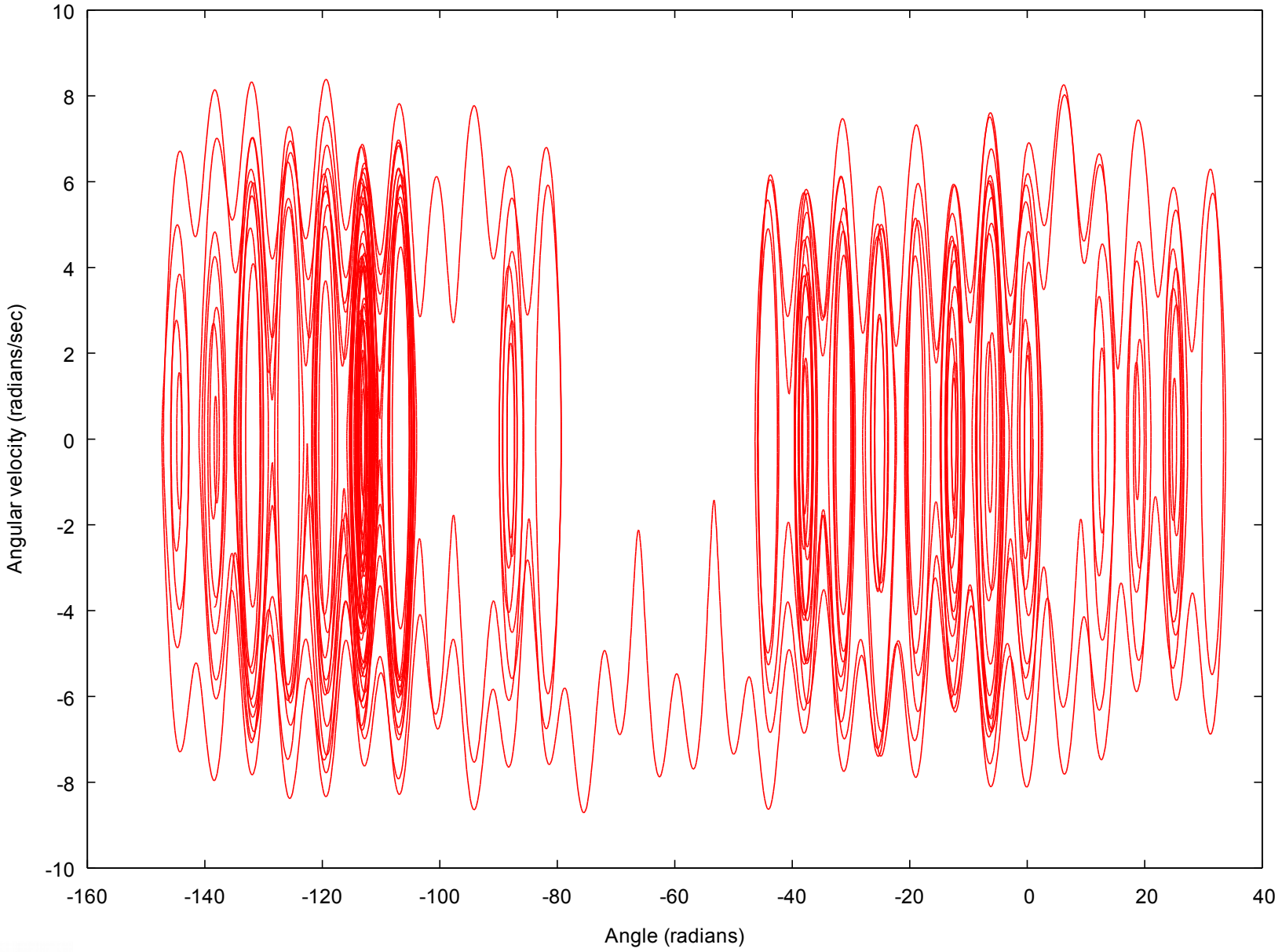
Example Chaotic Behavior



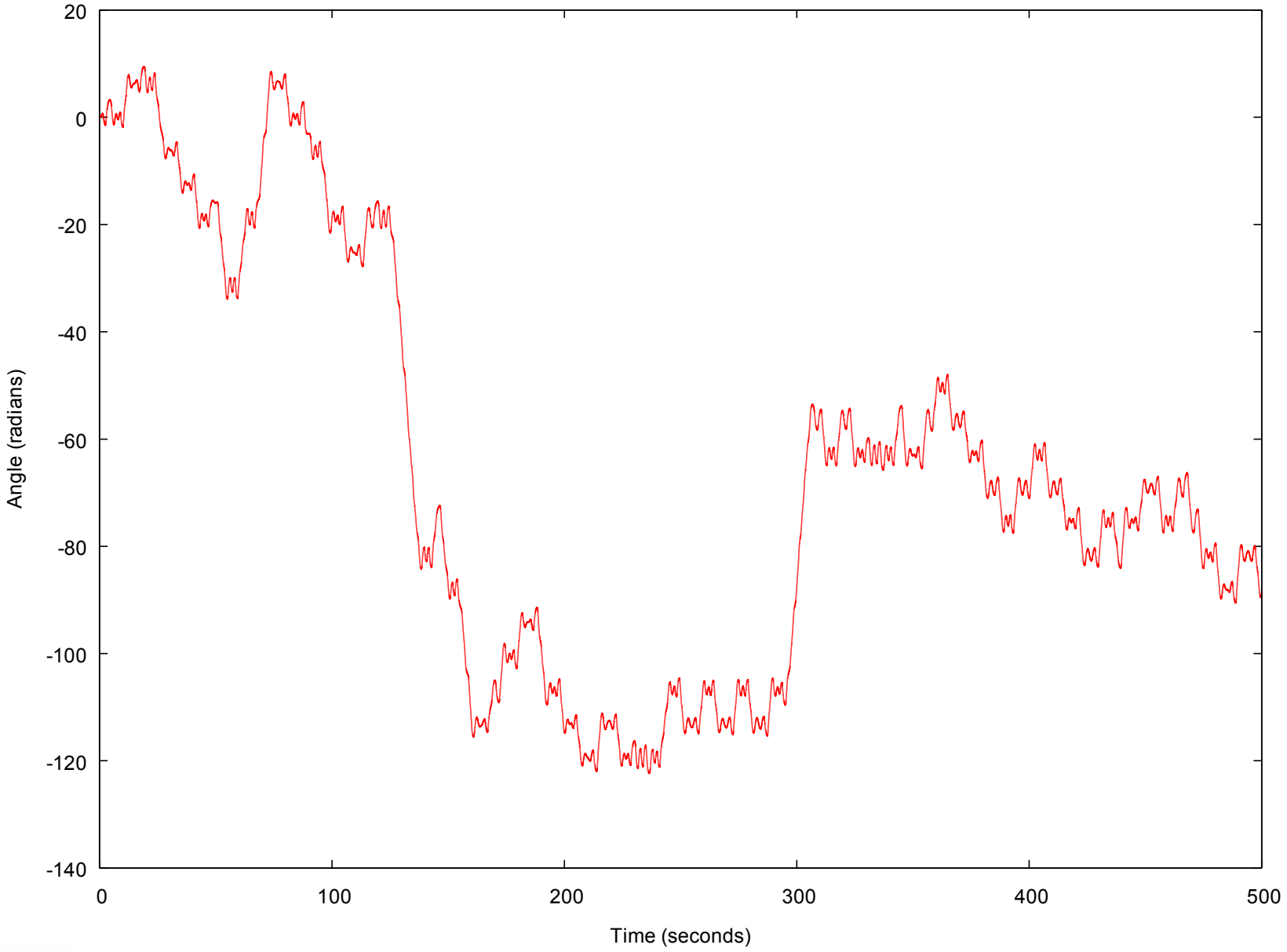
Example Chaotic Behavior



Example Chaotic Behavior

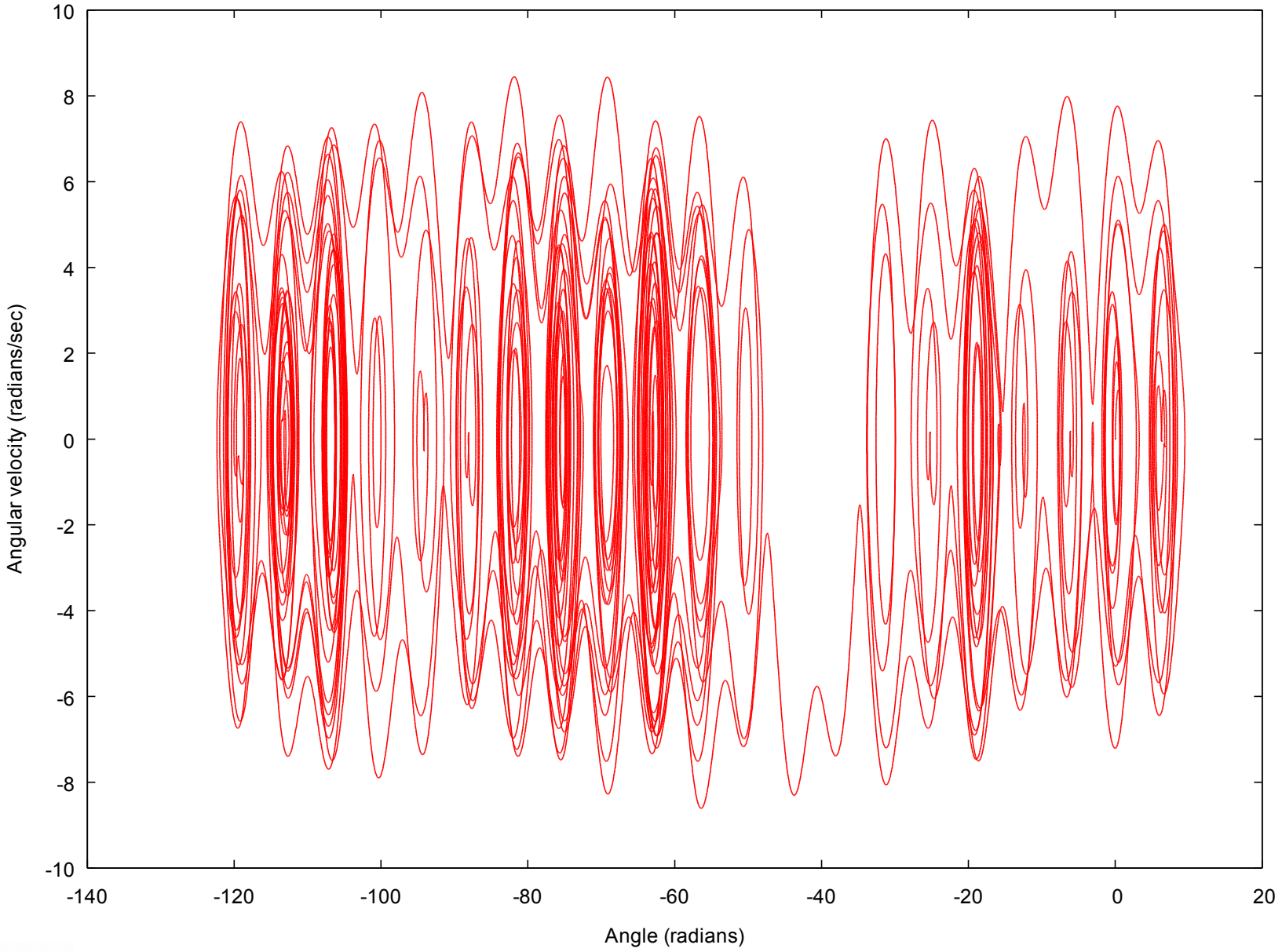


Example Chaotic Behavior



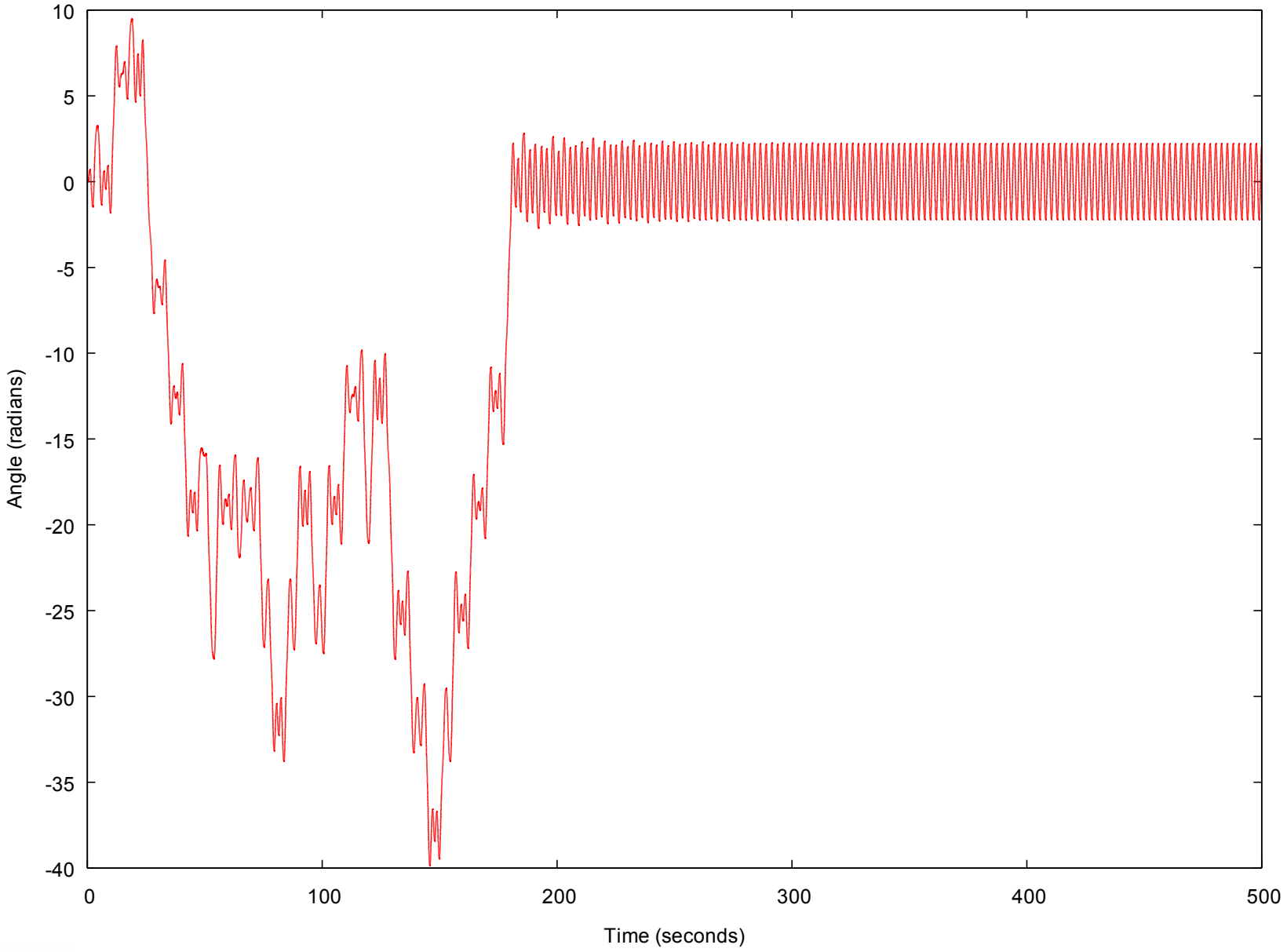
Example Chaotic Behavior

L=1 α=0.05 F=4 ω_f=2.5596



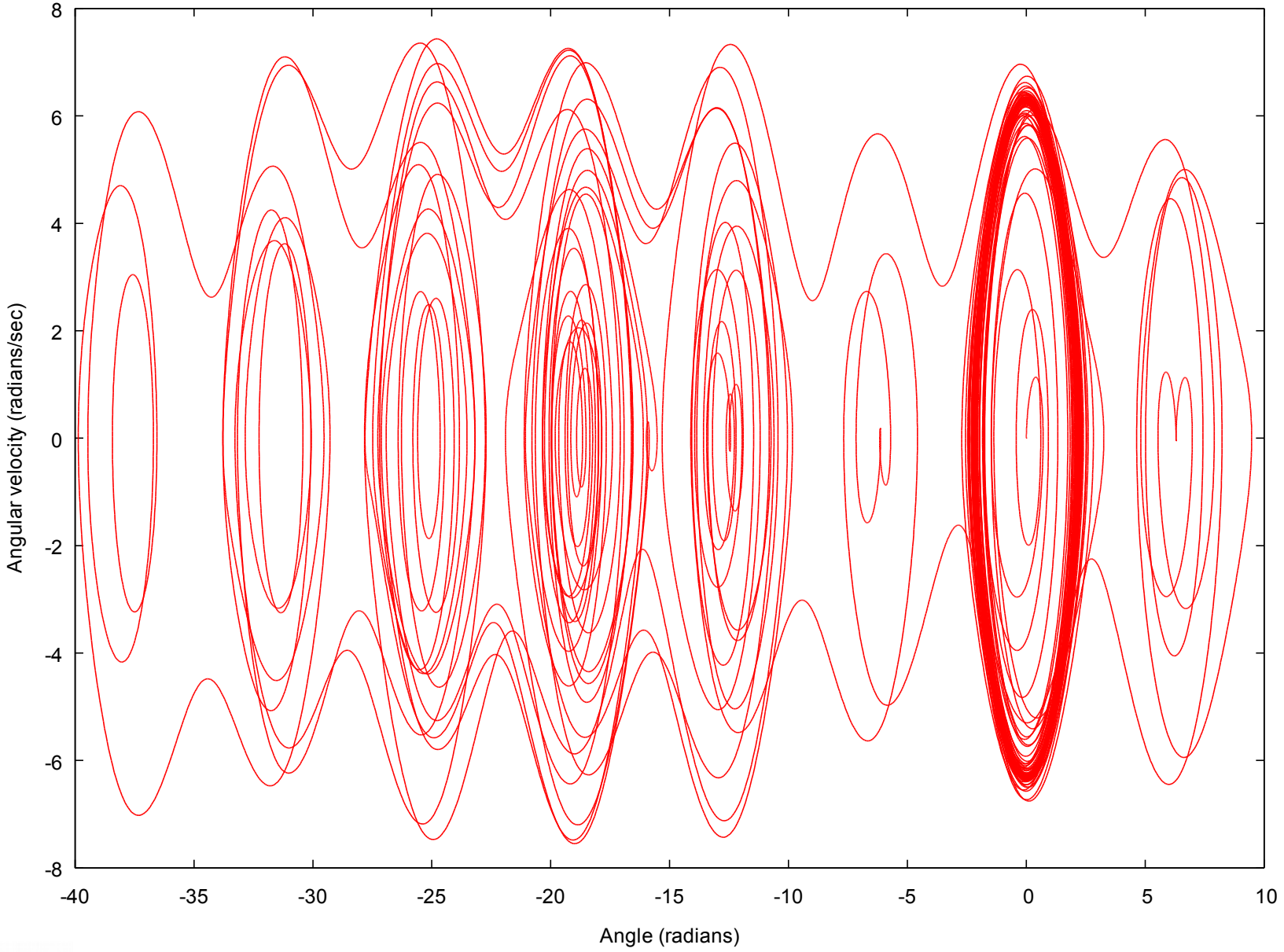
Example Chaotic Behavior

L=1 α=0.05 F=4 ω_f=2.559600001



Example Chaotic Behavior

L=1 α=0.05 F=4 ω_f=2.559600001



Lagrangian Derivation of Inverted Pendulum

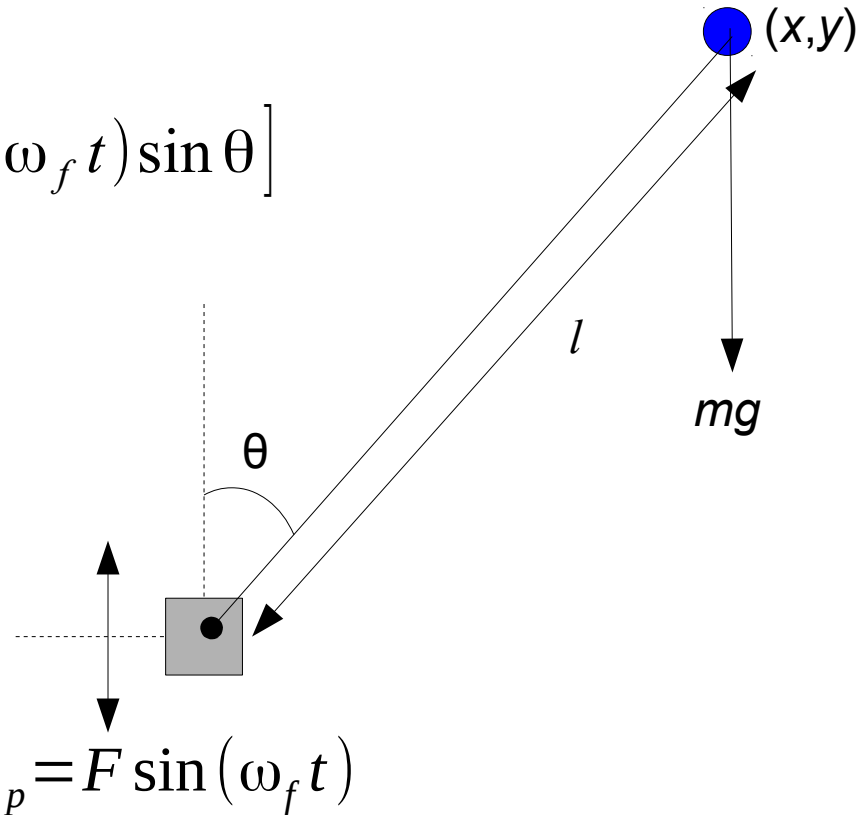
$$x = l\sin\theta \qquad y = l\cos\theta + F\sin(\omega_f t)$$

Lagrangian $L = T - V$

$$T = \frac{m|\vec{v}|^2}{2} = \frac{m}{2}(\dot{x}^2 + \dot{y}^2)$$

$$T = \frac{m}{2}\left[l^2\dot{\theta}^2 + \omega_f^2 F^2\cos^2(\omega_f t) - 2\omega_f F l\,\dot{\theta}\cos(\omega_f t)\sin\theta\right]$$

$$V = mgy = mgl\cos\theta + mgF\sin(\omega_f t)$$



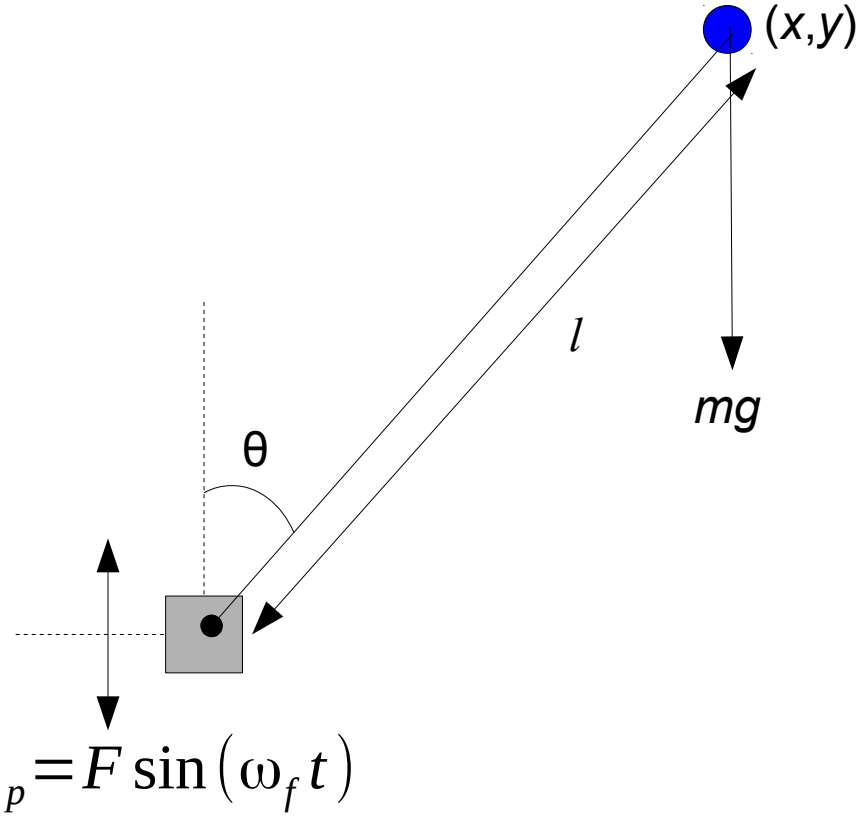
Pendulum pivot location is forced to be at an oscillating *y* coordinate

Lagrangian Derivation of Inverted Pendulum

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial L}{\partial \dot{\theta}}\right]=\frac{\partial L}{\partial \theta}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[ml^2\dot{\theta}-m\omega_f F l\cos(\omega_f t)\sin\theta\right] = mgl\sin\theta-m\omega_f F l\dot{\theta}\cos(\omega_f t)\cos\theta$$

$$\ddot{\theta}=\frac{1}{l}\left[g\sin\theta-\omega_f^2 F\sin(\omega_f t)\sin\theta\right]$$



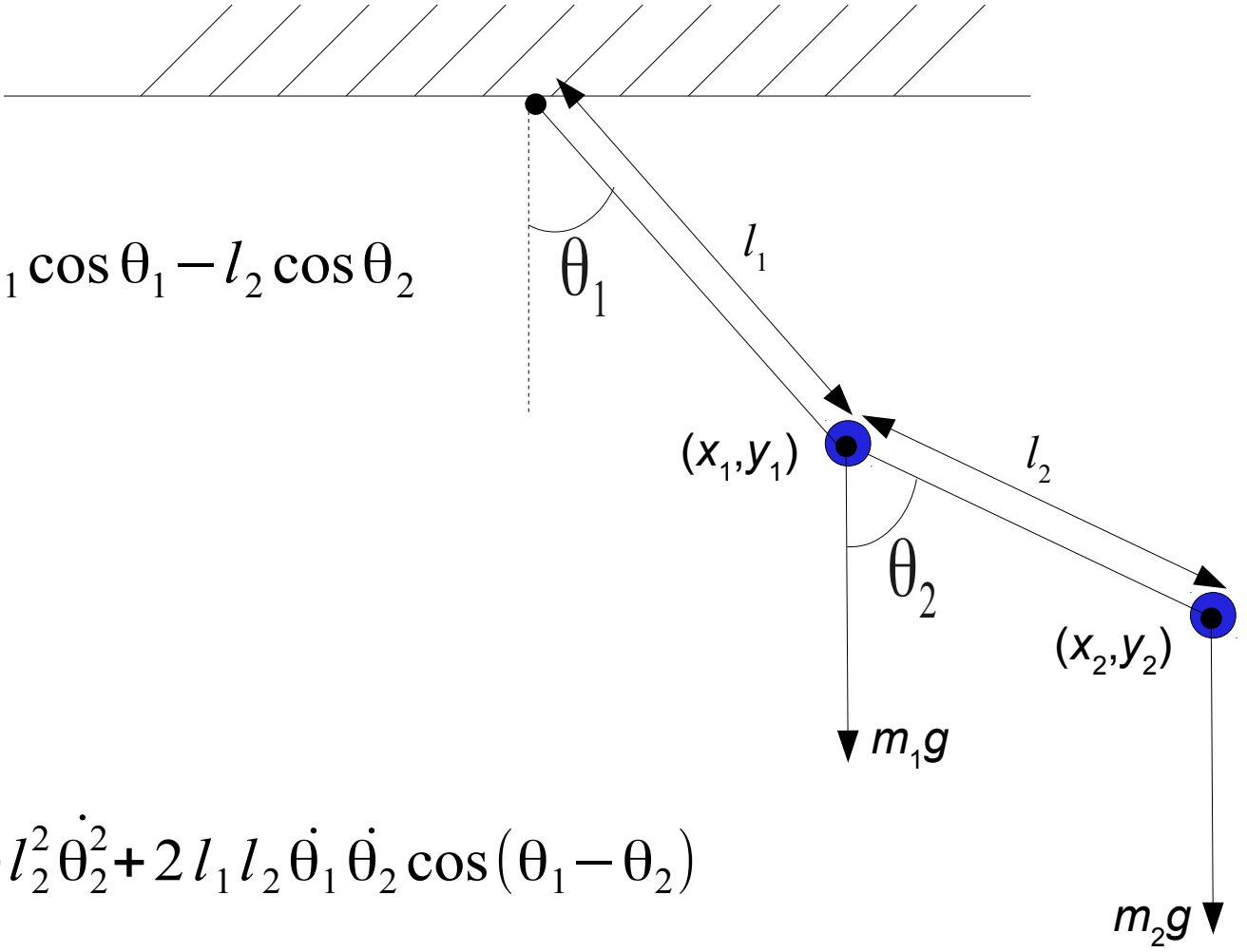
Transform to System of First Order ODEs

Original 2nd order ODE is $$\frac{d^2\theta}{dt^2} = \frac{1}{l}\left[g\sin\theta - \omega_f^2 F\sin(\omega_f t)\sin\theta\right]$$

Let $y_1 = \theta$ $y_2 = \frac{dy_1}{dt}$

So system becomes
$$\frac{dy_1}{dt} = y_2$$
$$\frac{dy_2}{dt} = \frac{1}{l}\left[g - \omega_f^2 F\sin(\omega_f t)\right]\sin(y_1)$$

Lagrangian Derivation of Double Pendulum



$$x_1 = l_1 \sin\theta_1 \quad y_1 = -l_1 \cos\theta_1$$
$$x_2 = l_1 \sin\theta_1 + l_2 \sin\theta_2 \quad y_2 = -l_1 \cos\theta_1 - l_2 \cos\theta_2$$

Lagrangian $L = T - V$

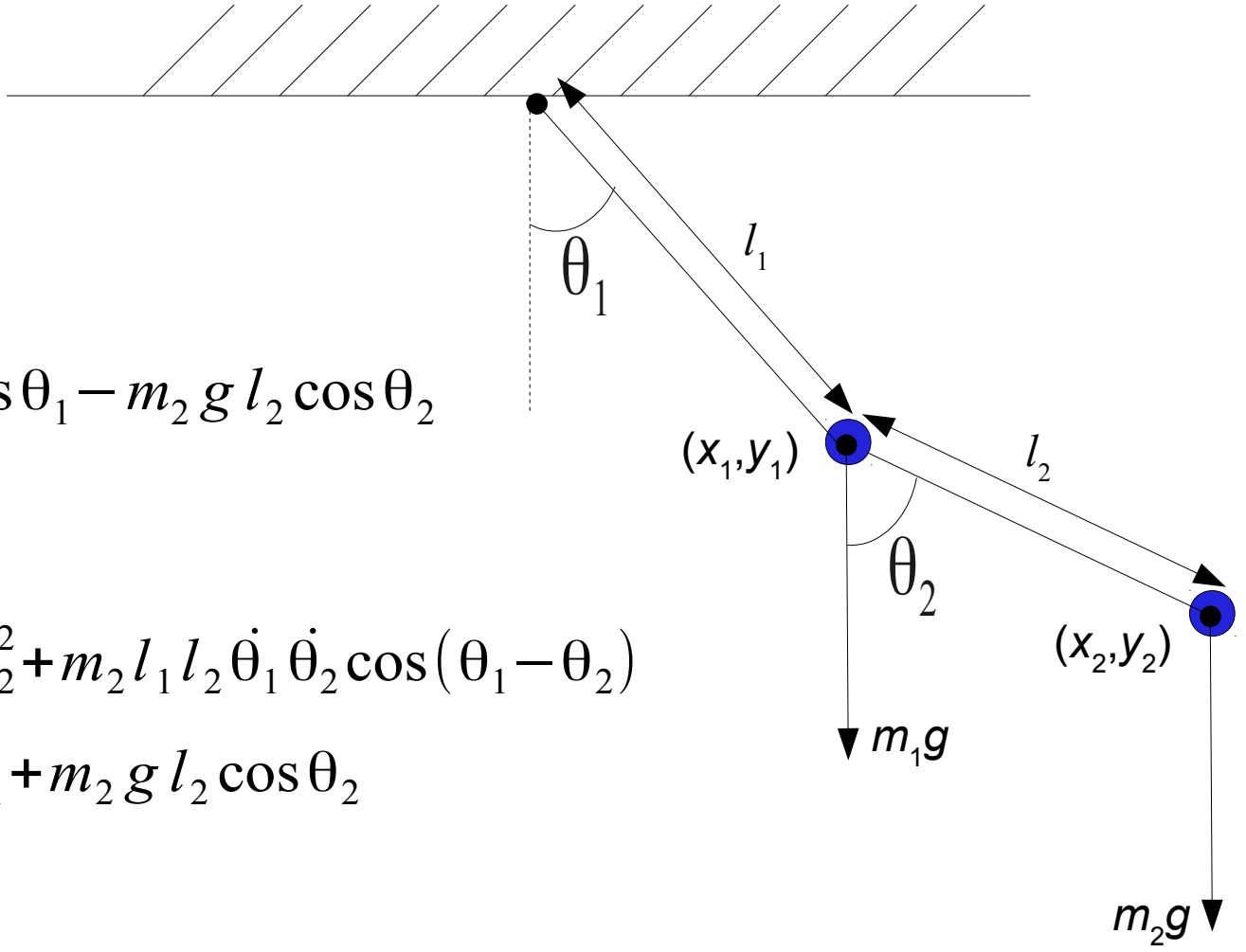
$$T = \frac{m_1 |\vec{v}_1|^2}{2} + \frac{m_2 |\vec{v}_2|^2}{2}$$

$$|\vec{v}_1|^2 = \left(\frac{\mathrm{d}\, x_1}{\mathrm{d}\, t}\right)^2 + \left(\frac{\mathrm{d}\, y_1}{\mathrm{d}\, t}\right)^2 = l_1^2 \dot{\theta}_1^2$$

$$|\vec{v}_2|^2 = \left(\frac{\mathrm{d}\, x_2}{\mathrm{d}\, t}\right)^2 + \left(\frac{\mathrm{d}\, y_2}{\mathrm{d}\, t}\right)^2 = l_1^2 \dot{\theta}_1^2 + l_2^2 \dot{\theta}_2^2 + 2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2)$$

$$T = \frac{m_1 l_1^2}{2} \dot{\theta}_1^2 + \frac{m_2 l_1^2}{2} \dot{\theta}_1^2 + \frac{m_2 l_2^2}{2} \dot{\theta}_2^2 + m_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2)$$

Lagrangian Derivation of Double Pendulum

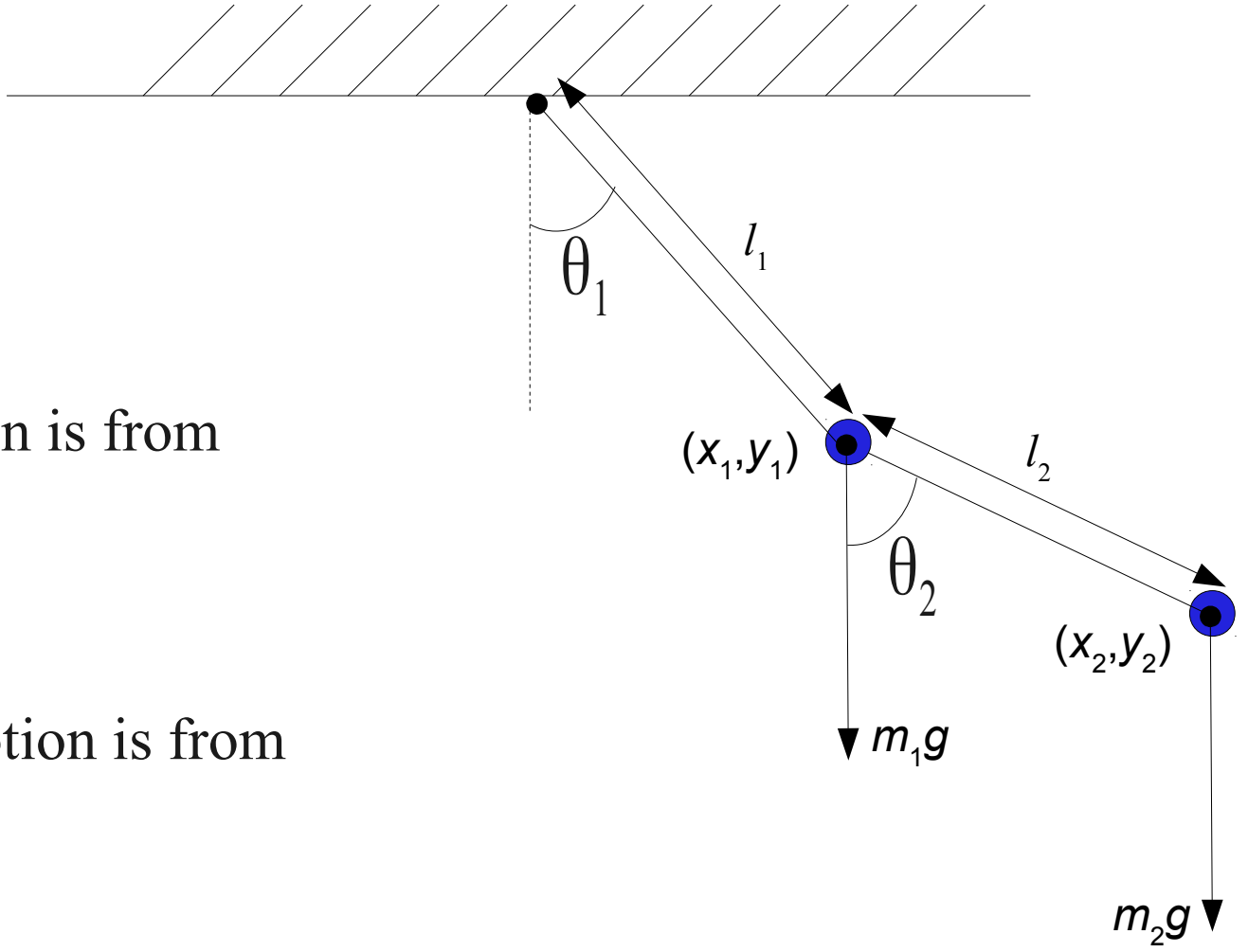


$$V = m_1 g\, y_1 + m_2 g\, y_2$$
$$= -m_1 g\, l_1 \cos\theta_1 - m_2 g\, l_1 \cos\theta_1 - m_2 g\, l_2 \cos\theta_2$$

so

$$L = \frac{m_1 l_1^2}{2}\dot{\theta}_1^2 + \frac{m_2 l_1^2}{2}\dot{\theta}_1^2 + \frac{m_2 l_2^2}{2}\dot{\theta}_2^2 + m_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2)$$
$$+ m_1 g\, l_1 \cos\theta_1 + m_2 g\, l_1 \cos\theta_1 + m_2 g\, l_2 \cos\theta_2$$

Lagrangian Derivation of Double Pendulum



First equation of motion is from

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial L}{\partial \dot{\theta}_1}\right]=\frac{\partial L}{\partial \theta_1}$$

Second equation of motion is from

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial L}{\partial \dot{\theta}_2}\right]=\frac{\partial L}{\partial \theta_2}$$

Lagrangian Derivation of Double Pendulum

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial L}{\partial \dot{\theta}_1}\right]=\frac{\partial L}{\partial \theta_1}$$

$$(m_1+m_2)\,l_1^2\ddot{\theta}_1+m_2\,l_1\,l_2\,\ddot{\theta}_2\cos(\theta_1-\theta_2)+m_2\,l_1\,l_2\,\dot{\theta}_2{}^2\sin(\theta_1-\theta_2)$$
$$+(m_1+m_2)\,g\,l_1\sin(\theta_1)=0$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial L}{\partial \dot{\theta}_2}\right]=\frac{\partial L}{\partial \theta_2}$$

$$m_2\,l_2^2\ddot{\theta}_2+m_2\,l_1\,l_2\,\ddot{\theta}_1\cos(\theta_1-\theta_2)-m_2\,l_1\,l_2\,\dot{\theta}_1{}^2\sin(\theta_1-\theta_2)$$
$$+m_2\,g\,l_2\sin(\theta_2)=0$$

Transform to System of First Order ODEs

Let $y_1 = \theta_1 \quad y_2 = \theta_2 \quad y_3 = \dot{\theta}_1 \quad y_4 = \dot{\theta}_2$

So system becomes

$$\dot{y}_1 = y_3$$
$$\dot{y}_2 = y_4$$
$$(m_1 + m_2) l_1^2 \dot{y}_3 + m_2 l_1 l_2 \dot{y}_4 \cos(y_1 - y_2) + m_2 l_1 l_2 y_4^2 \sin(y_1 - y_2) + (m_1 + m_2) g l_1 \sin(y_1) = 0$$
$$m_2 l_2^2 \dot{y}_4 + m_2 l_1 l_2 \dot{y}_3 \cos(y_1 - y_2) - m_2 l_1 l_2 y_3^2 \sin(y_1 - y_2) + m_2 g l_2 \sin(y_2) = 0$$

Unfortunately this does not conform to the standard form for systems of first order ODEs to be solved by the Runge-Kutta algorithm. There must be only one first derivative of each state variable, moved to the left side of each equation. Fix this by eliminating one derivative between each of the last two equations.

Solve third equation for $\dot{y}_3$ and substitute into fourth equation

Solve fourth equation for $\dot{y}_4$ and substitute back into third equation

Also define $\alpha = \dfrac{m_2}{(m_1 + m_2)}$

as the fraction of the total mass in the bottom mass

Transform to System of First Order ODEs

So with $y_1 = \theta_1 \quad y_2 = \theta_2 \quad y_3 = \dot{\theta}_1 \quad y_4 = \dot{\theta}_2$

The final system for R-K solution becomes

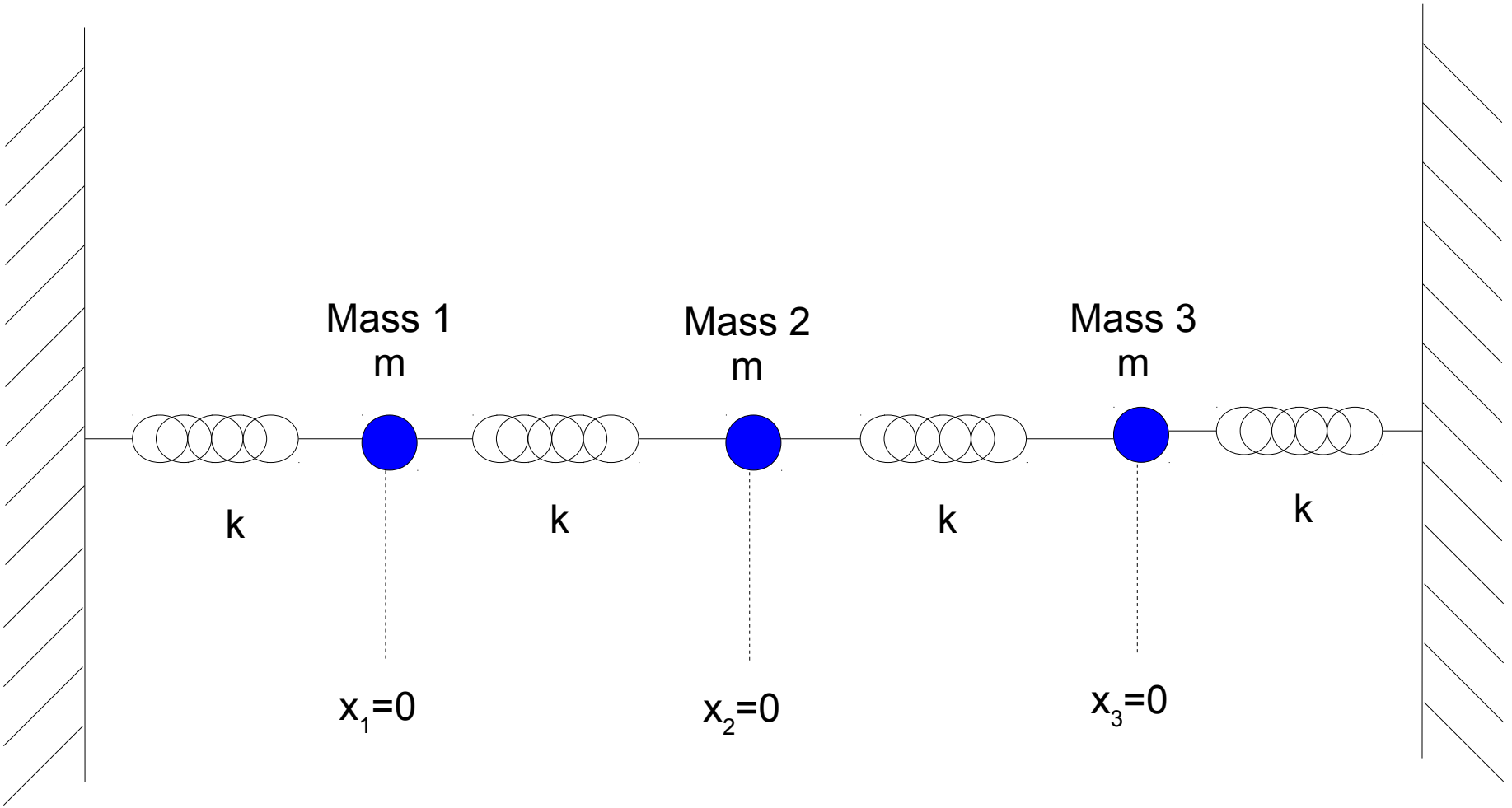
$$\dot{y}_1 = y_3$$

$$\dot{y}_2 = y_4$$

$$\dot{y}_3 = \frac{[\alpha l_1 y_3^2 \sin(y_1 - y_2)\cos(y_1 - y_2) + \alpha g \sin(y_2)\cos(y_1 - y_2) - \alpha l_2 y_4^2 \sin(y_1 - y_2) - g \sin(y_1)]}{l_1(1 - \alpha \cos^2(y_1 - y_2))}$$

$$\dot{y}_4 = \frac{[\alpha l_2 y_4^2 \sin(y_1 - y_2)\cos(y_1 - y_2) + g \sin(y_1)\cos(y_1 - y_2) + l_1 y_3^2 \sin(y_1 - y_2) - g \sin(y_2)]}{l_2(1 - \alpha \cos^2(y_1 - y_2))}$$

Mass/Spring Coupled Oscillator



Equations of Motion for Three Mass Oscillator

Let $\omega^2 = \frac{k}{m}$

$$\frac{d^2x_1}{dt^2} = -2\omega^2 x_1 + \omega^2 x_2 \qquad \frac{d^2x_2}{dt^2} = \omega^2 x_1 - 2\omega^2 x_2 + \omega^2 x_3 \qquad \frac{d^2x_3}{dt^2} = \omega^2 x_2 - 2\omega^2 x_3$$

Let $y_1 = x_1 \quad y_2 = \frac{dy_1}{dt} = v_1 \quad y_3 = x_2 \quad y_4 = \frac{dy_3}{dt} = v_2 \quad y_5 = x_3 \quad y_6 = \frac{dy_5}{dt} = v_3$

So system becomes

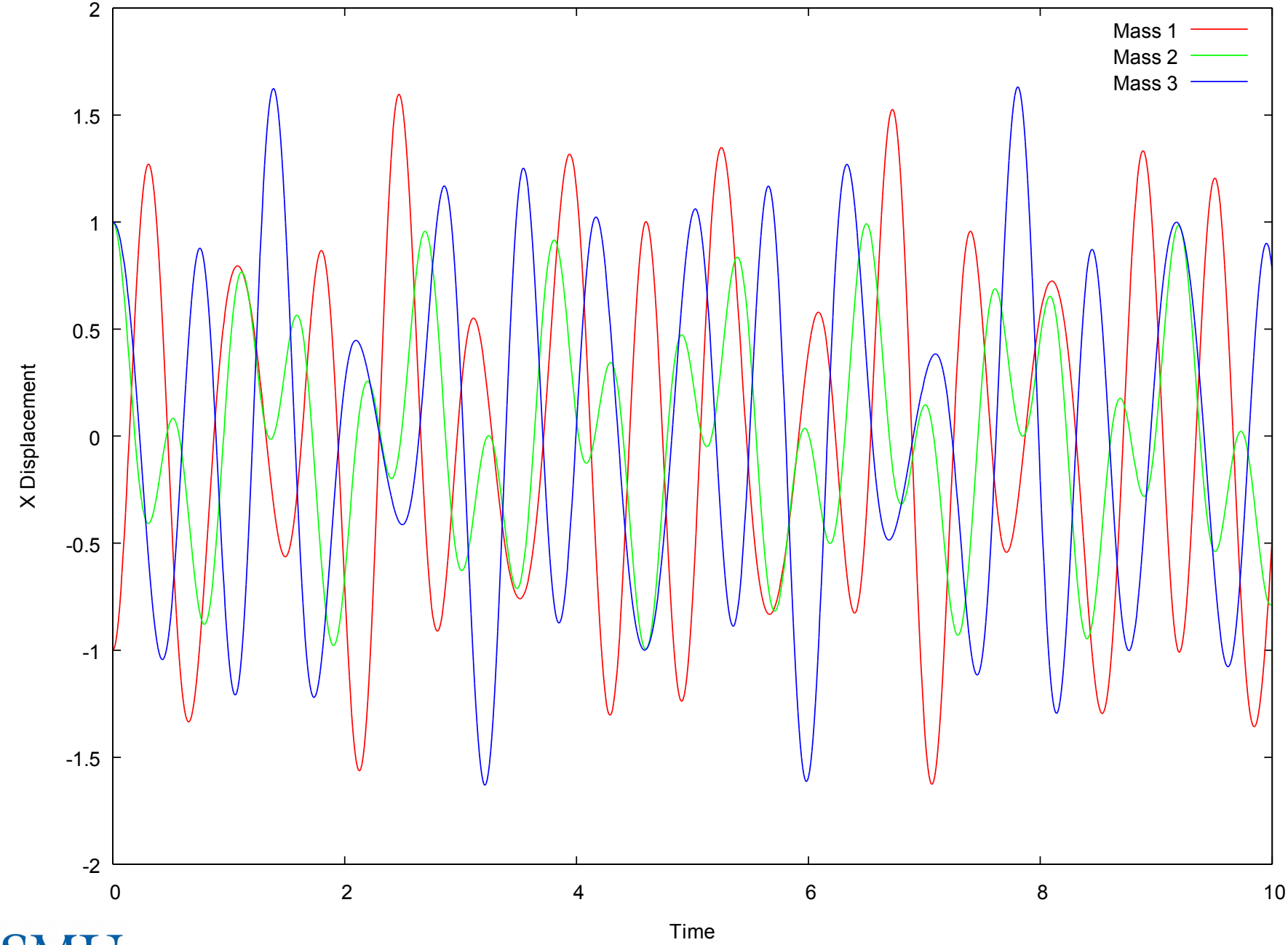
$$\frac{dy_1}{dt} = y_2 \qquad \frac{dy_2}{dt} = -2\omega^2 y_1 + \omega^2 y_3$$

$$\frac{dy_3}{dt} = y_4 \qquad \frac{dy_4}{dt} = \omega^2 y_1 - 2\omega^2 y_3 + \omega^2 y_5$$

$$\frac{dy_5}{dt} = y_6 \qquad \frac{dy_6}{dt} = \omega^2 y_3 - 2\omega^2 y_5$$

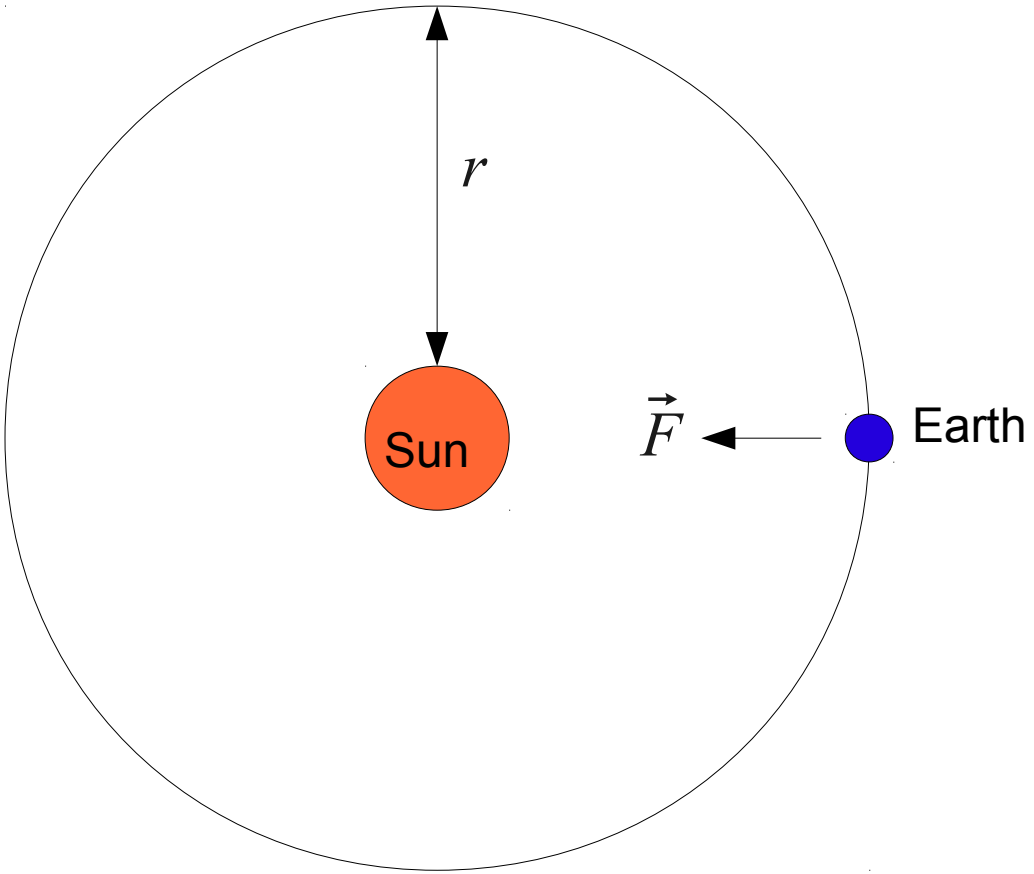
Example Dynamic Solution

Initial displacements of -1 1 1 match initial conditions used in multimode analysis in linear systems chapter



Solutions of Two-body Gravitational Problems

An analytic solution (for an orbital period around the sun) :



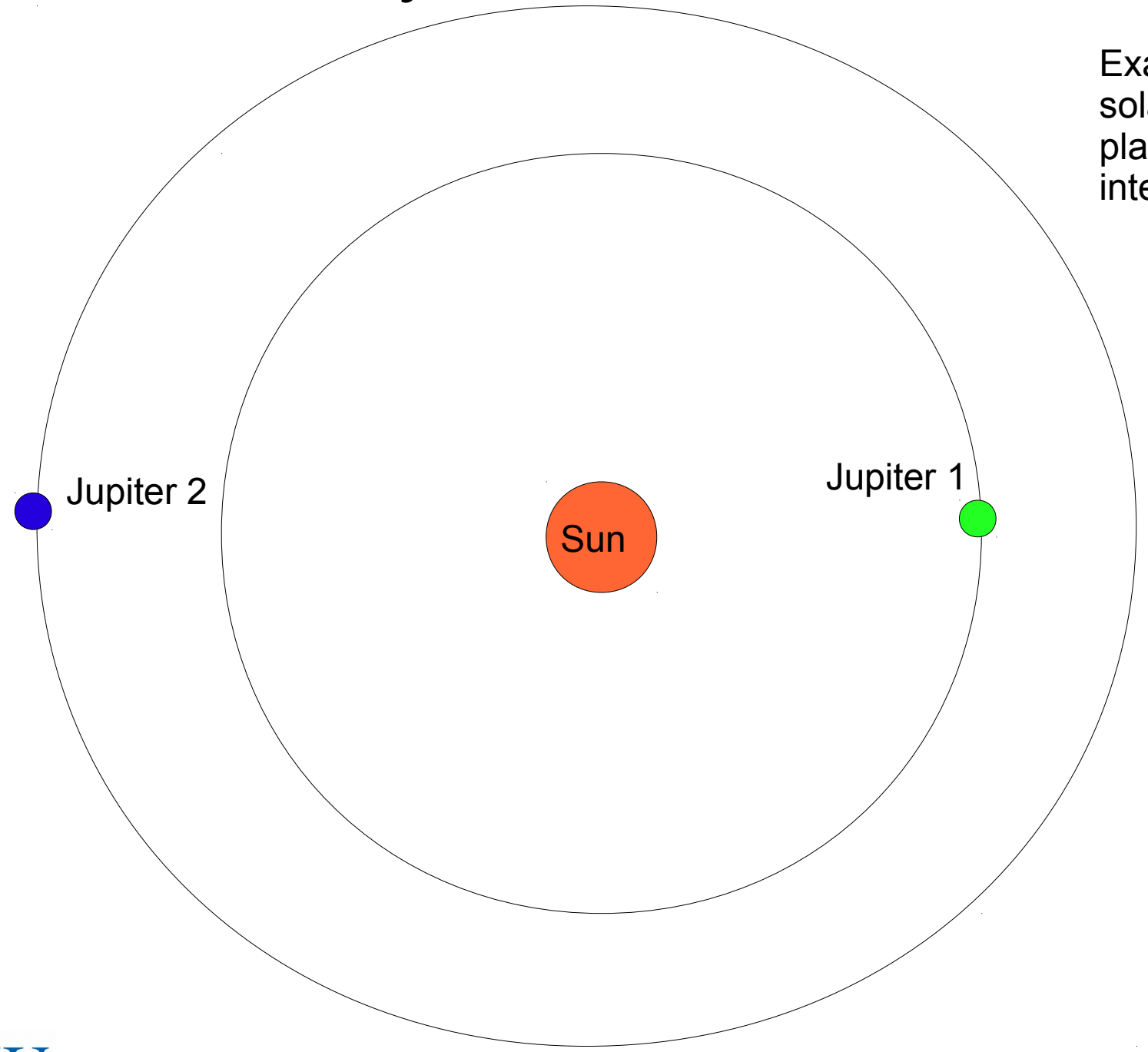
$$F = M_{earth} a = M_{earth} \frac{4\pi^2 r}{T^2}$$

$$F = \frac{G M_{earth} M_{sun}}{r^2}$$

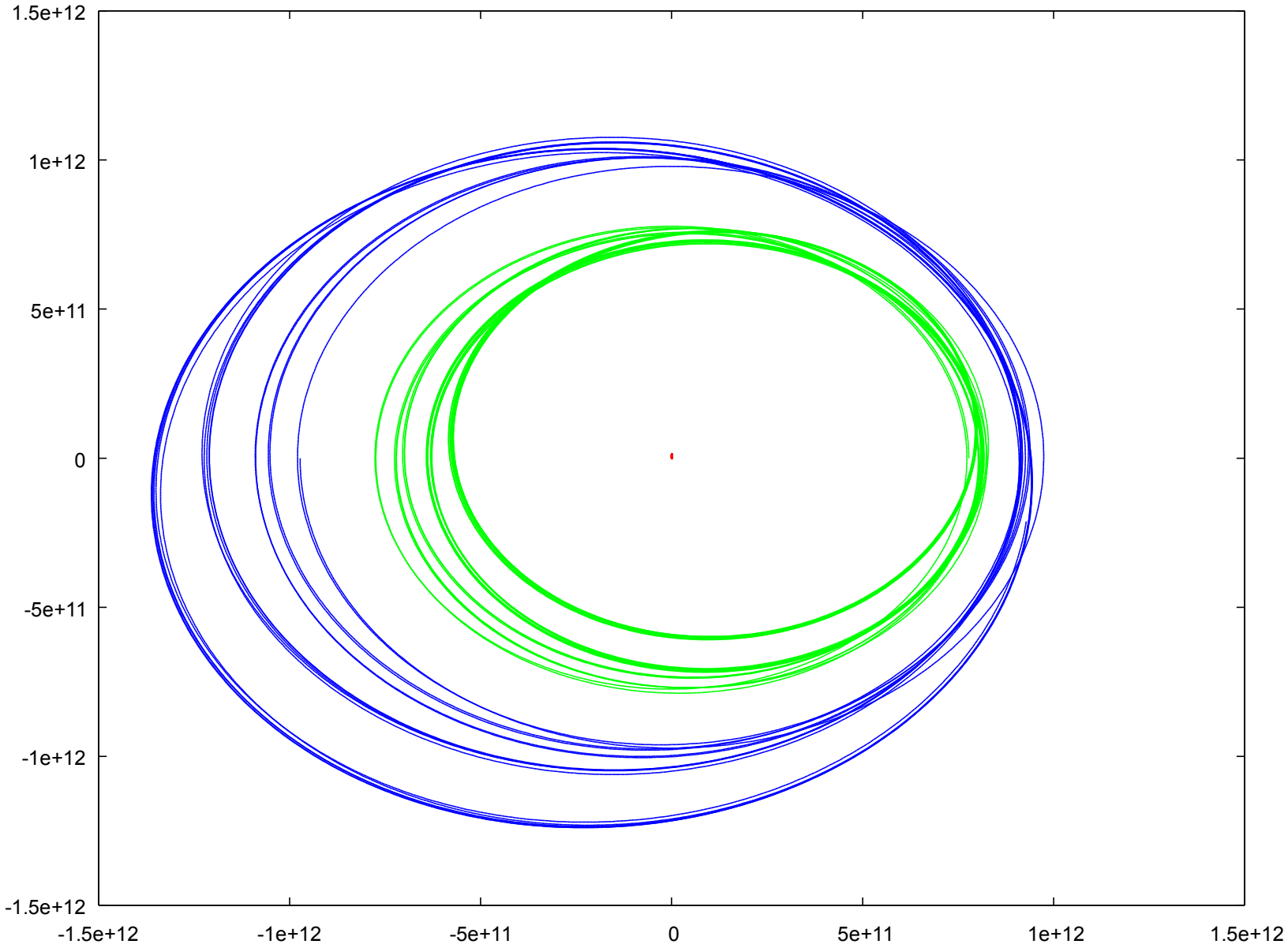
$$T = \sqrt{\frac{4\pi^2 r^3}{G M_{sun}}}$$

Three-body Gravitational Problems

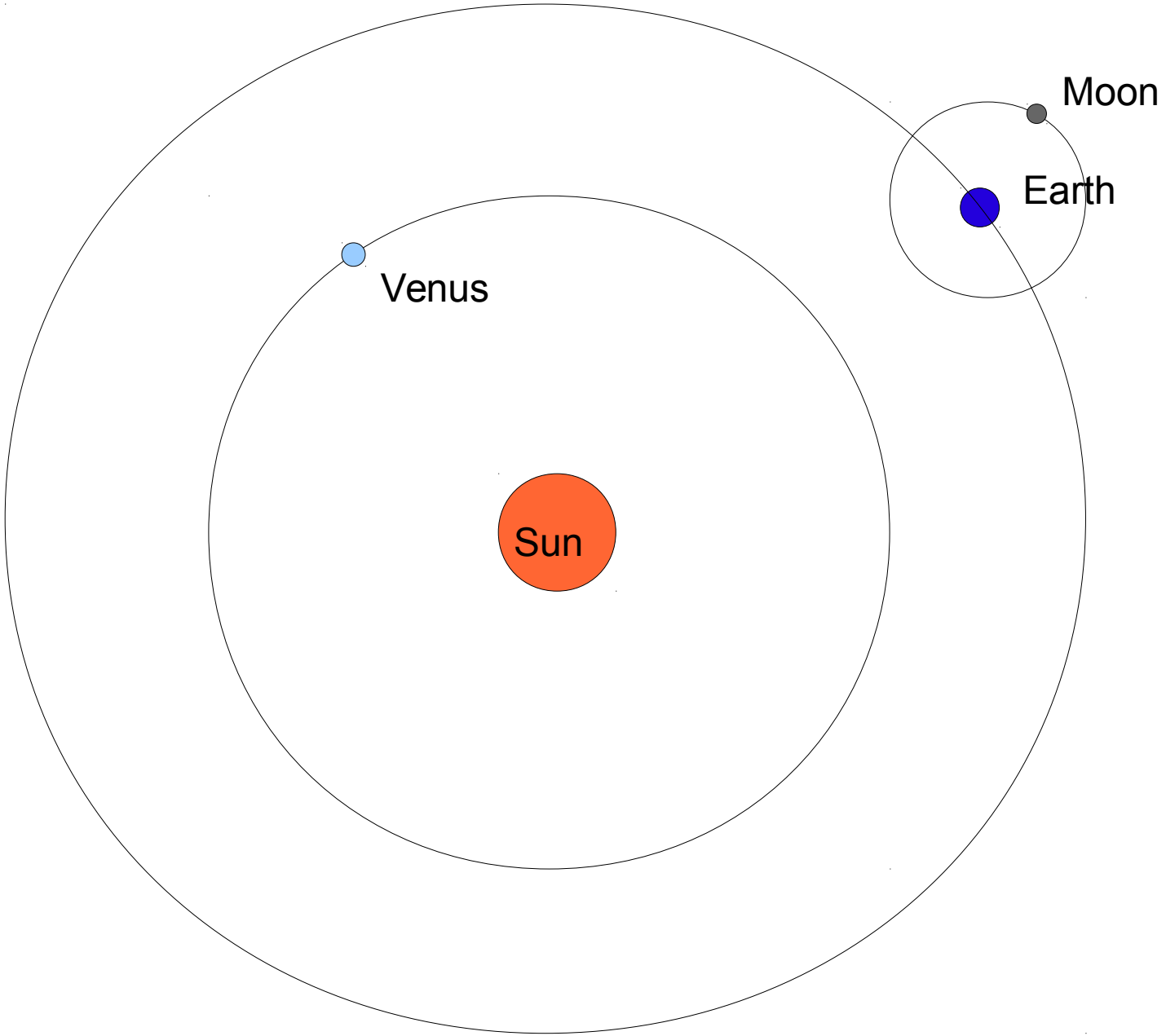
Example simple solar system with planet-planet interaction



Three-body Gravitational Simulation

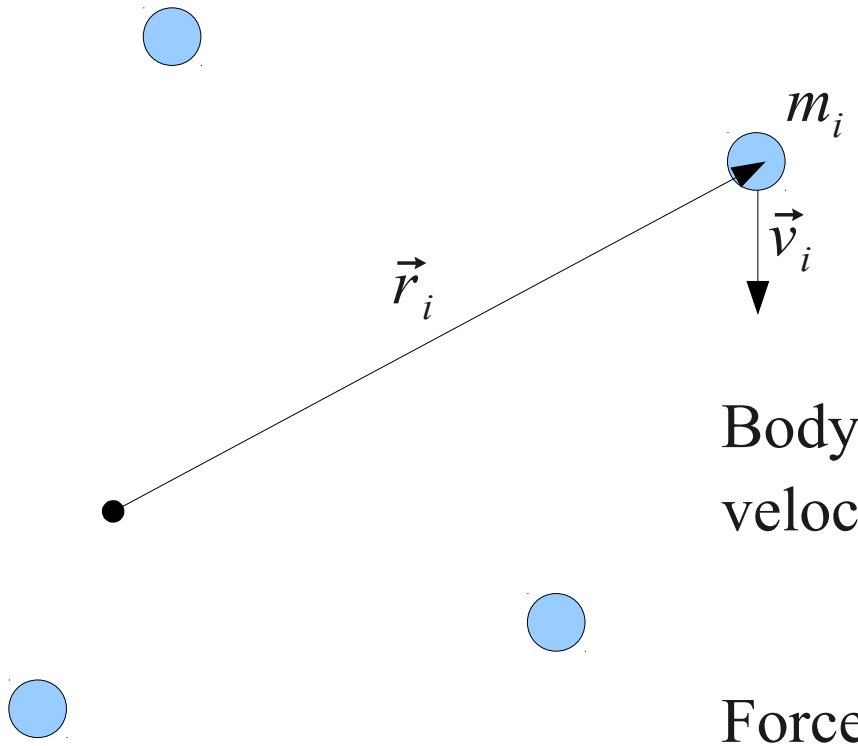


n-body Gravitational Problems



Complexity of gravitational interactions preclude general analytic solutions, except in trivial cases

n-body Gravitational Problems



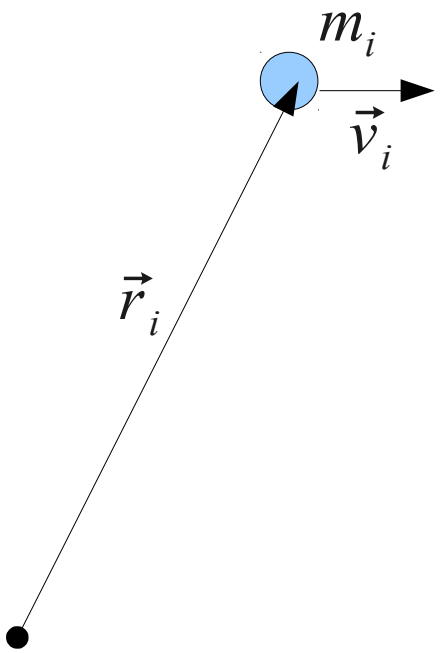
Body i has position vector $\vec{r}_i$
velocity vector $\vec{v}_i$ and mass m_i

Force on body i is $\vec{F}_i = m_i \vec{a}_i = m_i \frac{d\vec{v}_i}{dt} = m_i \frac{d^2\vec{r}_i}{dt^2}$

$$m_i \frac{d^2\vec{r}_i}{dt^2} = G \sum_{j \neq i} \frac{m_i m_j (\vec{r}_j - \vec{r}_i)}{|\vec{r}_j - \vec{r}_i|^3}$$

$$\frac{d^2\vec{r}_i}{dt^2} = G \sum_{j \neq i} \frac{m_j (\vec{r}_j - \vec{r}_i)}{|\vec{r}_j - \vec{r}_i|^3}$$

n-body Gravitational Problems



If system is solved in two dimensions, resolve each equation into two components

$$\frac{d^2 r_{ix}}{d t^2} = G \sum_{j \neq i} \frac{m_j \left(r_{jx} - r_{ix} \right)}{|\vec{r}_j - \vec{r}_i|^3}$$

$$\frac{d^2 r_{iy}}{d t^2} = G \sum_{j \neq i} \frac{m_j \left(r_{jy} - r_{iy} \right)}{|\vec{r}_j - \vec{r}_i|^3}$$

Transform to System of First Order ODEs

Let $y_1 = r_{ix}$ $y_2 = r_{iy}$ $y_3 = \frac{dy_1}{dt}$ $y_4 = \frac{dy_2}{dt}$

So system becomes

$$\frac{dy_1}{dt} = y_3$$

$$\frac{dy_2}{dt} = y_4$$

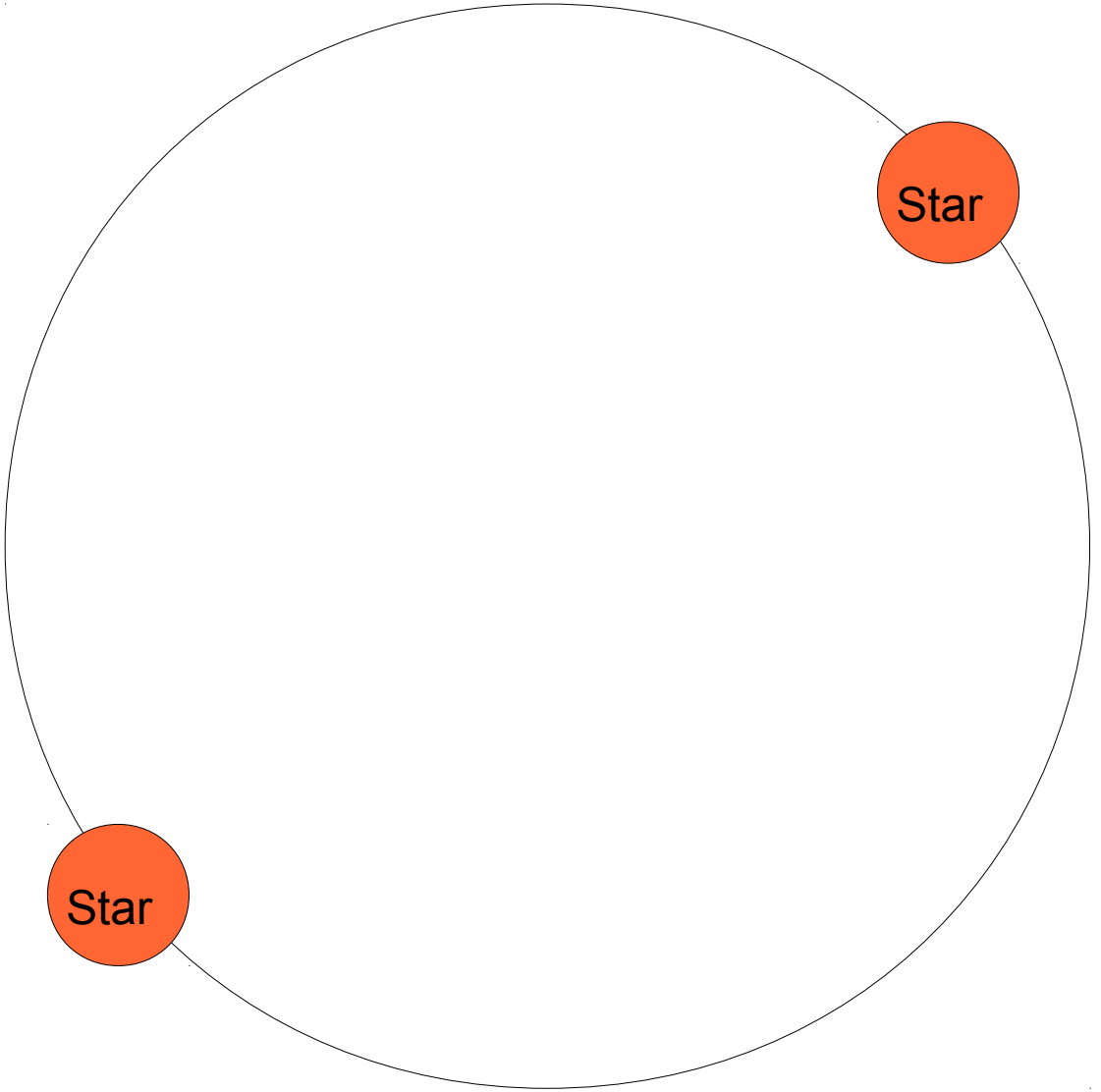
$$\frac{dy_3}{dt} = G\sum_{j\neq i}\frac{m_j\left(r_{jx}-r_{ix}\right)}{\left|\vec{r}_j-\vec{r}_i\right|^3}$$

$$\frac{dy_4}{dt} = G\sum_{j\neq i}\frac{m_j\left(r_{jy}-r_{iy}\right)}{\left|\vec{r}_j-\vec{r}_i\right|^3}$$

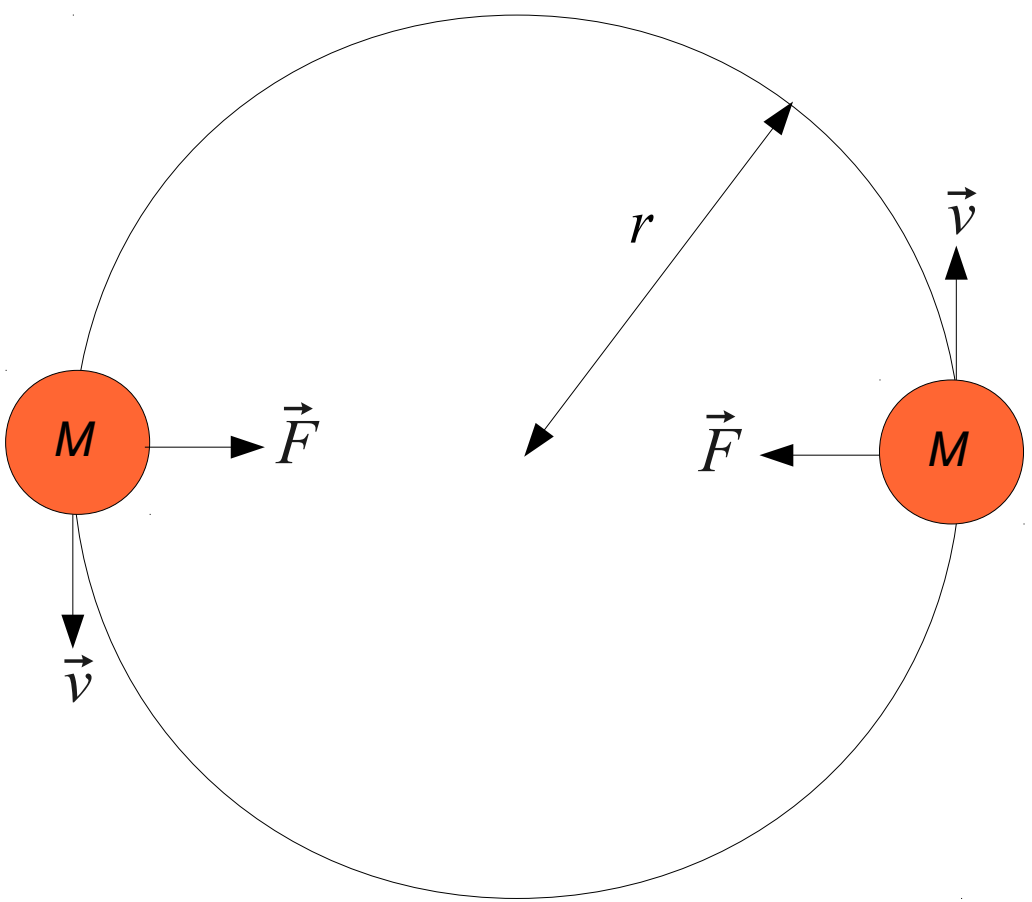
Each body contributes four first order equations. State variables y_1 through y_4 are for the first body, state variables y_5 through y_8 are for the second body, etc...

Simulating system with *n* bodies in two dimensions requires 4*n* state variables (6n if simulated in three dimensions).

Binary Star Systems Can Be Analyzed Analytically



Binary Star Systems Can Be Analyzed Analytically



$$F = Ma = M\frac{4\pi^2 r}{T^2}$$

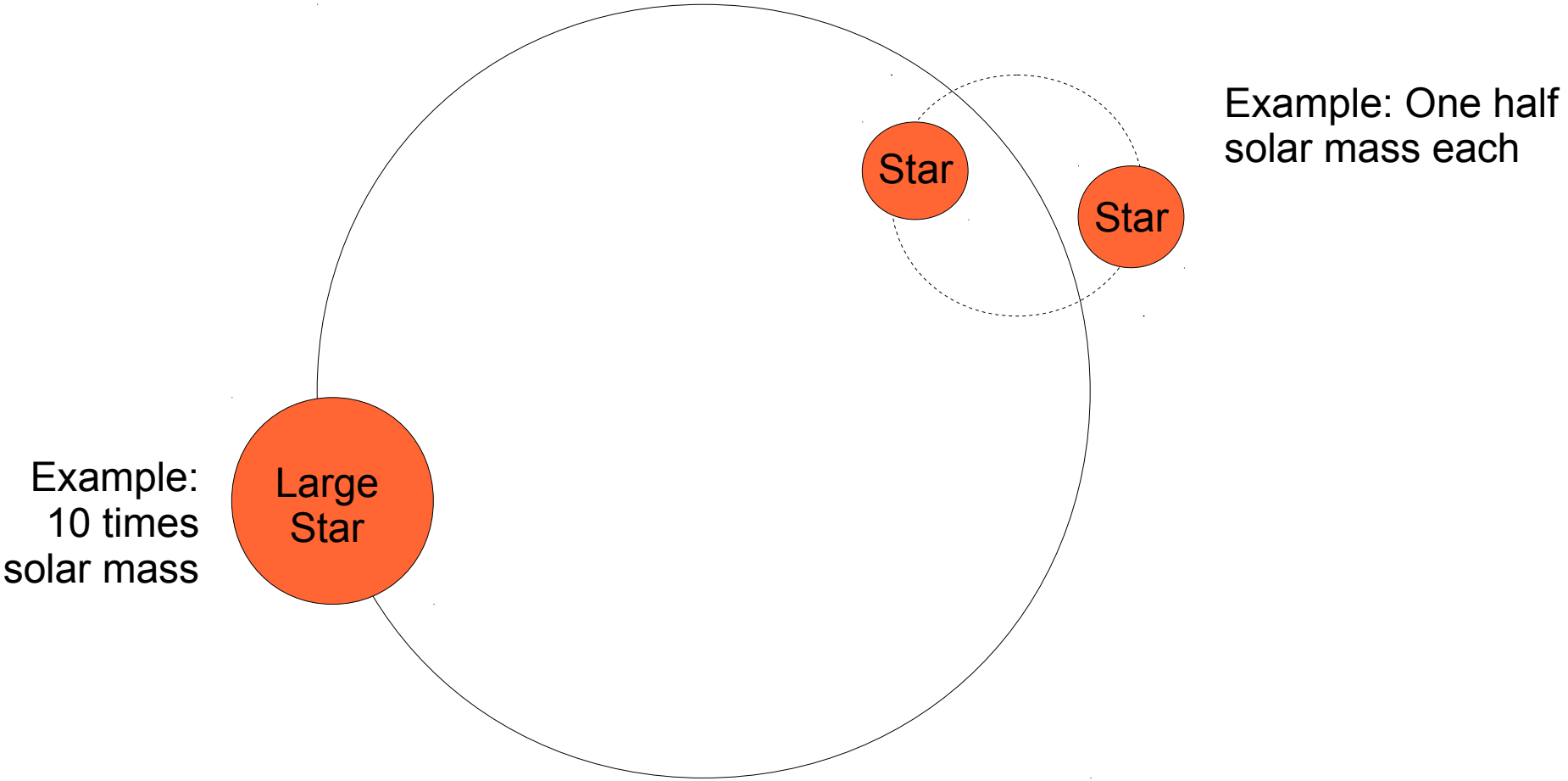
$$F = \frac{G M^2}{4 r^2}$$

$$T = \sqrt{\frac{16\pi^2 r^3}{GM}} \quad \text{or} \quad r = \sqrt[3]{\frac{GMT^2}{16\pi^2}}$$

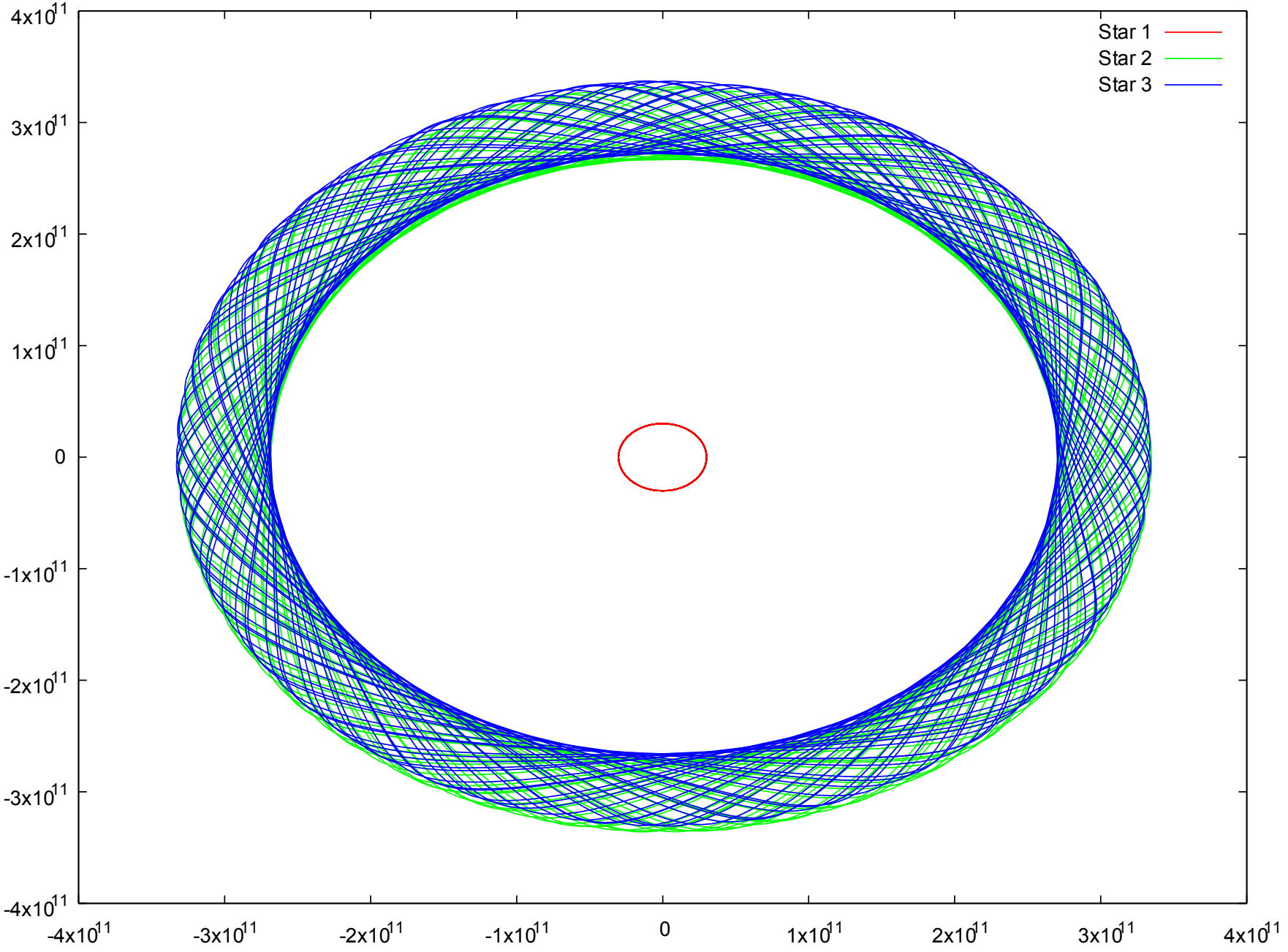
$$\text{also } v = \frac{2\pi r}{T}$$

Are There Stable Trinary Star Systems?

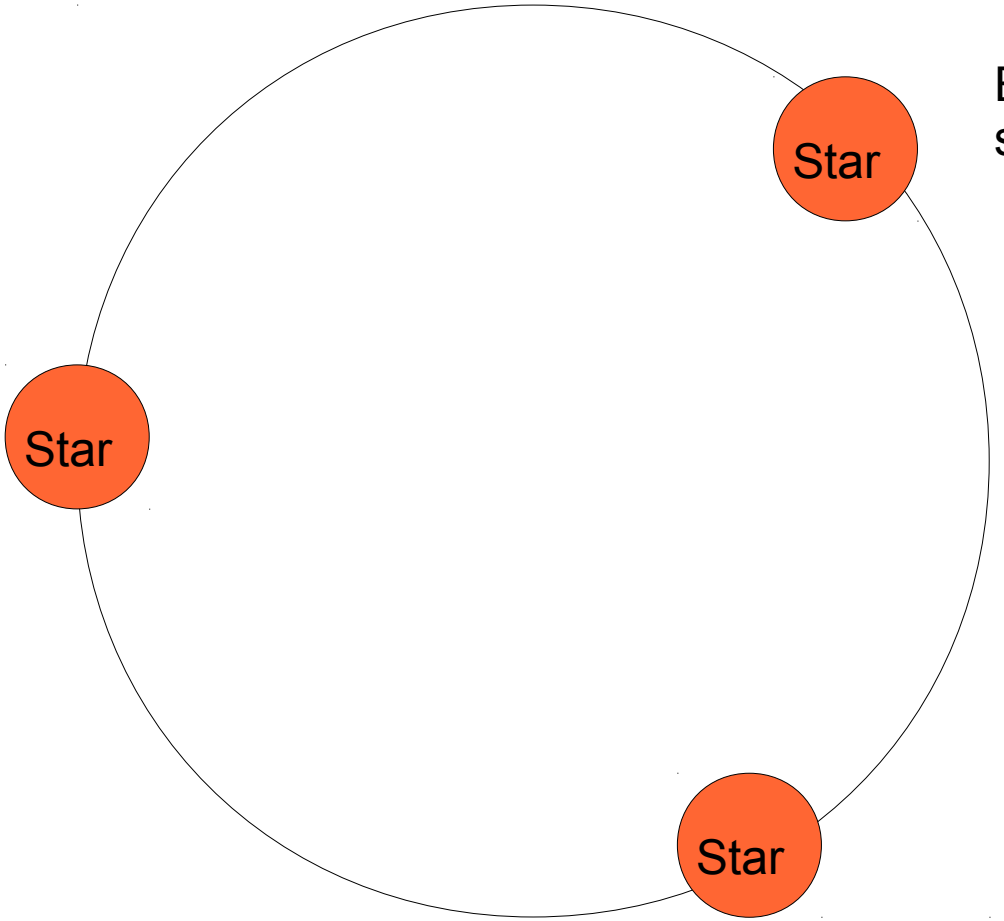
One possible configuration is a tight binary combination of two smaller stars, in orbit around a large dominant star



Numerical Solution to Trinary Star System

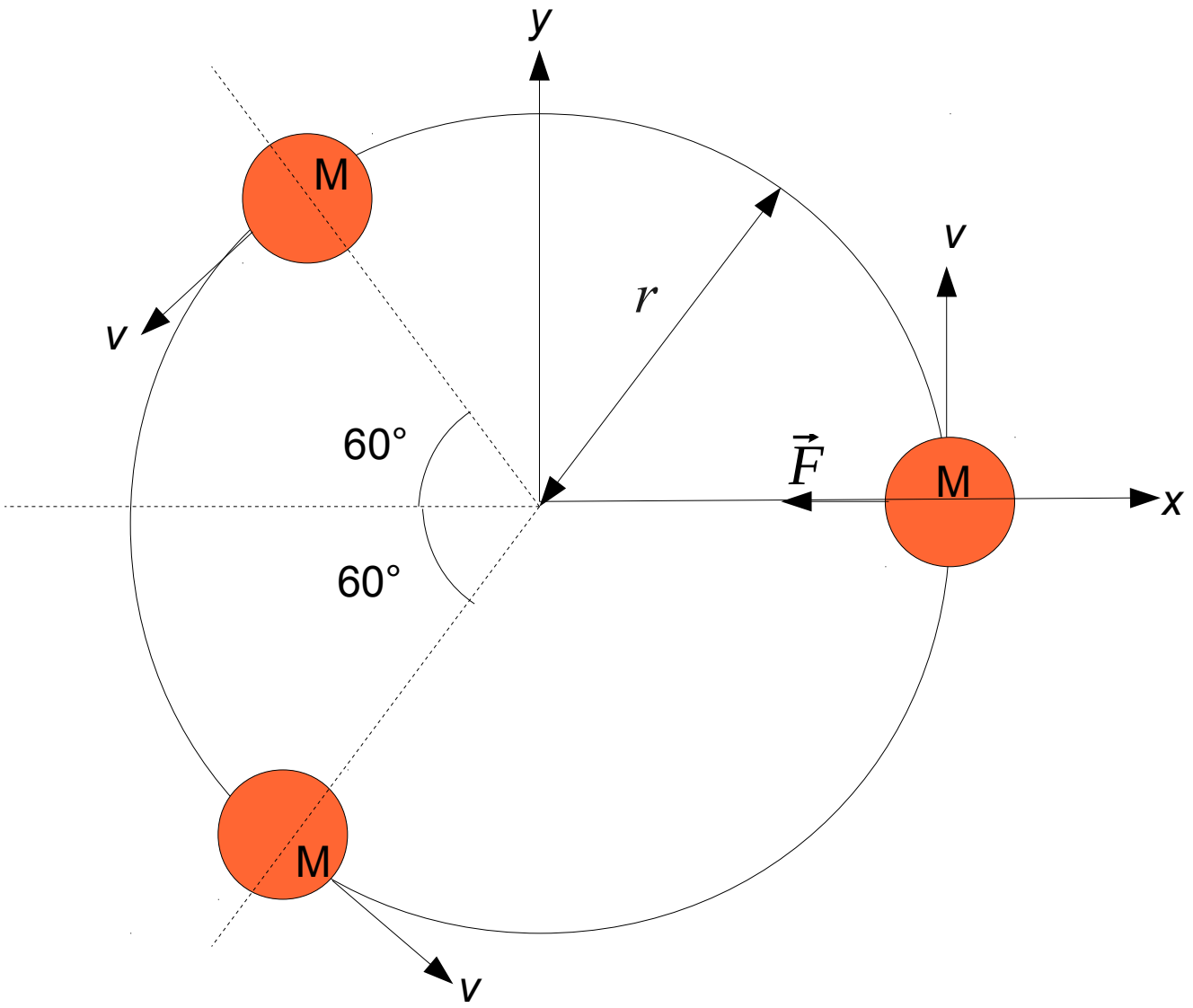


Are There Stable Symmetrical Trinary Star Systems?



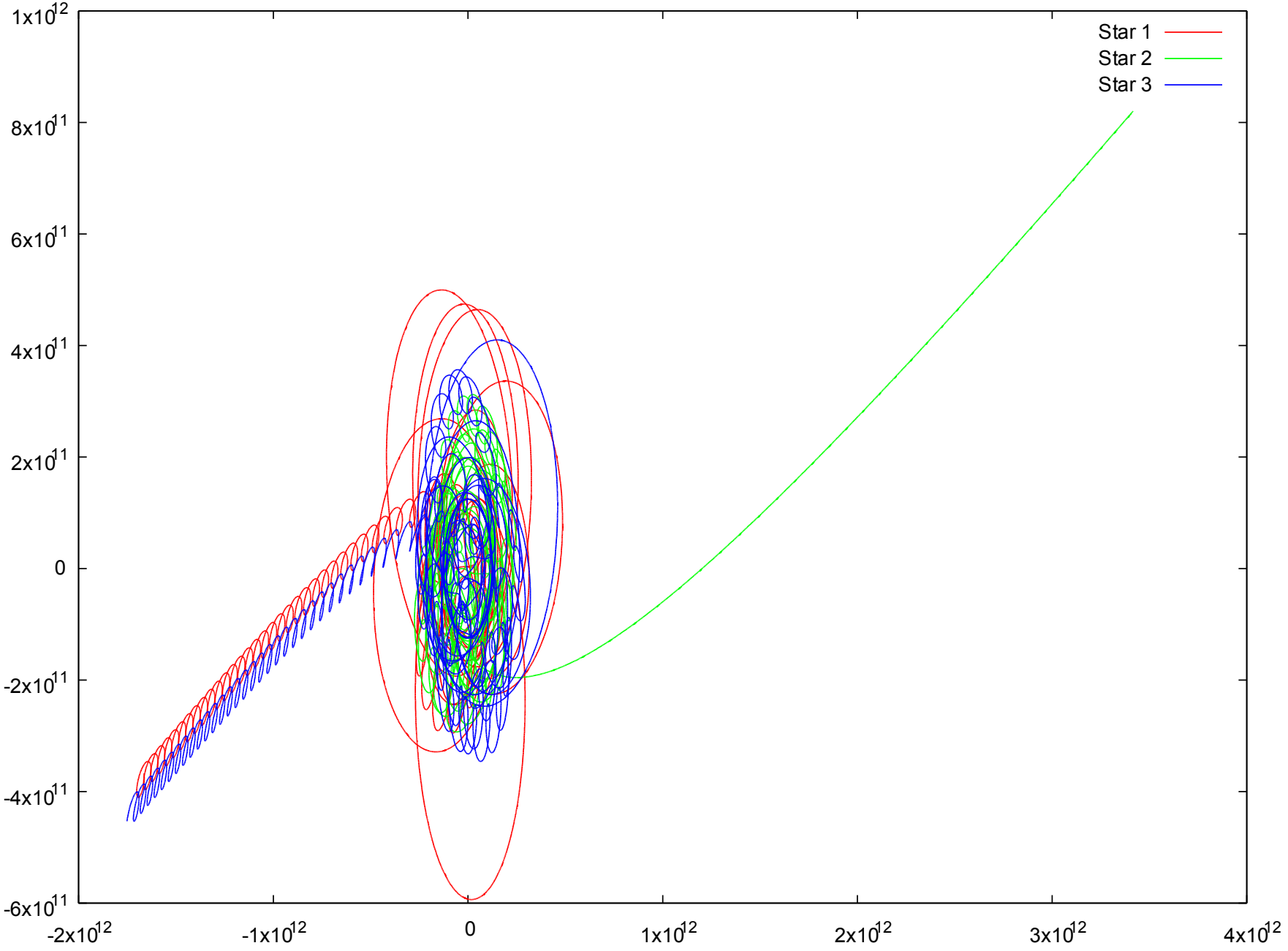
Example: One solar mass each

Setting Initial Simulation Conditions

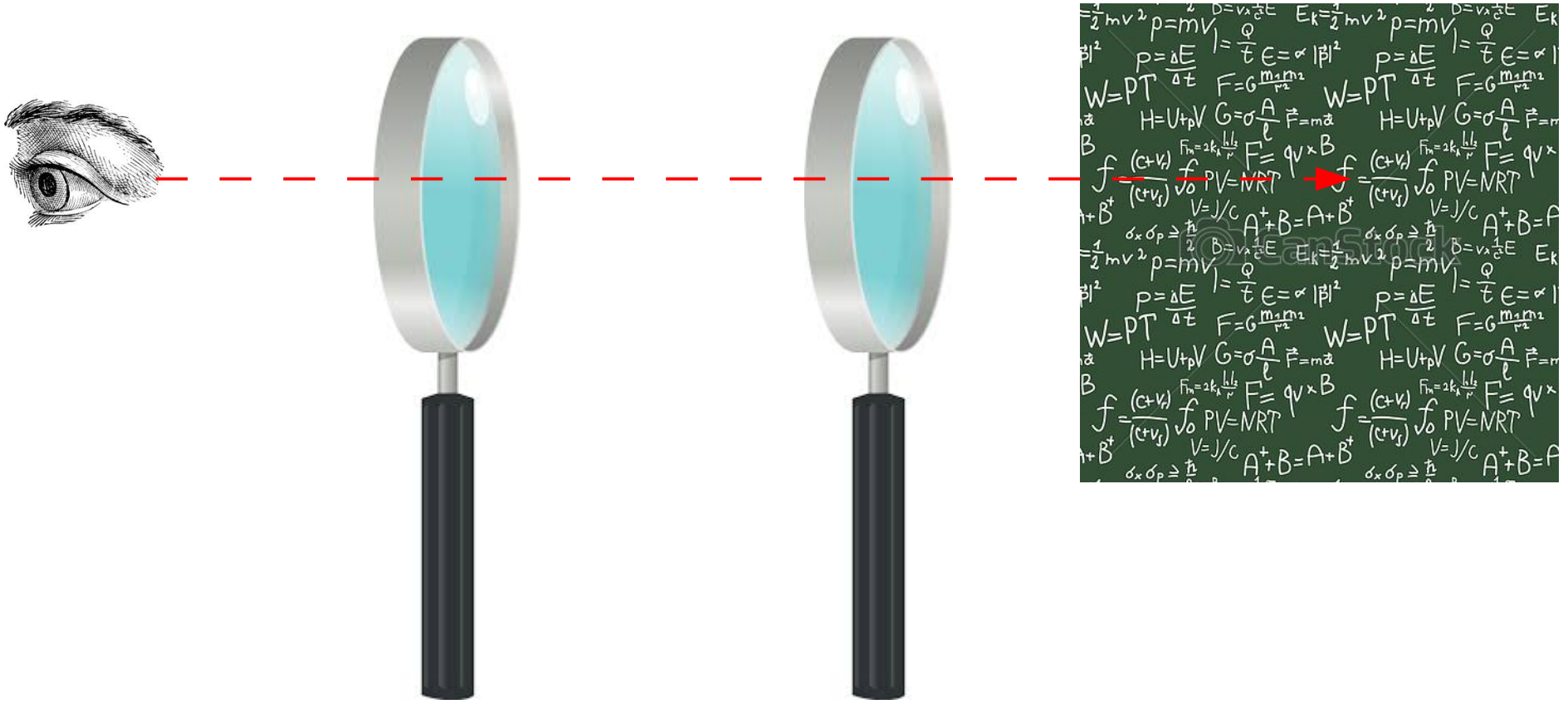


Find the gravitational force F on the right hand mass due to the other two masses, and equate to the required centripetal force

Numerical Solution to Unstable Star System



Solving Physics Problems Numerically

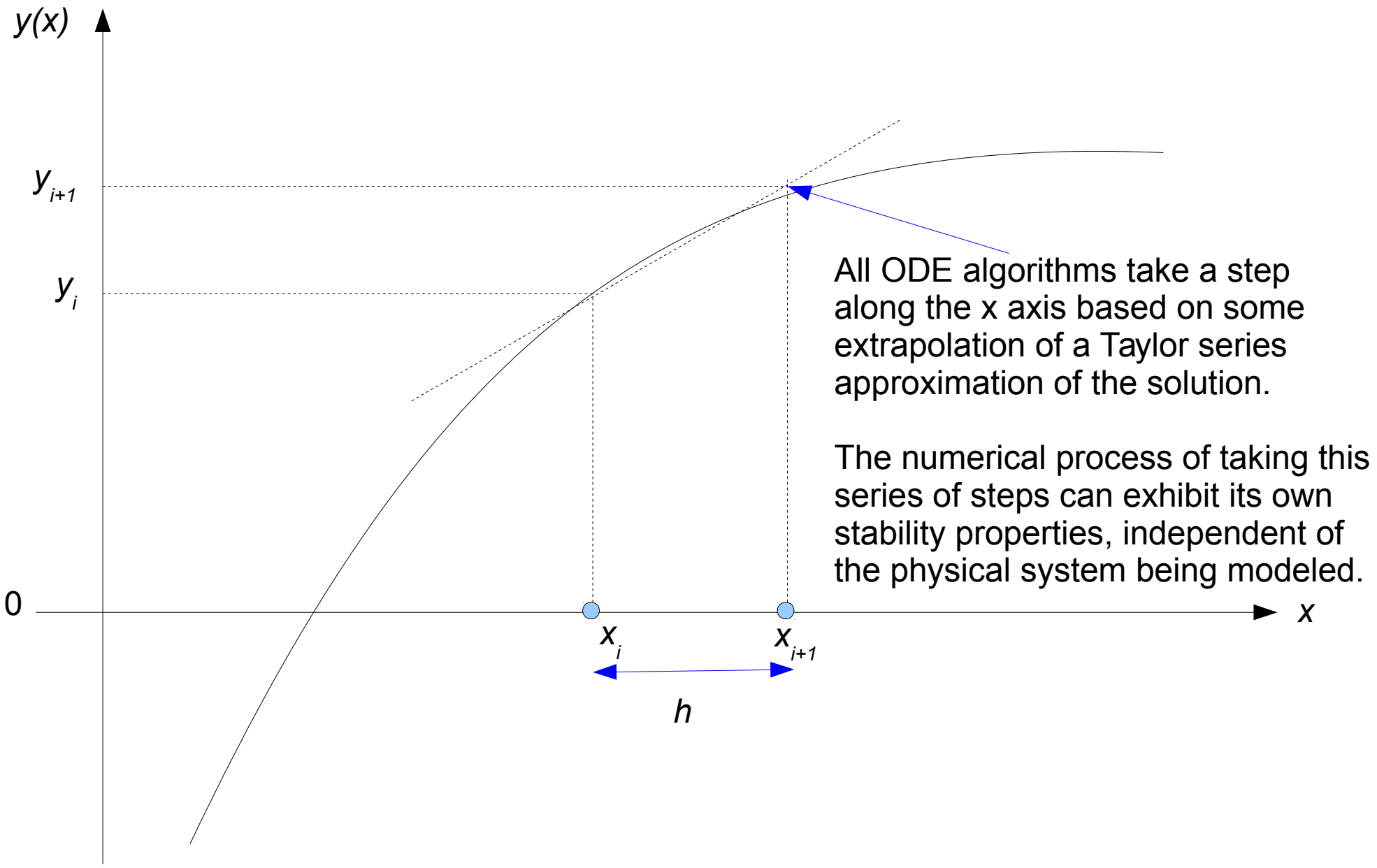


Computer Hardware

Numerical Algorithm

Each of these tools has its own properties and limitations that must be understood

Stability of ODE Algorithms



All ODE algorithms take a step along the x axis based on some extrapolation of a Taylor series approximation of the solution.

The numerical process of taking this series of steps can exhibit its own stability properties, independent of the physical system being modeled.

"Stiff" System of ODEs from Cheney and Kincaid

$$\frac{d\,y_1}{dx} = -20\,y_1 - 19\,y_2$$

$$\frac{d\,y_2}{dx} = -20\,y_1 - 19\,y_2$$

Initial conditions: $y_1(0) = 2 \qquad y_2(0) = 0$

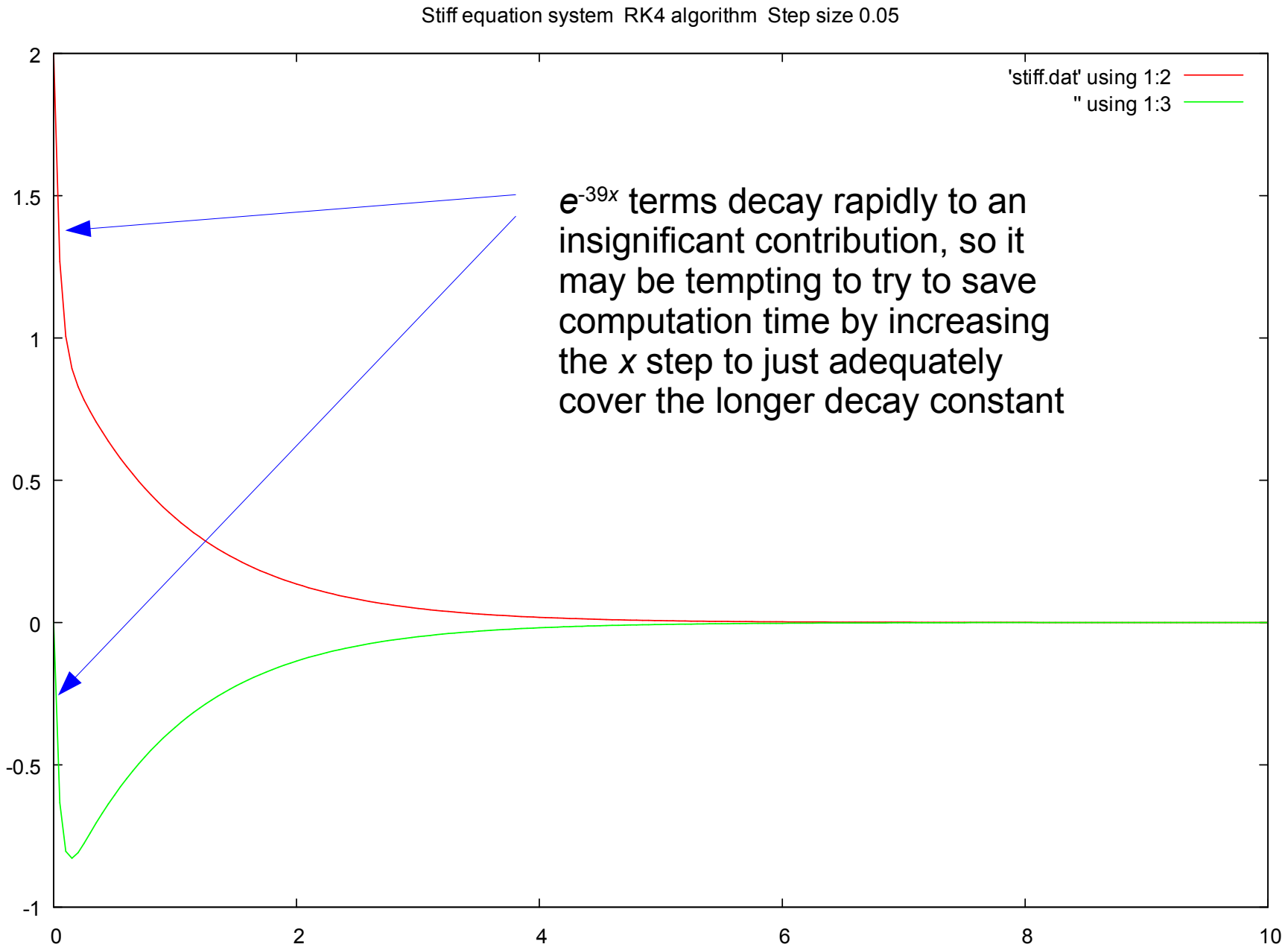
Solution:

$$y_1(x) = e^{-39x} + e^{-x}$$

$$y_2(x) = e^{-39x} - e^{-x}$$

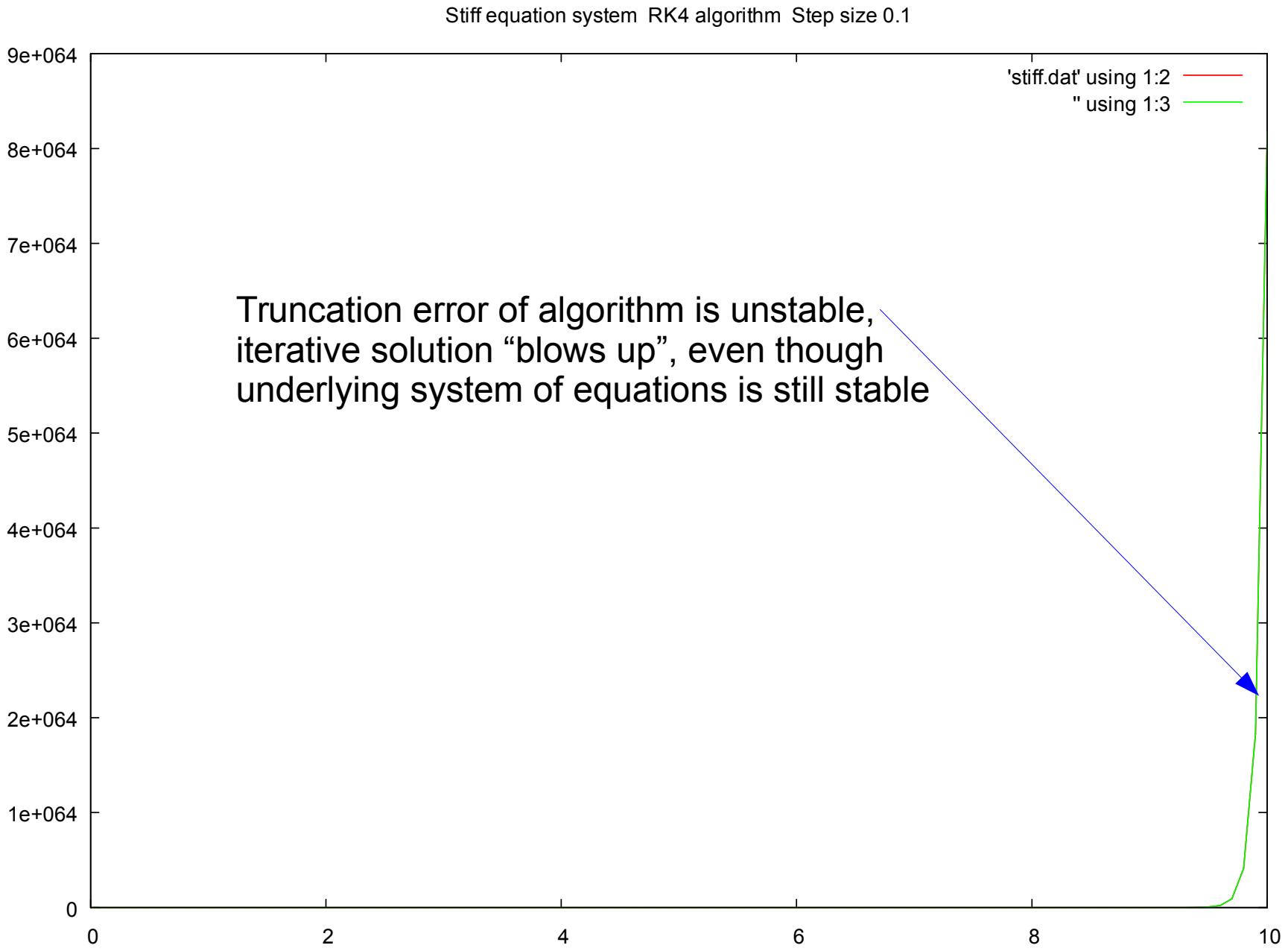
Solution is a linear superposition of exponentials with widely varying decay constants

Runge-Kutta Algorithm with Stable Step Size

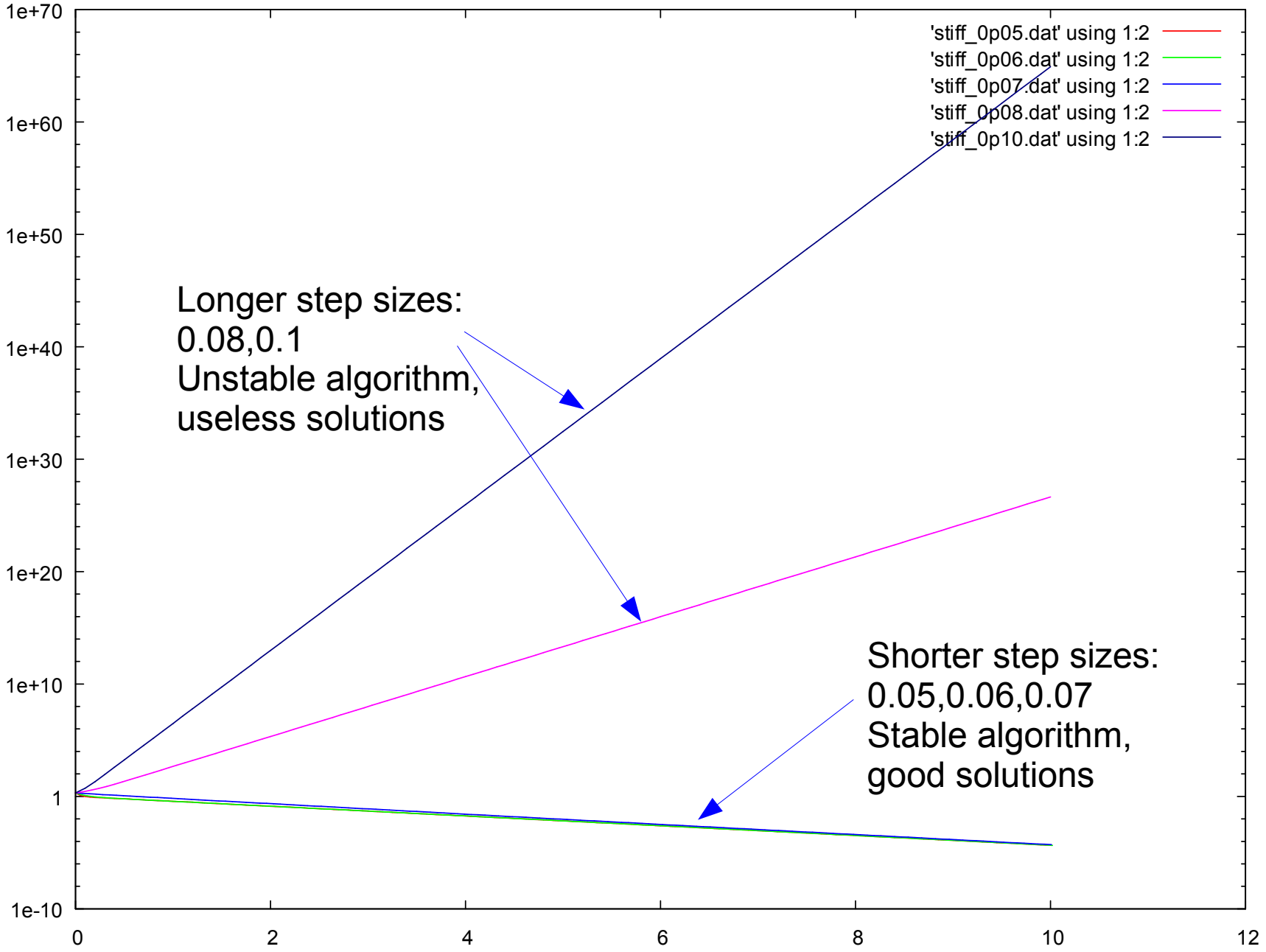


e^{-39x} terms decay rapidly to an insignificant contribution, so it may be tempting to try to save computation time by increasing the *x* step to just adequately cover the longer decay constant

Runge-Kutta Algorithm with Larger Step Size

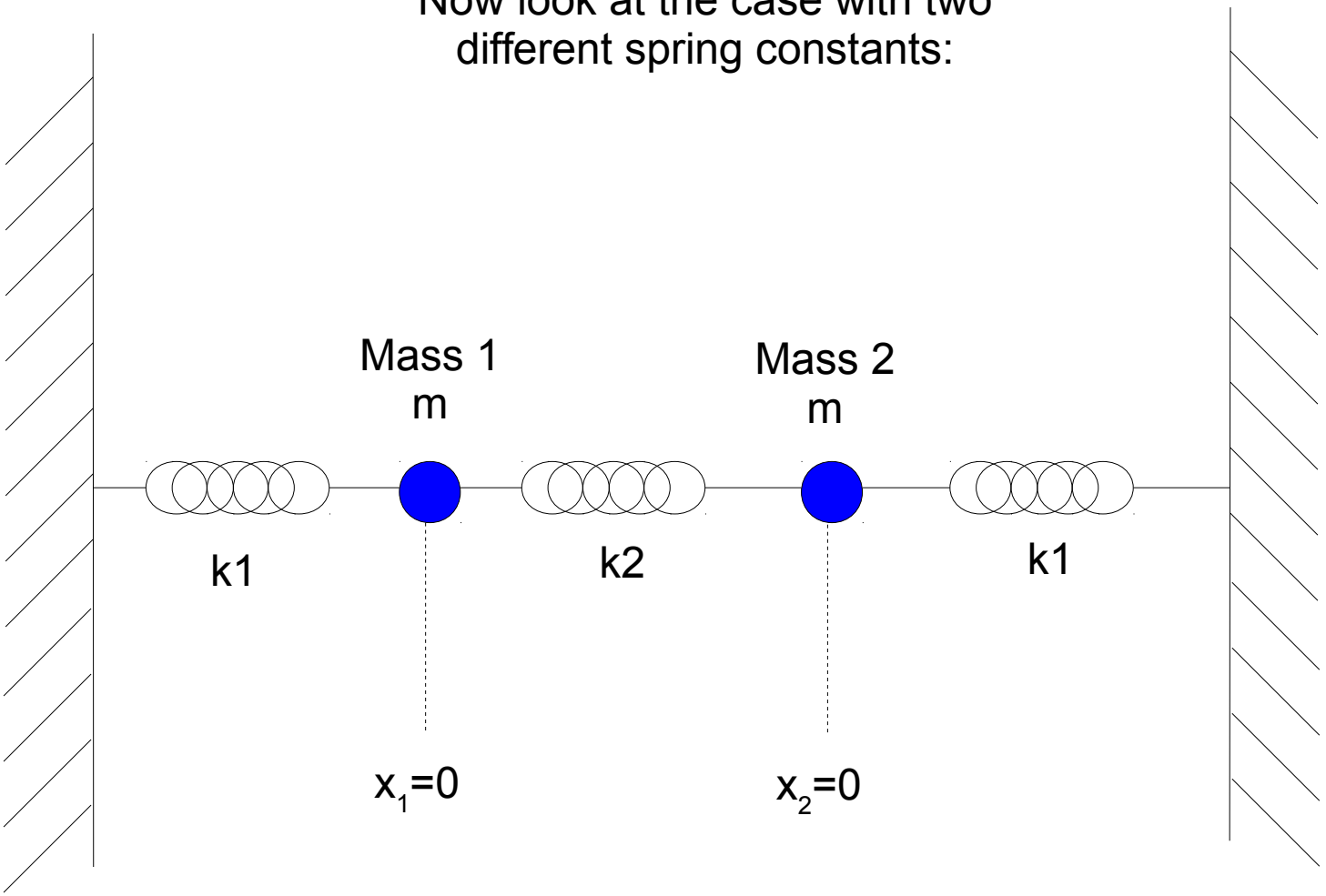


Onset of Instability of ODE Algorithm



Mass/Spring Coupled Oscillator

Now look at the case with two different spring constants:



Equations of Motion for Two Mass Oscillator

$$\text{Let} \quad \omega_1^2 = \frac{k_1}{m} \text{ and } \omega_2^2 = \frac{k_2}{m} \qquad \text{so}$$

$$\frac{d^2 x_1}{dt^2} = -(\omega_1^2 + \omega_2^2) x_1 + \omega_2^2 x_2 \qquad \frac{d^2 x_2}{dt^2} = \omega_2^2 x_1 - (\omega_1^2 + \omega_2^2) x_2$$

Assume solutions in the form: $x_1 = a_1 e^{i\alpha t} \qquad x_2 = a_2 e^{i\alpha t}$

$$-\alpha^2 a_1 e^{i\alpha t} = -(\omega_1^2 + \omega_2^2) a_1 e^{i\alpha t} + \omega_2^2 a_2 e^{i\alpha t} \qquad -\alpha^2 a_2 e^{i\alpha t} = \omega_2^2 a_1 e^{i\alpha t} - (\omega_1^2 + \omega_2^2) a_2 e^{i\alpha t}$$

Then cancel the time dependent factors:

$$-\alpha^2 a_1 = -(\omega_1^2 + \omega_2^2) a_1 + \omega_2^2 a_2 \qquad -\alpha^2 a_2 = \omega_2^2 a_1 - (\omega_1^2 + \omega_1^2) a_2$$

$$\begin{bmatrix} \omega_1^2 + \omega_2^2 & -\omega_2^2 \\ -\omega_2^2 & \omega_1^2 + \omega_2^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \alpha^2 \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

$$\begin{bmatrix} \frac{k_1}{k_2} + 1 & -1 \\ -1 & \frac{k_1}{k_2} + 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \lambda \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad \text{where} \quad \lambda = \frac{\alpha^2}{\omega_2^2}$$

Solutions to Eigenvalue Problem

Solution to this eigenvalue and eigenvector problem is

$$\lambda_1 = \frac{k_1}{k_2} + 2 : \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad \lambda_2 = \frac{k_1}{k_2} : \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Any linear combination of eigenvectors are valid solutions, where the linear combination coefficients are to be determined by the initial conditions

$$\lambda = \frac{\alpha^2}{\omega_2^2} \quad \text{so} \quad \alpha = \pm\sqrt{\lambda}\,\omega_2$$

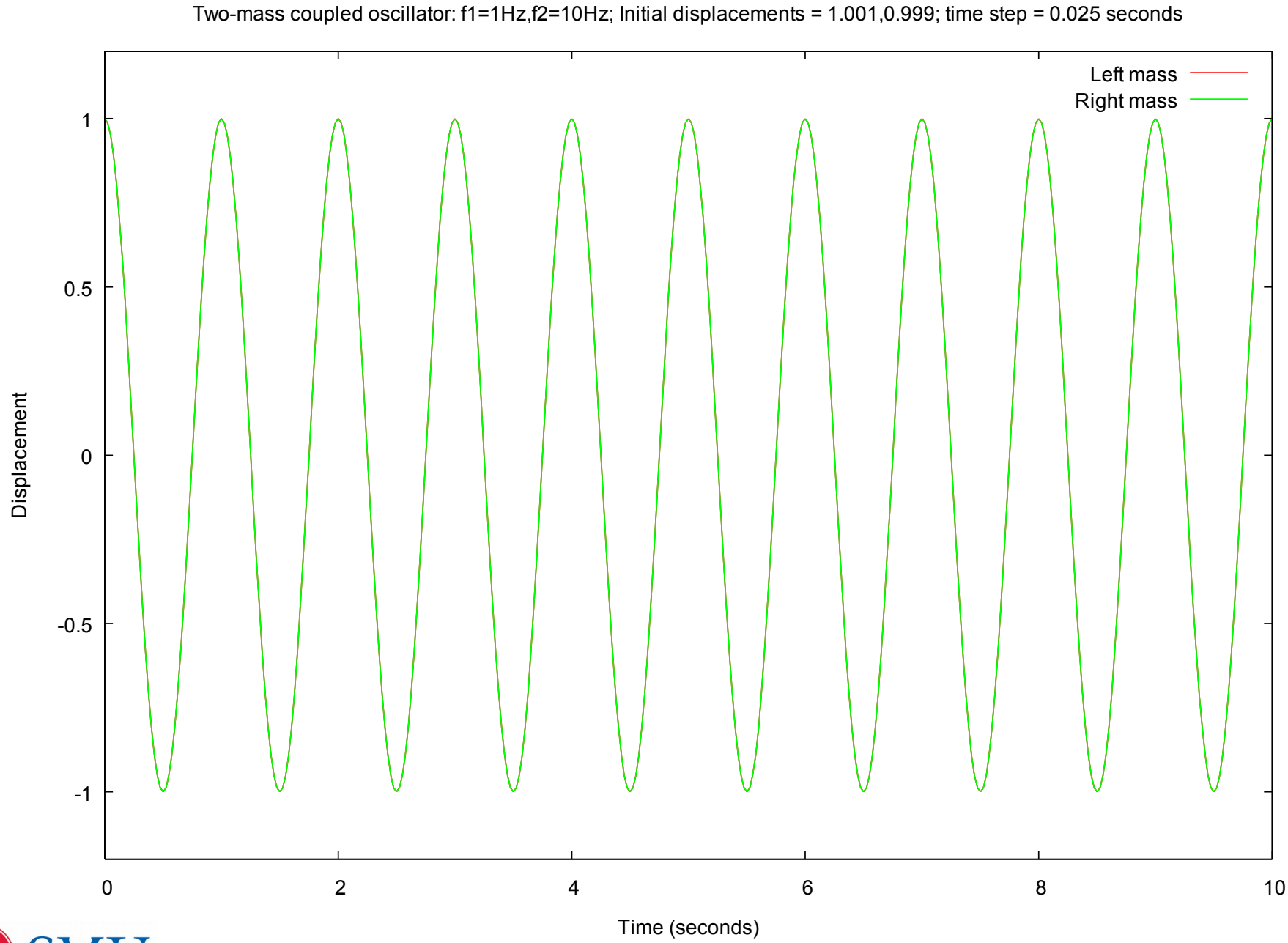
The general solution is

$$x_1(t) = b_1 e^{i\sqrt{\lambda_1}\omega_2 t} + b_2 e^{-i\sqrt{\lambda_1}\omega_2 t} + b_3 e^{i\sqrt{\lambda_2}\omega_2 t} + b_4 e^{-i\sqrt{\lambda_2}\omega_2 t}$$
$$x_2(t) = -b_1 e^{i\sqrt{\lambda_1}\omega_2 t} - b_2 e^{-i\sqrt{\lambda_1}\omega_2 t} + b_3 e^{i\sqrt{\lambda_2}\omega_2 t} + b_4 e^{-i\sqrt{\lambda_2}\omega_2 t}$$

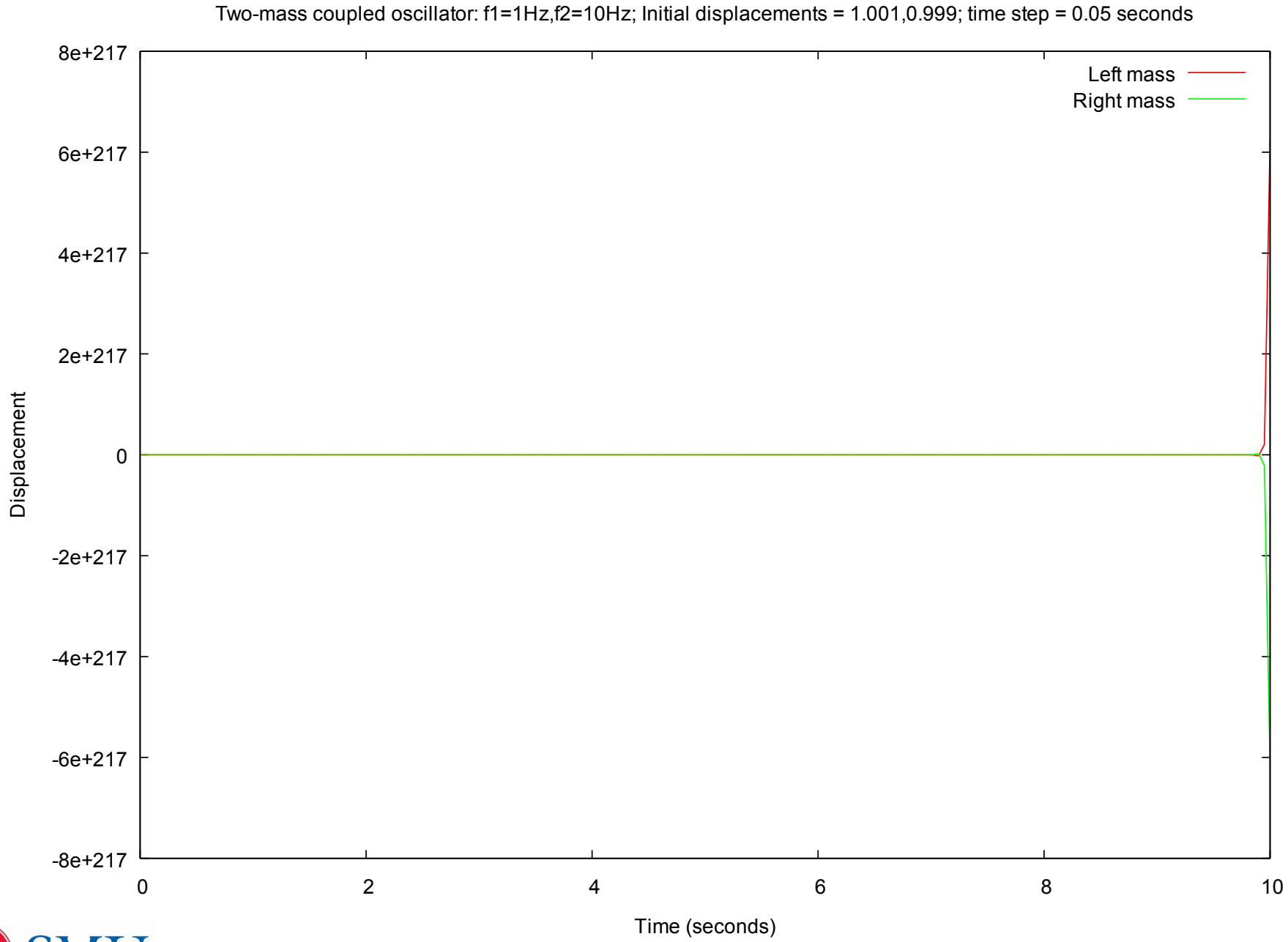
Or in a more convenient form

$$x_1(t) = c_1 \cos\left(\sqrt{\omega_1^2 + 2\omega_2^2}\,t\right) + c_2 \sin\left(\sqrt{\omega_1^2 + 2\omega_2^2}\,t\right) + c_3 \cos(\omega_1 t) + c_4 \sin(\omega_1 t)$$
$$x_2(t) = -c_1 \cos\left(\sqrt{\omega_1^2 + 2\omega_2^2}\,t\right) - c_2 \sin\left(\sqrt{\omega_1^2 + 2\omega_2^2}\,t\right) + c_3 \cos(\omega_1 t) + c_4 \sin(\omega_1 t)$$

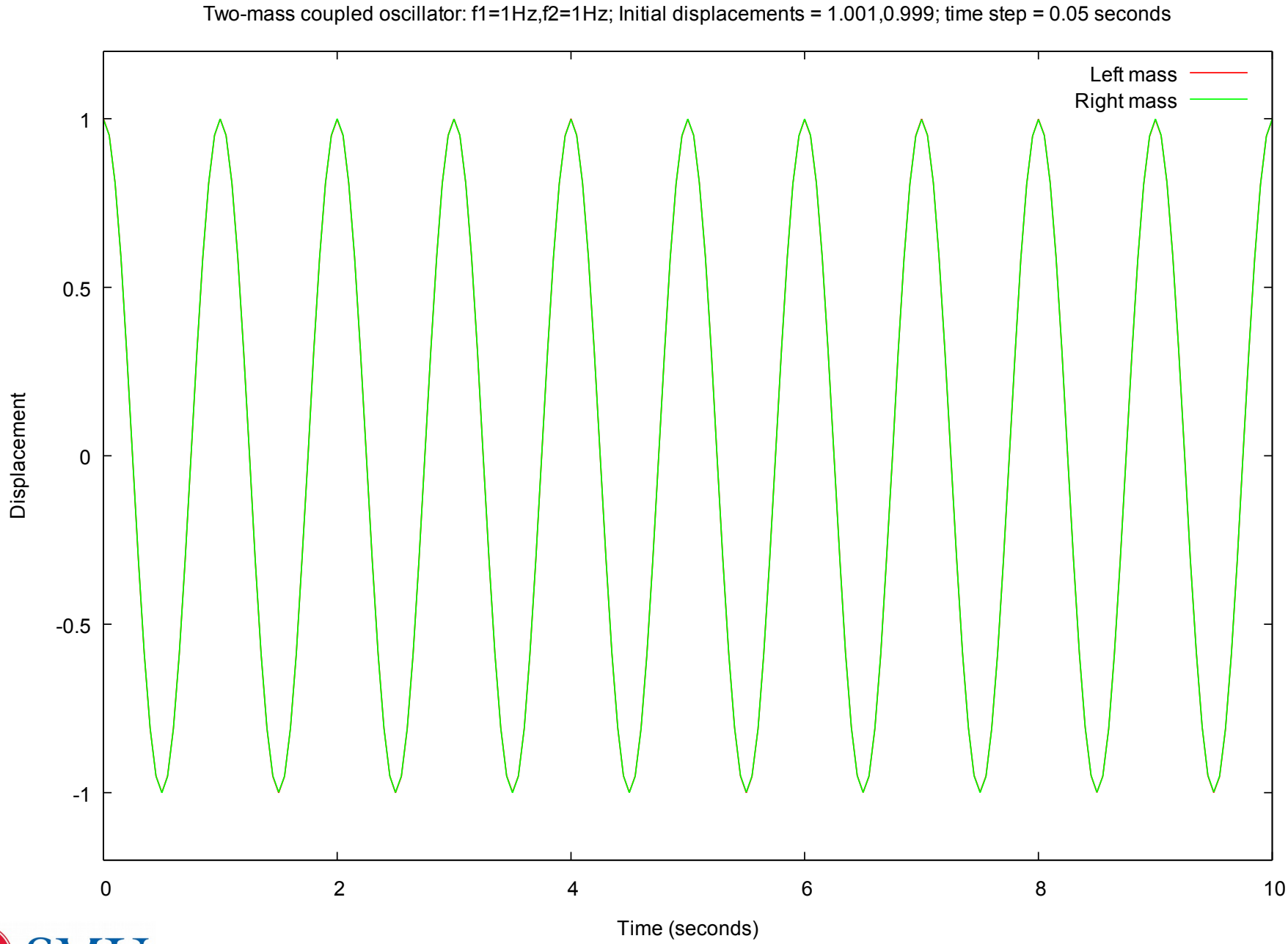
Two-mass Coupled Oscillator with Widely Separated Natural Frequencies



Two-mass Coupled Oscillator with Widely Separated Natural Frequencies



Two-mass Coupled Oscillator with Identical Natural Frequencies



Fifth-Fourth Order Runge-Kutta-Fehlberg Algorithm

$$y_{i+1}^{4\text{th}} = y_i + h\cdot\left[\frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5\right]$$

where

$$k_1 = f\left(x_i, y_i\right)$$

$$k_2 = f\left(x_i + \frac{h}{4}, y_i + \frac{h}{4}\cdot k_1\right)$$

$$k_3 = f\left(x_i + \frac{3h}{8}, y_i + \frac{3h}{32}\cdot k_1 + \frac{9h}{32}\cdot k_2\right)$$

$$k_4 = f\left(x_i + \frac{12h}{13}, y_i + \frac{1932h}{2197}\cdot k_1 - \frac{7200h}{2197}\cdot k_2 + \frac{7296h}{2197}\cdot k_3\right)$$

$$k_5 = f\left(x_i + h, y_i + \frac{439h}{216}\cdot k_1 - 8h\cdot k_2 + \frac{3680h}{513}\cdot k_3 - \frac{845h}{4104}\cdot k_4\right)$$

Is also a 4th order Runge-Kutta algorithm. But why use it if it costs one more function evaluation than the classical 4th order R-K algorithm?

Fifth-Fourth Order Runge-Kutta-Fehlberg Algorithm

Because we can get a 5th order Runge-Kutta algorithm from:

$$y_{i+1} = y_i + h \cdot \left[\frac{16}{135} k_1 + \frac{6656}{12825} k_3 + \frac{28561}{56430} k_4 - \frac{9}{50} k_5 + \frac{2}{55} k_6 \right]$$

when just one more slope is computed:

$$k_6 = f\left(x_i + \frac{h}{2}, y_i - \frac{8\mathrm{h}}{27} \cdot k_1 + 2\mathrm{h} \cdot k_2 - \frac{3544\mathrm{h}}{2565} \cdot k_3 + \frac{1859\mathrm{h}}{4104} \cdot k_4 - \frac{11\mathrm{h}}{40} \cdot k_5 \right)$$

and in addition an estimate of the truncation error from the difference of the 5th and 4th order steps:

$$\epsilon = \left| y_{i+1}^{4\mathrm{th}} - y_{i+1} \right|$$

Runge-Kutta-Fehlberg Algorithm with Adaptive Step Size

Take a step in *x* with the current step size *h* and compute candidate values for the 4th order and 5th order solution step

$$y_{i+1}^{4\text{th}} \text{ and } y_{i+1}$$

and an estimate of the truncation error from the difference of the 5th and 4th order steps:

$$\epsilon = |y_{i+1}^{4\text{th}} - y_{i+1}|$$

If the current truncation error is within specified target limits, keep the candidate value of the 5th order solution step as the next *y* value and go to the next *x* step keeping the step size unchanged

If the current truncation error is smaller than a specified minimum error allowed, keep the candidate value of the 5th order solution step as the next *y* value and go to the next *x* step after doubling the step size

If the current truncation error is larger than a specified maximum error allowed, discard the candidate values and retry another step from the current *x* value after halving the step size

Runge-Kutta-Fehlberg Algorithm with Adaptive Step Size

Compile time constant sets the maximum power of two by which the step size may be decreased

Compile time constant sets the fraction of the error tolerance that signals a step size increase

```
#define RKF54_MAX_DECREASE 14
#define RKF54_TEST_INCREASE 0.2

int rkf54_adaptive(int order,void (*func)(int n,double x,double y[],double yp[]),double xstart,
                   double xstop,double xincmax,double y[],double tol[]) {
  int dec;
    .
    .
    .
  dec = 0;
```

Pass the initial and maximum step size allowed

Pass an array of tolerance values, the limit of the truncation error for each state variable that is acceptable without decreasing the step size

Internal variable keeping track of the power of 2 by which the step size is decreased

Runge-Kutta-Fehlberg Algorithm with Adaptive Step Size

At the end of the simulation, a report is sent to the stderr stream summarizing the truncation error for each state variable, and the depth to which the step size had to be decreased:

State variable	Minimum error	Maximum error	Mean error
0	0	99.225792	1.2769709
1	0	97.354866	1.2808573
2	0	0.00055877871	4.9252261e-07
3	0	0.00052681007	5.8644326e-07
4	3.0517578e-05	99.924194	24.051726
5	3.0517578e-05	99.569489	18.790362
6	7.0940587e-11	0.00291373	0.00012197284
7	2.4374458e-10	0.0020341325	0.00015985743
8	4.5776367e-05	99.924194	23.955143
9	0.00074768066	99.569489	18.672032
10	9.4587449e-10	0.0029137297	0.00012215824
11	2.4174369e-09	0.0020341325	0.00016002856

```
Total number of steps = 17555
Maximum step size decrease exponent = 9
```

Electron Energies in Quantum Wells

Solve Schroedinger's equation in one dimension to find allowed energy levels:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + (V(x)-E)\psi(x) = 0$$

$$\text{or} \quad \frac{d^2\psi(x)}{dx^2} = \frac{2m}{\hbar^2}(V(x)-E)\psi(x)$$

Allowed energy levels are those values of E that admit a continuous and finite solution $\psi(x)$ that is able to be normalized

$$\int \psi^* \psi \, dx = 1$$

Transform to System of First Order ODEs

Let $y_1 = \psi(x)$ $\qquad y_2 = \dfrac{dy_1}{dx}$

So system becomes

$$\frac{dy_1}{dx} = y_2$$
$$\frac{dy_2}{dx} = \frac{2m}{\hbar^2}\left(V(x) - E\right) y_1$$

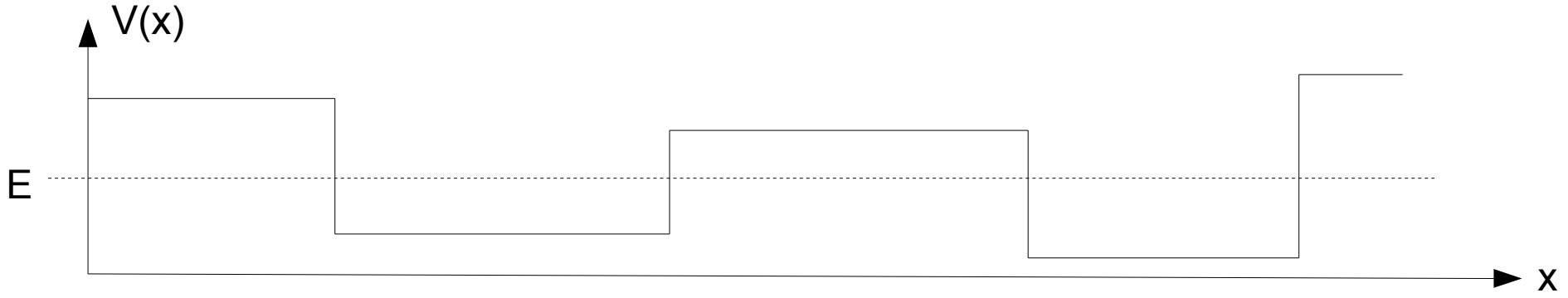
Follow the convention that masses are expressed in energy equivalents mc^2

Then the constant $\dfrac{2m}{\hbar^2} = \dfrac{2mc^2}{\hbar^2 c^2} = 5.1363071 \times 10^{13}\, eV^{-2} m^{-2} \cdot mc^2$

Boundary value problem established by normalization constraint

Electron Energies in Quantum Wells

For special case of constant potential regions



$$\frac{d^2\psi(x)}{dx^2} = \frac{2m}{\hbar^2}(V-E)\psi(x)$$

In regions where $E \leqslant V$ general solution is $\psi(x) = Ae^{k_1 x} + Be^{-k_1 x}$

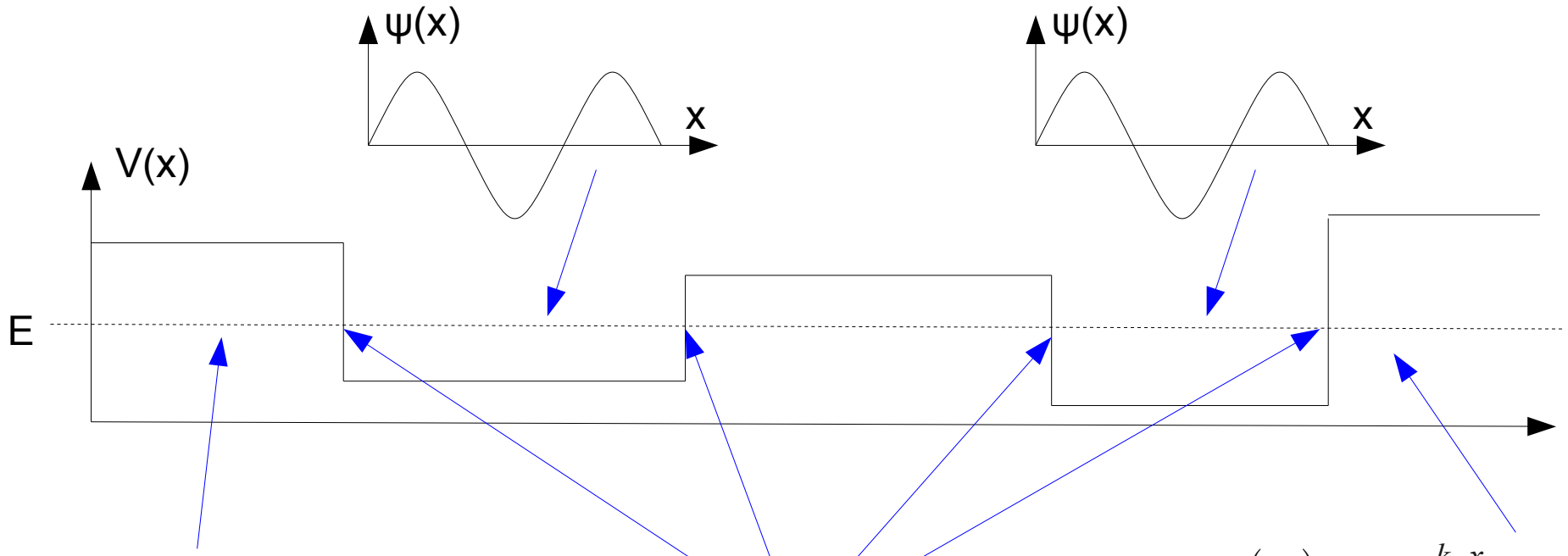
where $k_1 = \frac{\sqrt{2m(V-E)}}{\hbar}$

In regions where $E > V$ general solution is $\psi(x) = C\sin(k_2 x) + D\cos(k_2 x)$

where $k_2 = \frac{\sqrt{2m(E-V)}}{\hbar}$

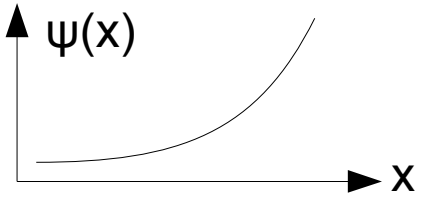
Solving for Electron Energies Analytically

Stitch together solutions in each potential region under normalization and continuity constraints



$\psi(x) = A e^{k_1 x} + B e^{-k_1 x}$

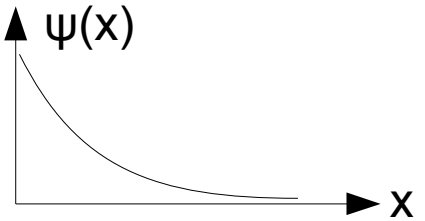
$B = 0$ required by normalization constraint



$\psi(x)$ and $\frac{d\psi(x)}{dx}$ required to be continuous across boundaries

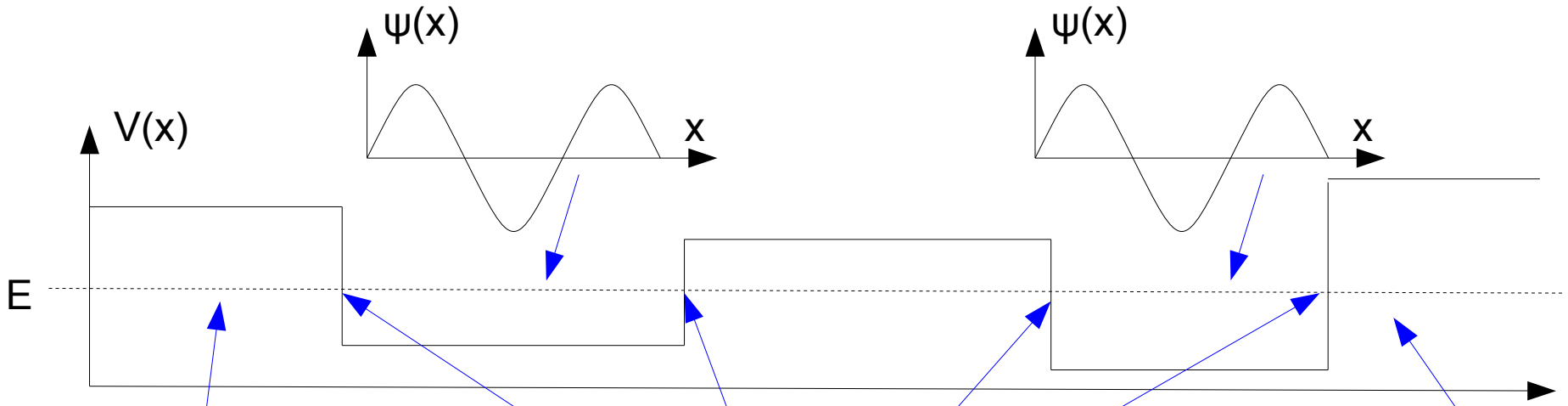
$\psi(x) = A e^{k_1 x} + B e^{-k_1 x}$

$A = 0$ required by normalization constraint



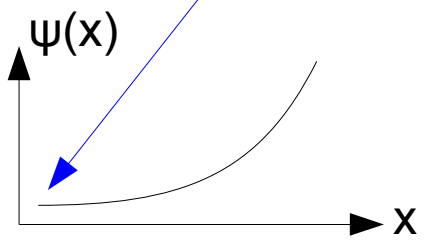
Solving for Electron Energies Numerically

Initialize ψ for low *x*, use iterative differential equation algorithm, such as Runge-Kutta, to enforce continuity across boundaries, and search for E values giving normalizable ψ solution for high *x*



$$\psi(x) = A e^{k_1 x} + B e^{-k_1 x}$$

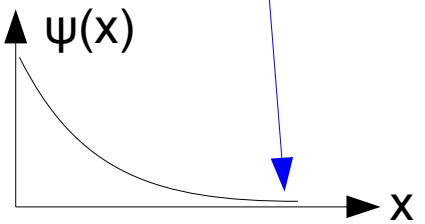
Initialize $\psi(x)$ and $\frac{d\psi(x)}{dx}$ to make $B = 0$



$\psi(x)$ and $\frac{d\psi(x)}{dx}$ will be continuous as R-K algorithm steps proceed

$$\psi(x) = A e^{k_1 x} + B e^{-k_1 x}$$

Shoot for ψ target to make $A = 0$



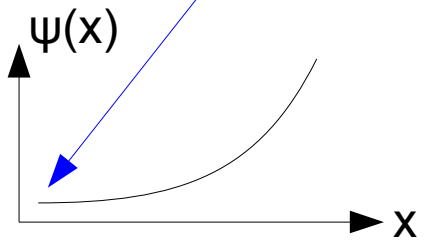
Solving for Electron Energies Numerically

Initialize ψ for low *x*, use iterative differential equation algorithm, such as Runge-Kutta, to enforce continuity across boundaries, and search for E values giving normalizable ψ solution for high *x*

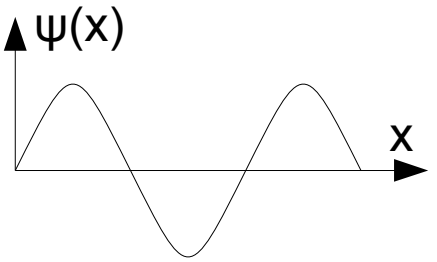
$$\psi(x) = A e^{k_1 x} + B e^{-k_1 x}$$

Initialize $\psi(x)$ and $\frac{d\psi(x)}{dx}$

to make $B = 0$



.....

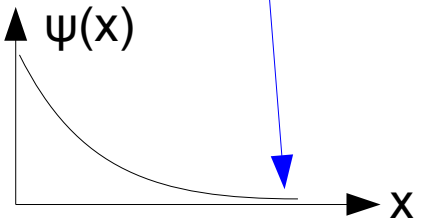


$$\psi(x) = A e^{k_1 x} + B e^{-k_1 x}$$

Shoot for ψ target

to make $A = 0$

.....



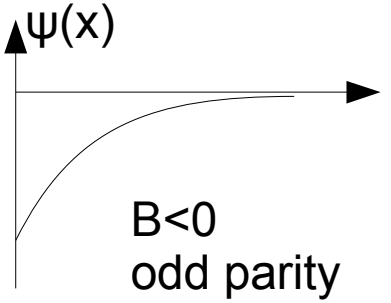
B>0
even parity

$$k = \sqrt{\frac{-2mE}{\hbar^2}} \qquad a = \text{ width of well}$$

Initialize $\psi(x_{init}) = e^{-ka}$

$$\frac{d\psi(x_{init})}{dx} = k e^{-ka}$$

But what is a good numerical target to shoot at? Define a **target error function** that will go to zero when A=0, then shoot for the target error = 0



B<0
odd parity

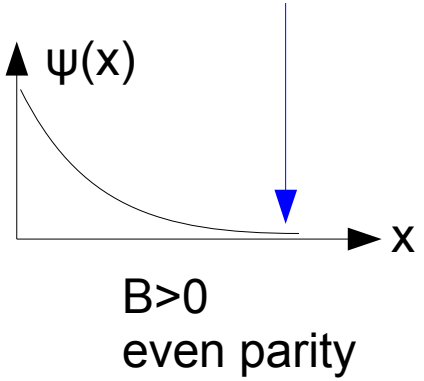
Where x_{init} is one half well width to the left of the first well. This will launch an exponential with only A≠0

Defining a Target Error Function

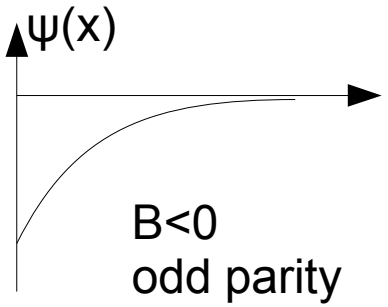
The R-K algorithm gives us $\psi(x)$ and $\frac{d\psi(x)}{dx}$ at the end of the integration.

The A and B coefficients that emerge on the right side will be functions of the electron energy that we launch on the left side. What function can we define that goes to zero when the solution is composed of only a decaying exponential, or A=0, B≠0? Use the opposite relationship between ψ and its derivative that was exploited when launching a solution on the left hand side.

$$\psi(x) = A e^{k_1 x} + B e^{-k_1 x}$$



B>0
even parity



B<0
odd parity

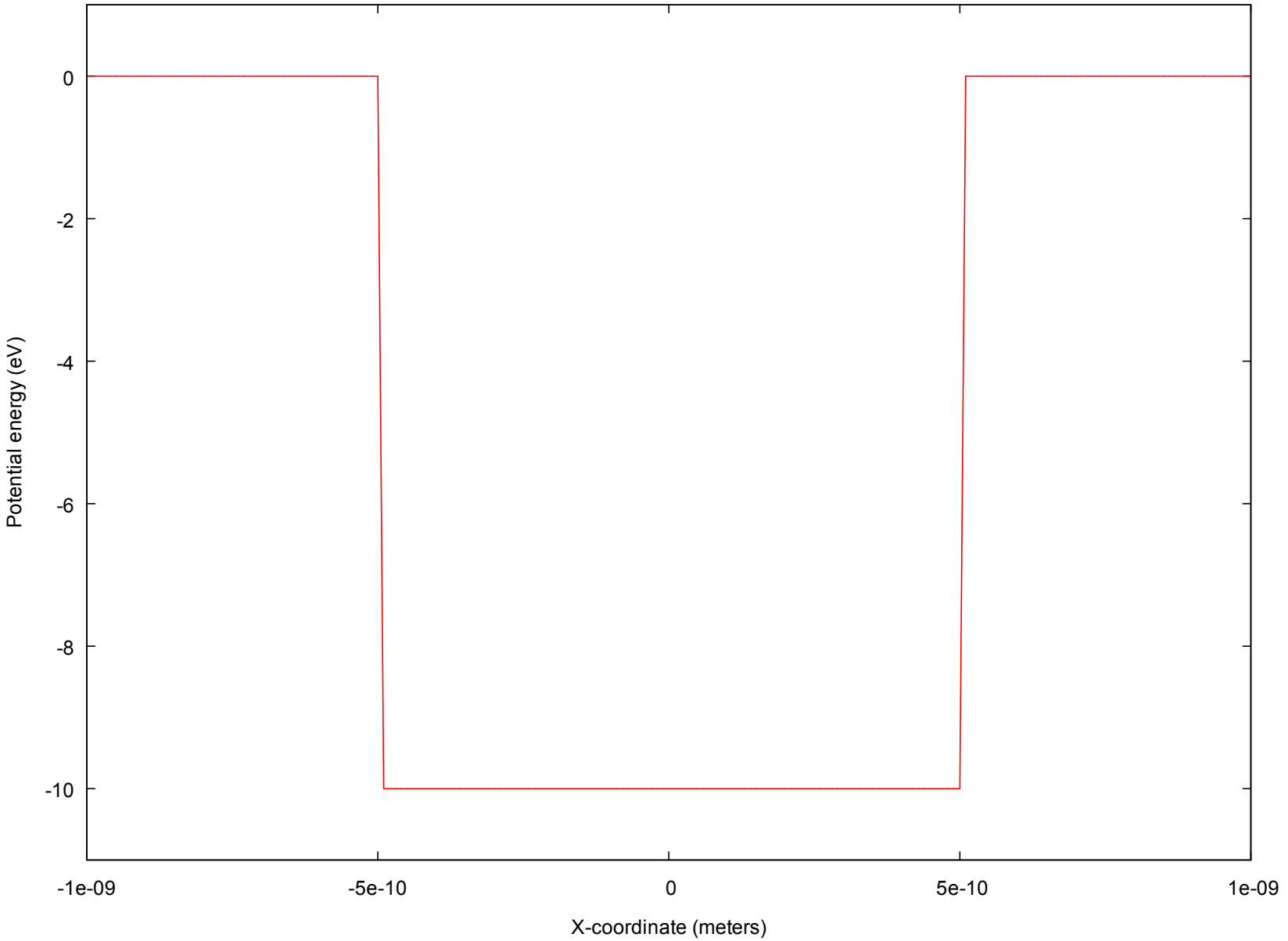
Define target error function as

$$f(E) = \psi(x_{final}) + \frac{1}{k_1}\frac{d\psi(x_{final})}{dx}$$

For $\psi(x) = A(E) e^{k_1 x} + B(E) e^{-k_1 x}$
then $f(E) = 2A(E) e^{k_1 x_{final}}$
which will be 0 for arbitrary x_{final}
only when A(E) = 0

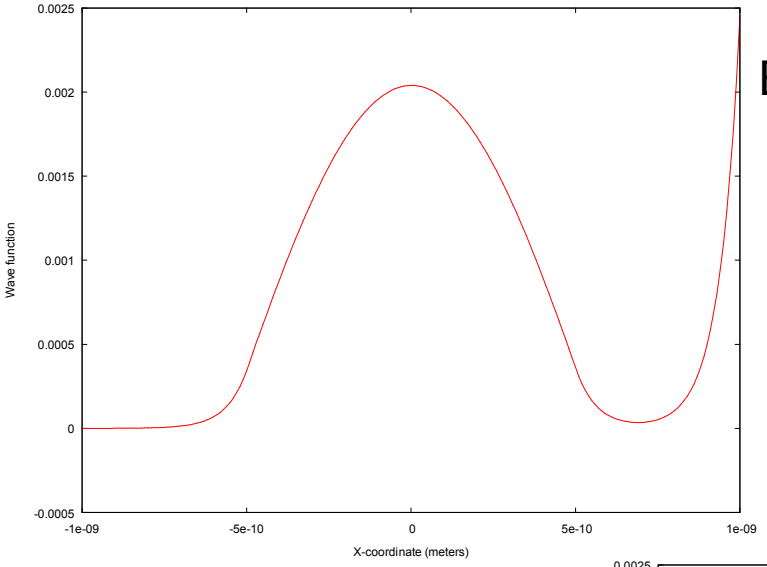
Set x_{final} to be one half well width to the right of the last well

Example Rectangular Potential Well



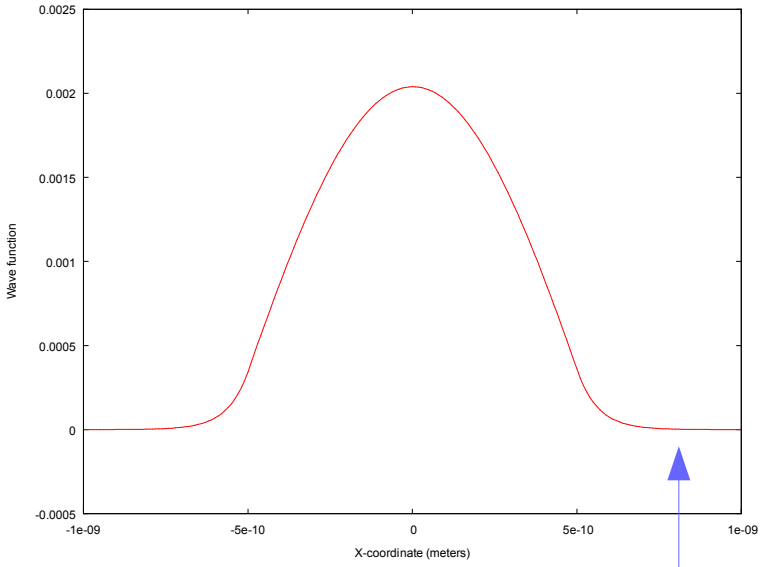
Shooting Converging on Energy Eigenvalue

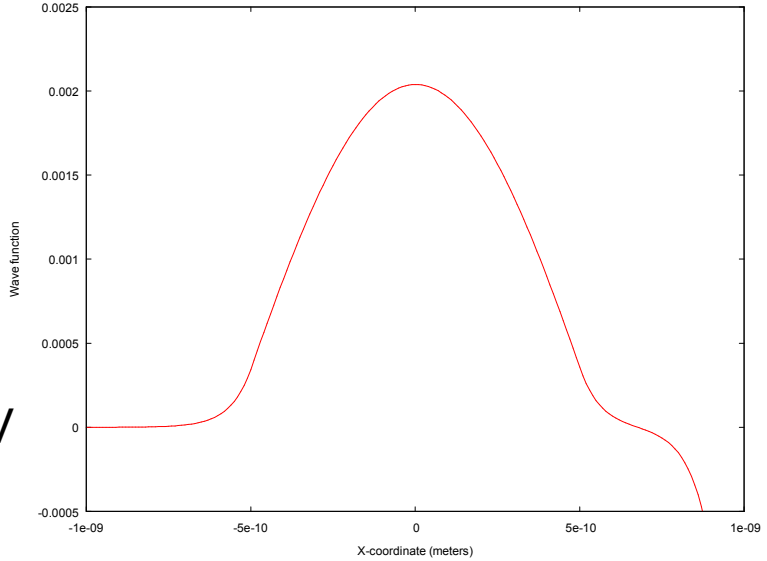
Well width = 1nm well depth = 10eV
Wave function trajectory for even parity lowest energy state



E=-9.7026 eV

E=-9.70244232262 eV



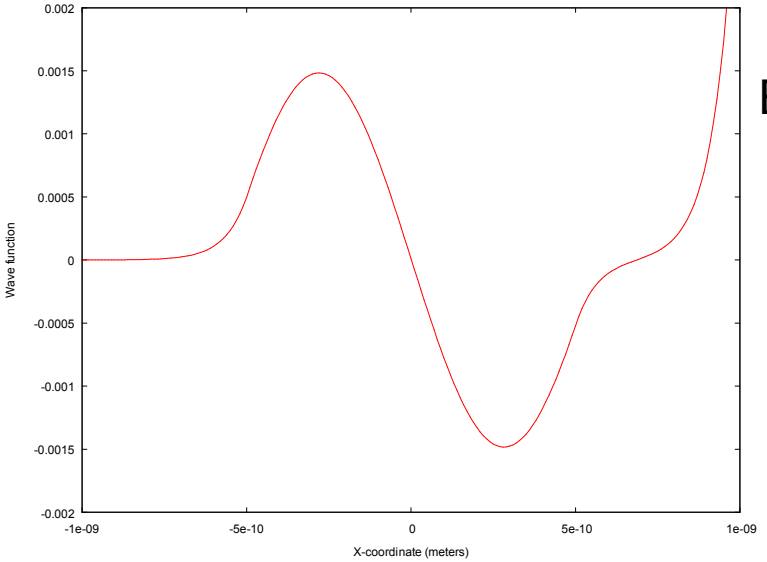


E=-9.7022 eV

Target error function will go to zero when solution on left side consists of only a decaying exponential. Use a root finding algorithm with respect to E to solve for the allowed value of E!

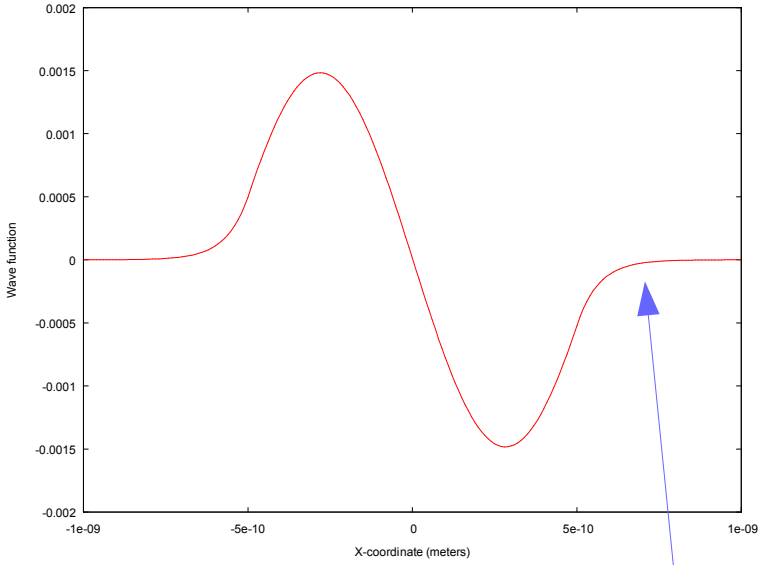
Shooting Converging on Energy Eigenvalue

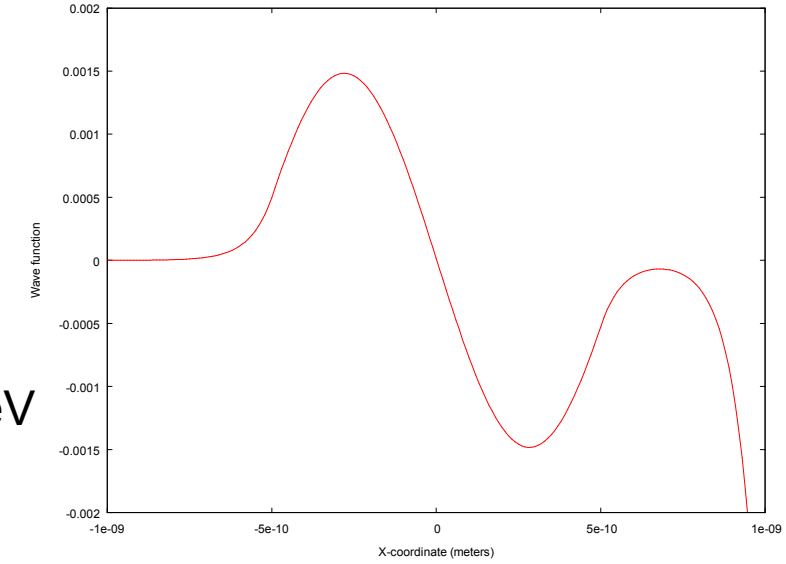
Well width = 1nm well depth = 10eV
Wave function trajectory for odd parity second energy state



E=-8.813 eV

E=-8.81389991413 eV

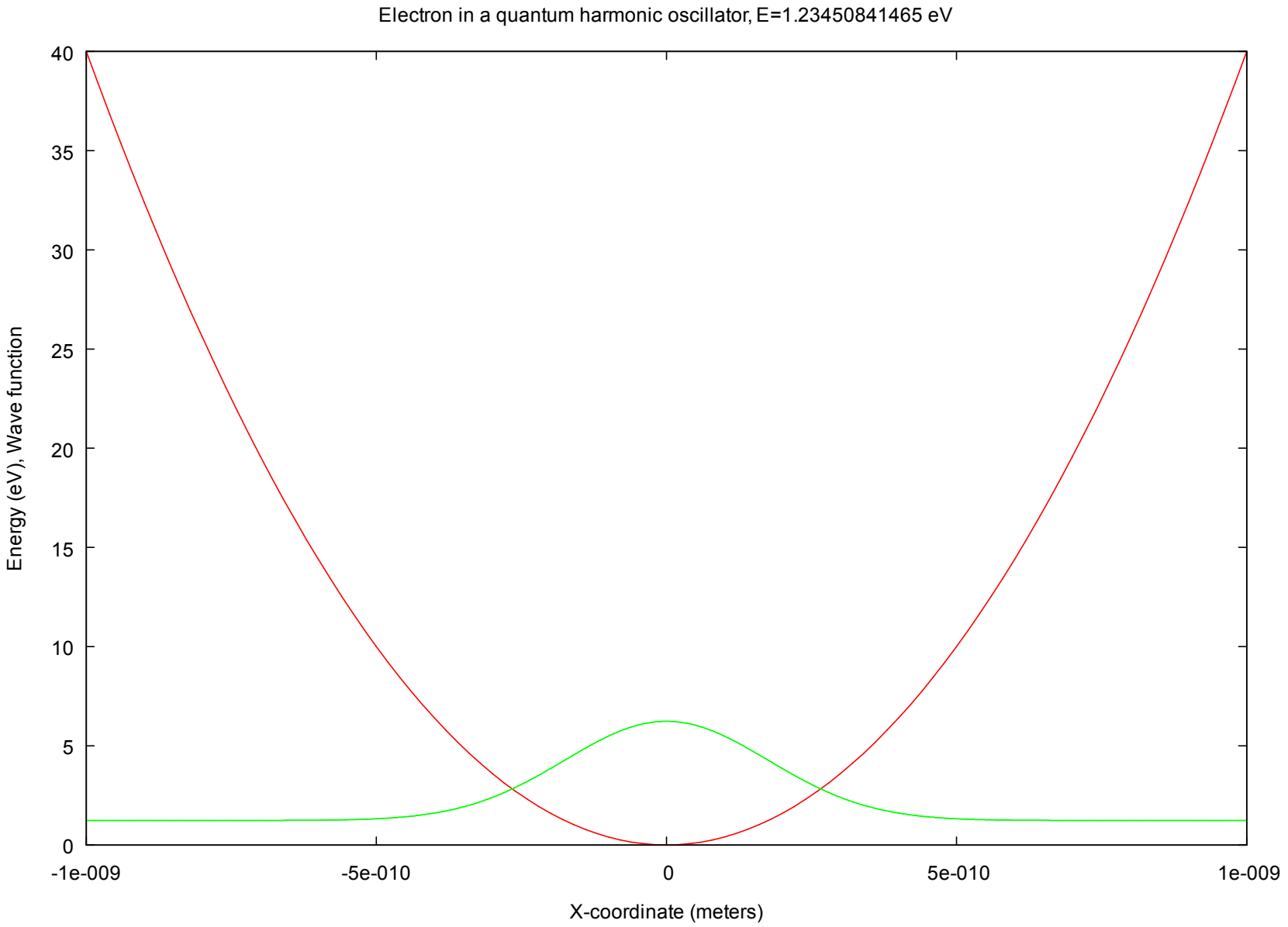




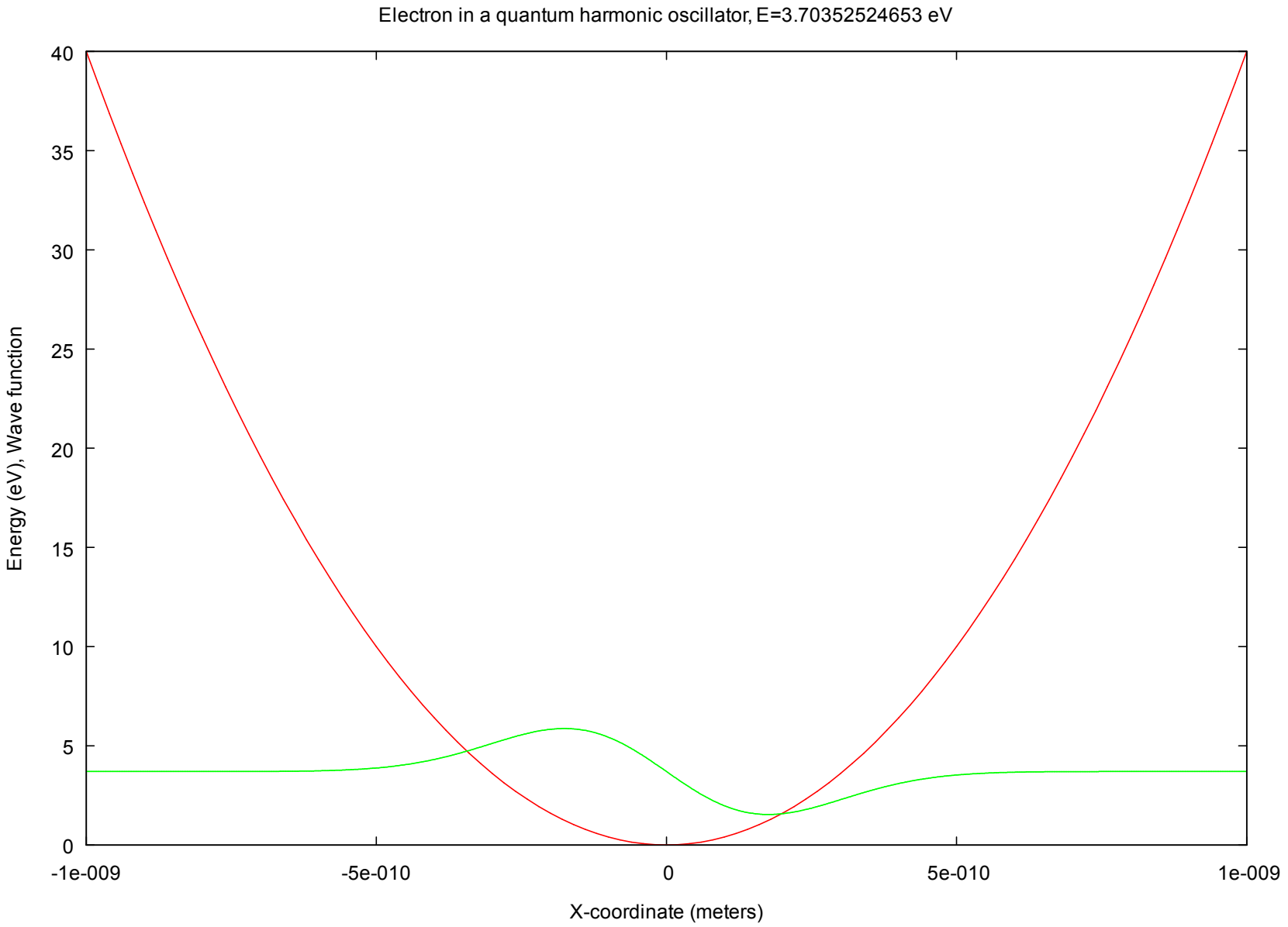
E=-8.815 eV

Target error function will go to zero when solution on left side consists of only a decaying exponential. Use a root finding algorithm with respect to E to solve for the allowed value of E!

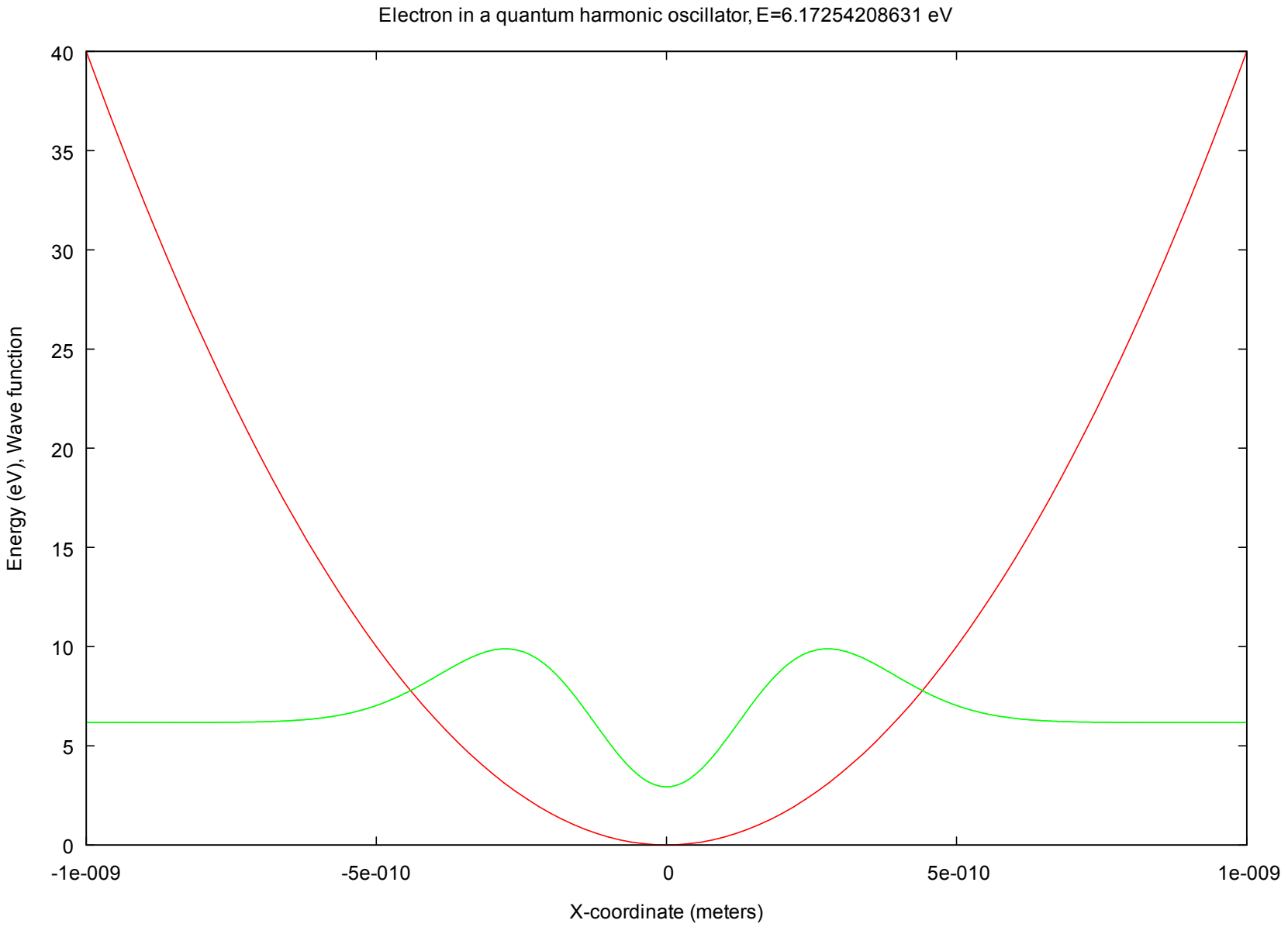
Solution for Electron in a Quantum Harmonic Oscillator



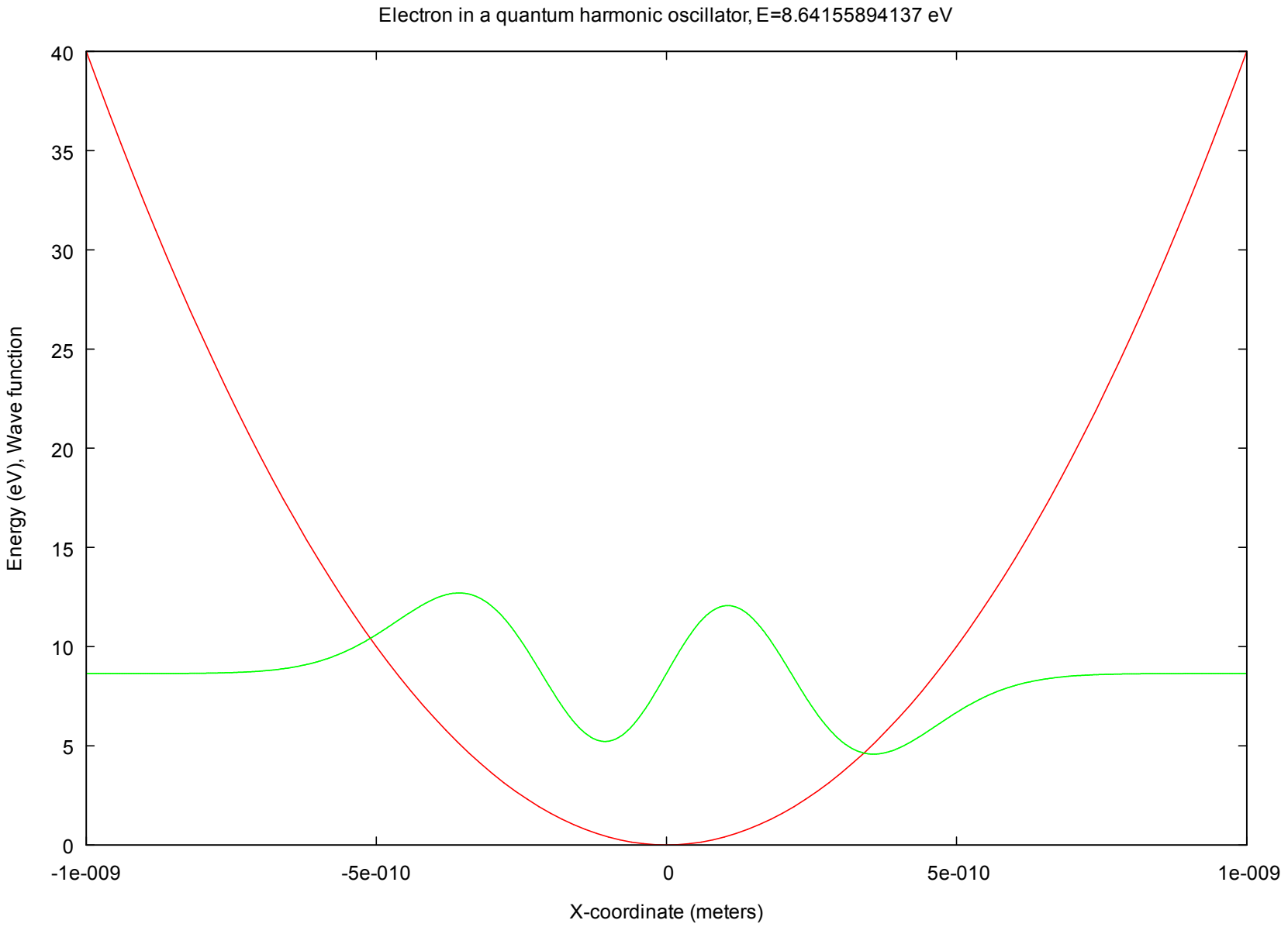
Solution for Electron in a Quantum Harmonic Oscillator



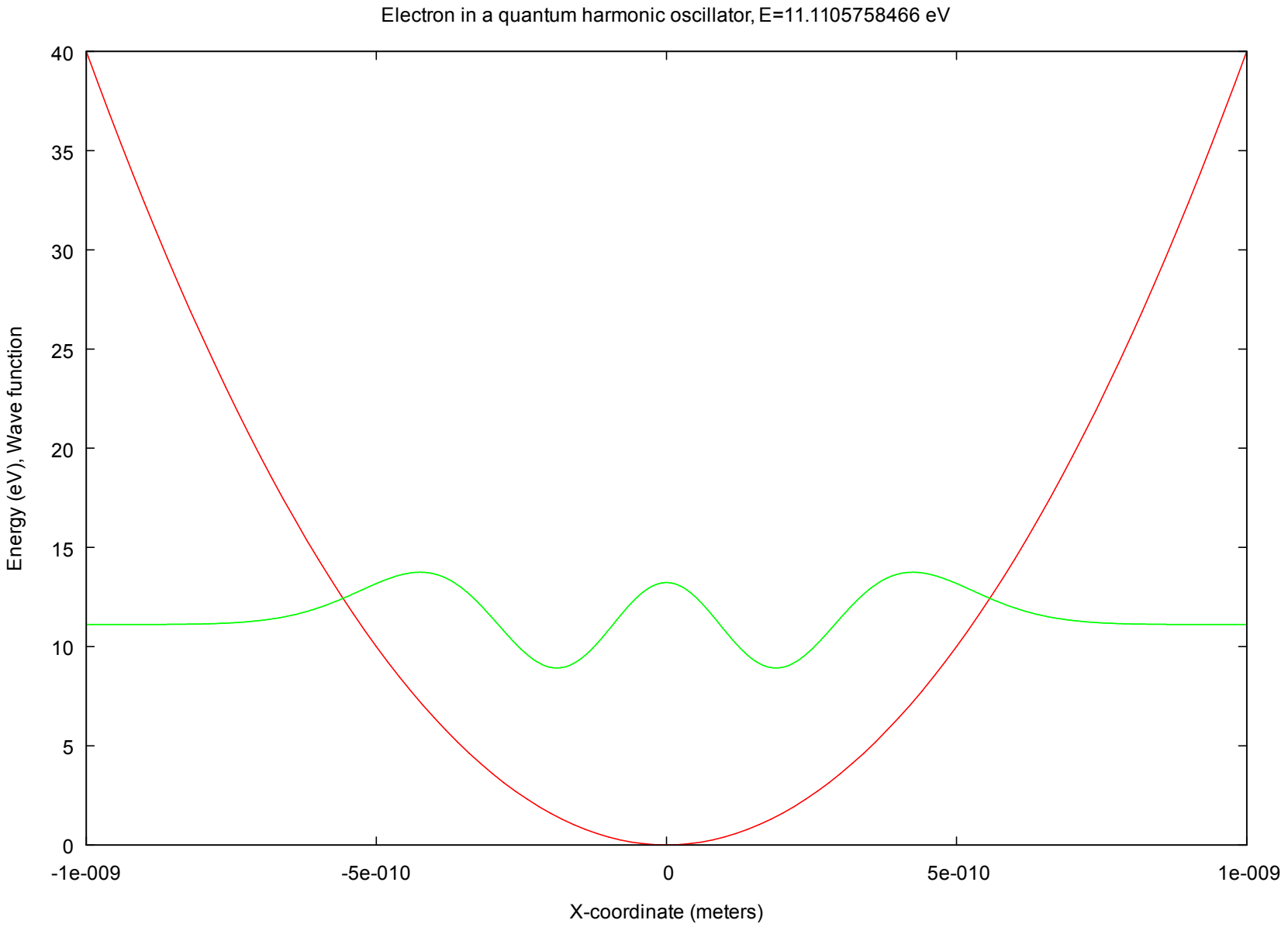
Solution for Electron in a Quantum Harmonic Oscillator



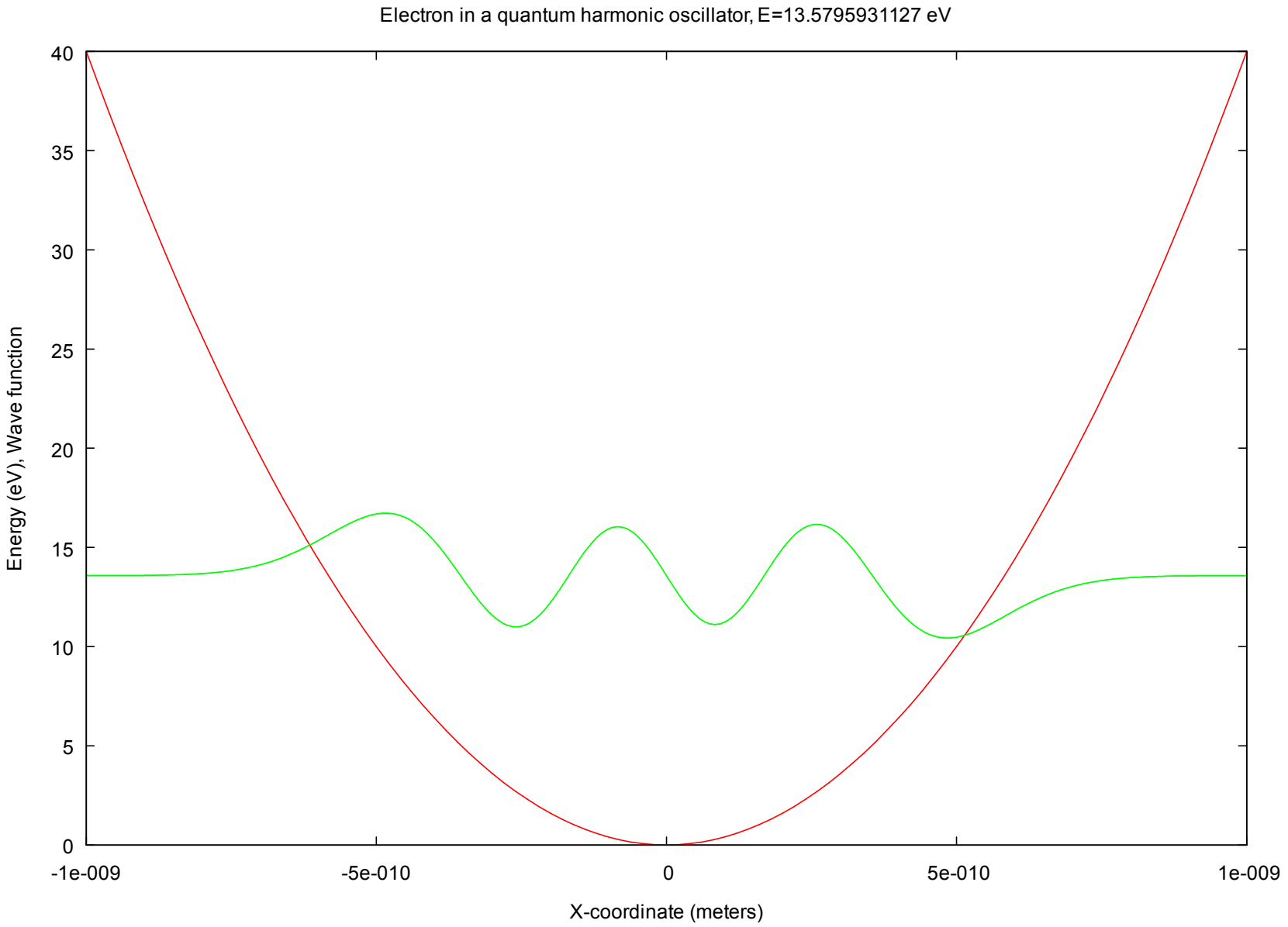
Solution for Electron in a Quantum Harmonic Oscillator



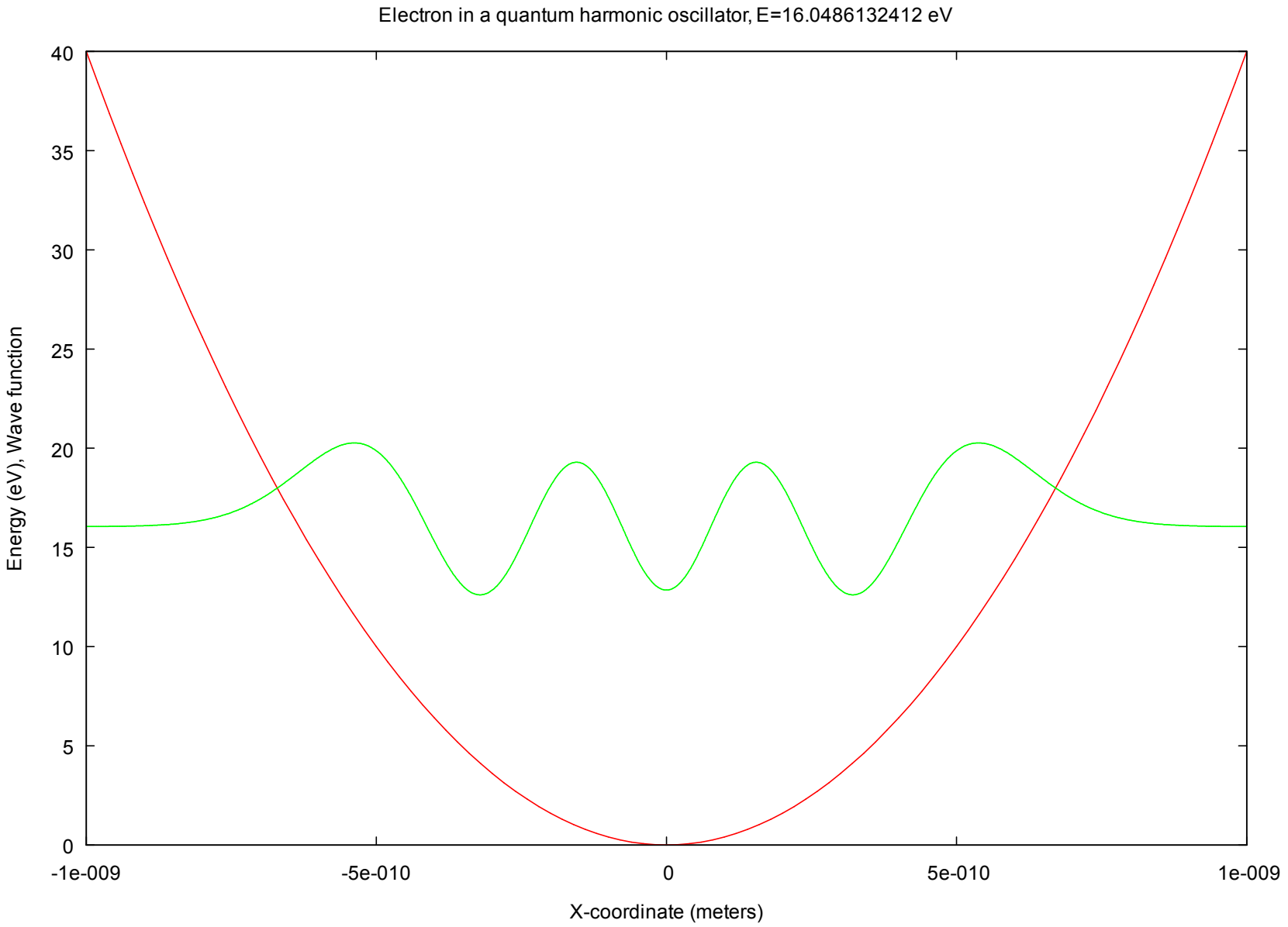
Solution for Electron in a Quantum Harmonic Oscillator



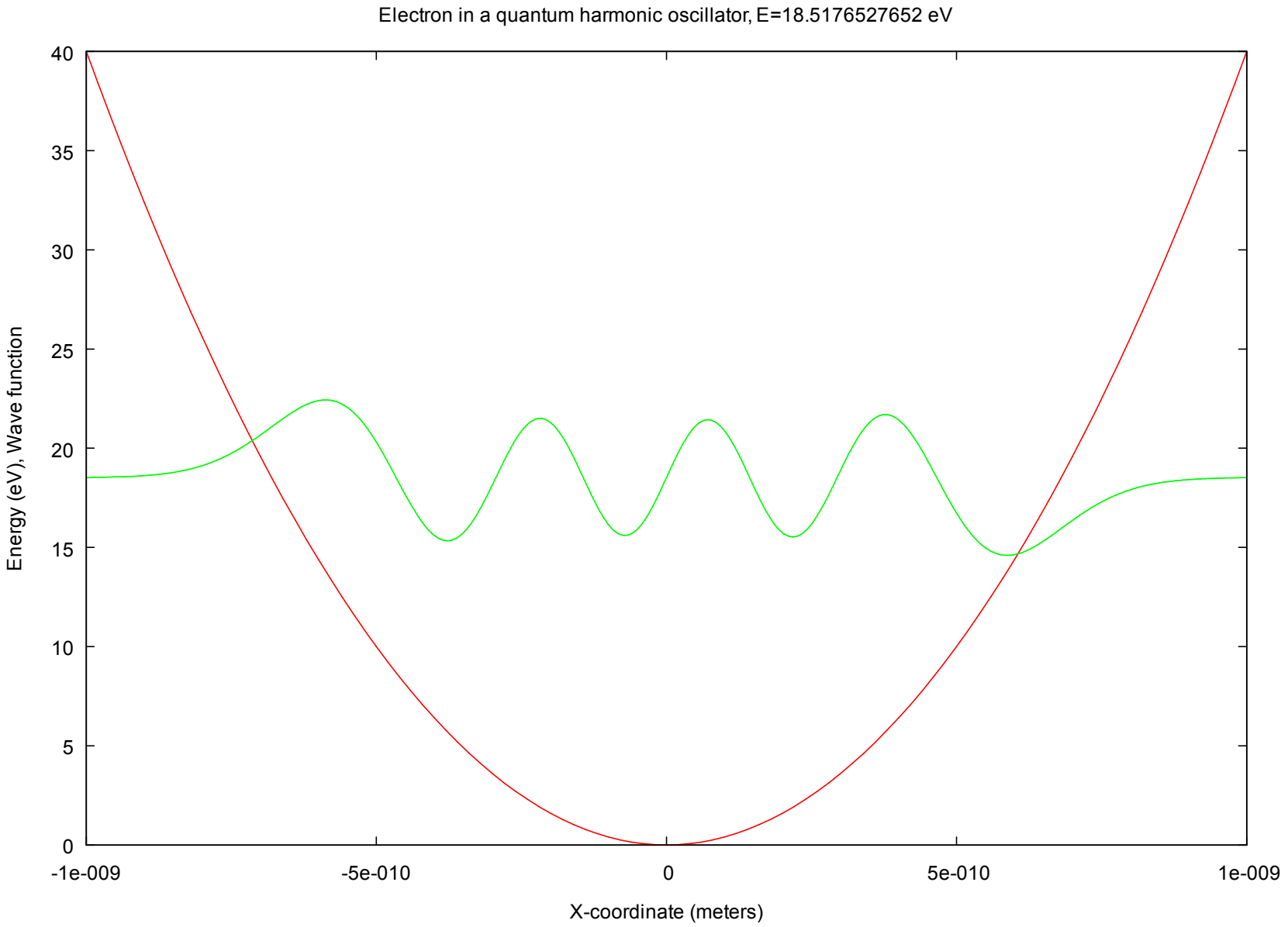
Solution for Electron in a Quantum Harmonic Oscillator



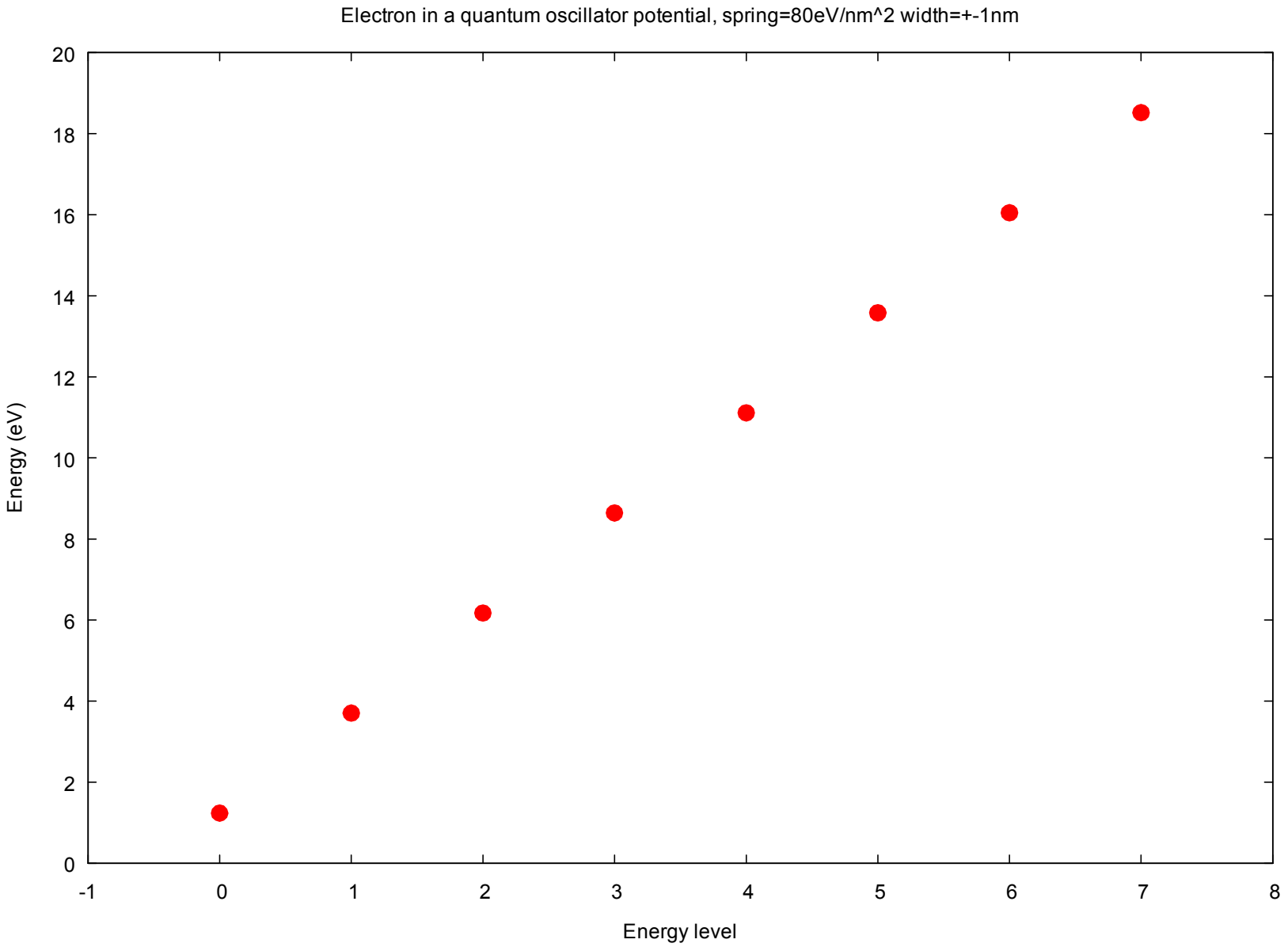
Solution for Electron in a Quantum Harmonic Oscillator



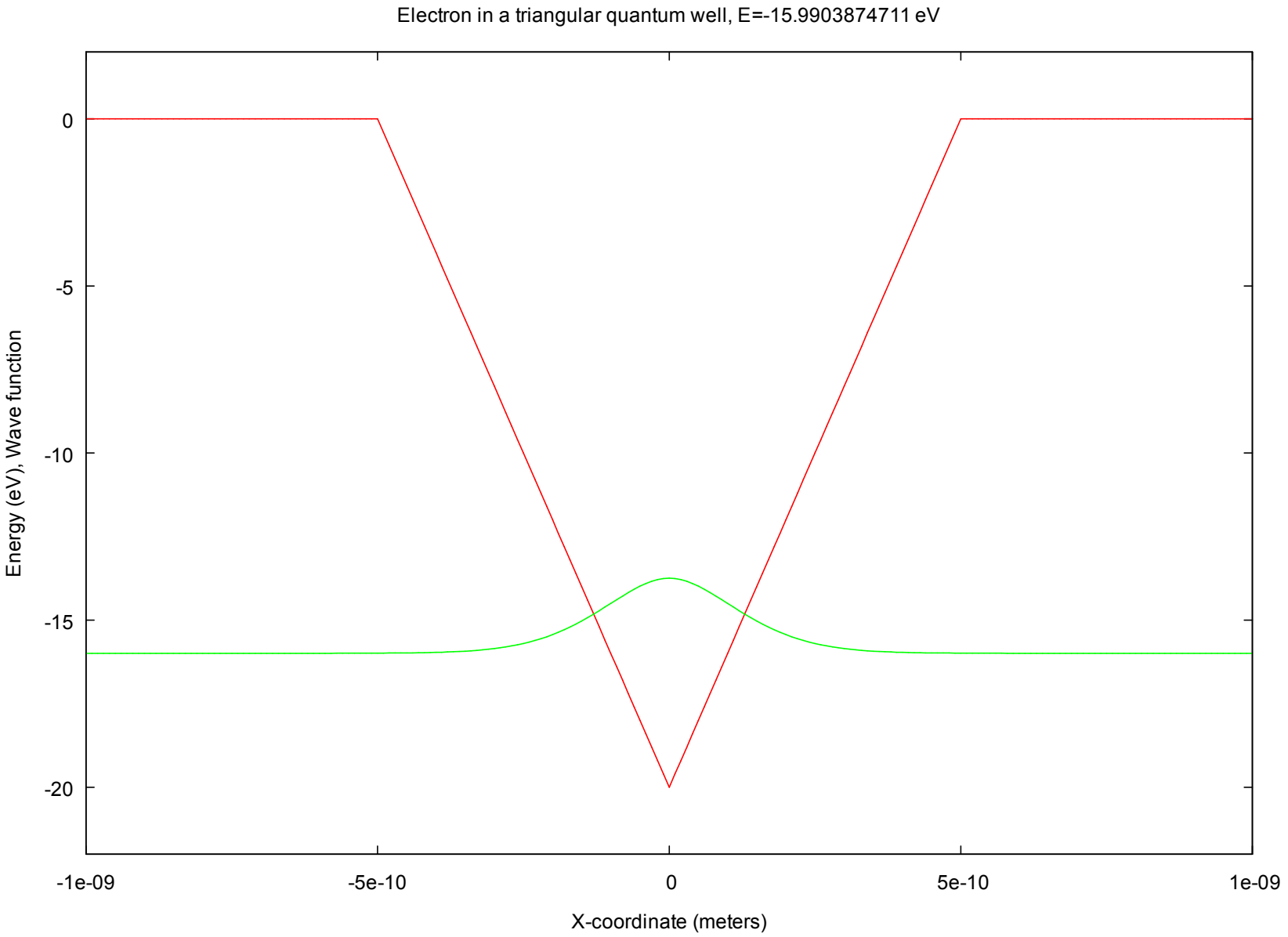
Solution for Electron in a Quantum Harmonic Oscillator



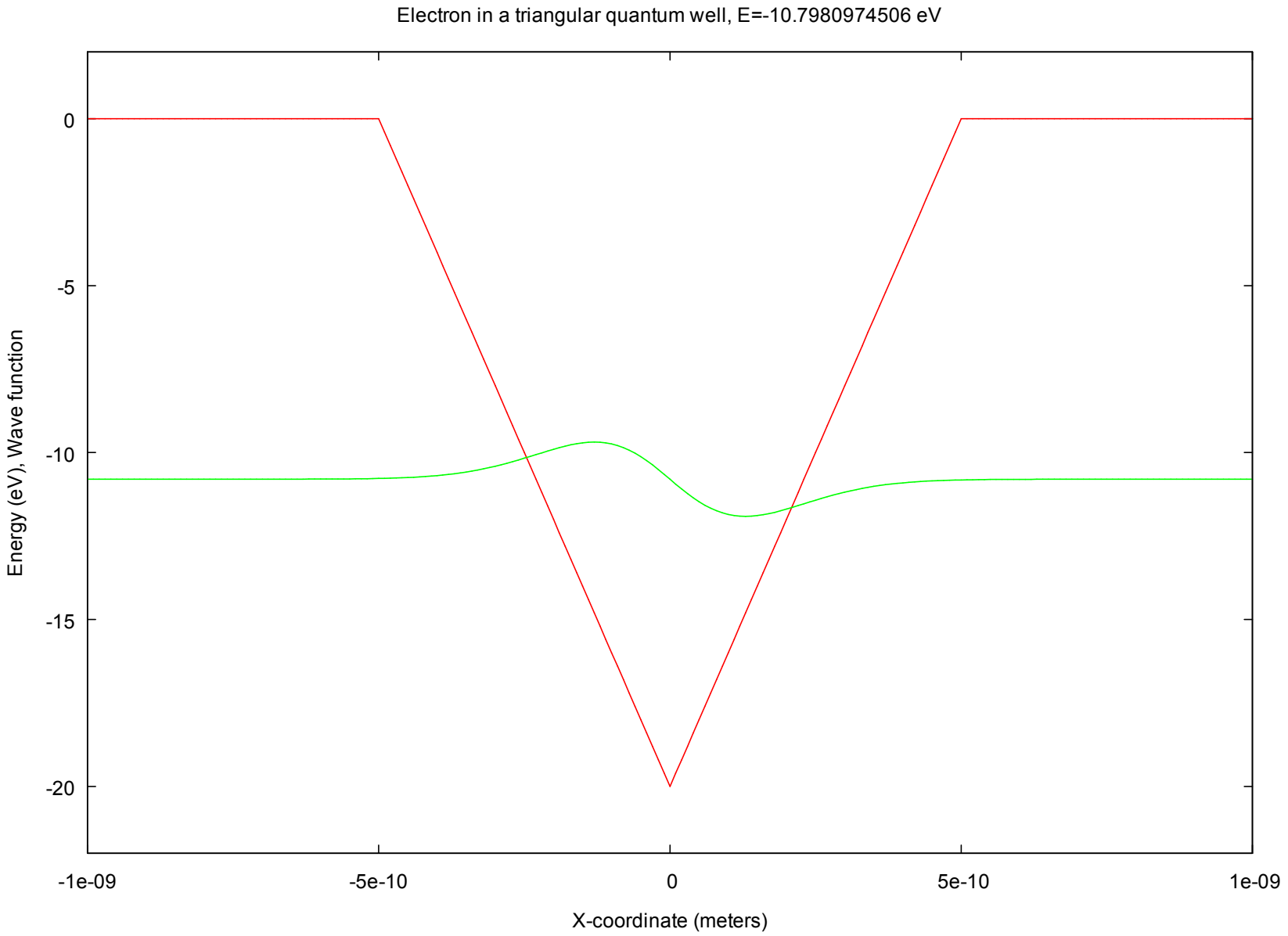
Eigenvalues for a Quantum Harmonic Oscillator



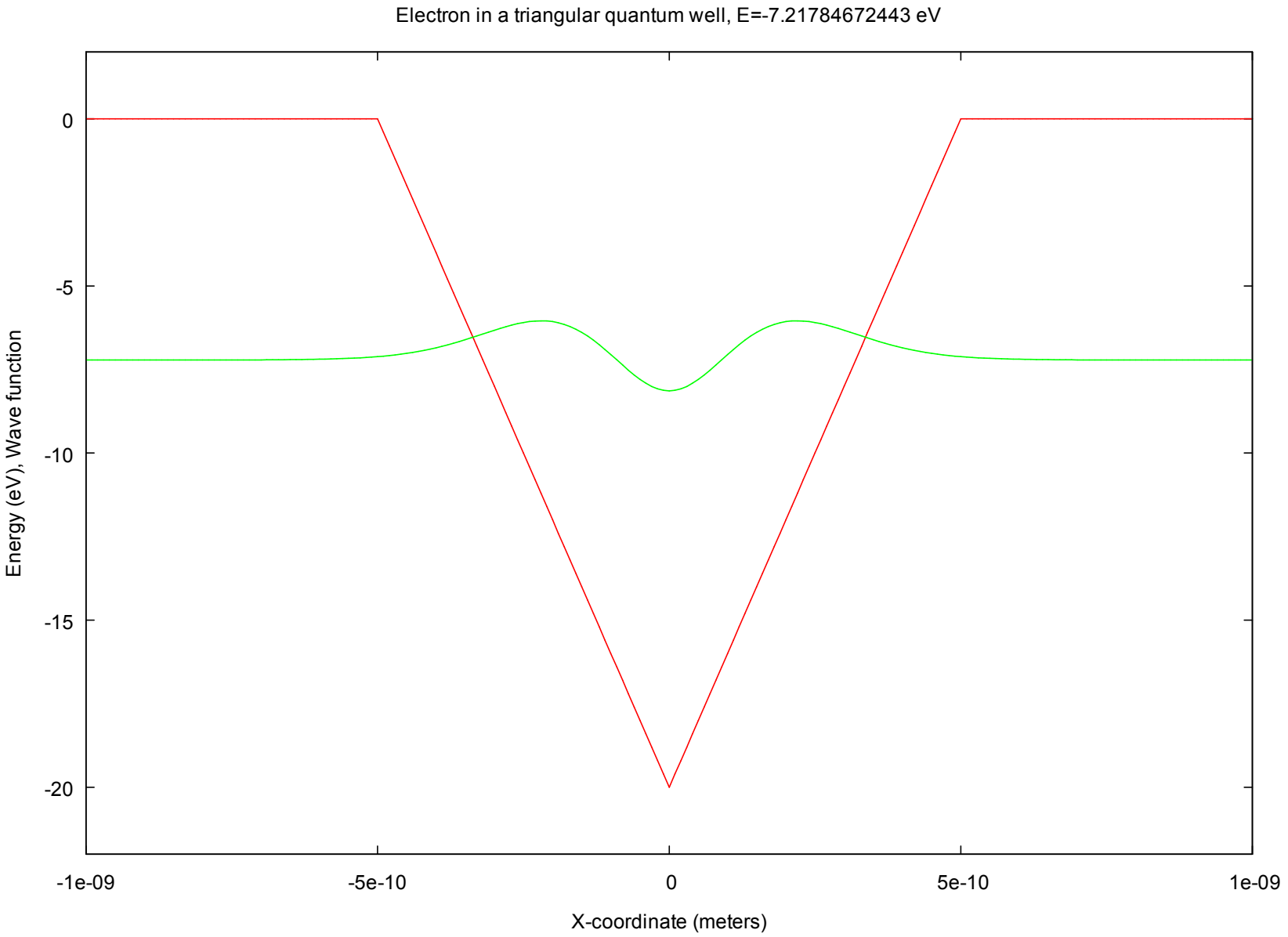
Eigenvalue Solution for Electron in a Quantum Well



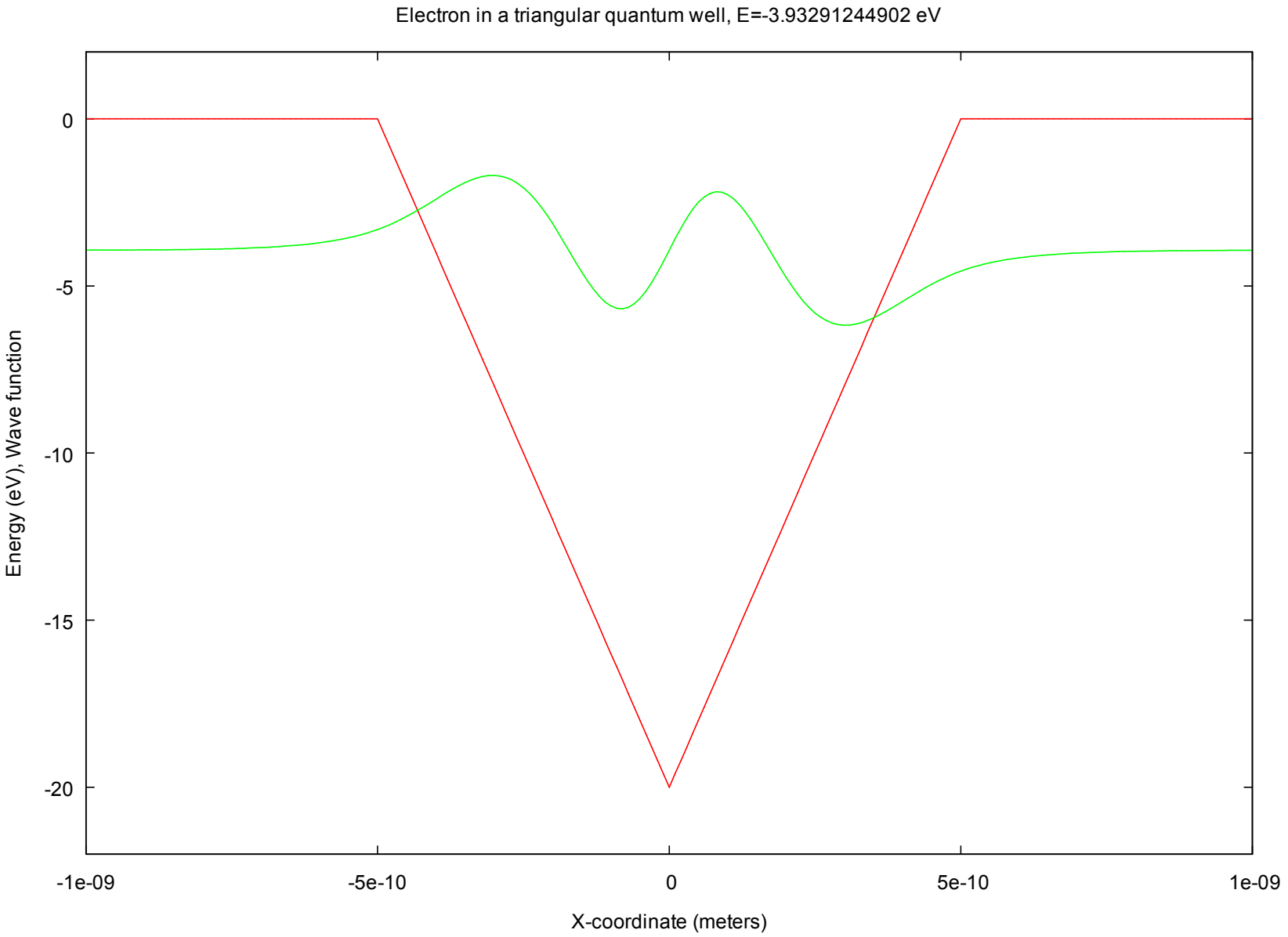
Eigenvalue Solution for Electron in a Quantum Well



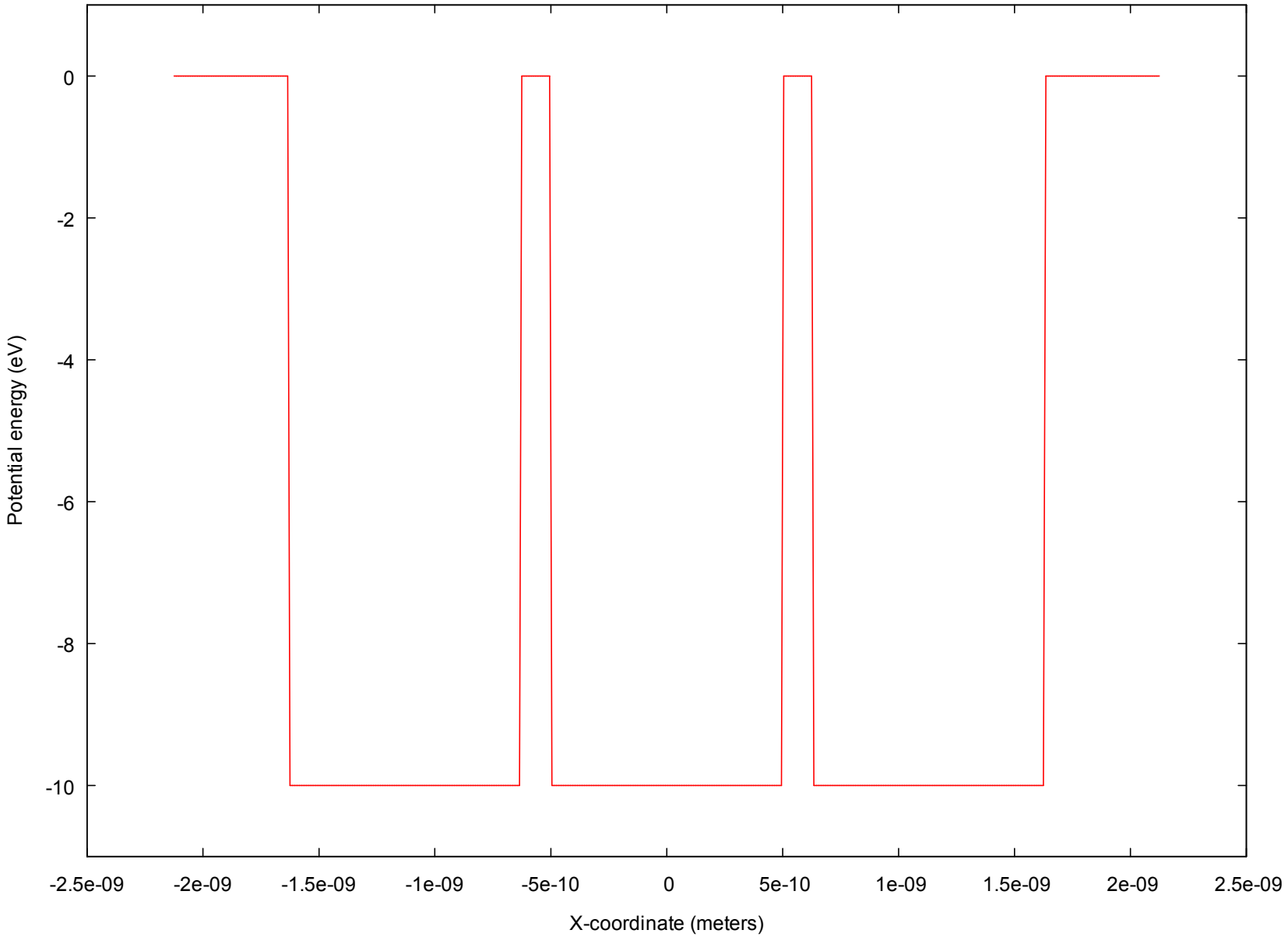
Eigenvalue Solution for Electron in a Quantum Well



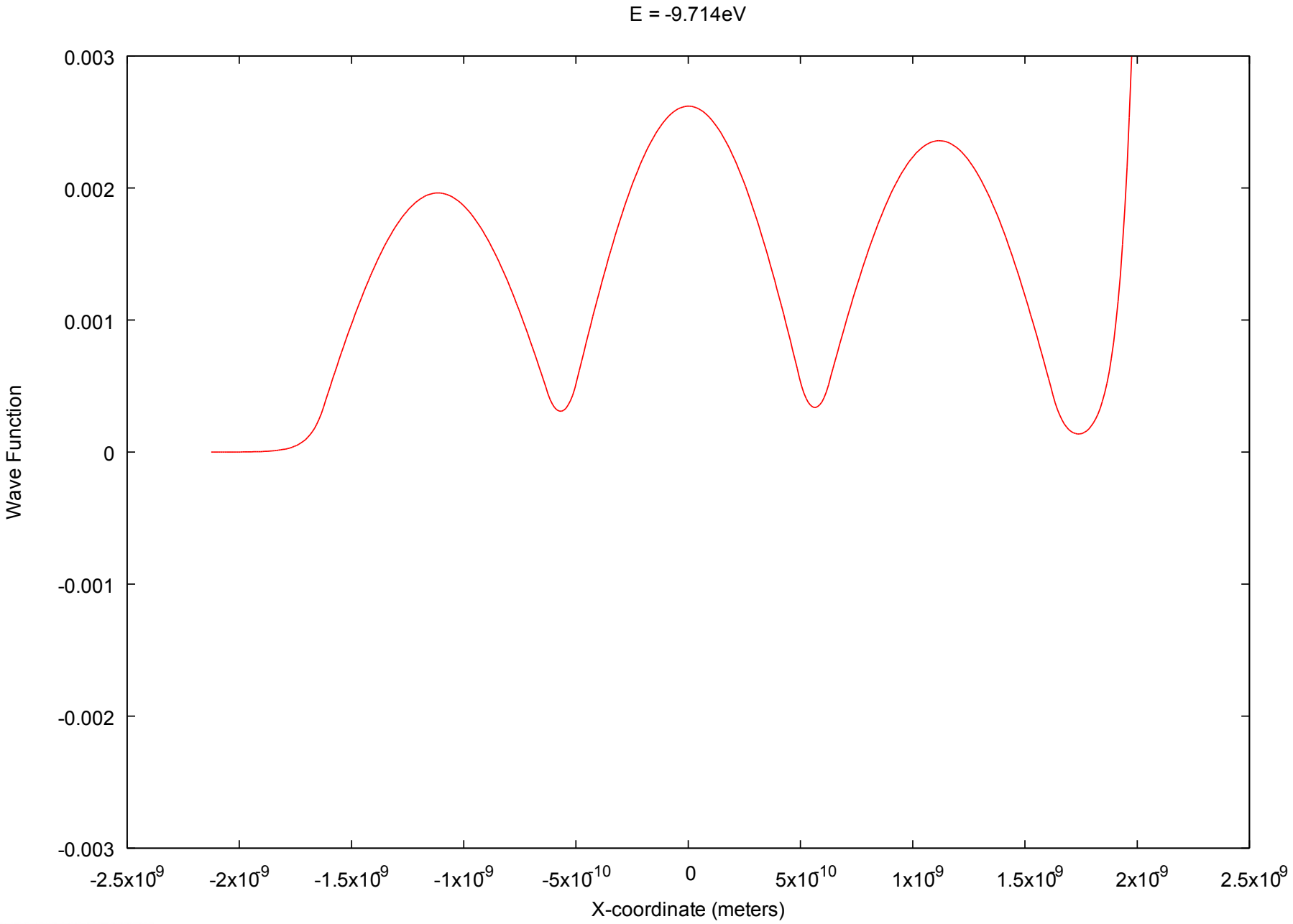
Eigenvalue Solution for Electron in a Quantum Well



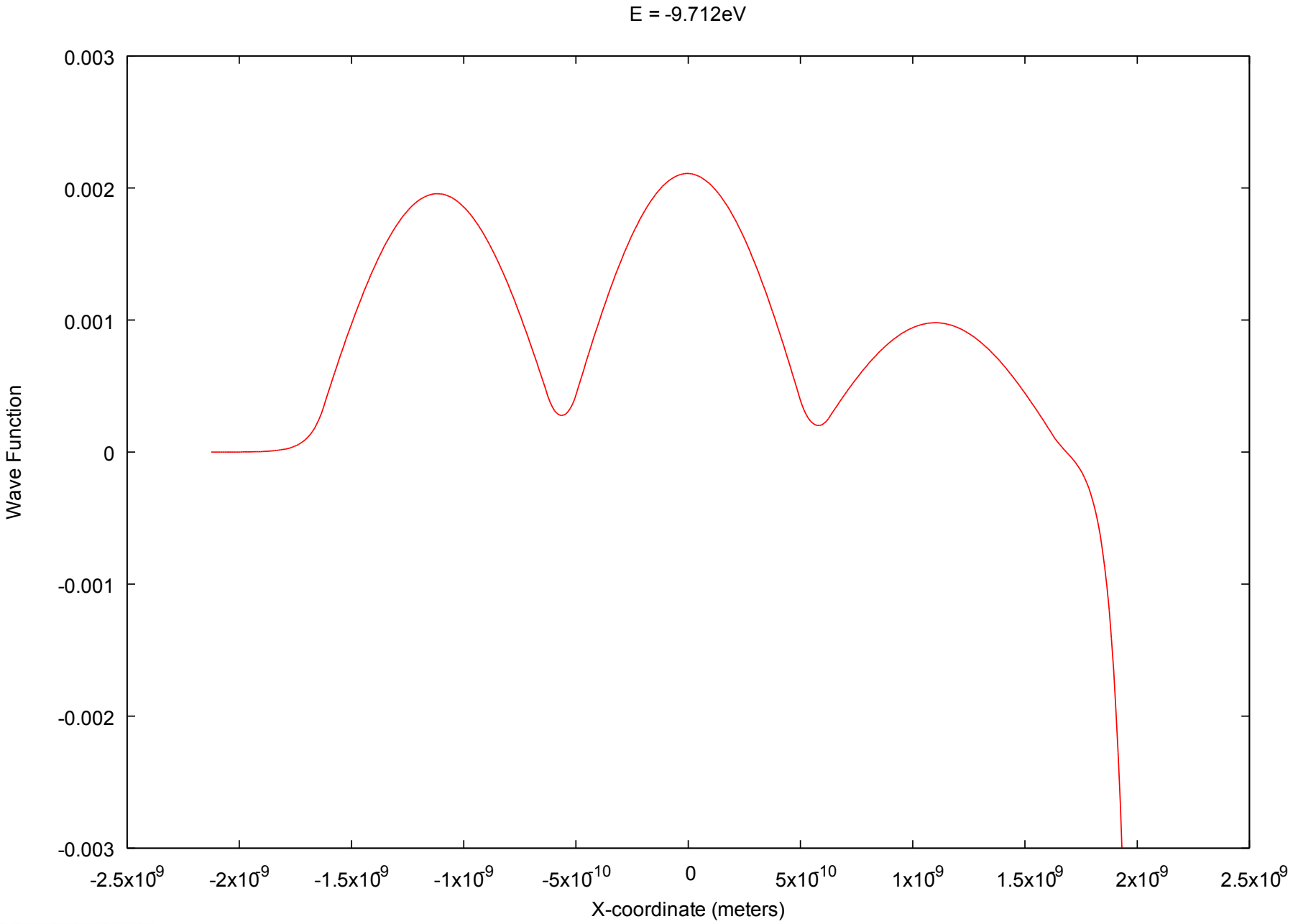
Example Potential with Three Rectangular Wells



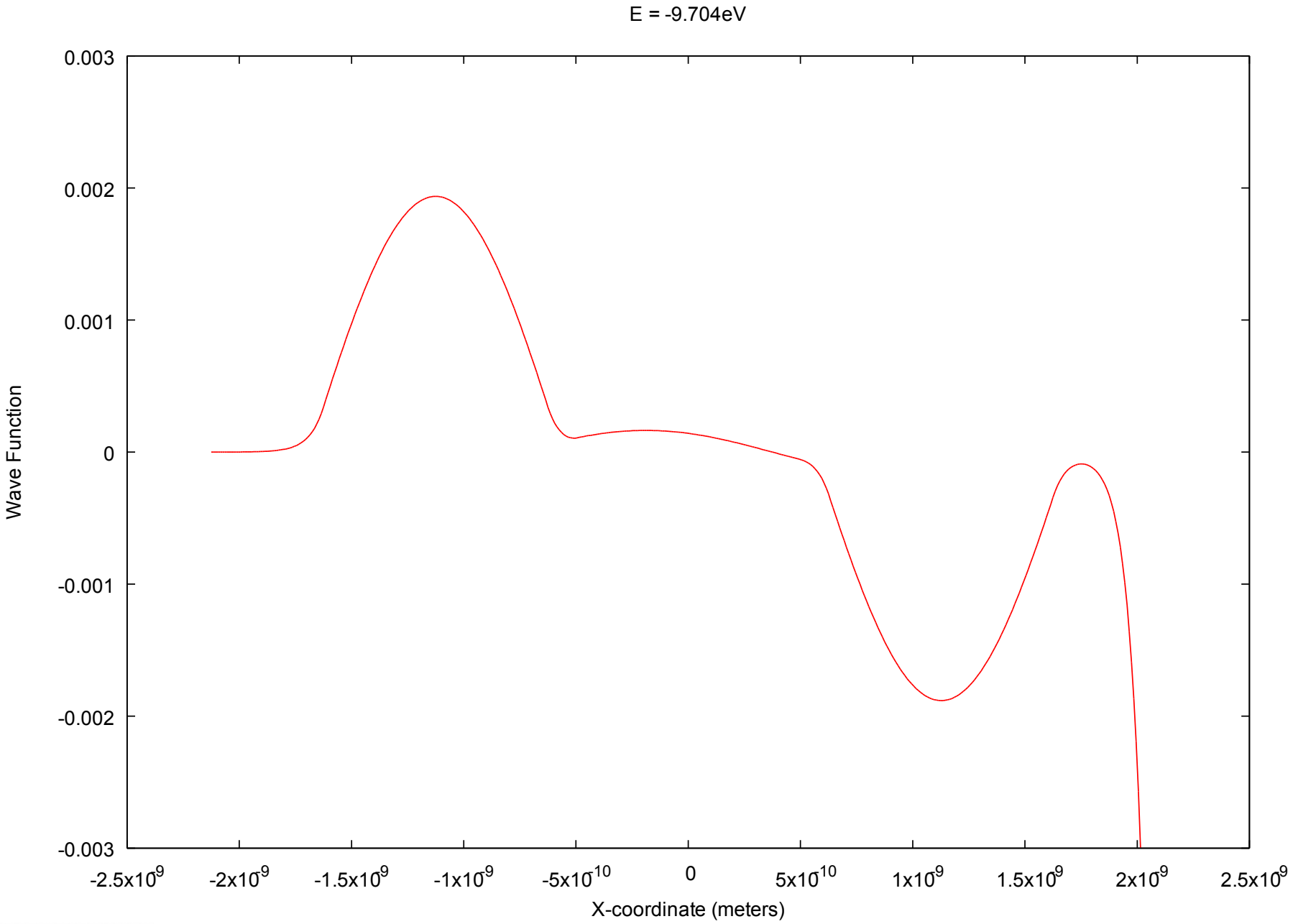
Homing In On Eigenvalues Manually



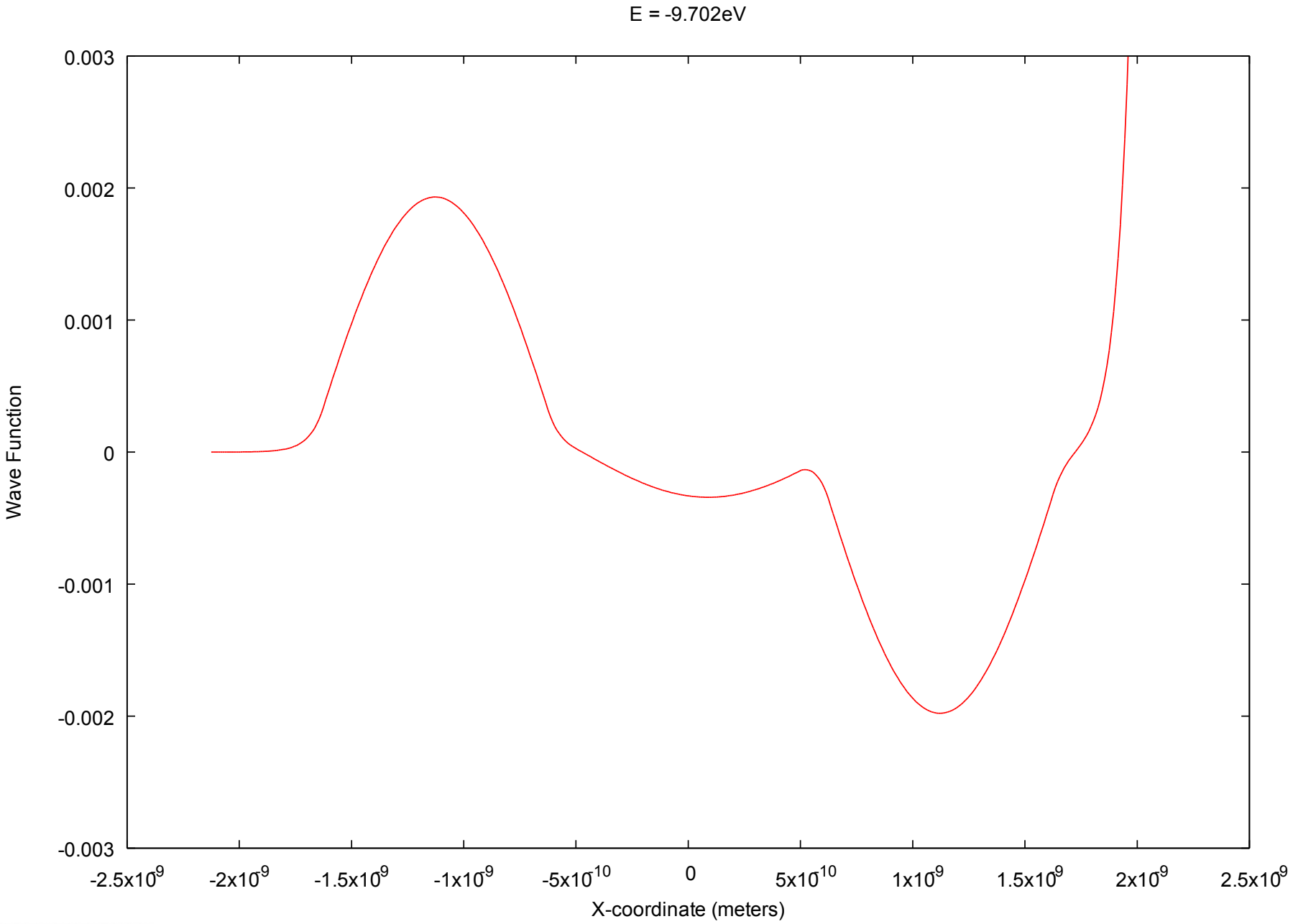
Homing In On Eigenvalues Manually



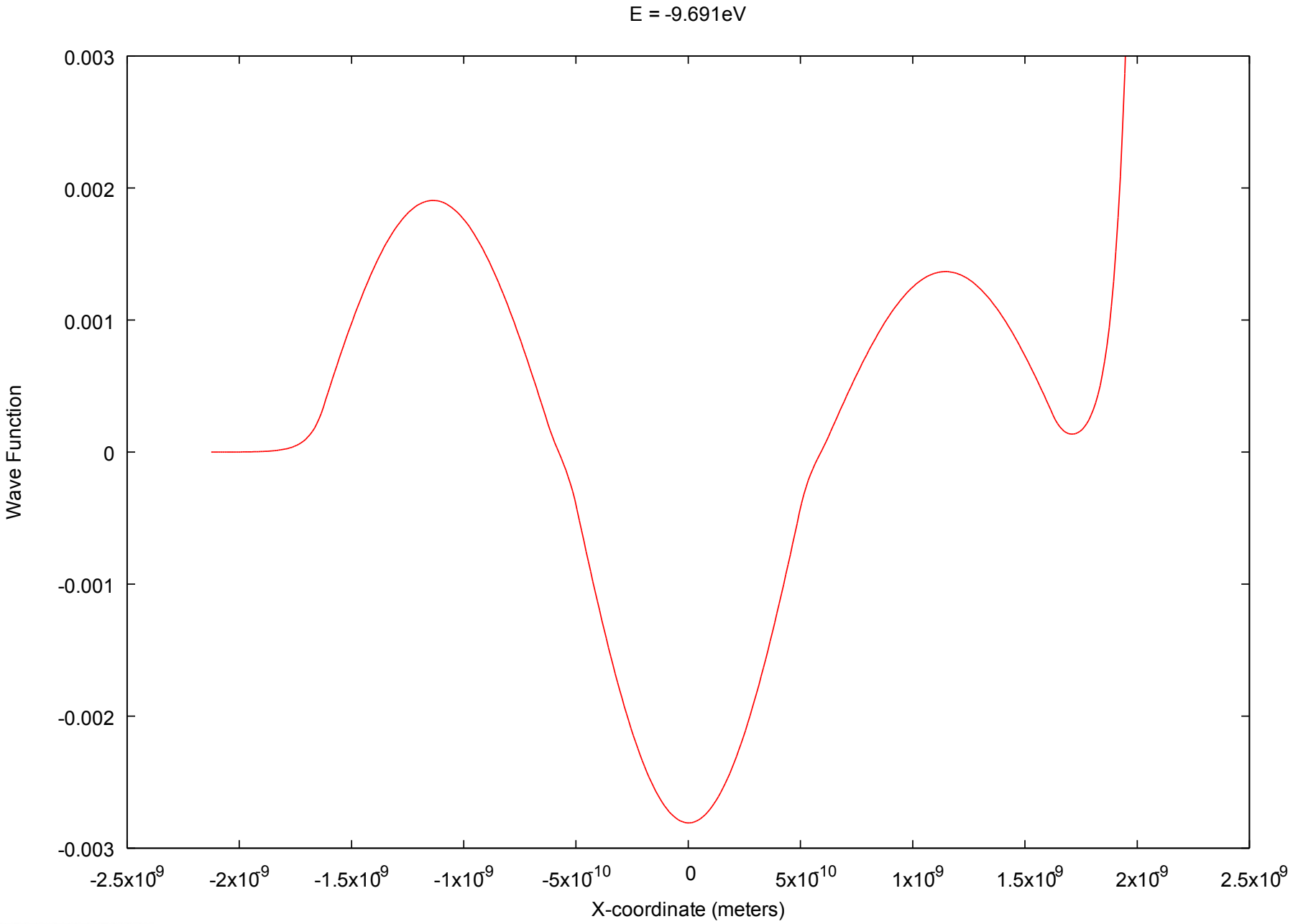
Homing In On Eigenvalues Manually



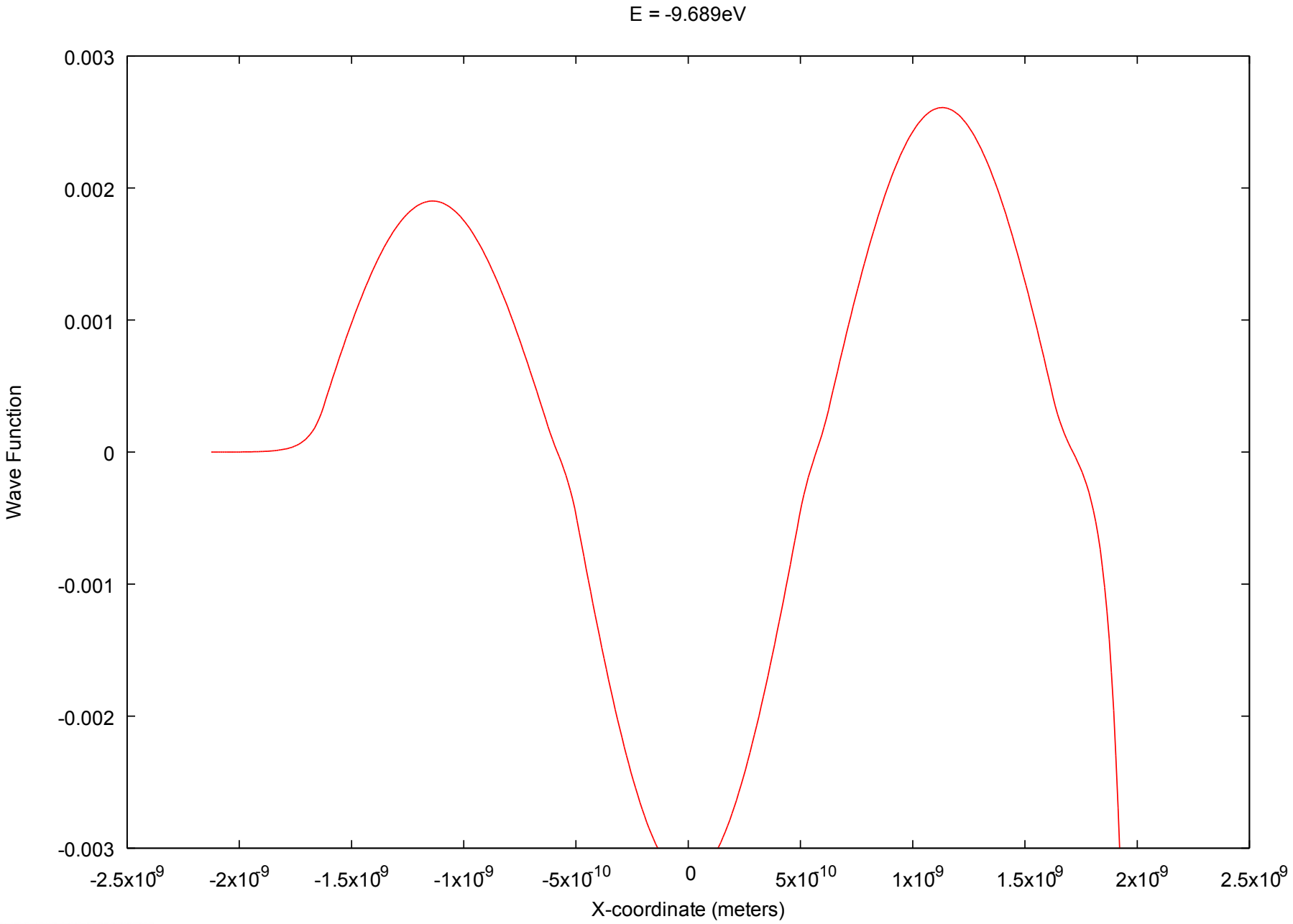
Homing In On Eigenvalues Manually



Homing In On Eigenvalues Manually

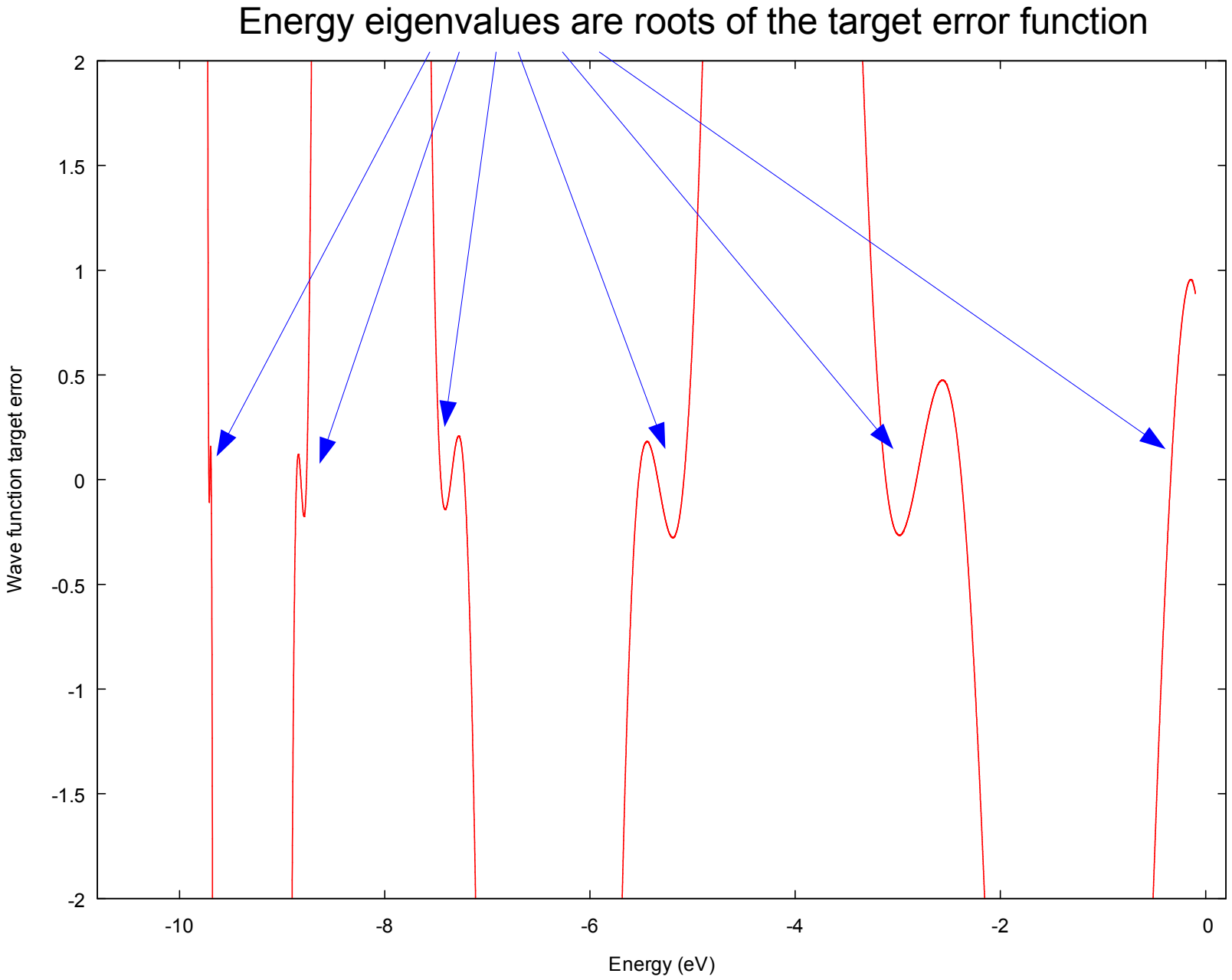


Homing In On Eigenvalues Manually

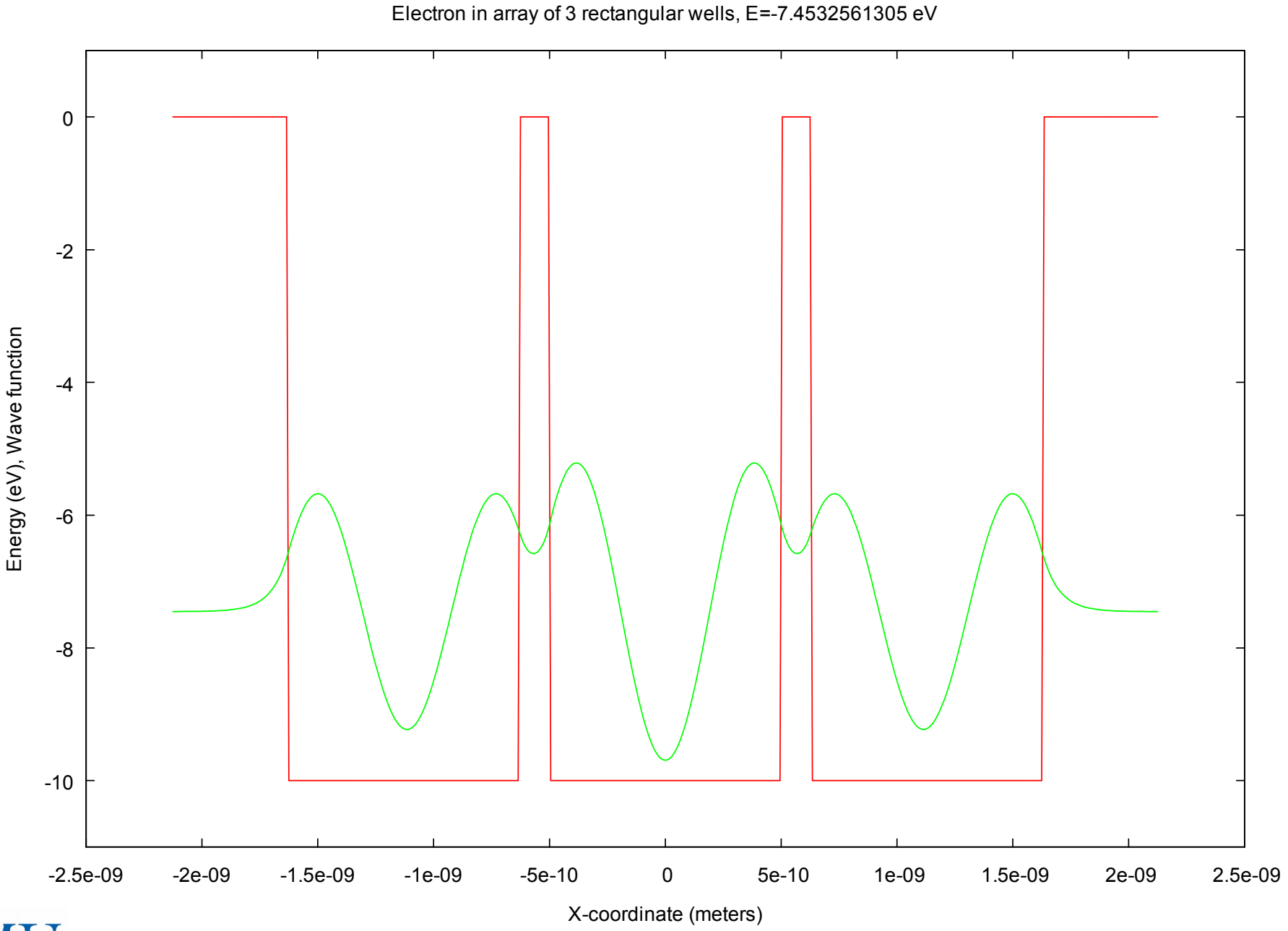


Shooting Converging on Energy Eigenvalues

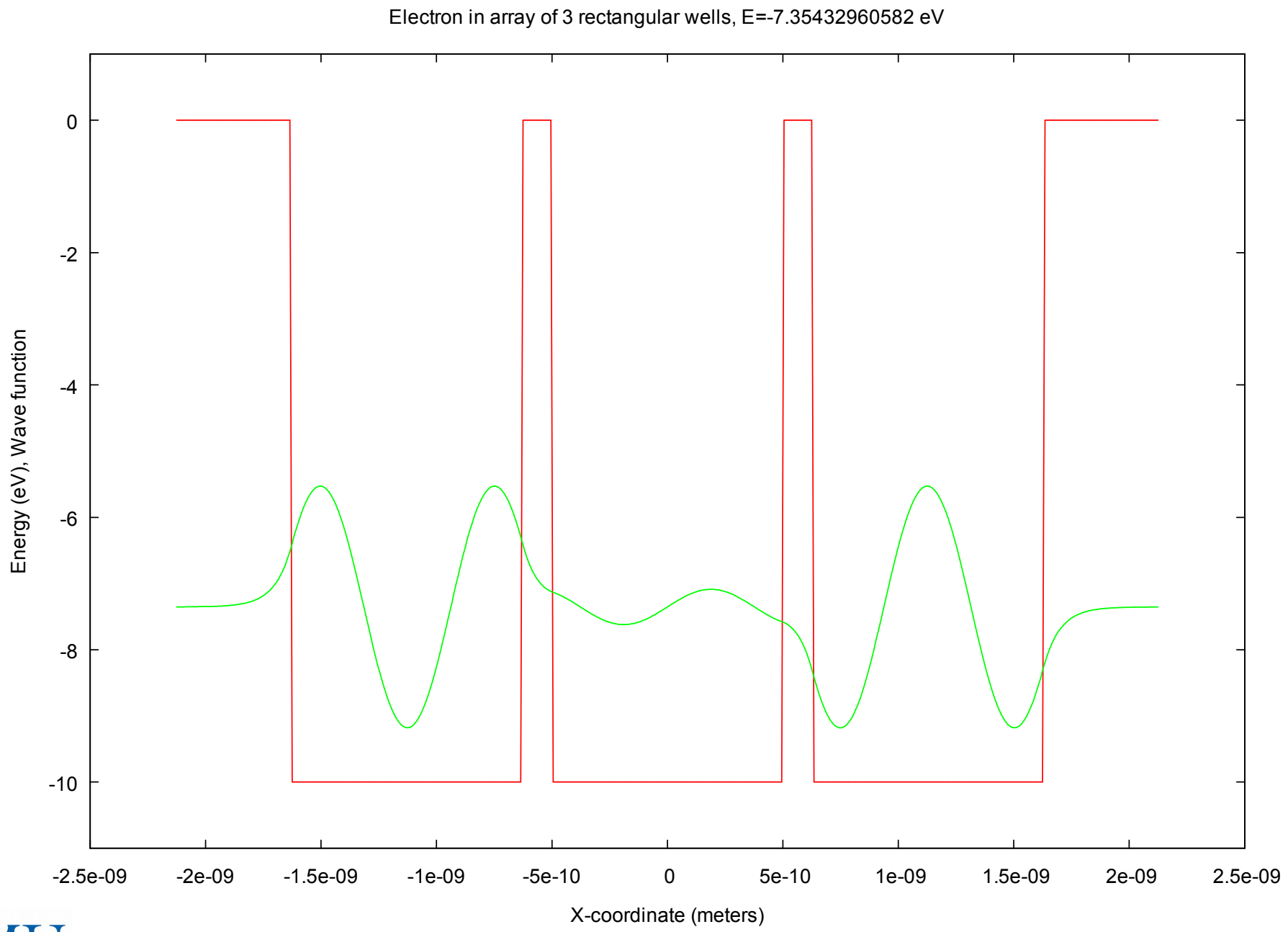
```
-9.71346399404 eV
-9.70340417958 eV
-9.69009897282 eV
-8.85911690107 eV
-8.81741476862 eV
-8.7622689821  eV
-7.45325624473 eV
-7.35432984079 eV
-7.22272013943 eV
-5.524595822   eV
-5.33839525754 eV
-5.08463221192 eV
-3.11786052982 eV
-2.81774564288 eV
-2.37927492949 eV
-0.34733856826 eV
```



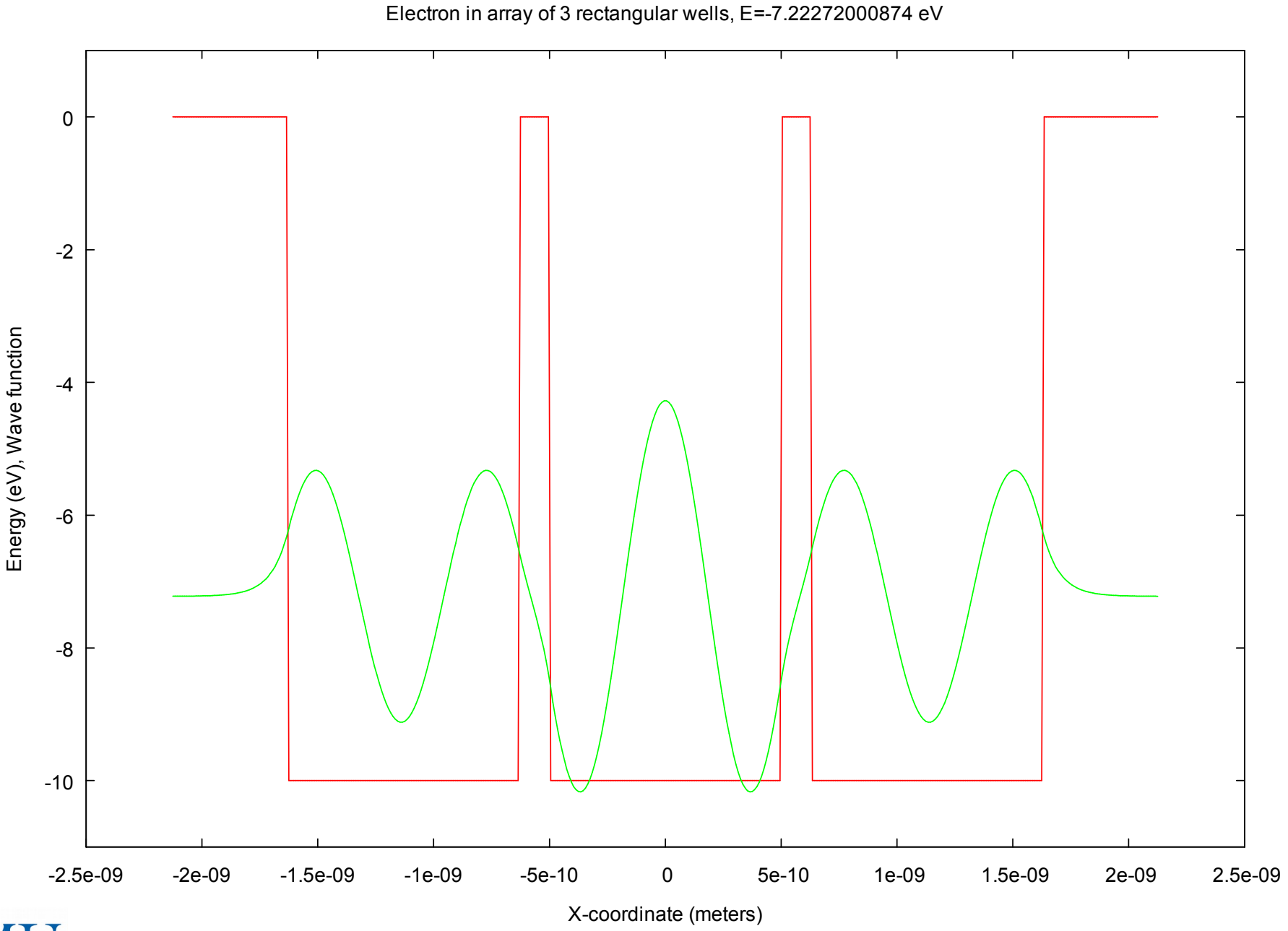
Eigenvalues for Electron in a Quantum Well Array



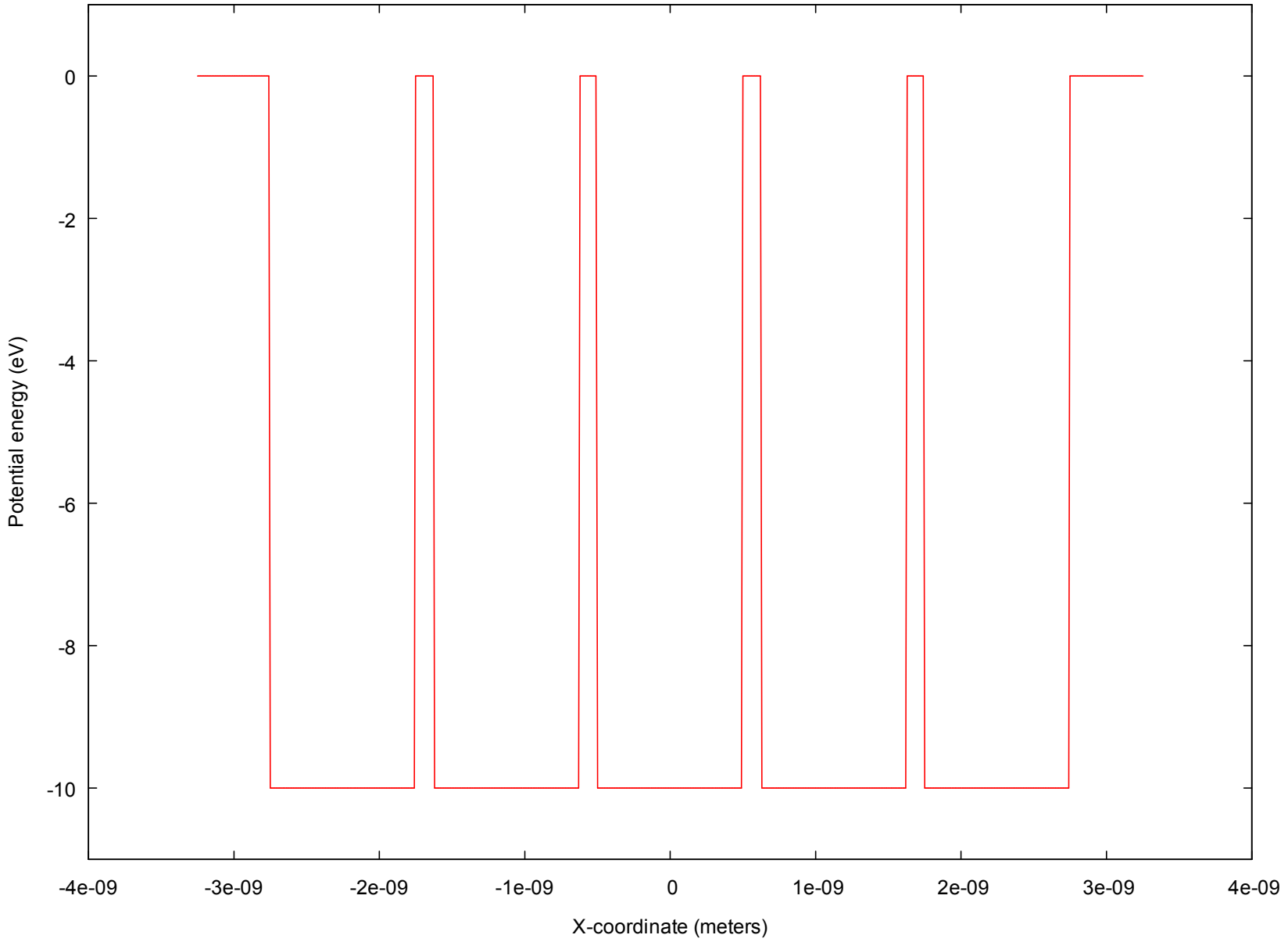
Eigenvalues for Electron in a Quantum Well Array



Eigenvalues for Electron in a Quantum Well Array

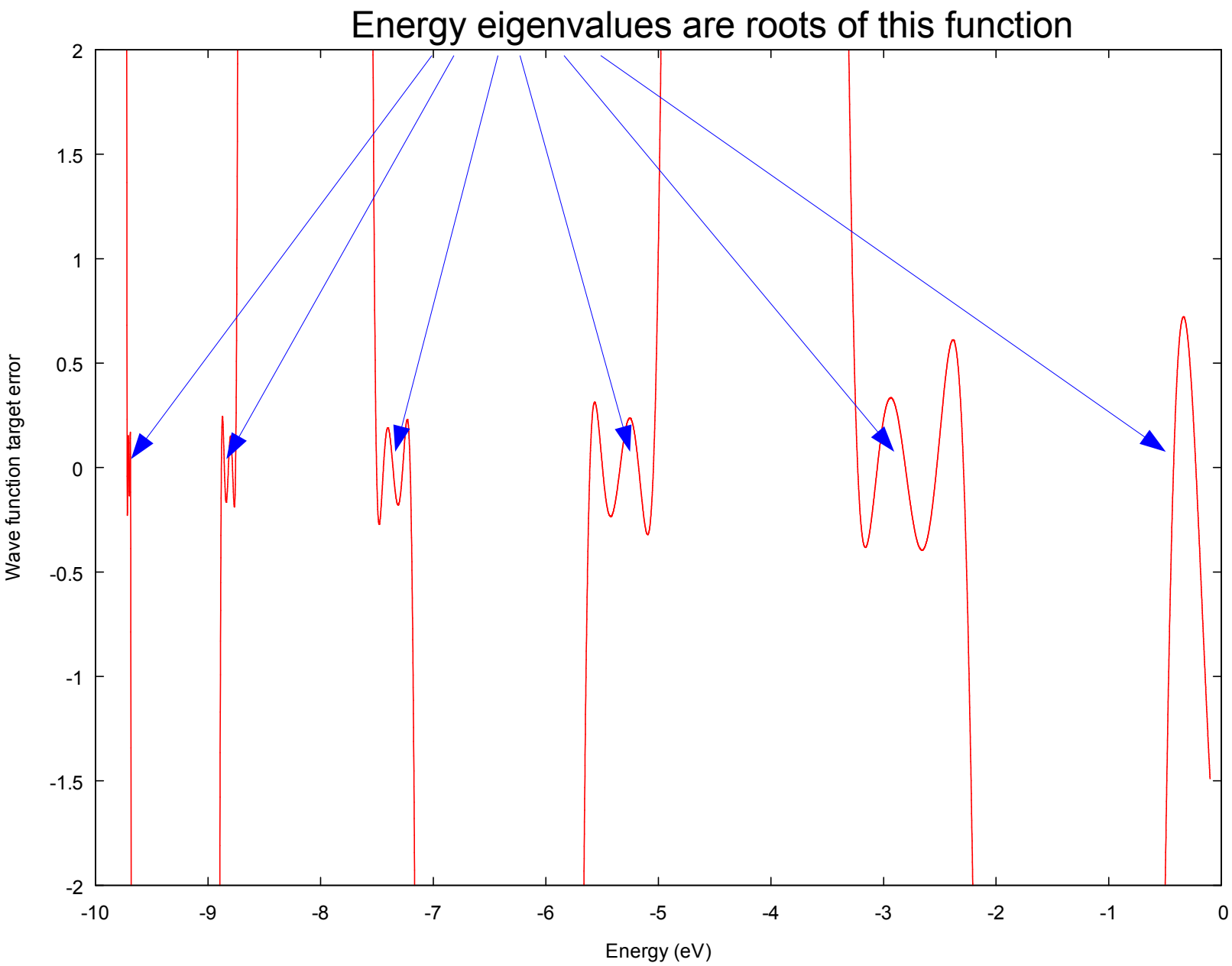


Example Potential with Five Rectangular Wells

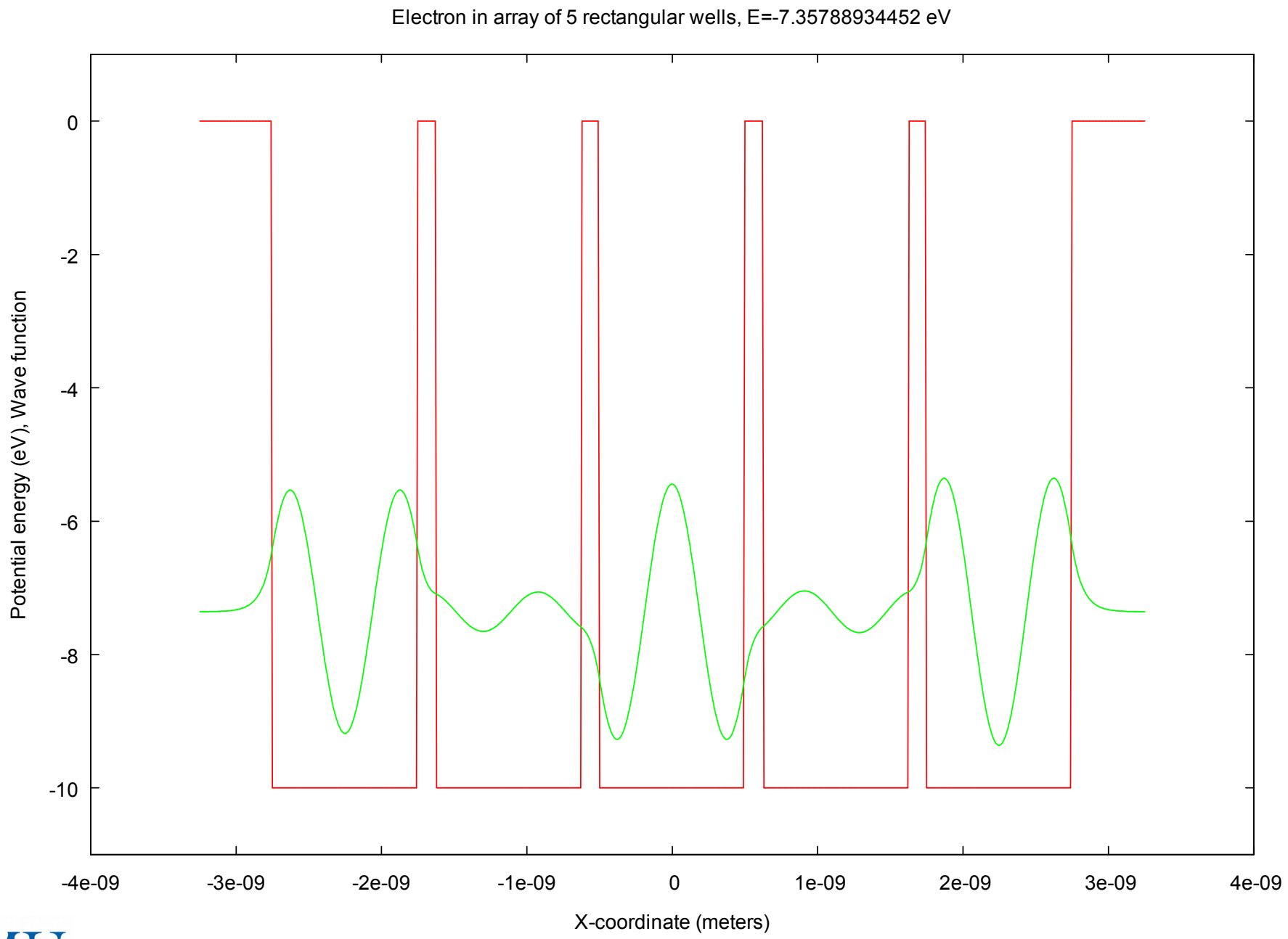


Shooting Converging on Energy Eigenvalues

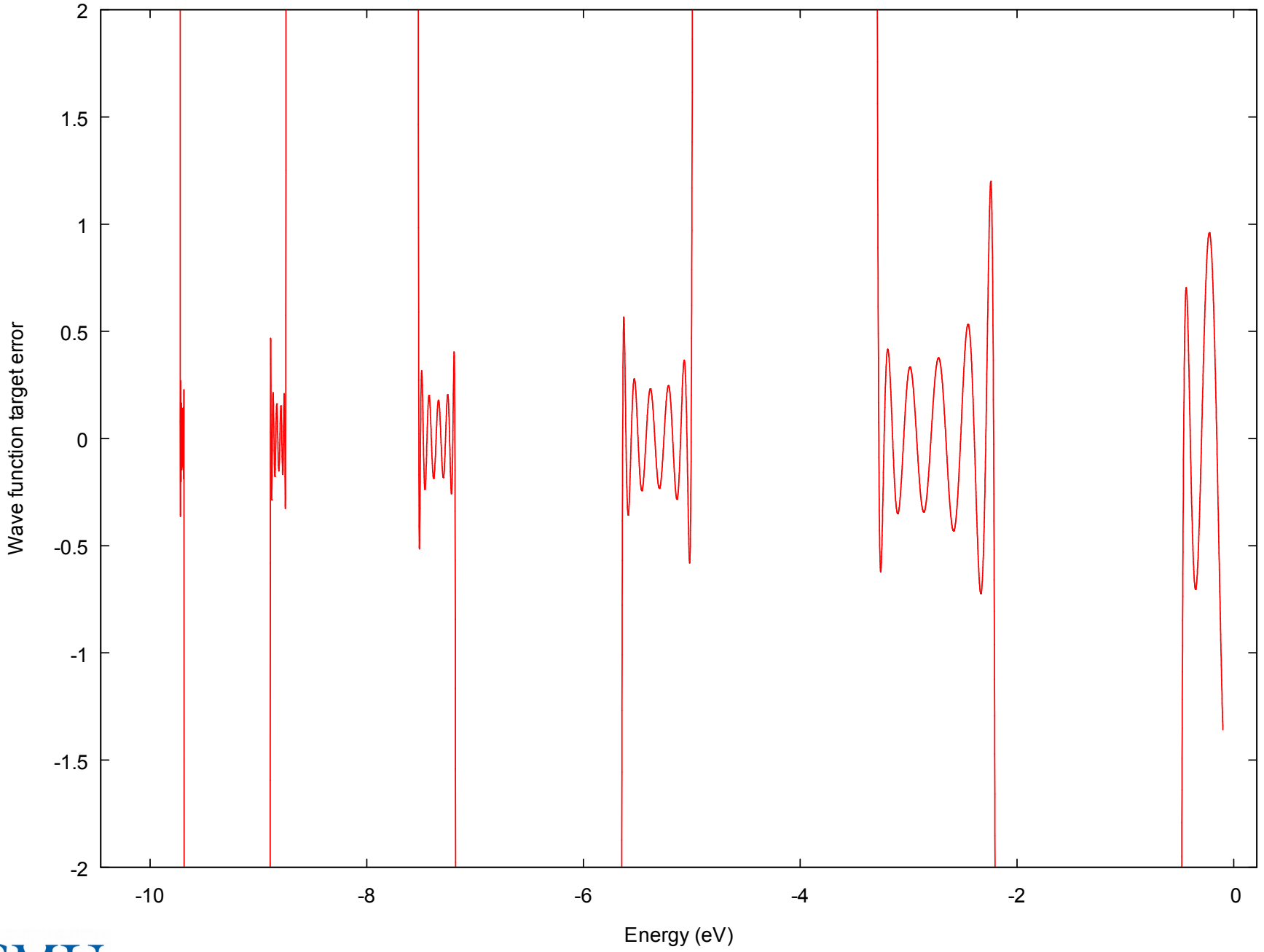
```
-9.71919638946 eV
-9.71279126092 eV
-9.70394993593 eV
-9.69509469861 eV
-9.6886352178  eV
-8.88186622227 eV
-8.85567286656 eV
-8.81942691172 eV
-8.78278388266 eV
-8.75572296675 eV
-7.5032508231  eV
-7.44243611652 eV
-7.35788949906 eV
-7.27091247592 eV
-7.20508392589 eV
-5.60823159971 eV
-5.49686872433 eV
-5.34065410468 eV
-5.17523889907 eV
-5.04437259971 eV
-3.22965701672 eV
-3.05577581474 eV
-2.80575648548 eV
-2.52801752779 eV
-2.28995145952 eV
-0.42608141845 eV
-0.23345845464 eV
```



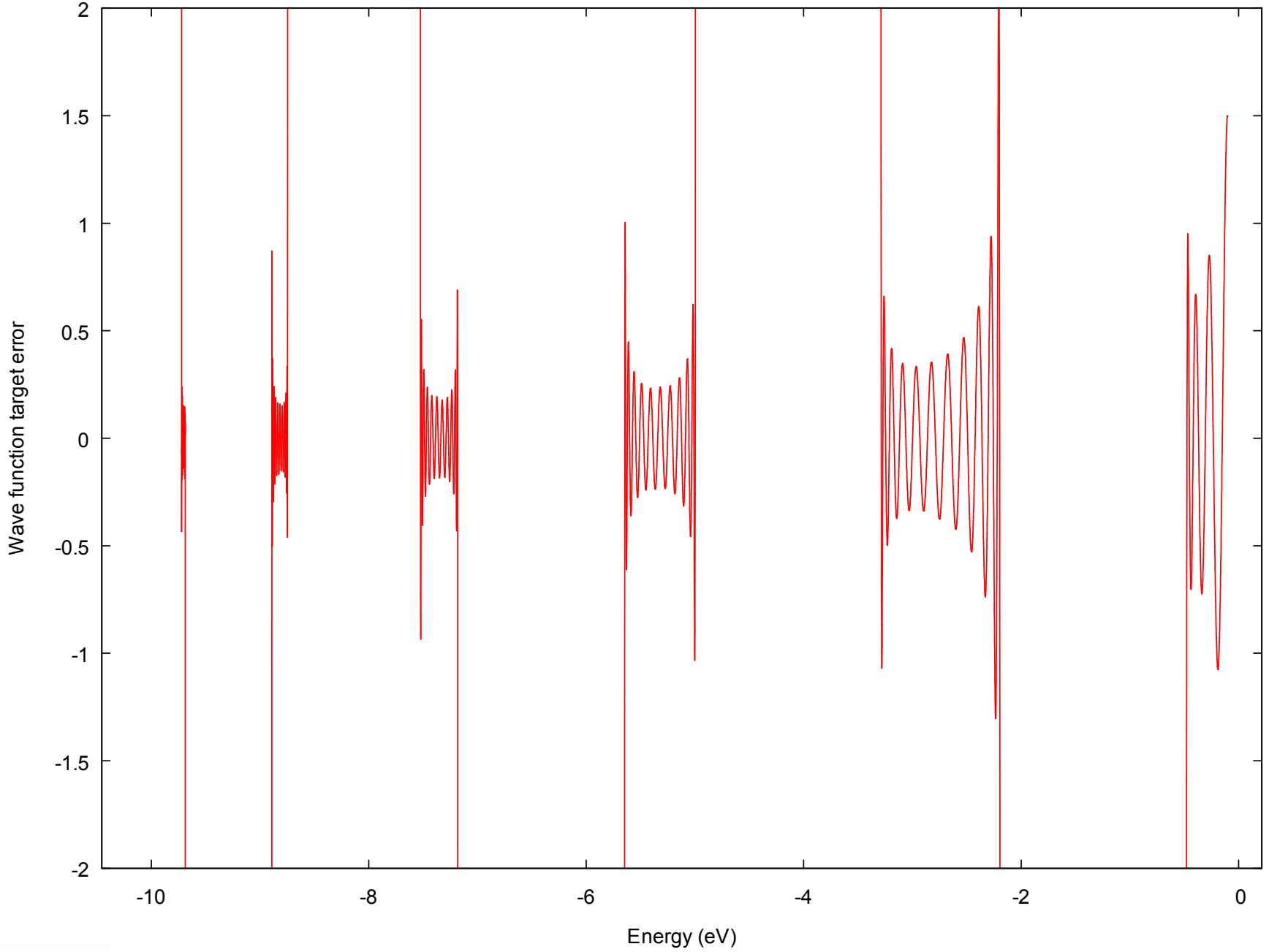
Eigenvalues for Electron in a Quantum Well Array



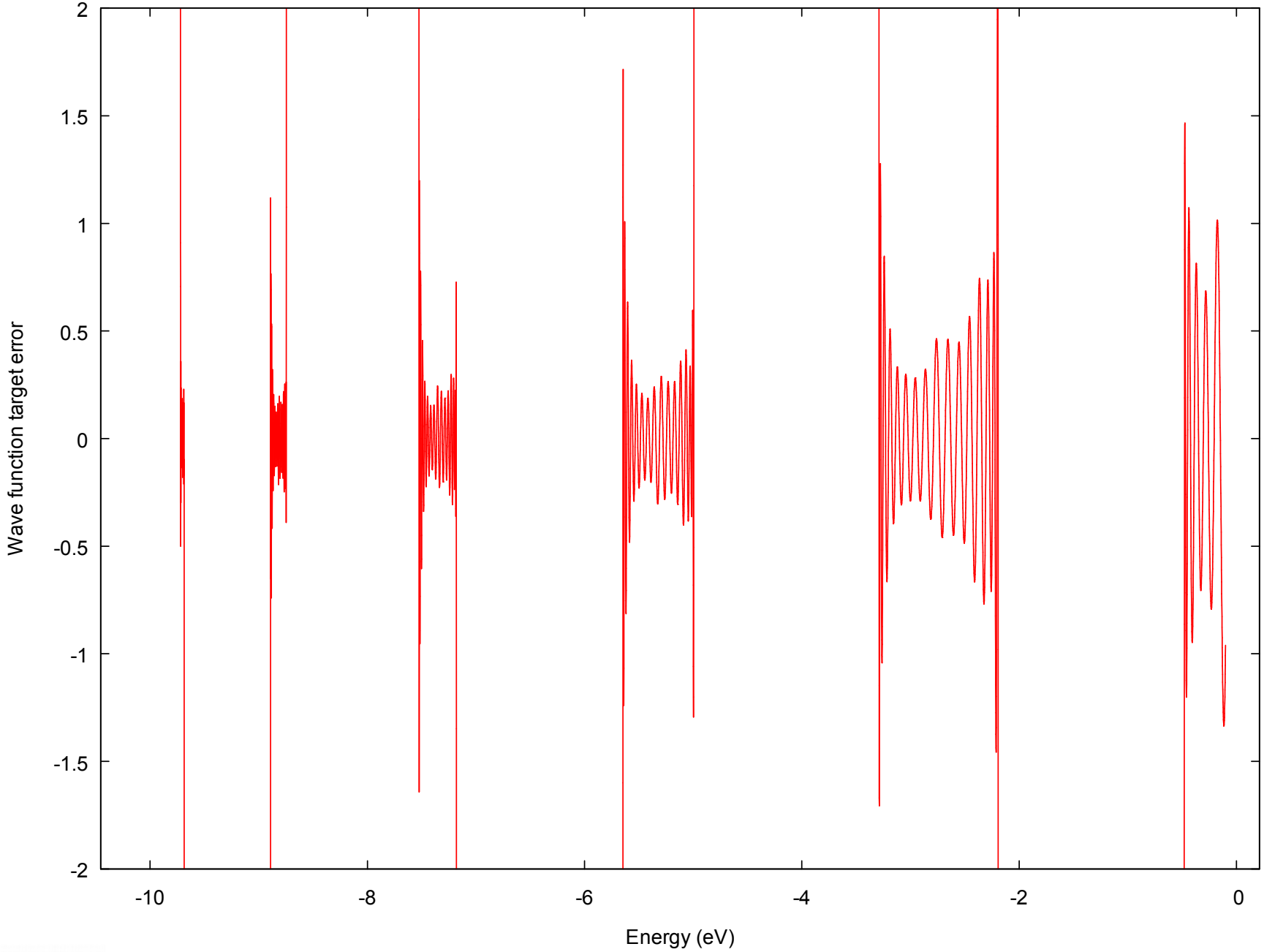
Wave Function Target Error with 11 Wells



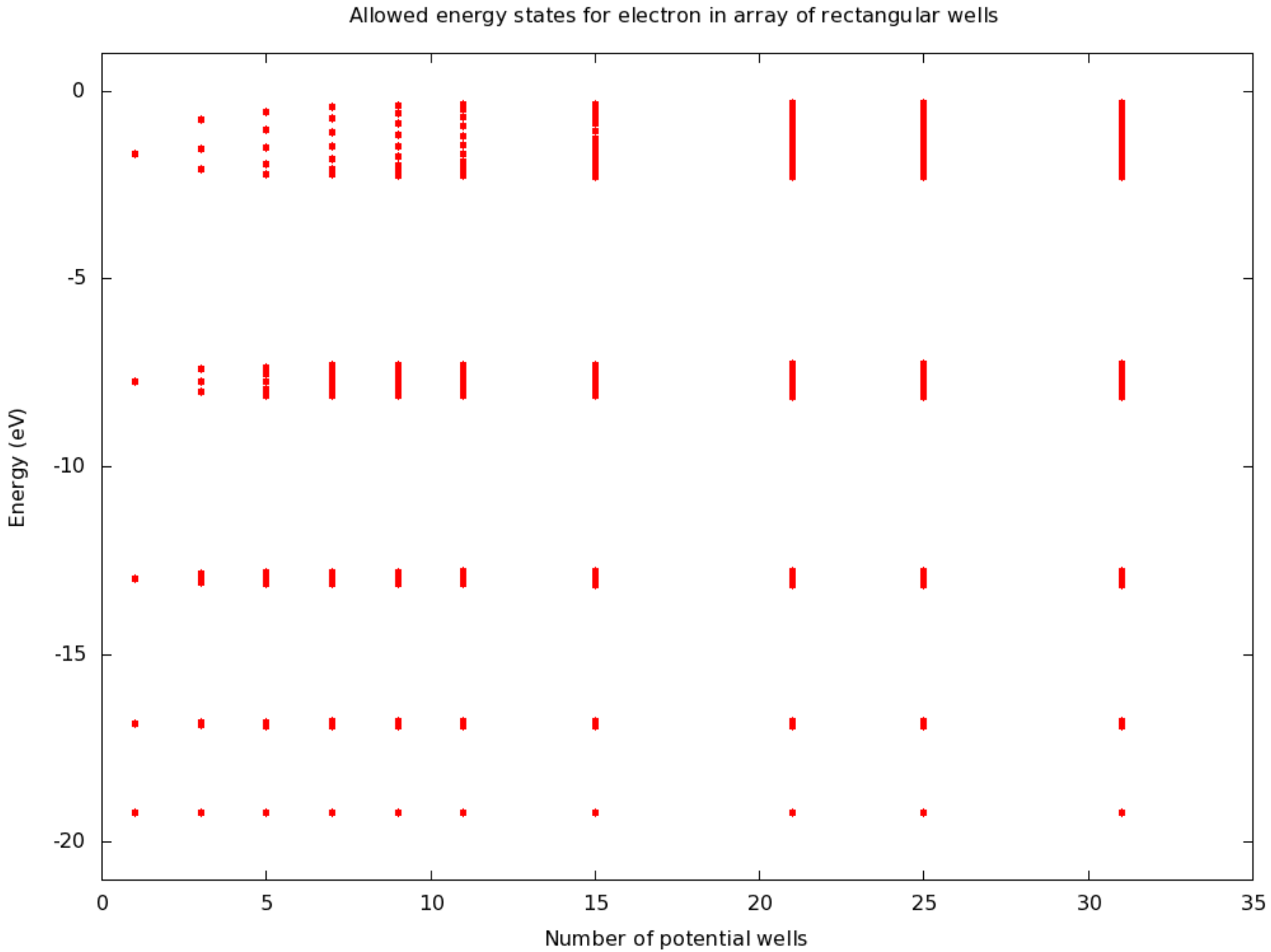
Wave Function Target Error with 21 Wells



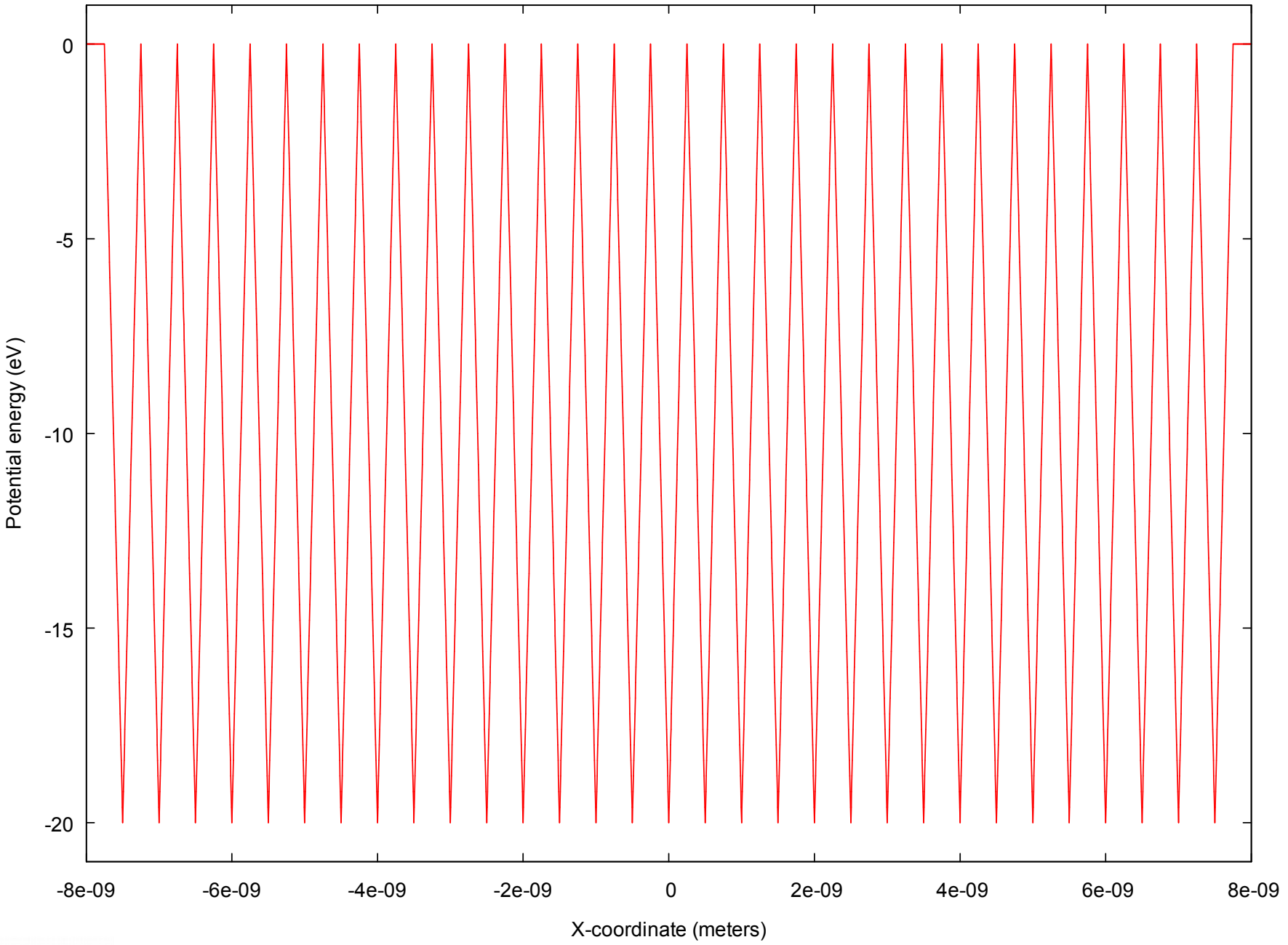
Wave Function Target Error with 31 Wells



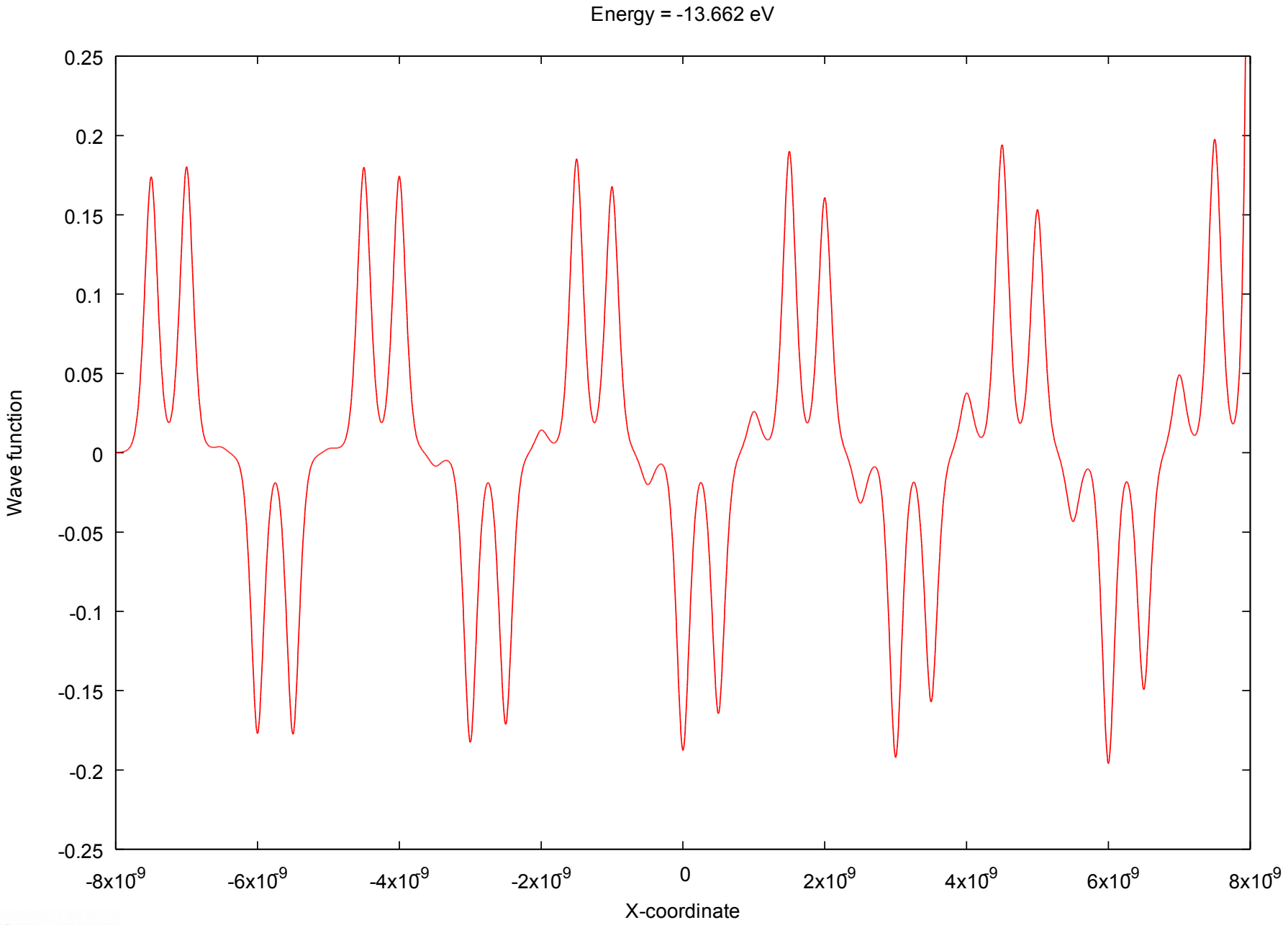
Allowed Energy States for Arrays of Wells



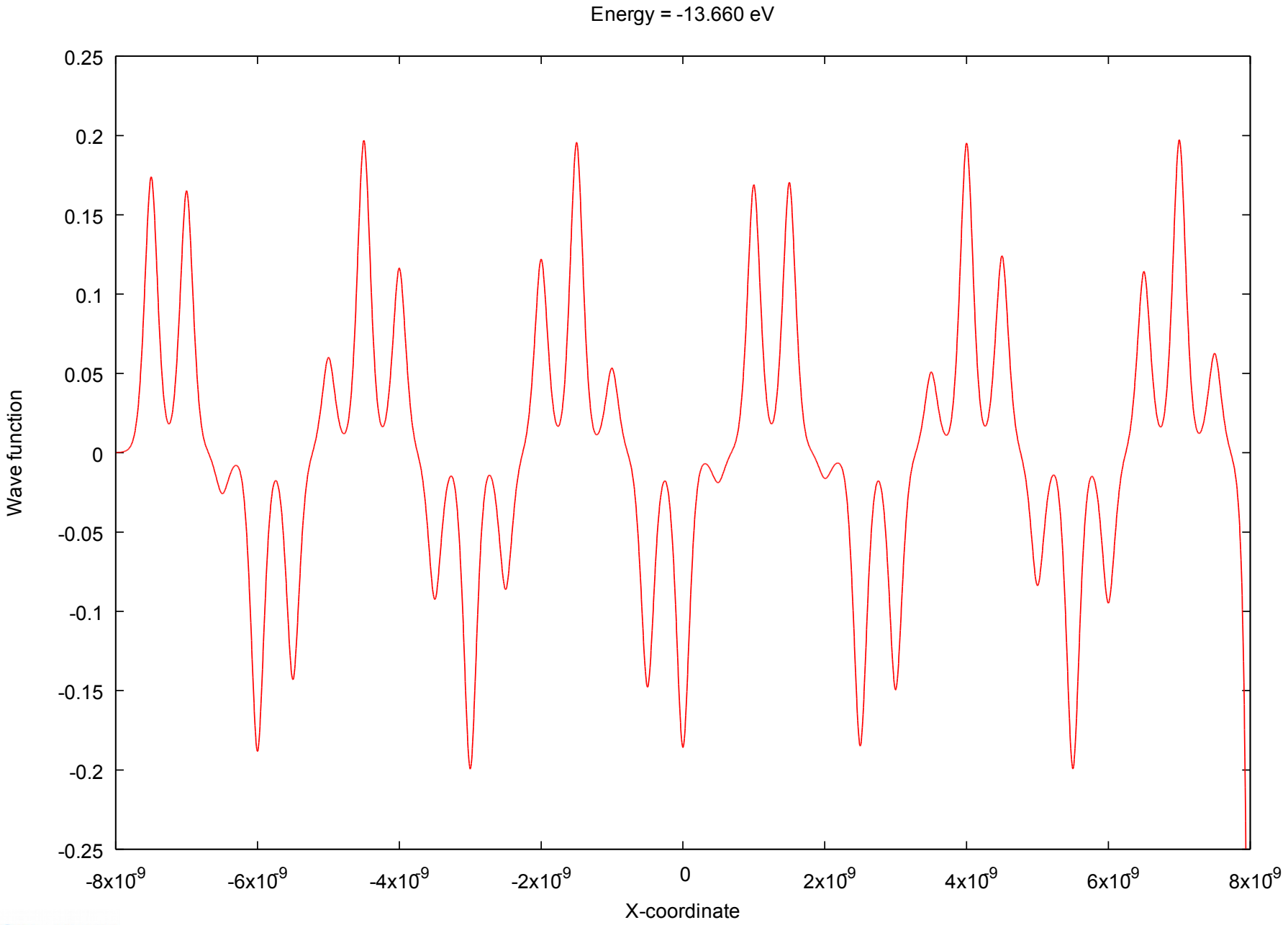
Example Potential with 31 Triangular Wells



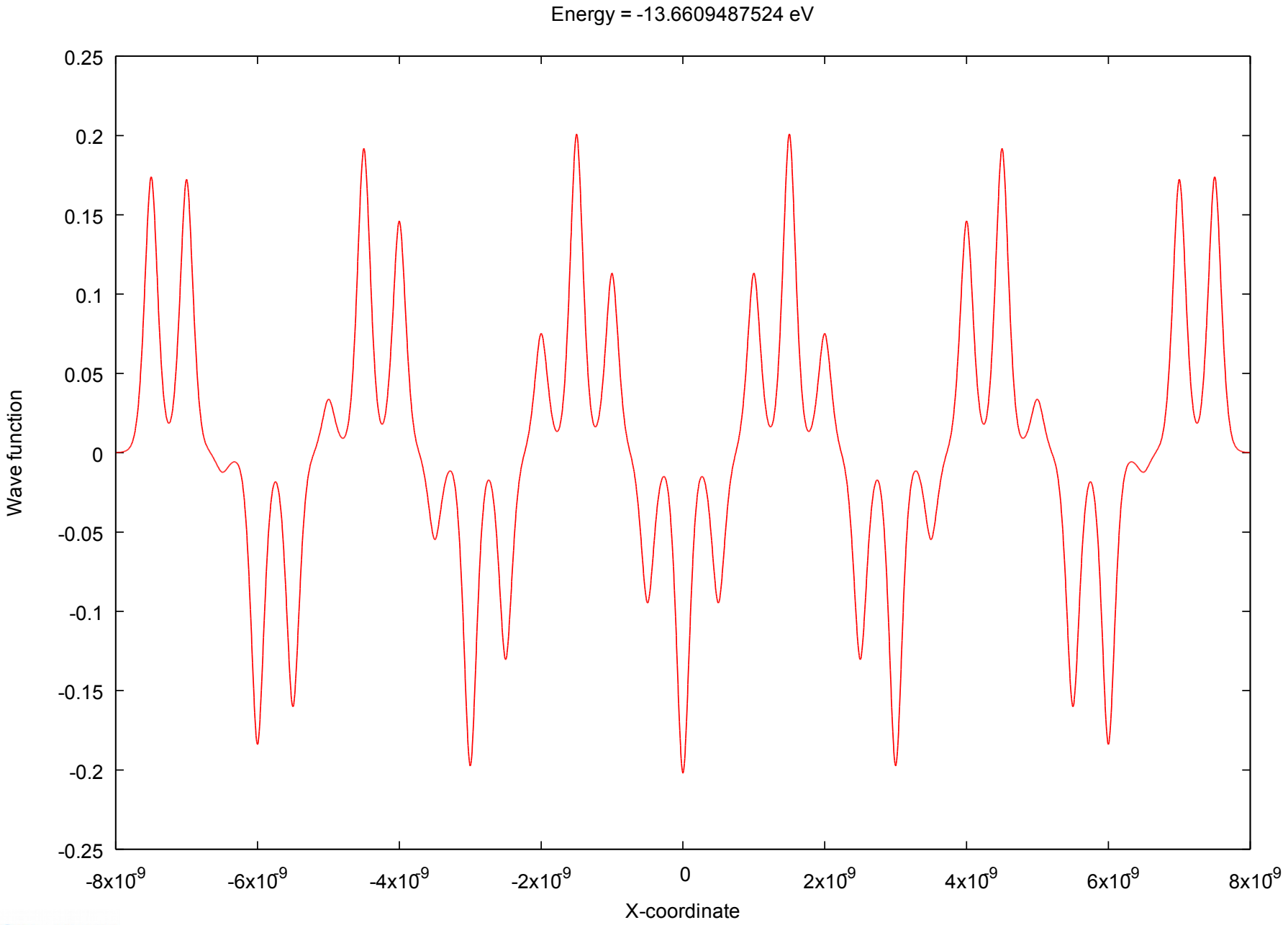
Wave Function Below Allowed Energy



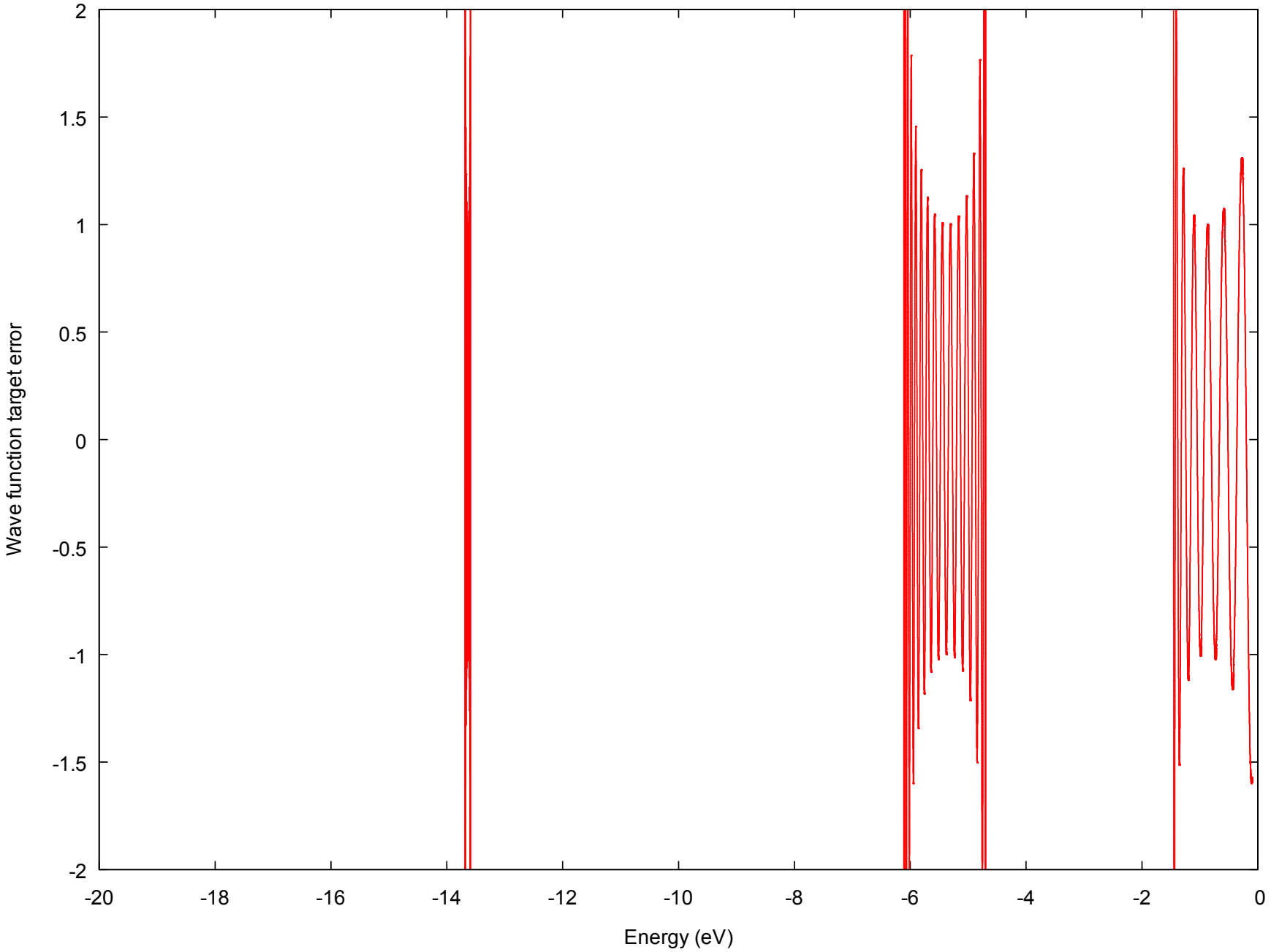
Wave Function Above Allowed Energy



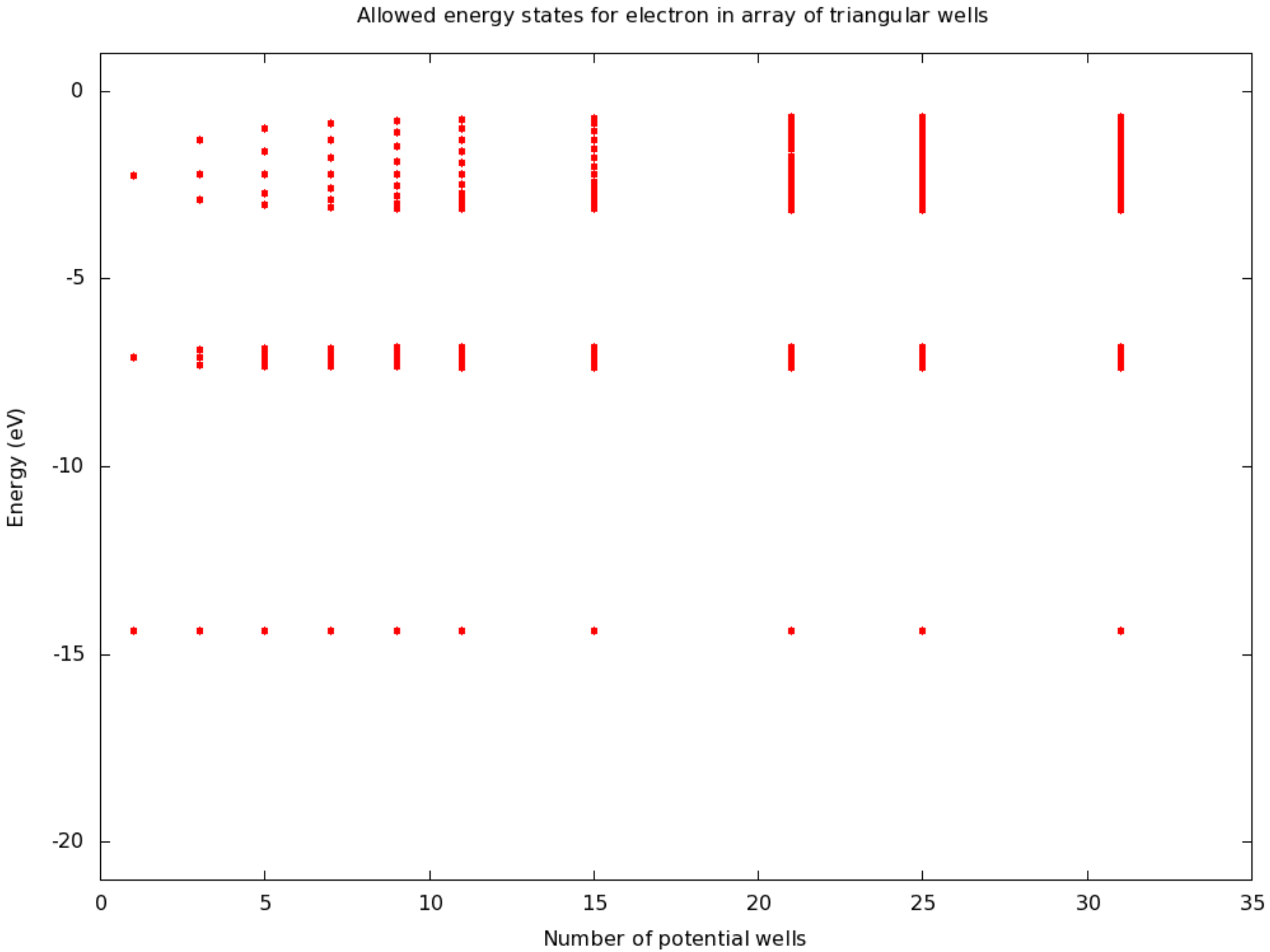
Wave Function at Example Allowed Energy



Wave Function Target Error with 31 Triangular Wells



Allowed Energy States for Arrays of Wells



Final Project: Part One

Use sweep.c module to produce plot of the potential wells for your crystal substance

```
double potential(double x) {
  ...
/* Calculate and return the potential energy as a function of x*/
  ...
}

int main(int argc,char **argv) {
  ...
  sweep(potential,...);
  ...
}
```

Final Project: Part Two

Use diffeq.c module to produce plots of the wave function for a given trial energy for your crystal substance, for two values of energy that bracket an allowed energy

```
double potential(double x) {
/* Calculate and return the potential energy as a function of x*/
  ...
}

void schroedinger(int n,double x,double y[2],double yp[2]) {
/* Calculate first order derivatives for R-K4 algorithm*/
  yp[0] = ...;
  yp[1] = ... potential(x)...;
}

int main(int argc,char **argv) {
  ...
  energy = atof(argv[]);
  ...
  rk4(2,schroedinger,...);
  ...
}
```

Final Project: Part Three

Use sweep.c module to produce plot of the wave function target error versus trial energy for your crystal substance

```
double potential(double x) {
/* Calculate and return the potential energy as a function of x*/
  ...
}

void schroedinger(int n,double x,double y[2],double yp[2]) {
/* Calculate first order derivatives for R-K4 algorithm*/
  yp[0] = ...;
  yp[1] = ... potential(x)...;
}

double wave_shoot(double energy) {
  ...
  rk4_quiet(2,schroedinger,...);
  ...
  return(...);
}

int main(int argc,char **argv) {
  ...
  sweep(wave_shoot,...);
  ...
}
```

Final Project: Part Four

Use sweep_roots.c module to find energy eigenvalues for your crystal substance

```
double potential(double x) {
/* Calculate and return the potential energy as a function of x*/
  ...
}

void schroedinger(int n,double x,double y[2],double yp[2]) {
/* Calculate first order derivatives for R-K4 algorithm*/
  yp[0] = ...;
  yp[1] = ... potential(x)...;
}

double wave_shoot(double energy) {
  ...
  rk4_quiet(2,schroedinger,...);
  ...
  return(...);
}

int main(int argc,char **argv) {
  ...
  sweep_regula_falsi(wave_shoot,...);
  ...
}
```