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

General Problem of Systems of Nonlinear Equation Roots

Find $x_1, x_2, x_3, \ldots, x_n$ for which

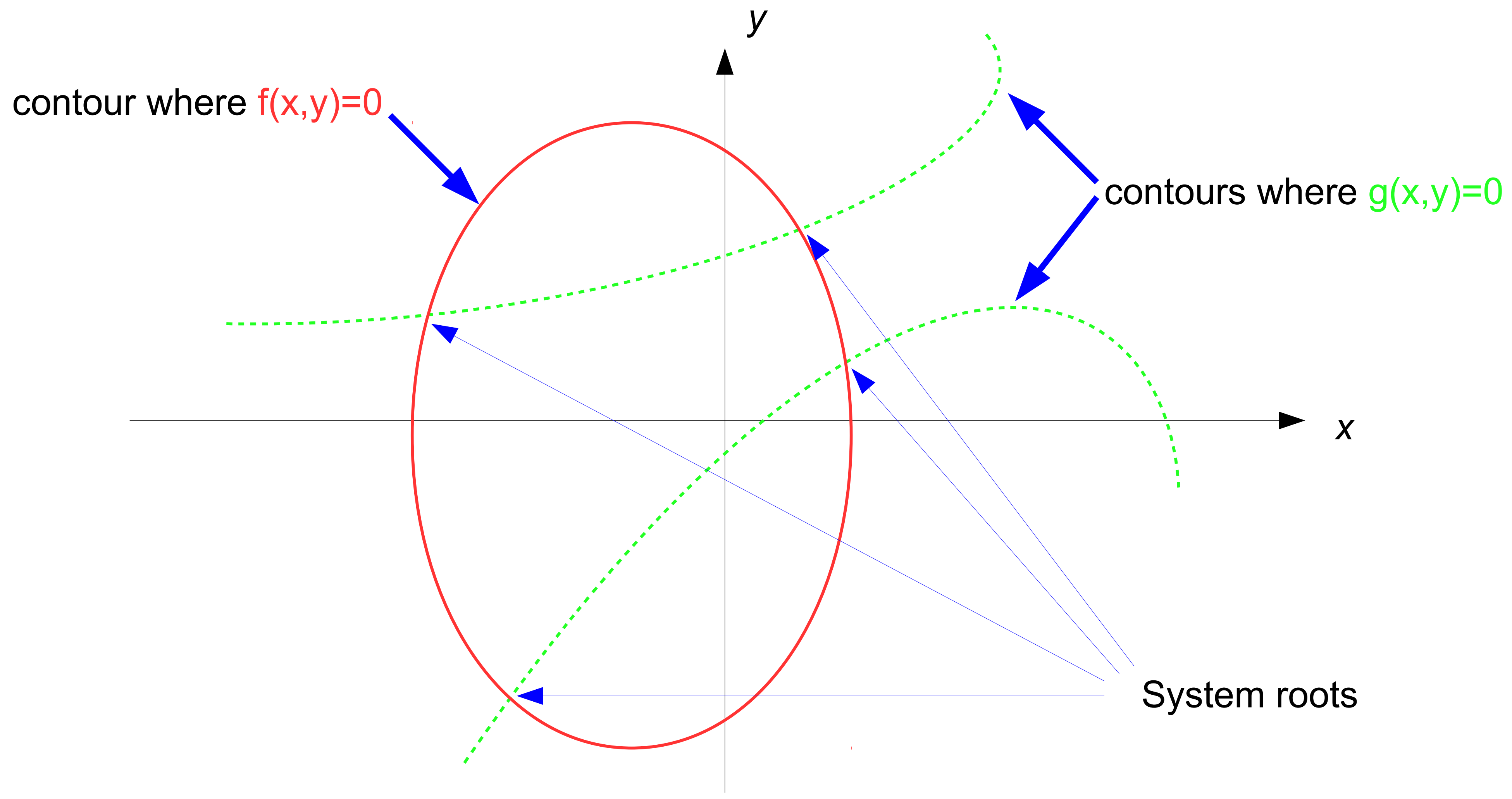
$$\begin{gathered} f_1(x_1, x_2, x_3, \ldots, x_n) = 0 \text{ and} \\ f_2(x_1, x_2, x_3, \ldots, x_n) = 0 \text{ and} \\ \vdots \\ f_n(x_1, x_2, x_3, \ldots, x_n) = 0 \end{gathered}$$

Example:

$$\begin{gathered} x^4 + 14y^3 - 23x^2 + 7z - 3 = 0 \\ \frac{e^{k_1 x} + e^{-k_2 y}}{8} = 0 \\ \sin(3x) - \cos(5y) - \sqrt{7z} + 13 = 0 \end{gathered}$$

Graphical Two-Dimensional Example

System: $f(x, y) = 0$ and $g(x, y) = 0$



Newton-Raphson Algorithm in One Dimension

given $f(x)$ and x_0, form Taylor approximation

$$f(x_{i+1}) = f(x_i) + f'(x_i) \cdot (x_{i+1} - x_i)$$

set $f(x_{i+1}) = 0$ and solve for x_{i+1}

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Extend Newton-Raphson Algorithm to Multiple Dimensions

First define the column vector $\vec{x}$ as $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

Then define the column vector $\vec{f}\left(\vec{x}\right)$ as $\begin{bmatrix} f_1\left(\vec{x}\right) \\ f_2\left(\vec{x}\right) \\ \vdots \\ f_n\left(\vec{x}\right) \end{bmatrix}$

Extend Newton-Raphson Algorithm to Multiple Dimensions

Now define the Jacobian matrix $\boldsymbol{J}\left(\vec{x}\right)$ as
$$\begin{bmatrix} \dfrac{\partial f_1(\vec{x})}{\partial x_1} & \dfrac{\partial f_1(\vec{x})}{\partial x_2} & \cdots & \dfrac{\partial f_1(\vec{x})}{\partial x_n} \\ \dfrac{\partial f_2(\vec{x})}{\partial x_1} & \dfrac{\partial f_2(\vec{x})}{\partial x_2} & \cdots & \dfrac{\partial f_2(\vec{x})}{\partial x_n} \\ & \vdots & & \\ \dfrac{\partial f_n(\vec{x})}{\partial x_1} & \dfrac{\partial f_n(\vec{x})}{\partial x_2} & \cdots & \dfrac{\partial f_n(\vec{x})}{\partial x_n} \end{bmatrix}$$

Extend Newton-Raphson Algorithm to Multiple Dimensions

given $\vec{f}(\vec{x})$ and $\vec{x}_0$, form multidimensional Taylor approximation

$$\vec{f}(\vec{x}_{i+1}) = \vec{f}(\vec{x}_i) + \boldsymbol{J}(\vec{x}_i)\cdot(\vec{x}_{i+1} - \vec{x}_i)$$

set $\vec{f}(\vec{x}_{i+1}) = 0$ and solve for $\vec{x}_{i+1}$

$$\vec{x}_{i+1} = \vec{x}_i - \boldsymbol{J}^{-1}(\vec{x}_i)\cdot\vec{f}(\vec{x}_i)$$

Note: $\boldsymbol{J}^{-1}(\vec{x}_i)\cdot\vec{f}(\vec{x}_i)$ is just the solution of a system of linear equations with $\boldsymbol{J}(\vec{x}_i)$ as the coefficient matrix and $\vec{f}(\vec{x}_i)$ as the right hand side vector

Multidimensional Newton-Raphson Algorithm

- Requires coding of both the functions being solved and its partial derivatives
- Requires any single vector close to a root to start
- Even more important that starting vector is sufficiently close to root than with one-dimensional case
- Fast convergence rate

C Code for Newton-Raphson Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "sys_roots.h"
#include "lineq.h"

int sys_newton_raphson(int n,void (*func)(int n,double *x,double
*f),void (*jacobian)(int n,double *x,double **coeff),double
*init,double *tol,int n_iters) {
  double **a,*d,*xcurrent,*rhs;
  int *rindex,i,i_iter,done;

  a = alloc_matrix(n,n);
  d = malloc(n * sizeof(double));
  xcurrent = malloc(n * sizeof(double));
  rhs = malloc(n * sizeof(double));
  rindex = (int *) malloc(n * sizeof(int));
  i = 0;
  while (i < n) {
    xcurrent[i] = init[i];
    i++;
  }
```

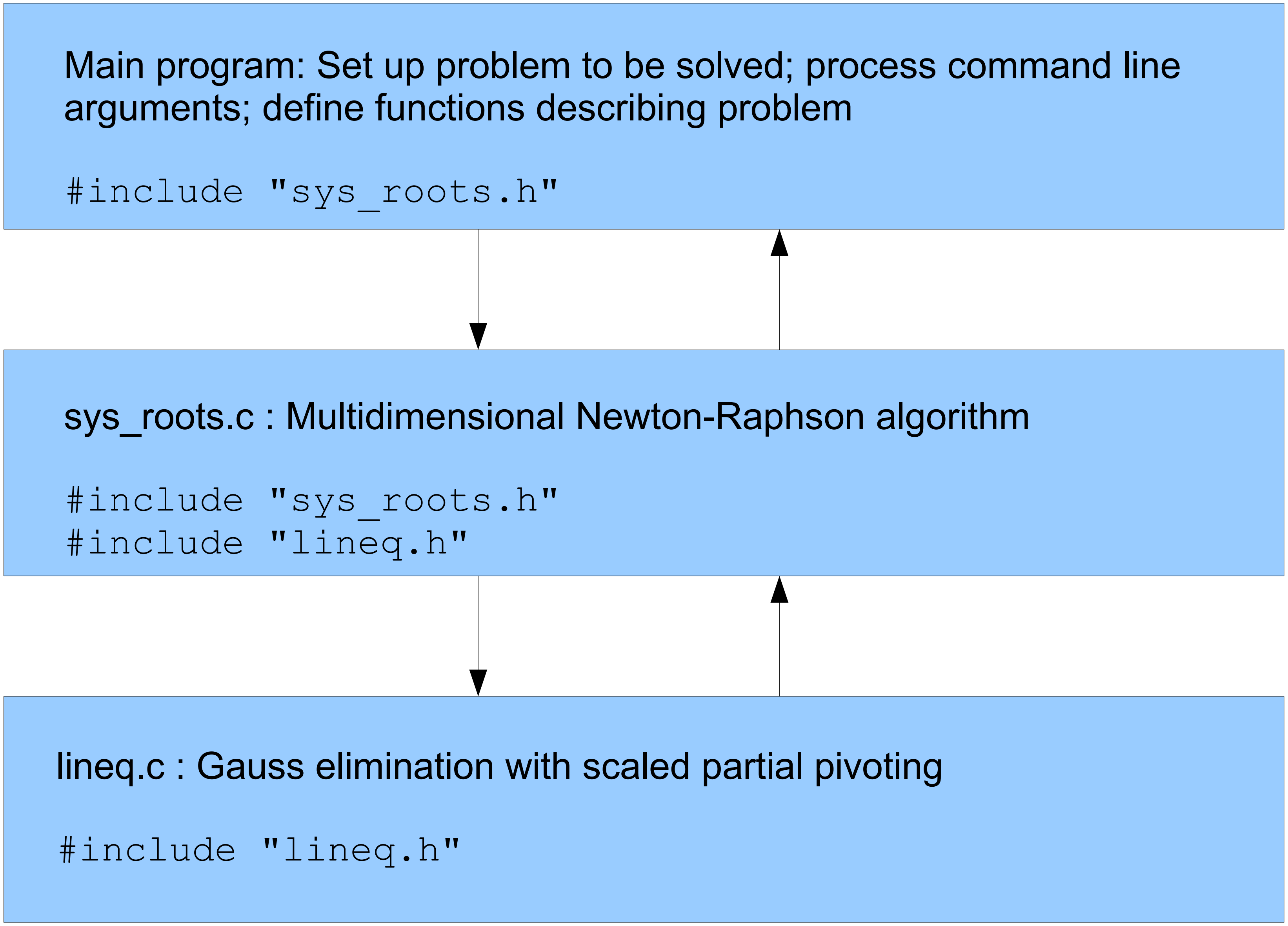
C Code for Newton-Raphson Algorithm (cont'd)

```
  i_iter = 0;
  while (i_iter < n_iters) {
    (*jacobian)(n,xcurrent,a);   /*fill current Newton-Raphson A matrix and RHS
vector*/
    (*func)(n,xcurrent,rhs);
    if (gauss_pivotmax(n,a,rhs,d,rindex,1.0e-12)) {
      fprintf(stderr,"Singular Jacobian matrix\n");
      free_matrix(n,a);
      free(d);
      free(xcurrent);
      free(rhs);
      free(rindex);
      return(1);
    }
    done = 1;
    i = 0;   /*test if all x values are within convergence tolerance*/
    while (i < n) {
      xcurrent[i] -= d[rindex[i]];   /*update current x vector*/
      printf("%.12g ",xcurrent[i]);
      if (fabs(d[rindex[i]]) > tol[i]) done = 0;
      i++;
    }
    putchar('\n');
    if (done) break;
    i_iter++;
  }
```

C Code for Newton-Raphson Algorithm (cont'd)

```
  free_matrix(n,a);
  free(d);
  free(xcurrent);
  free(rhs);
  free(rindex);
  if (i_iter == n_iters) {
    fprintf(stderr,"sys_newton_raphson: Iteration limit of %d
reached\n",n_iters);
    return(1);
  }
  return(0);
}
```

C Function Dependencies



Main program: Set up problem to be solved; process command line arguments; define functions describing problem

```
#include "sys_roots.h"
```

sys_roots.c : Multidimensional Newton-Raphson algorithm

```
#include "sys_roots.h"
#include "lineq.h"
```

lineq.c : Gauss elimination with scaled partial pivoting

```
#include "lineq.h"
```

Numerically Calculating Jacobian Entries

Start with simple one-sided numerical approximation to the first derivative

$$f(x+h) = f(x) + h f'(x) + \frac{f''(\xi)}{2} h^2$$

Solve for *f′(x)*

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(\xi)}{2} h$$

Error term is of order *h*

Numerically Calculating Jacobian Entries

Improve error order with two-sided numerical approximation to the first derivative

$$f(x+h) = f(x) + h f'(x) + \frac{h^2 f''(x)}{2} + \frac{f'''(\xi_1)}{6} h^3$$

$$f(x-h) = f(x) - h f'(x) + \frac{h^2 f''(x)}{2} - \frac{f'''(\xi_2)}{6} h^3$$

Subtract these two equations, then solve for *f′(x)*

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f'''(\xi)}{6} h^2$$

Now error term is of order *h*^{2} !

Numerically Calculating Jacobian Entries

Use two-sided numerical approximation to the partial derivatives in the Jacobian as well

$$\frac{\partial f_i(\vec{x})}{\partial x_j} \approx \frac{f_i(\vec{x}+\vec{h}_j) - f_i(\vec{x}-\vec{h}_j)}{2h_j}$$

where

$$\vec{h}_j = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ h_j \\ \vdots \\ 0 \end{bmatrix}$$

is a vector incrementing only the *j*th argument component

Example Code for 2-D Numerical Jacobian

```
void jacobian_numerical(int n,double *x,double **a) {
  double f1[2],f2[2],xtest[2];

  xtest[0] = x[0] - inc[0];
  xtest[1] = x[1];
  func(n,xtest,f1);
  xtest[0] = x[0] + inc[0];
  func(n,xtest,f2);
  a[0][0] = (f2[0] - f1[0]) / (2.0 * inc[0]);
  a[1][0] = (f2[1] - f1[1]) / (2.0 * inc[0]);
  xtest[0] = x[0];
  xtest[1] = x[1] - inc[1];
  func(n,xtest,f1);
  xtest[1] = x[1] + inc[1];
  func(n,xtest,f2);
  a[0][1] = (f2[0] - f1[0]) / (2.0 * inc[1]);
  a[1][1] = (f2[1] - f1[1]) / (2.0 * inc[1]);
  return;
}
```

Code with `while` or `for` loops for more than two dimensions

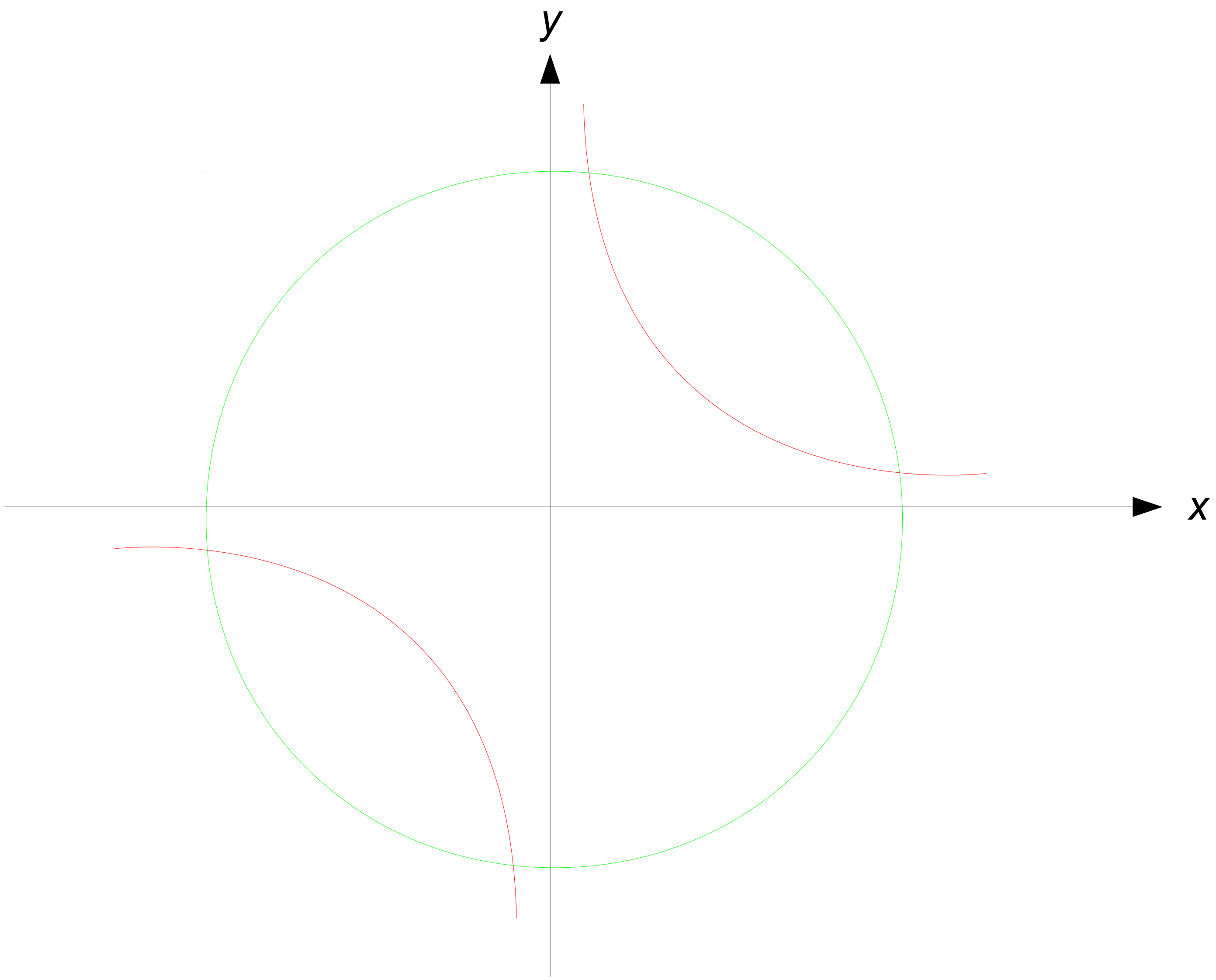
General Code for Numerical Jacobian

```
void jacobian_numerical(int n,double *x,double **a) {
  double *f1,*f2,*xtest;
  int i,j;

  f1 = (double *) malloc(n * sizeof(double));
  f2 = (double *) malloc(n * sizeof(double));
  xtest = (double *) malloc(n * sizeof(double));
  for (j = 0; j < n; j++) {
/* initialize xtest vector */
    for (i = 0; i < n; i++) {
      xtest[i] = x[i];
    }
/* find partial derivative with respect to x[j] */
    xtest[j] = x[j] - inc[j];
    func(n,xtest,f1);
    xtest[j] = x[j] + inc[j];
    func(n,xtest,f2);
    for (i = 0; i < n; i++) {
      a[i][j] = (f2[i] - f1[i]) / (2.0 * inc[j]);
    }
  }
  free(f1);
  free(f2);
  free(xtest);
  return;
}
```

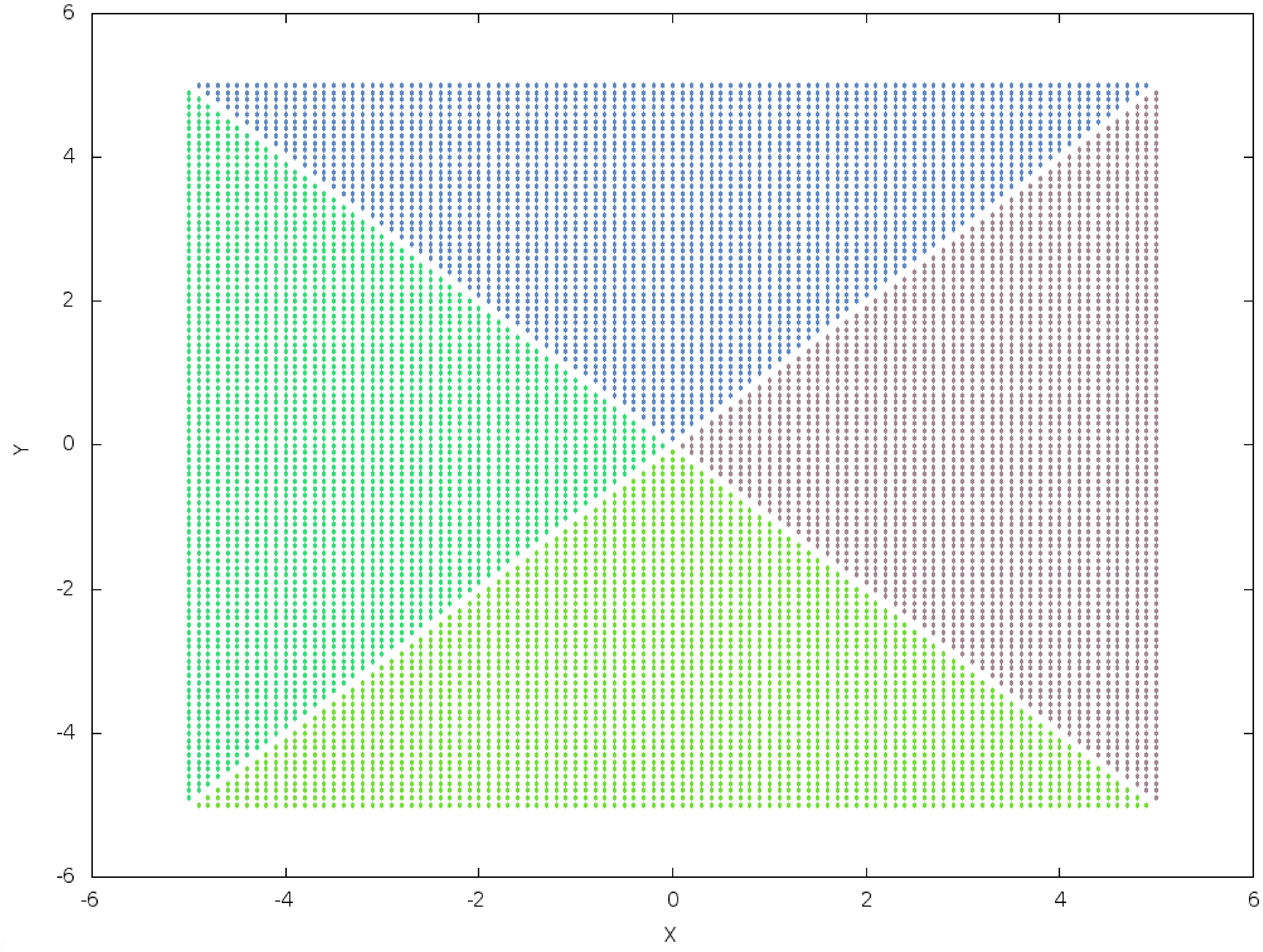
Works for any dimension, passed as argument *n*

Example System of Nonlinear Equations

$$f(x,y) = xy - 1 \qquad g(x,y) = x^2 + y^2 - 4$$

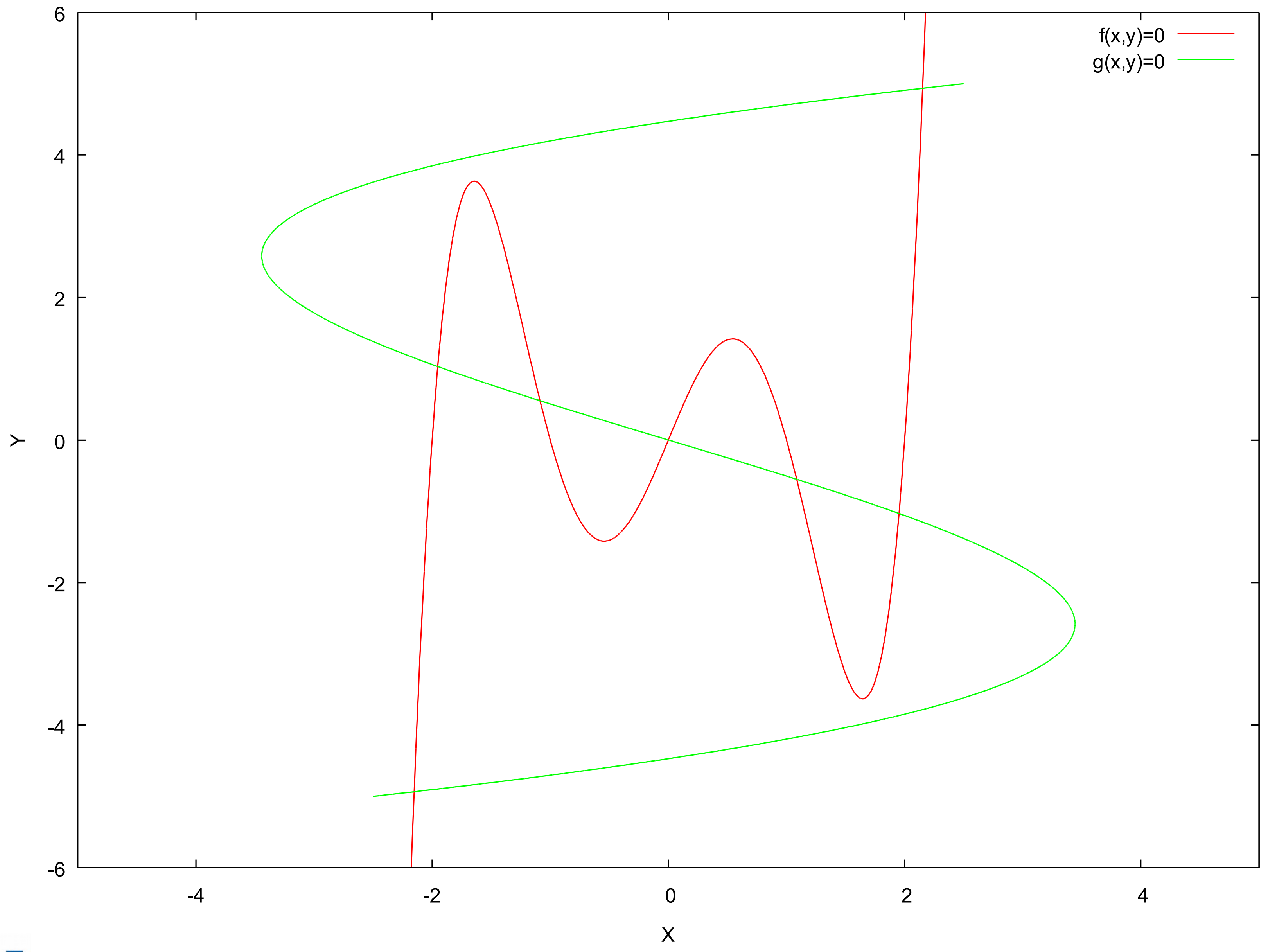


Map of Initial Points to Converged Roots

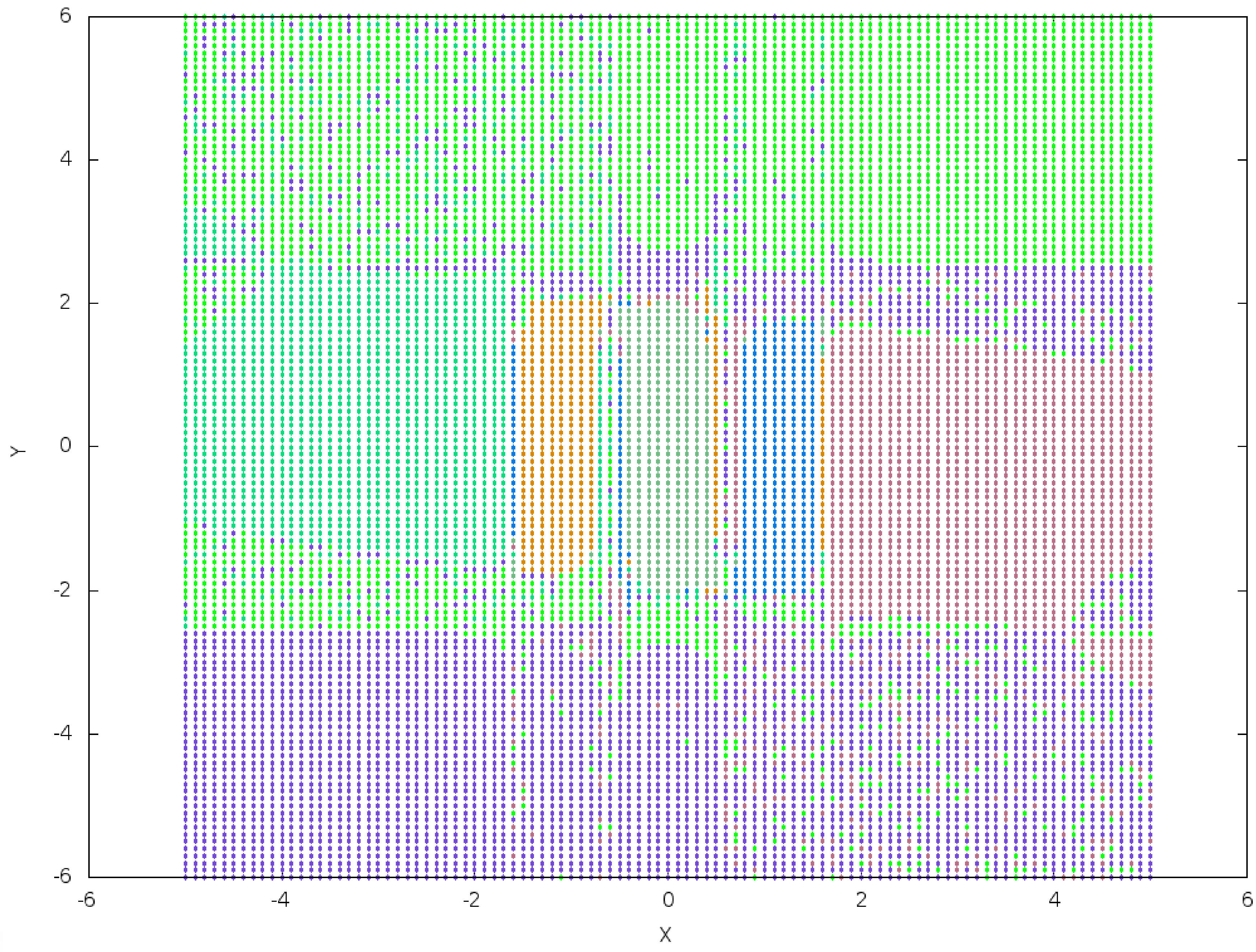


Example System of Nonlinear Equations

$$f(x,y) = x^5 - 5x^3 + 4x - y \qquad g(x,y) = \frac{y^3}{10} - 2y - x$$



Map of Initial Points to Converged Roots



Add Relaxation Factor to Multi-dimensional N-R Algorithm

$$\vec{x}_{i+1} = \vec{x}_i - \alpha \boldsymbol{J}^{-1}(\vec{x}_i) \cdot \vec{f}(\vec{x}_i)$$

where α<1 is the “relaxation” factor. This tends to be an empirically determined “fudge factor”, typically in the range of 0.1 to 0.25. The value of α trades off between convergence speed and convergence reliability.

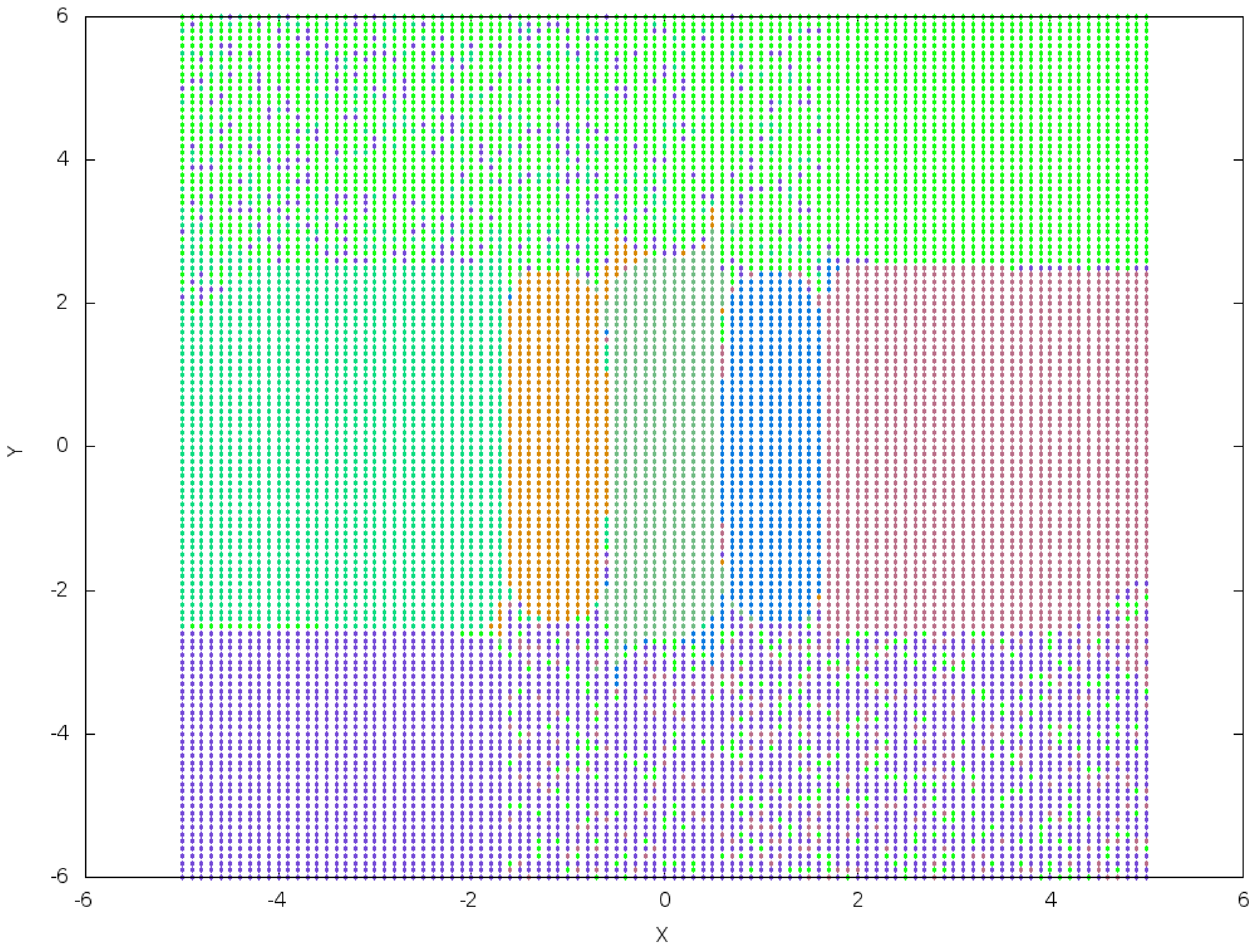
As before, $\boldsymbol{J}^{-1}(\vec{x}_i)\vec{f}(\vec{x}_i)$ is the displacement from the current $\vec{x}_i$ to the next $\vec{x}_i$

and is just the solution of a system of linear equations with

$\boldsymbol{J}(\vec{x}_i)$ as the coefficient matrix and

$\vec{f}(\vec{x}_i)$ as the right hand side vector

Map of Initial Points to Converged Roots



Relaxation factor α = 0.1

Root Finding Algorithms

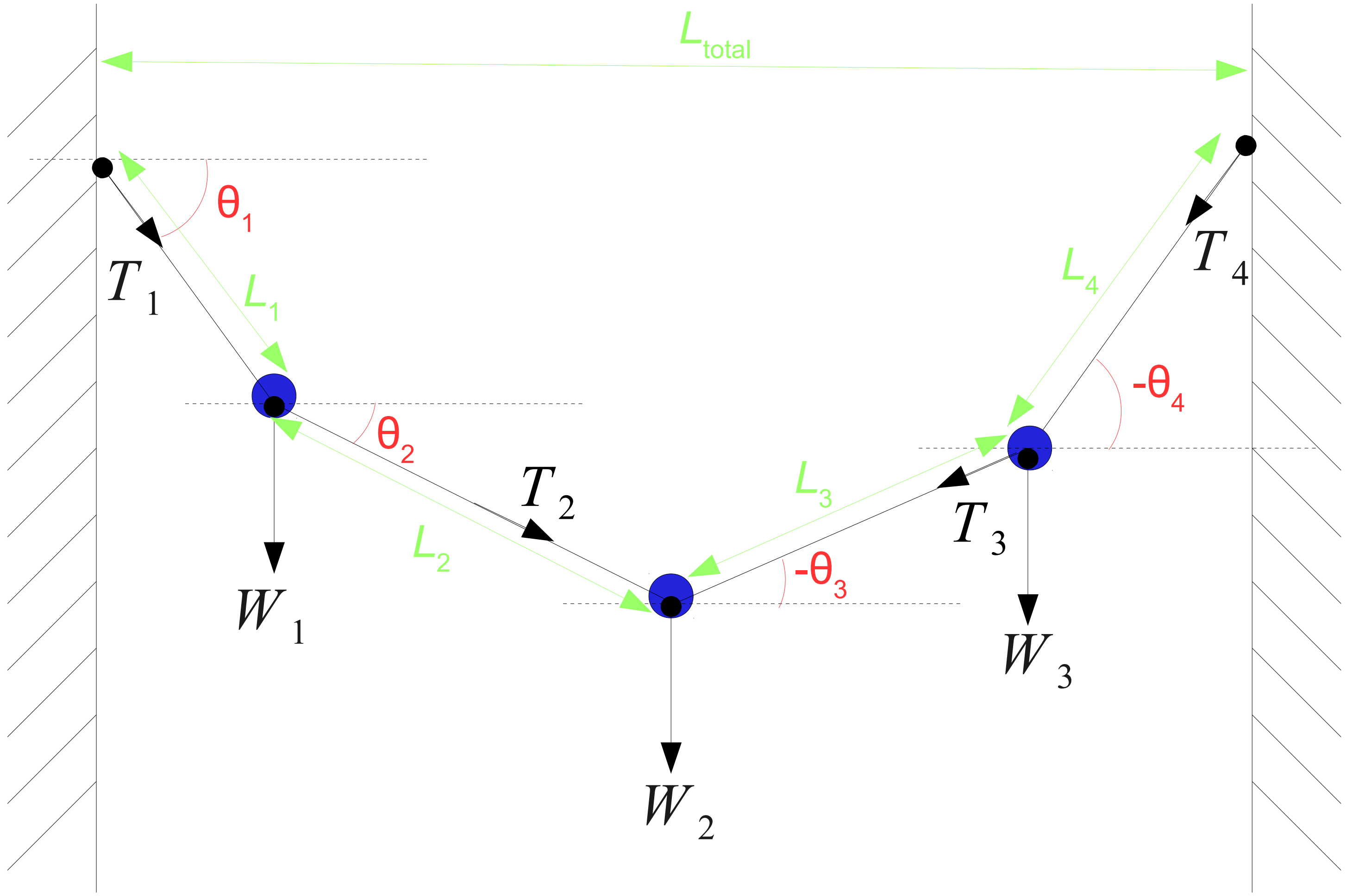
Robustness ← In one dimension: → *Convergence rate*

Bisection — Regula Falsi — Secant — Newton-Raphson

Robustness ← In multiple dimensions: → *Convergence rate*

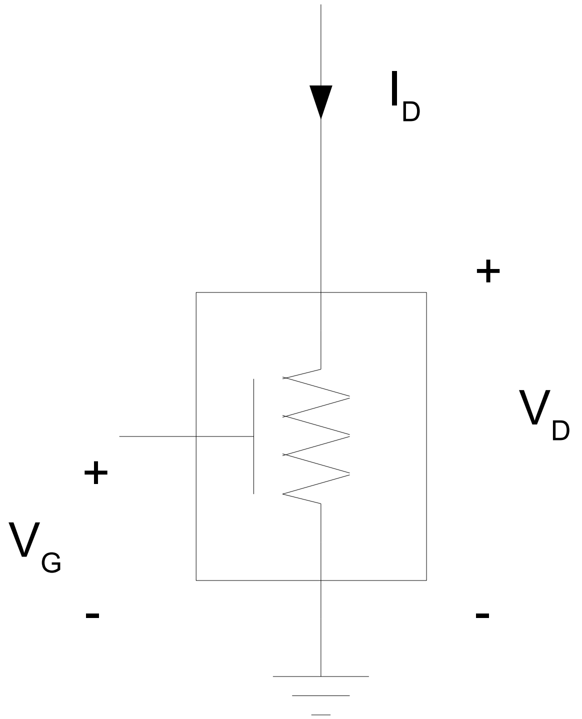
Multidimensional Newton-Raphson with Relaxation, small α	Multidimensional Newton-Raphson with Relaxation, larger α	Multidimensional Newton-Raphson

Statics Problem with No Analytical Solution



Given the four lengths L_1 through L_4 and the three weights W_1 through W_3, solve for four tensions T_1 through T_4 and four variable angles θ_1 through θ_4. The angles θ_i are the angles *below* the horizontal.

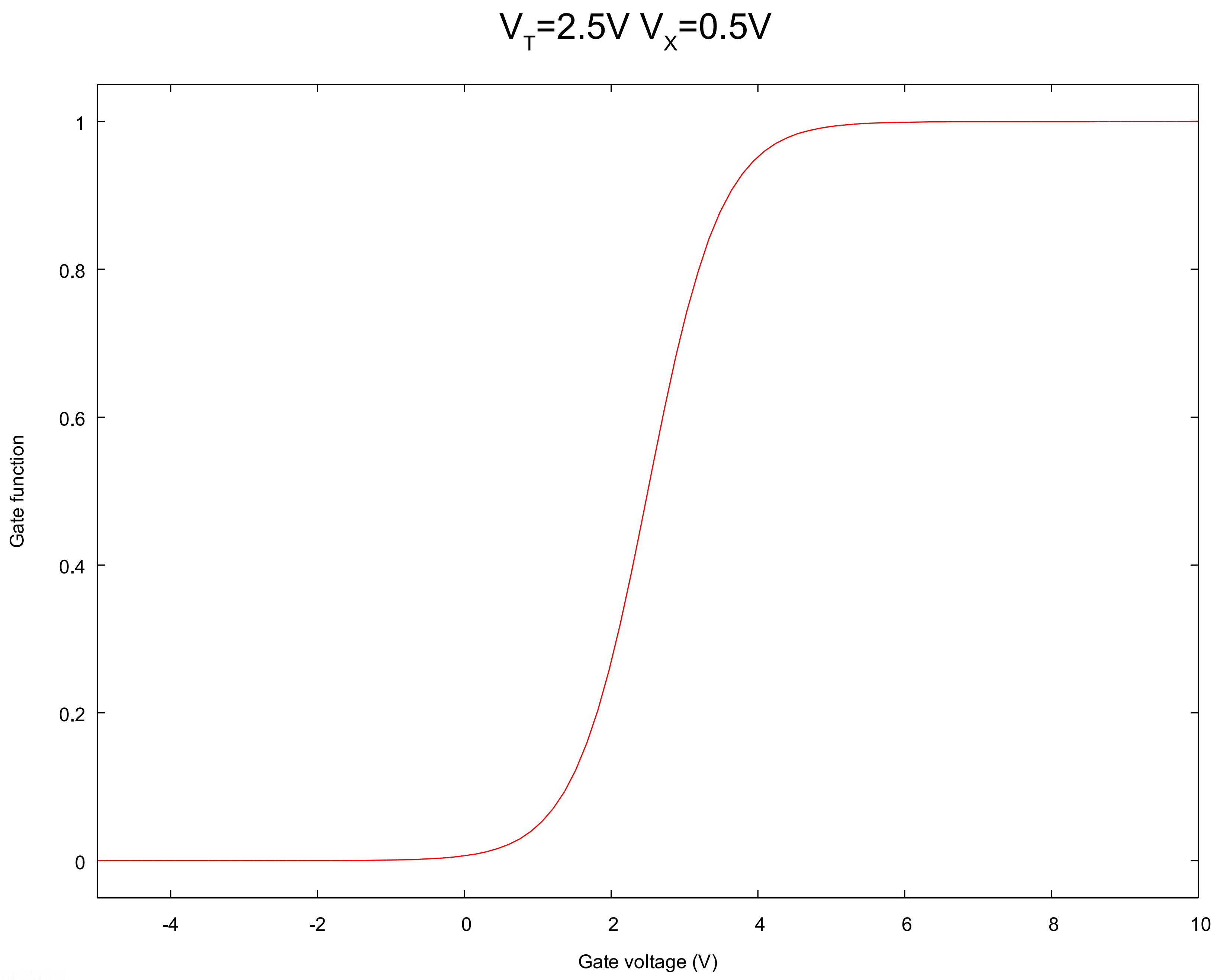
Hypothetical “Transistor” Device



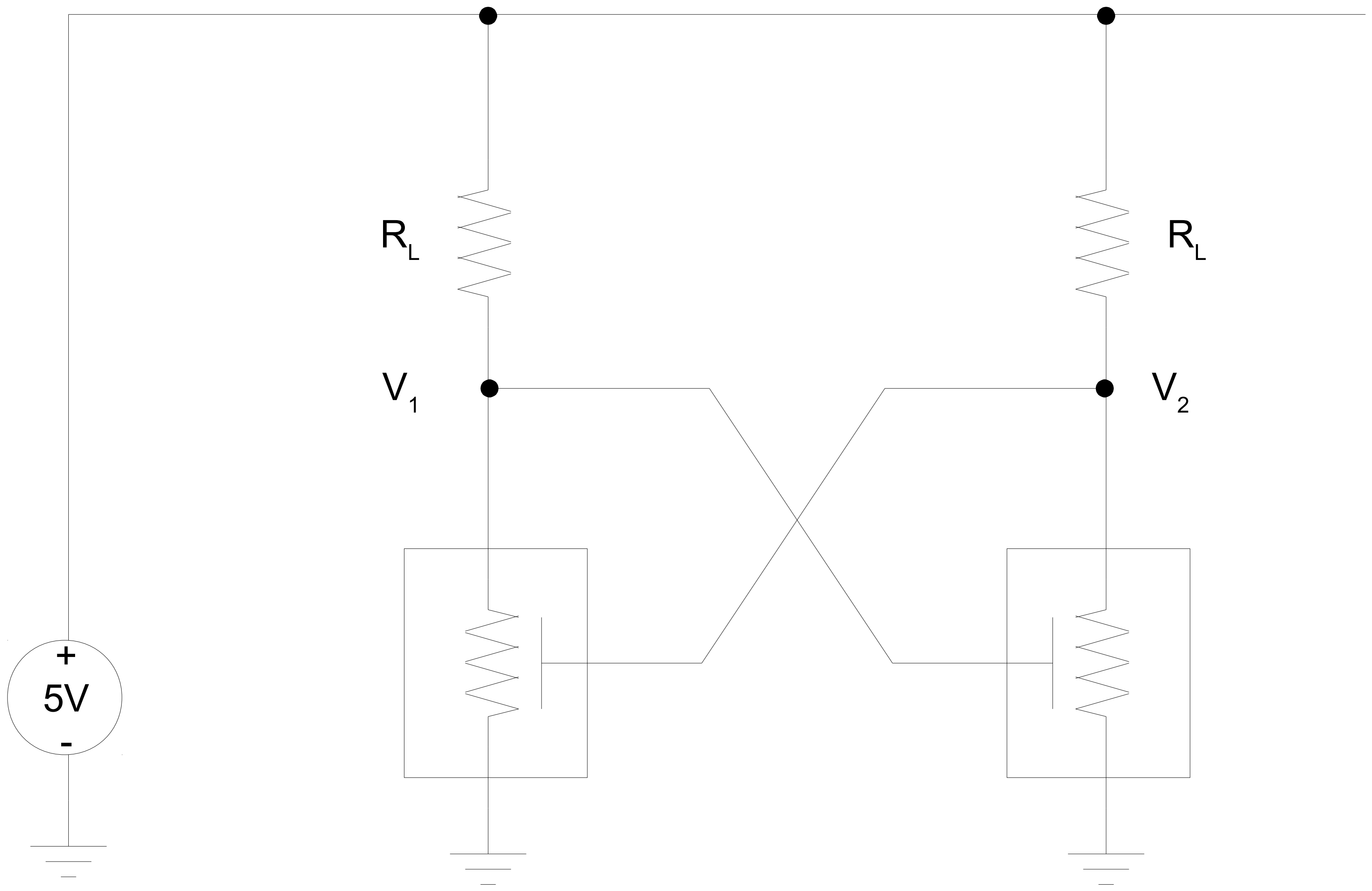
$$I_D = \frac{V_D}{R_0} \cdot g(V_G)$$

where the gate function $g(V_G) = \dfrac{e^{\left(\frac{V_G - V_T}{V_X}\right)}}{1 + e^{\left(\frac{V_G - V_T}{V_X}\right)}}$

Gate Function of Hypothetical Device

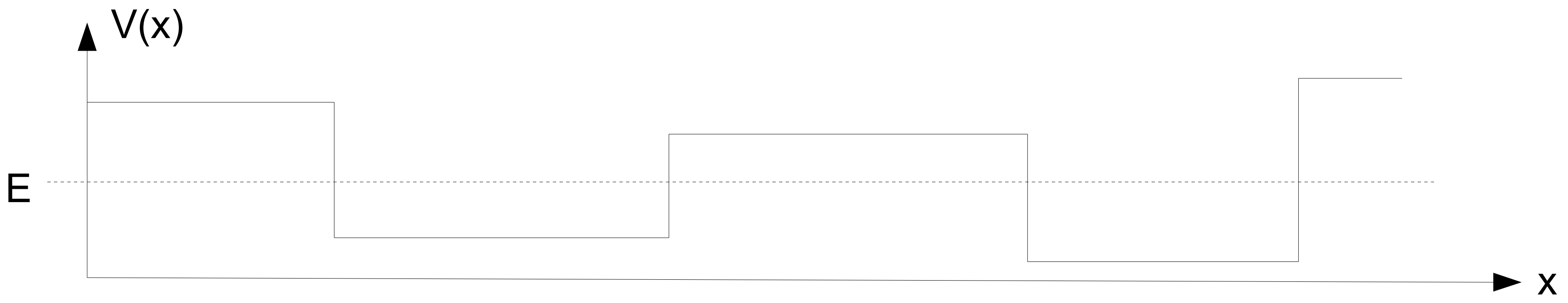


Analyze a Flip-Flop Bit Storage Circuit with KCL



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

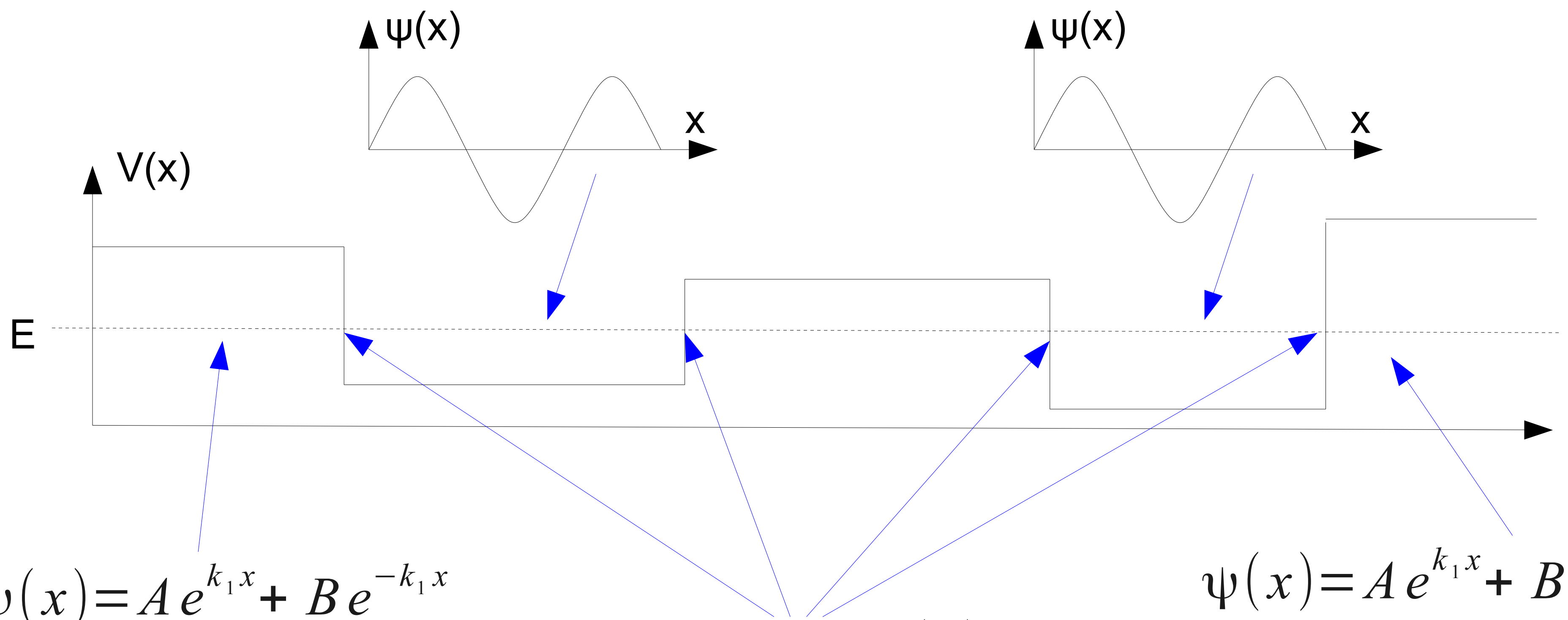
where $k_1 = \frac{\sqrt{2m(V-E)}}{\hbar}$

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

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

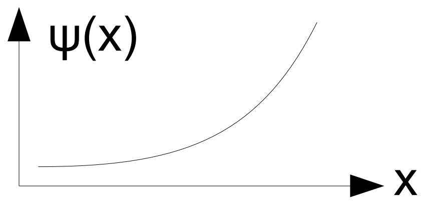
Solving for Electron Energies Analytically

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



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

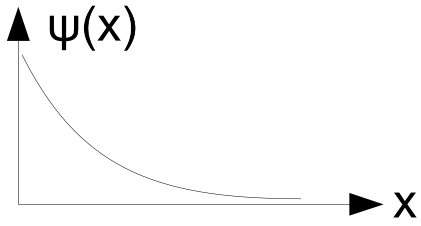
$B = 0$ required by normalization constraint



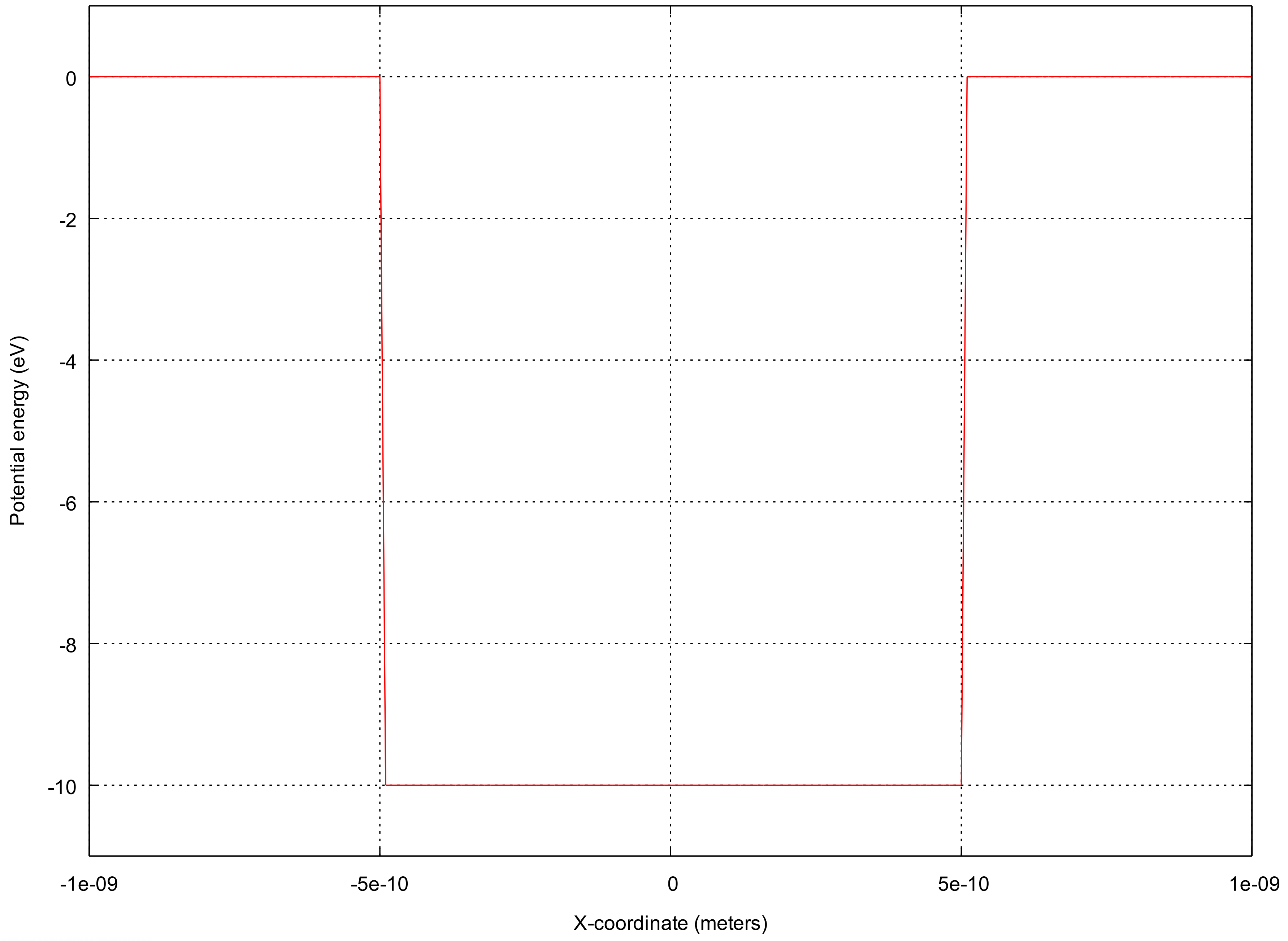
$\psi(x)$ and $\dfrac{d\psi(x)}{dx}$ required to be continuous across boundaries

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

$A = 0$ required by normalization constraint

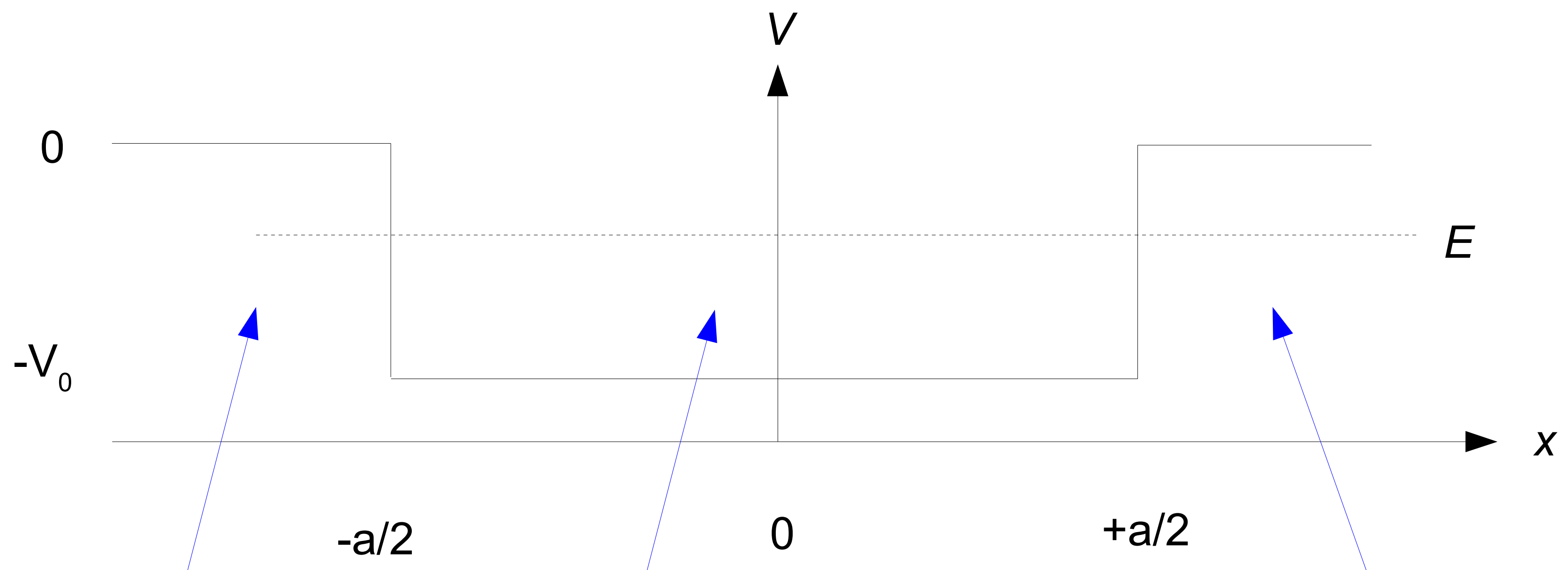


Example Rectangular Potential Well



Solving for Eigenvalues

well width $a = 1\,\mu\text{m}$ well depth $V_0 = 10\,\text{eV}$



$\psi(x) = A e^{k_1 x}$ $\psi(x) = C\cos(k_2 x) + D\sin(k_2 x)$ $\psi(x) = B e^{-k_1 x}$

$$k_1 = \sqrt{\frac{-2mE}{\hbar^2}} \qquad k_2 = \sqrt{\frac{2m(E+V_0)}{\hbar^2}}$$

Solving for Eigenvalues for Even Parity

Assume *A* = *B* and *D* = 0

Apply continuity of Ψ(*x*) at x=*a*/2 :

$$A e^{\frac{-k_1 a}{2}} = C\cos\left(\frac{k_2 a}{2}\right)$$

Apply continuity of Ψ'(*x*) at x=*a*/2 :

$$k_1 A e^{\frac{-k_1 a}{2}} = k_2 C\sin\left(\frac{k_2 a}{2}\right)$$

Take ratio of these two equations

$$k_2 \sin\left(\frac{k_2 a}{2}\right) = k_1 \cos\left(\frac{k_2 a}{2}\right)$$

So define a function for root finding

$$f_{even} = k_2 \sin\left(\frac{k_2 a}{2}\right) - k_1 \cos\left(\frac{k_2 a}{2}\right)$$

Solving for Eigenvalues for Odd Parity

Assume *A* = -*B* and *C* = 0

Apply continuity of Ψ(*x*) at x=*a*/2 :

$$B e^{\frac{-k_1 a}{2}} = D \sin(\frac{k_2 a}{2})$$

Apply continuity of Ψ'(*x*) at x=*a*/2 :

$$k_1 B e^{\frac{-k_1 a}{2}} = -k_2 D \cos(\frac{k_2 a}{2})$$

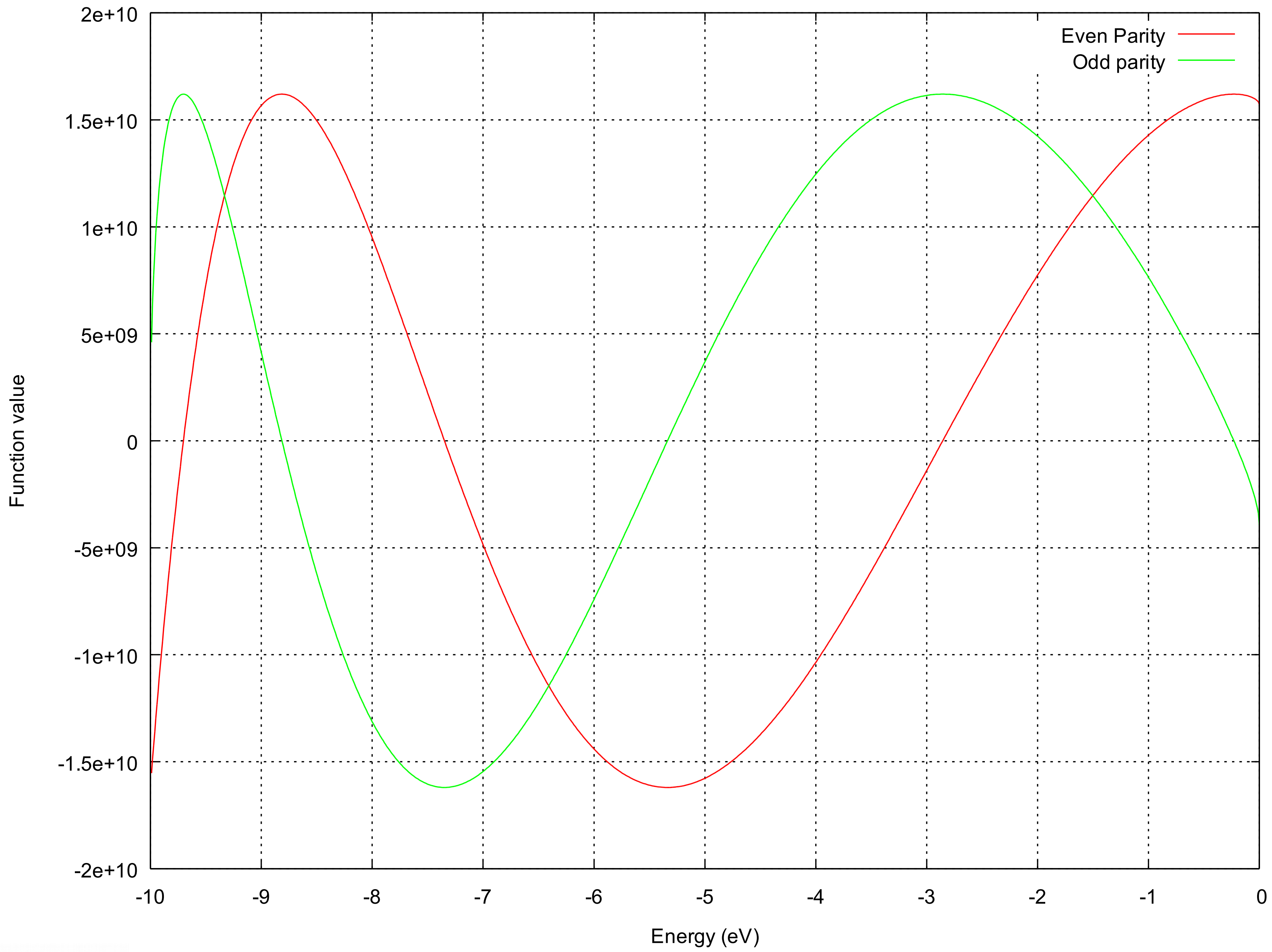
Take ratio of these two equations

$$k_2 \cos(\frac{k_2 a}{2}) = -k_1 \sin(\frac{k_2 a}{2})$$

So define a function for root finding

$$f_{odd} = k_2 \cos(\frac{k_2 a}{2}) + k_1 \sin(\frac{k_2 a}{2})$$

Functions for Even and Odd Eigenvalues

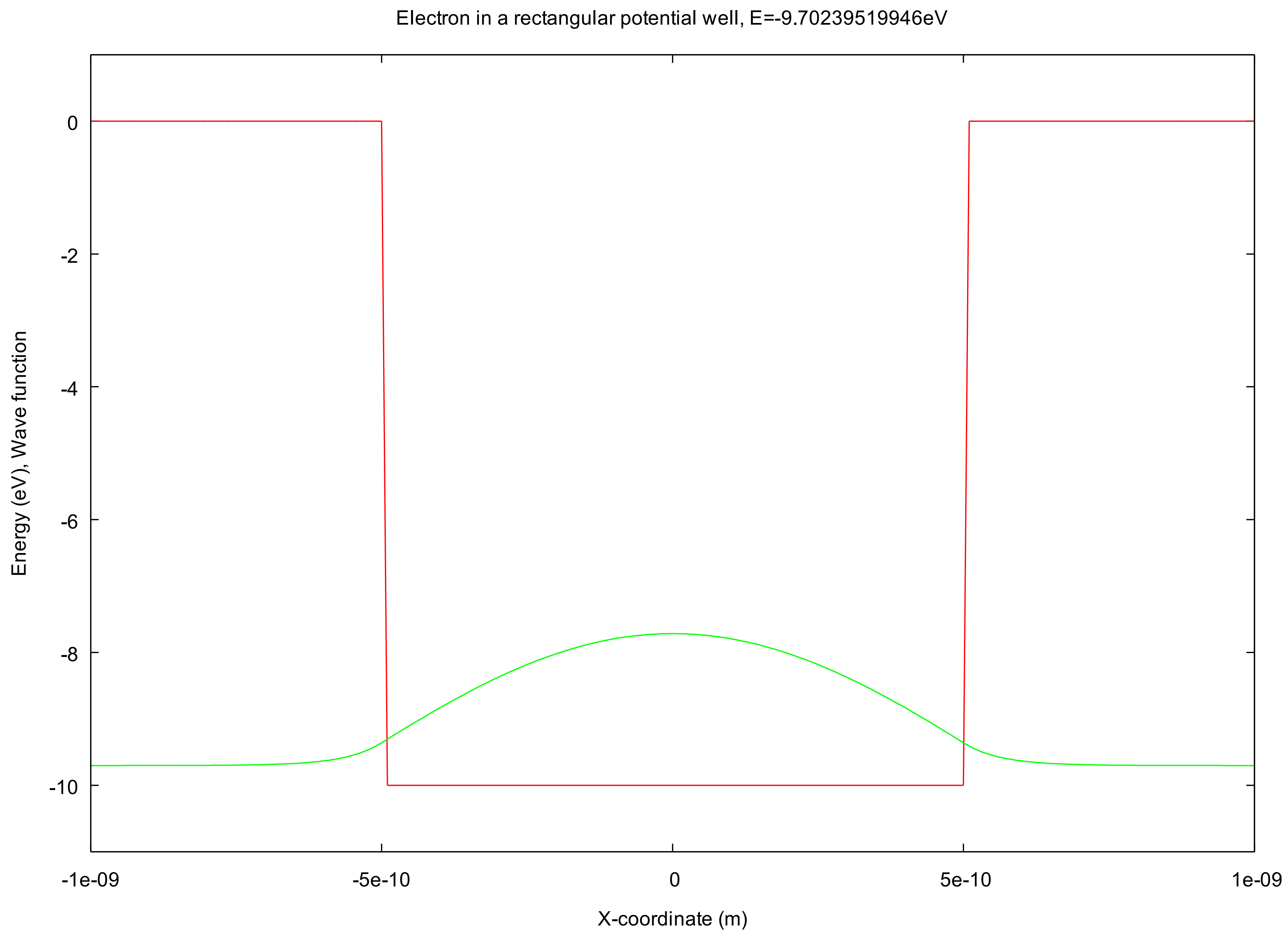


Numerical Solutions for Eigenvalues

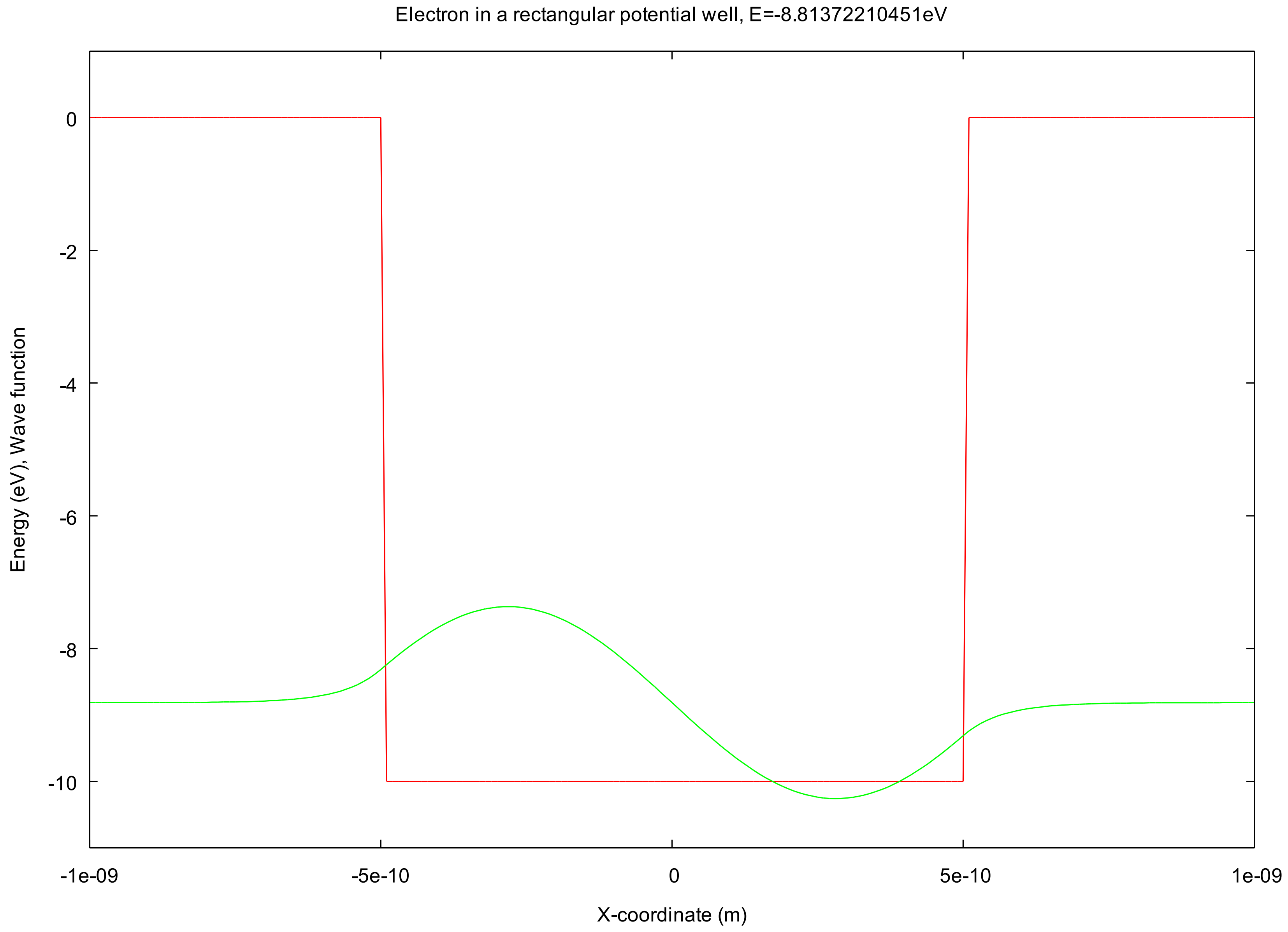
Found with sweep_secant() routine calling the two nonlinear functions:

```
Even wave function:
-9.70239519946
-7.34810530302
-2.85536474272
Odd wave function:
-8.81372210451
-5.33690334777
-0.230628910536
```

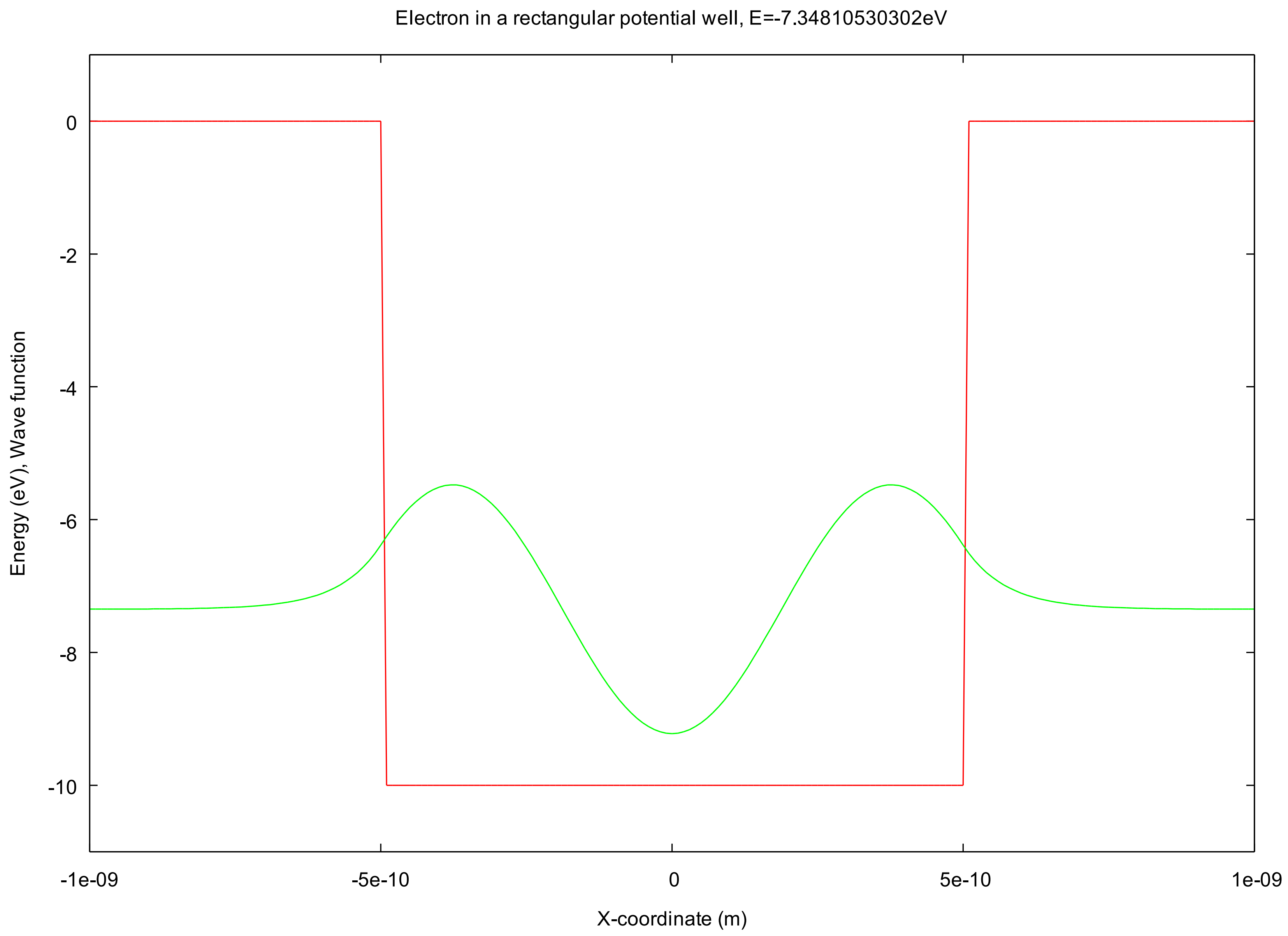
Eigenvalue Solution for Electron in a Quantum Well



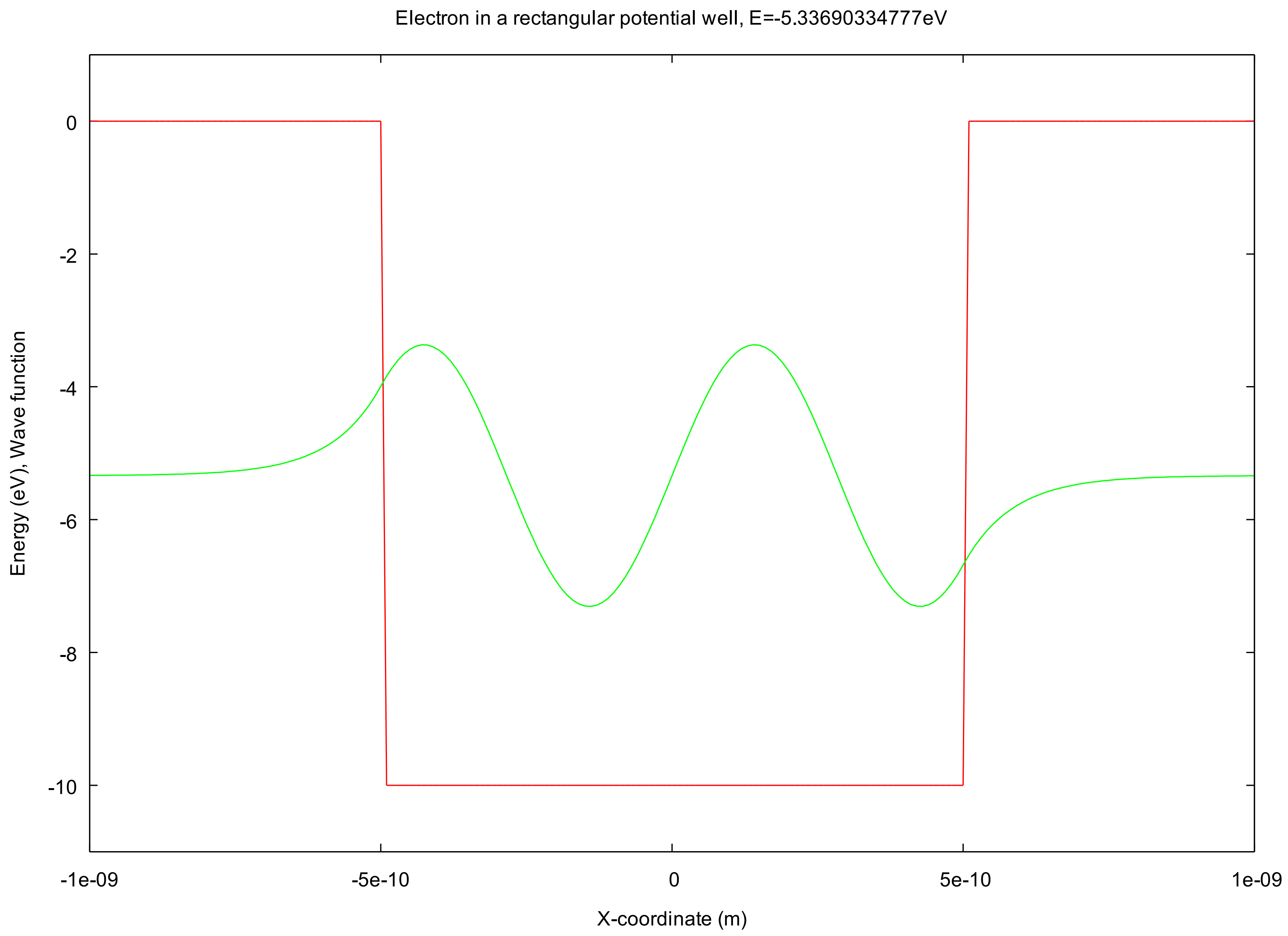
Eigenvalue Solution for Electron in a Quantum Well



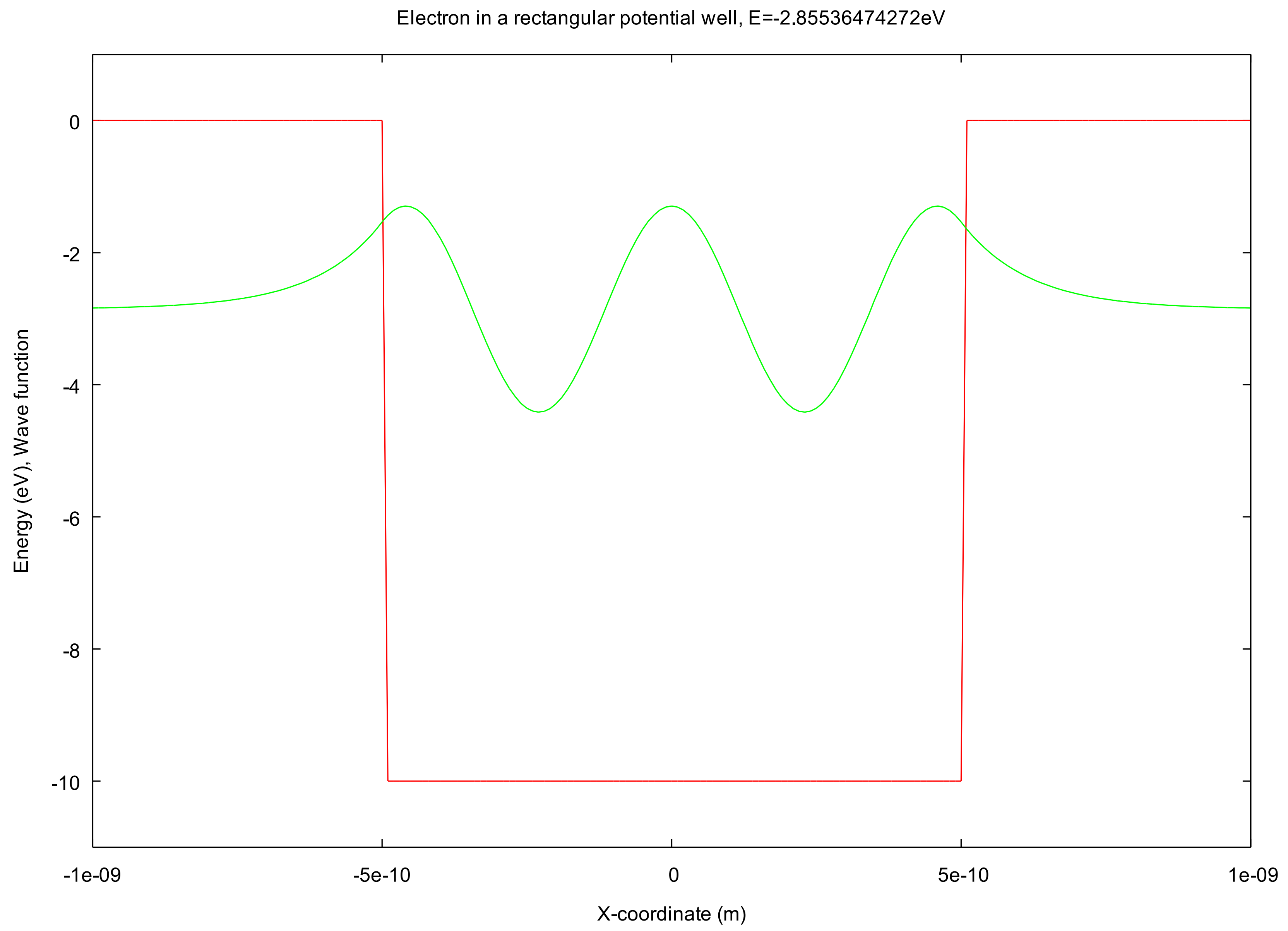
Eigenvalue Solution for Electron in a Quantum Well



Eigenvalue Solution for Electron in a Quantum Well



Eigenvalue Solution for Electron in a Quantum Well



Verify for Case of Infinite Well Depth

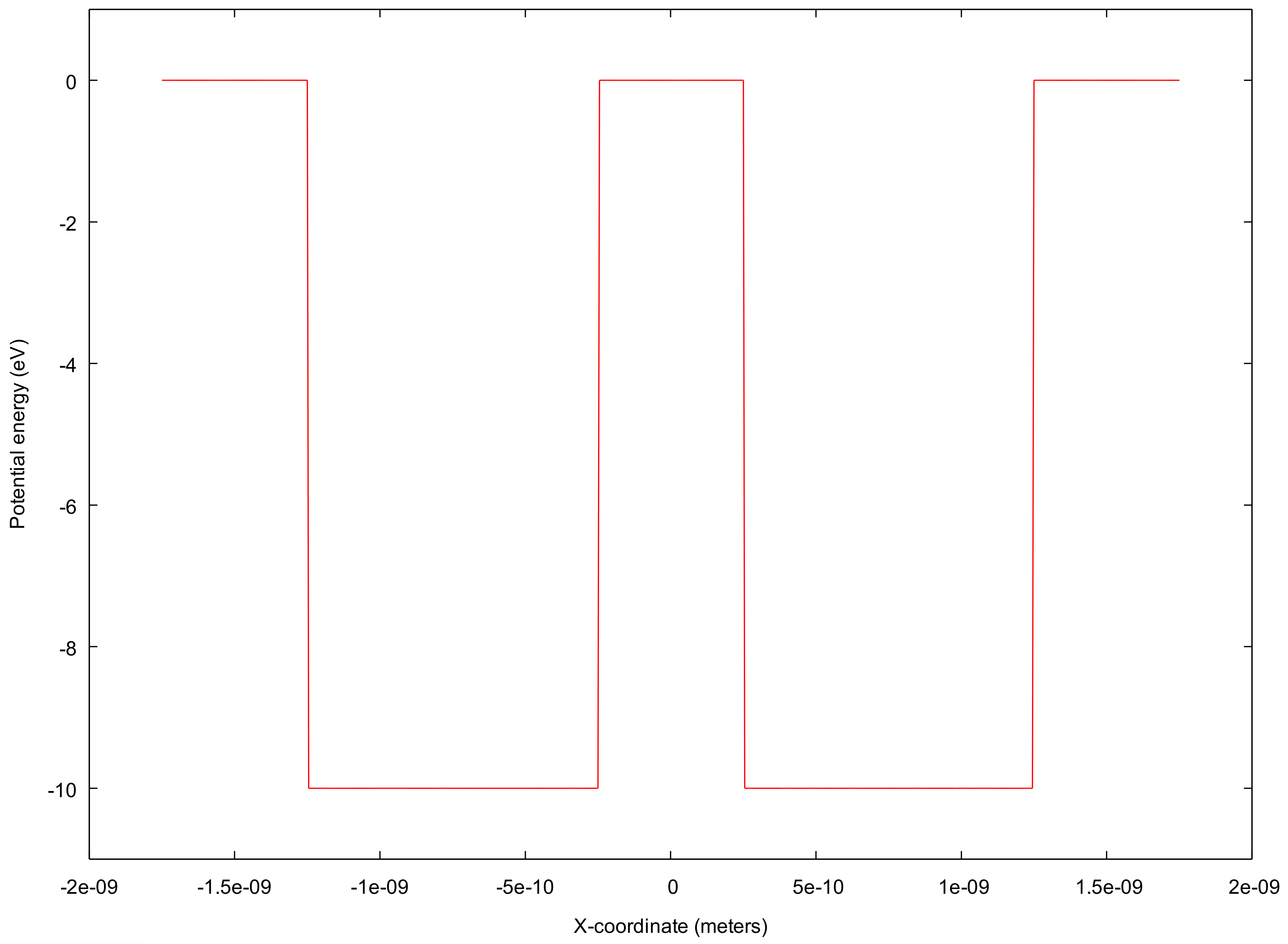
Found with sweep_secant() routine calling the two nonlinear functions for the case of V_0 = 100eV:

$$E_n = \frac{\pi^2 \hbar^2 n^2}{2 m a^2} - V_0$$

```
Even wave function:
-99.6517063762
-96.8664678478
-91.3021469481
-82.9723028236
Odd wave function:
-98.607009151
-94.4310434277
-87.4817113328
```

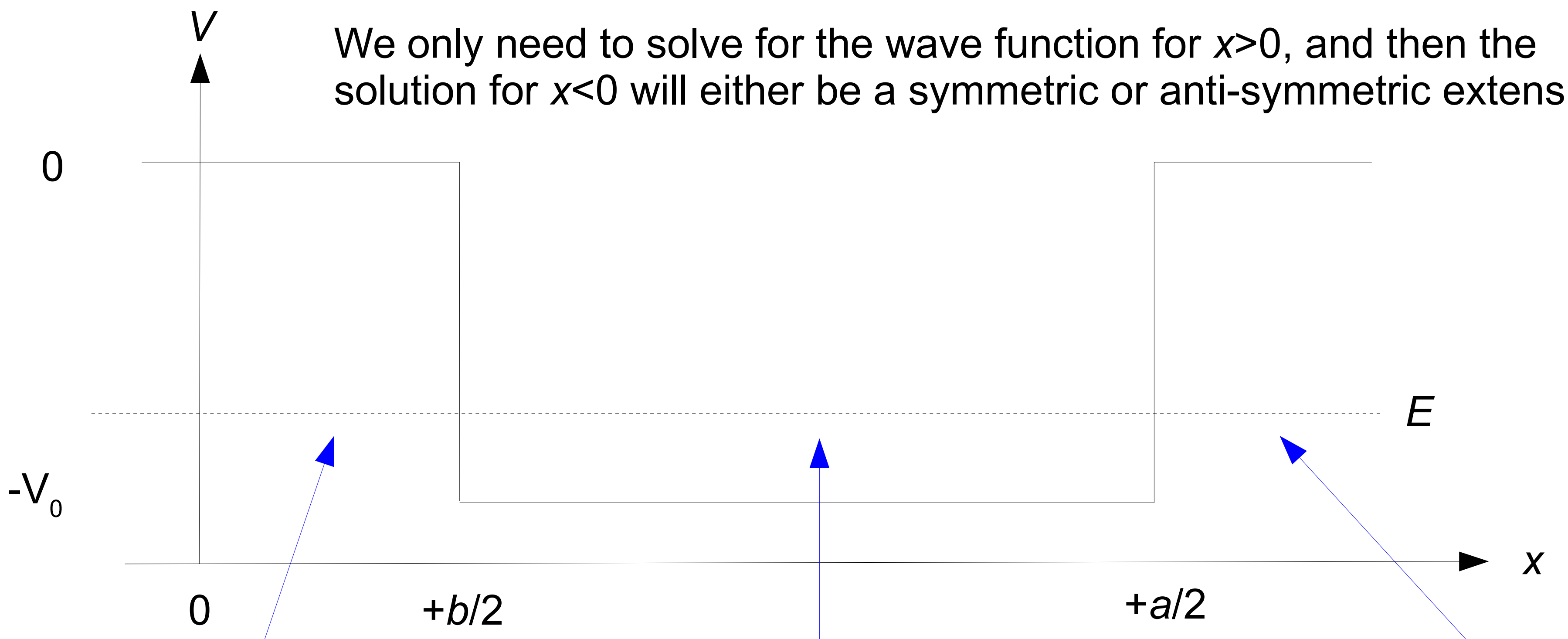
```
n=1 :   -99.623997eV
n=3 :   -96.615975 eV
n=5 :   -90.59993 eV
n=7 :   -81.575863 eV

n=2 :   -98.495989 eV
n=4 :   -93.983955 eV
n=6 :   -86.4639 eV
```

Example Double Rectangular Potential Well



Solving for Eigenvalues

We only need to solve for the wave function for *x*>0, and then the solution for *x*<0 will either be a symmetric or anti-symmetric extension



$$\psi(x) = D e^{k_1 x} + F e^{-k_1 x}$$

$$\psi(x) = A e^{-k_1 (x - \frac{a}{2})}$$

$$\psi(x) = B \cos k_2 (x - \frac{a+b}{4}) + C \sin k_2 (x - \frac{a+b}{4})$$

$$\text{where} \quad k_1 = \sqrt{\frac{-2mE}{\hbar^2}} \quad \text{and} \quad k_2 = \sqrt{\frac{2m(E+V_0)}{\hbar^2}}$$

Solving for Energies of Even Parity States

For a symmetric wave function: $F = D$

Apply continuity of Ψ(*x*) at x=*a*/2 : $A = B\cos k_2(\frac{a-b}{4}) + C\sin k_2(\frac{a-b}{4})$

Apply continuity of Ψ'(*x*) at x=*a*/2 : $-k_1 A = -k_2 B\sin k_2(\frac{a-b}{4}) + k_2 C\cos k_2(\frac{a-b}{4})$

Apply continuity of Ψ(*x*) at x=*b*/2 :

$$De^{\frac{k_1 b}{2}} + De^{\frac{-k_1 b}{2}} = B\cos k_2(-\frac{a-b}{4}) + C\sin k_2(-\frac{a-b}{4})$$

Apply continuity of Ψ'(*x*) at x=*b*/2 :

$$k_1 De^{\frac{k_1 b}{2}} - k_1 De^{\frac{-k_1 b}{2}} = -k_2 B\sin k_2(-\frac{a-b}{4}) + k_2 C\cos k_2(-\frac{a-b}{4})$$

Solving for Energies of Even Parity States

Arbitrarily set *A*=1 and use four dimensional solution of system of four nonlinear equations to find coefficients *B*, *C*, *D*, and energy level *E*:

$$f_1(B,C,D,E) = A - B\cos k_2\left(\frac{a-b}{4}\right) - C\sin k_2\left(\frac{a-b}{4}\right) = 0$$

$$f_2(B,C,D,E) = -k_1 A + k_2 B\sin k_2\left(\frac{a-b}{4}\right) - k_2 C\cos k_2\left(\frac{a-b}{4}\right) = 0$$

$$f_3(B,C,D,E) = De^{\frac{k_1 b}{2}} + De^{\frac{-k_1 b}{2}} - B\cos k_2\left(-\frac{a-b}{4}\right) - C\sin k_2\left(-\frac{a-b}{4}\right) = 0$$

$$f_4(B,C,D,E) =$$

$$k_1 De^{\frac{k_1 b}{2}} - k_1 De^{\frac{-k_1 b}{2}} + k_2 B\sin k_2\left(-\frac{a-b}{4}\right) - k_2 C\cos k_2\left(-\frac{a-b}{4}\right) = 0$$

Solving for Energies of Odd Parity States

For an anti- symmetric wave function: $F = -D$

Apply continuity of Ψ(*x*) at x=*a*/2 : $A = B\cos k_2(\frac{a-b}{4}) + C\sin k_2(\frac{a-b}{4})$

Apply continuity of Ψ'(*x*) at x=*a*/2 : $-k_1 A = -k_2 B\sin k_2(\frac{a-b}{4}) + k_2 C\cos k_2(\frac{a-b}{4})$

Apply continuity of Ψ(*x*) at x=*b*/2 :

$$De^{\frac{k_1 b}{2}} - De^{\frac{-k_1 b}{2}} = B\cos k_2(-\frac{a-b}{4}) + C\sin k_2(-\frac{a-b}{4})$$

Apply continuity of Ψ'(*x*) at x=*b*/2 :

$$k_1 De^{\frac{k_1 b}{2}} + k_1 De^{\frac{-k_1 b}{2}} = -k_2 B\sin k_2(-\frac{a-b}{4}) + k_2 C\cos k_2(-\frac{a-b}{4})$$

Solving for Energies of Odd Parity States

Arbitrarily set *A*=1 and use four dimensional solution of system of four nonlinear equations to find coefficients *B*, *C*, *D*, and energy level *E*:

$$f_1(B,C,D,E) \;=\; A - B\cos k_2\left(\frac{a-b}{4}\right) - C\sin k_2\left(\frac{a-b}{4}\right) \;=\; 0$$

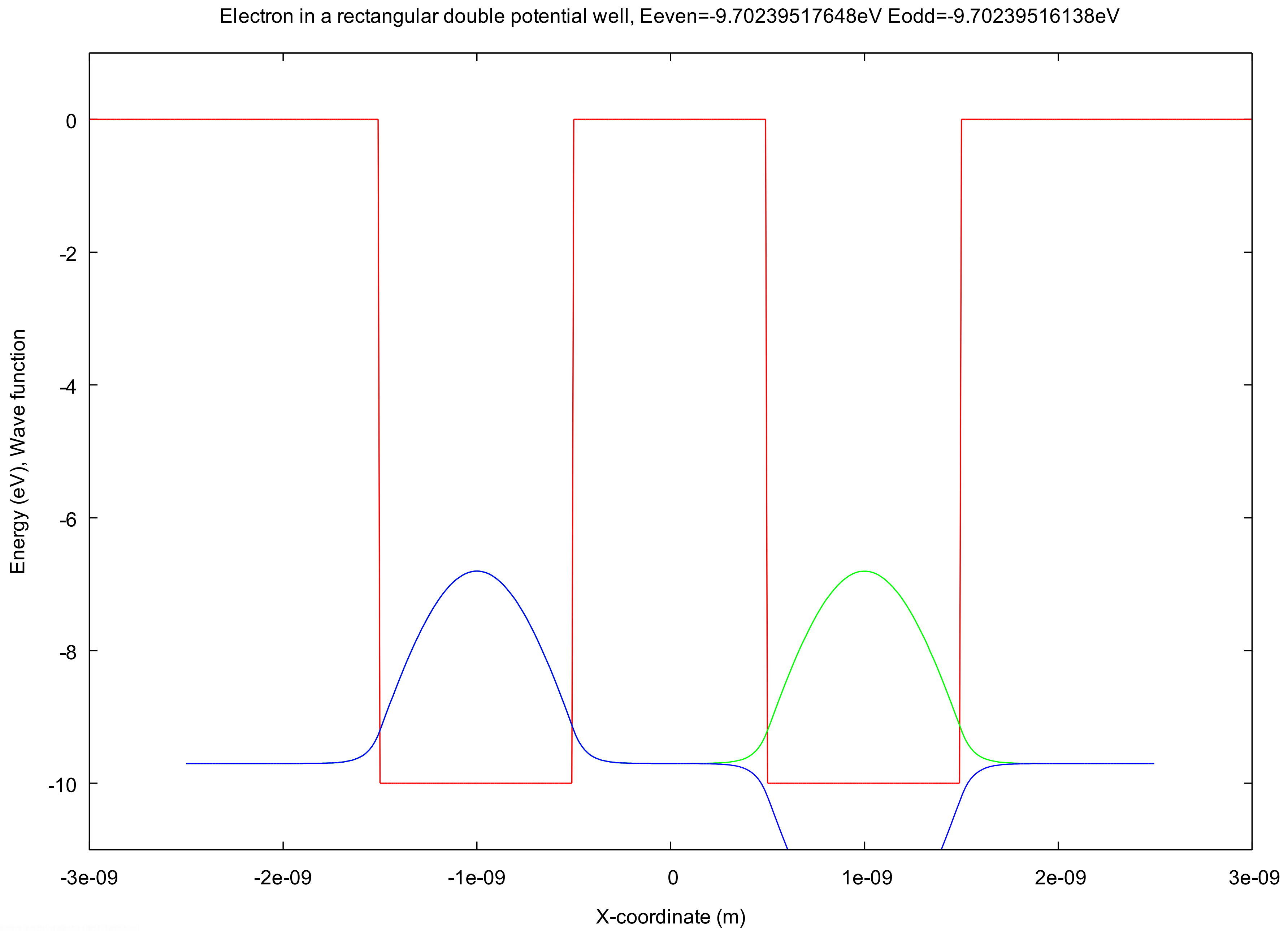
$$f_2(B,C,D,E) \;=\; -k_1 A + k_2 B\sin k_2\left(\frac{a-b}{4}\right) - k_2 C\cos k_2\left(\frac{a-b}{4}\right) \;=\; 0$$

$$f_3(B,C,D,E) \;=\; D e^{\frac{k_1 b}{2}} - D e^{\frac{-k_1 b}{2}} - B\cos k_2\left(-\frac{a-b}{4}\right) - C\sin k_2\left(-\frac{a-b}{4}\right) \;=\; 0$$

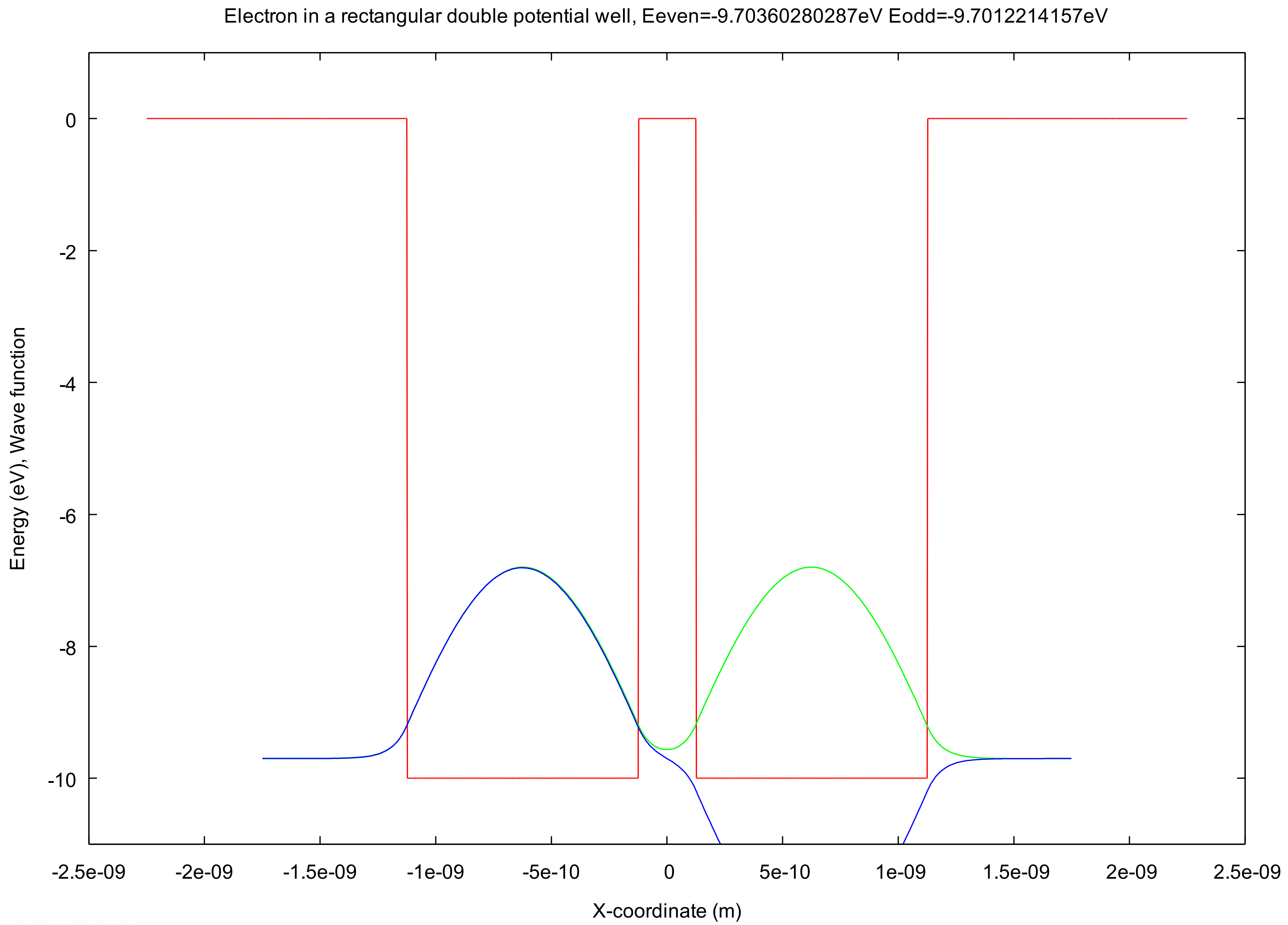
$$f_4(B,C,D,E) \;=$$

$$k_1 D e^{\frac{k_1 b}{2}} + k_1 D e^{\frac{-k_1 b}{2}} + k_2 B\sin k_2\left(-\frac{a-b}{4}\right) - k_2 C\cos k_2\left(-\frac{a-b}{4}\right) \;=\; 0$$

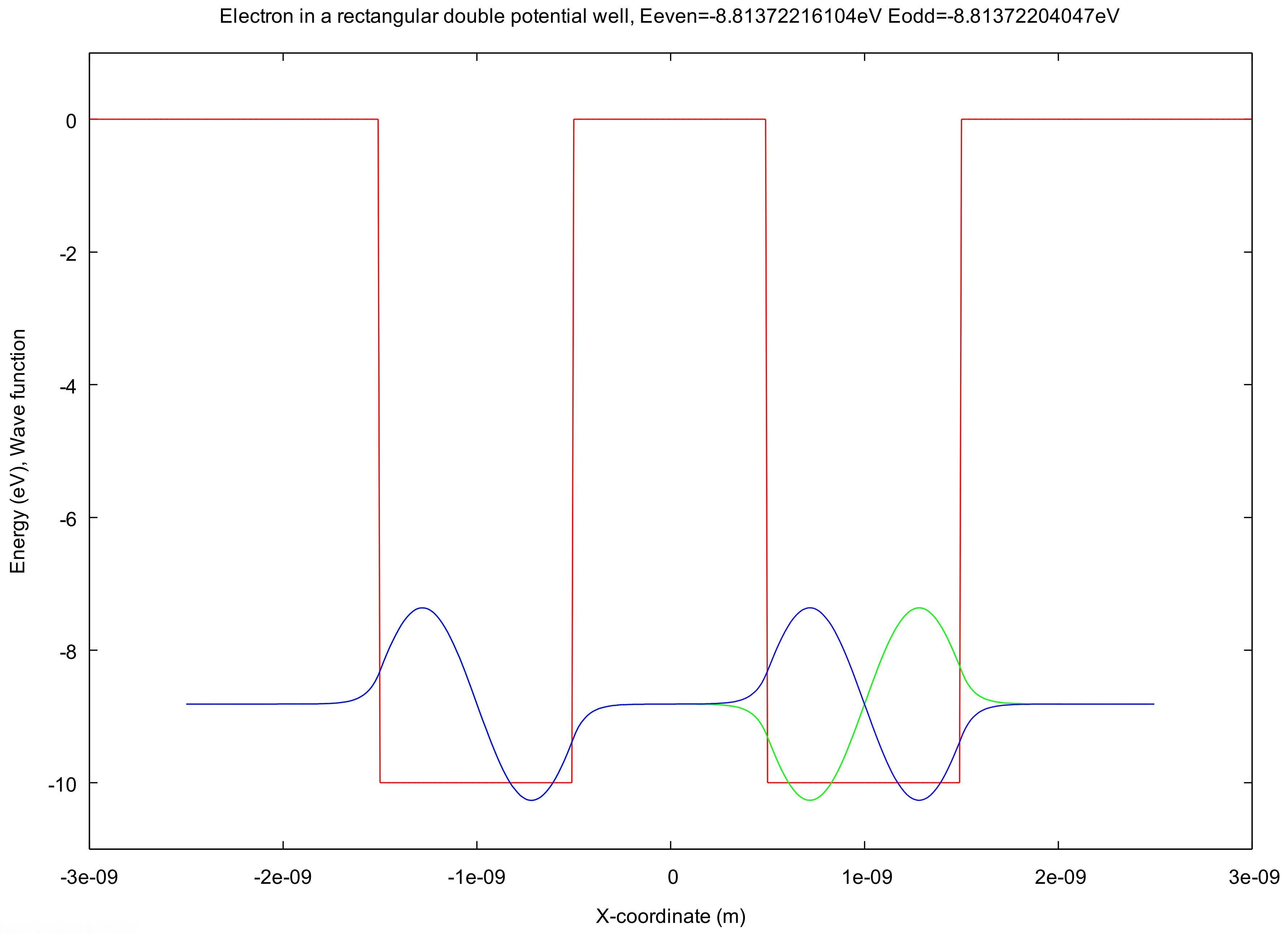
Lowest Energy States with 1μm Well Separation



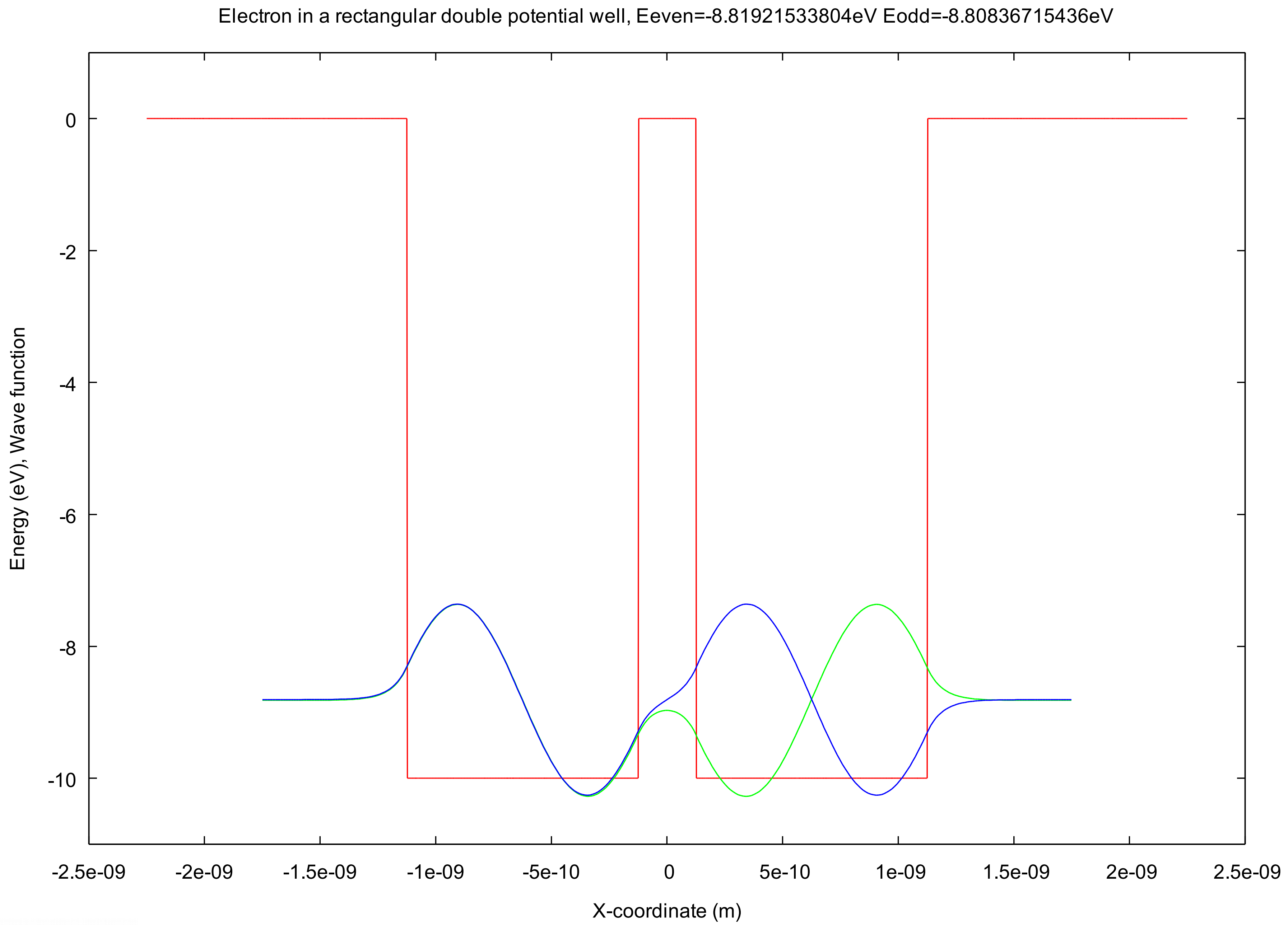
Lowest Energy States with 0.25μm Well Separation



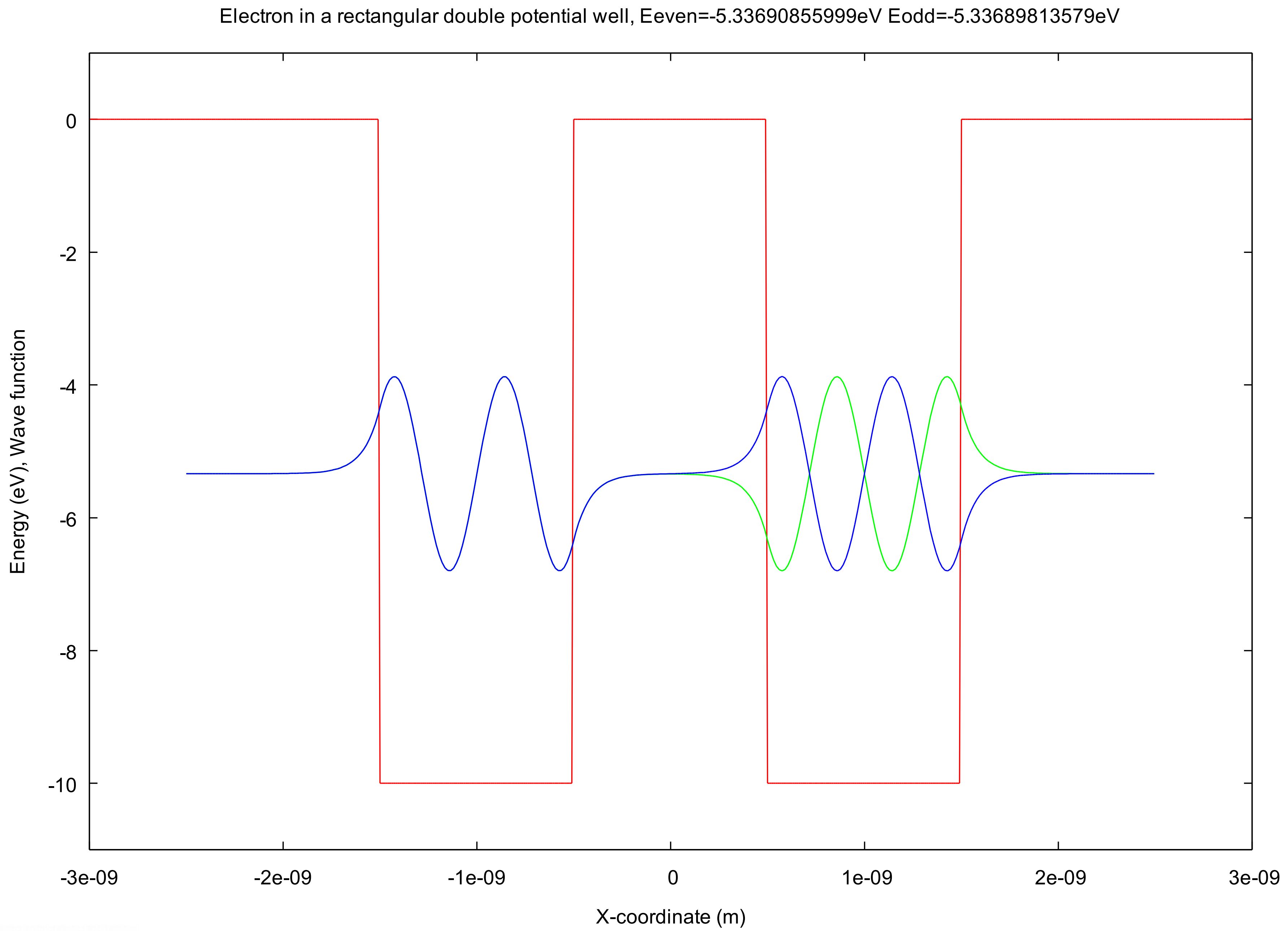
Next Energy States with 1μm Well Separation



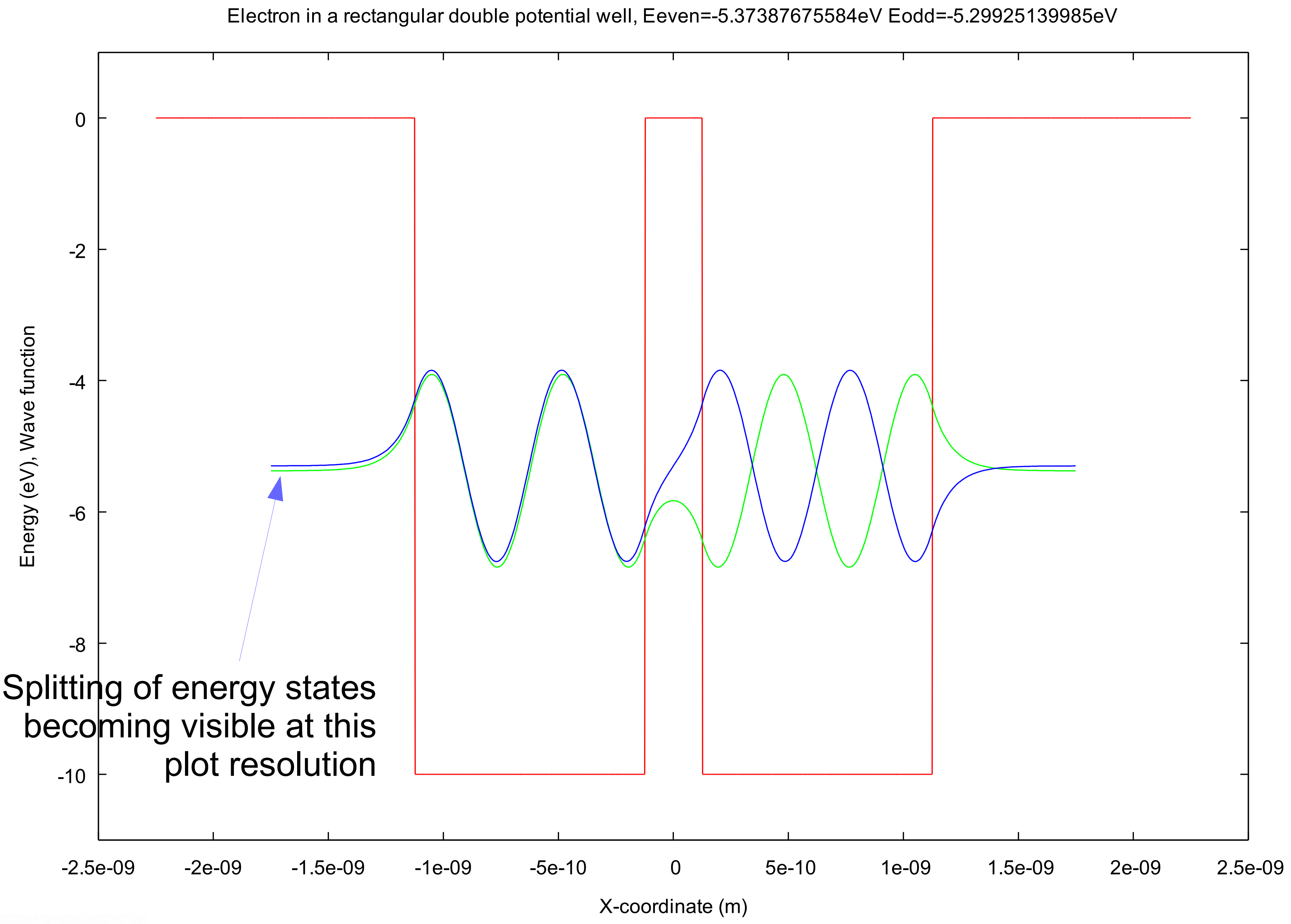
Next Energy States with 0.25μm Well Separation



Higher Energy States with 1μm Well Separation

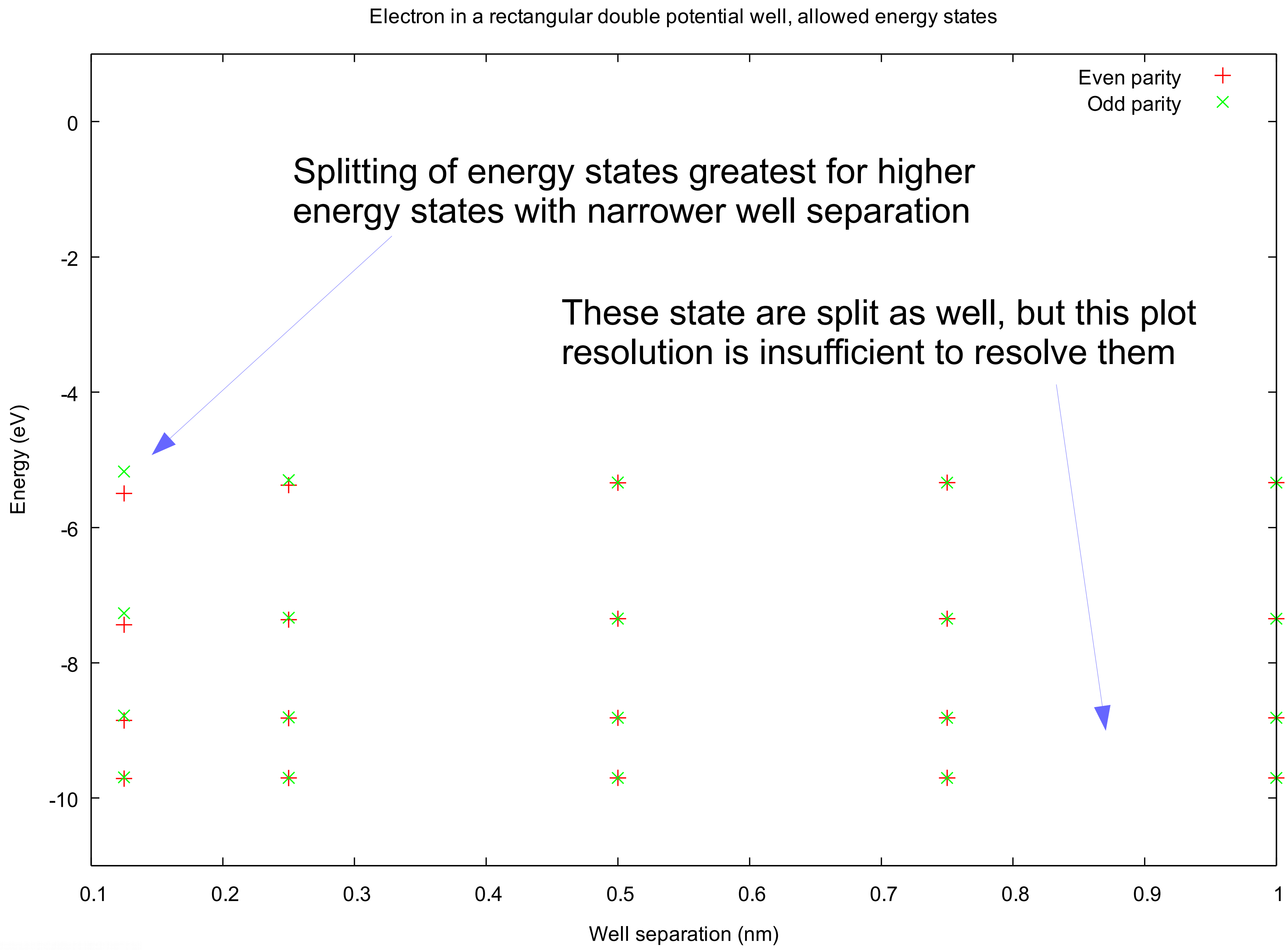


Higher Energy States with 0.25μm Well Separation



Splitting of energy states becoming visible at this plot resolution

Splitting of Energy States


Electron in a rectangular double potential well, allowed energy states
Even parity +
Odd parity ×
Splitting of energy states greatest for higher energy states with narrower well separation
These state are split as well, but this plot resolution is insufficient to resolve them
Energy (eV)
Well separation (nm)