# Numerically Solving for Eigenvalues and Eigenvectors of Schrödinger's Equation

Prof. Volker Crede

Department of Physics
Florida State University
April 2009

# Extra Credit Project: Energy Eigenvalues & Eigenvectors

- Schrödinger's Equation
  - Time-independent, one-dimensional

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

  - In units of $\hbar^2/m=1$

$$\frac{-1}{2}\frac{d^2\Psi}{dx^2}=[E-V(x)]\Psi$$

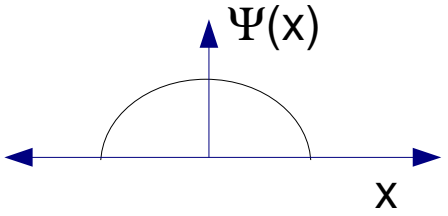
# Solving Schrödinger's Equation

- Numerical Procedure
  - Identical to any other $2^{nd}$ order ODE with known boundary conditions
    - Impliment standard 4th order Runge-Kutta method
  - Identical except for:
    - The Energy **E** is unknown
    - **Ψ(x)** must vanish as **x** becomes large
      - **Ψ(x)** must be normalizeable

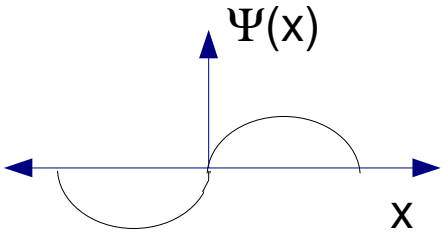
Boundary Condition

# Initial Boundary Conditions

- Exploit Symmetry
  - Symmetric Potential
    - Wavefunctions are Parity Eigenstates
      - Solutions are purely odd or purely even functions
  - Even Parity State Requires
    - **$\Psi(x)=\Psi(-x)$**
    - **$d\Psi(x=0)/dx = 0$**
    - **$\Psi(x=0) \neq 0$**
      - Choose **$\Psi(x=0)=1$** and renormalize later




  - Odd Parity State Requires
    - **$\Psi(x) = -\Psi(-x)$**
    - **$\Psi(x=0) = 0$**
    - **$d\Psi/dx(x=0) \neq 0$**



# Method of Solution

1) Pick a value of E

2) Propagate the solution to large x

   - Use 4$^{th}$ Order Runge Kutta method

3) Determine if boundary conditions match

   - If they do not
     - Adjust the value of E and try again

**Utilize zero-finding techniques!**

# Implementing RungeKutta()

```
#include<RungeKutta.h>

double F(int dim, double x, double Psi, double dPsi,       int
nparms, double parm[]) {
                    ...
}

void main() {
  int ndim=2;
  double *Psi = new double[ndim];
  int nparms = 2;
  double *parm = new double[nparms];
                    ...
  // Set initial conditions
                    ...
  // loop over dependent interval
  for (x=x_min; x<(x_min + h*msteps); x += h) {
     RungeKutta(ndim, F, Psi, x, h, nparms, parm);
     cout<<x<<" "<<Psi[0]<<" "<<Psi[1]<<endl;
  }
```

Include file

User must define

Library routine

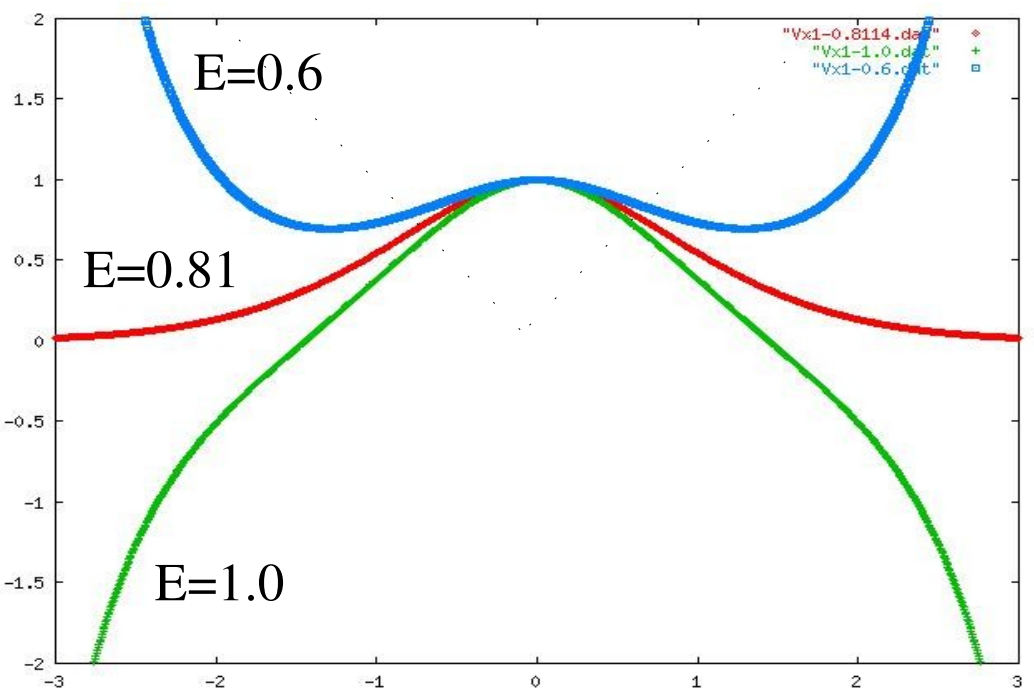
libRungeKutta.a

# Finding the Eigenvalues

```
while ( fabs(Psi[0]) > Psi_precision ) {
   init(Psi,sym);
   for(x=x_min; x<L; x += dx) {
     parm[1] = Vfun(x,useV);
     RungeKutta(ndim,f,Psi,x,dx,nparms,parm);
     if ( fabs(Psi[0]) > 2 )
       break; // Psi is diverging
   }
   if ( fabs(dE) < E_precision )
     break; // this is good enough

   if( Psi[0] > 0 ) // Psi@infinity
     divergence = +1;
   else
     divergence = -1;
   if (last_divergence*divergence < 0)
     dE = -dE/(double (2.0));
   parm[0] += dE;
   last_divergence = divergence;
}
```

$V(x) = |x|$



# Divergence & Precision

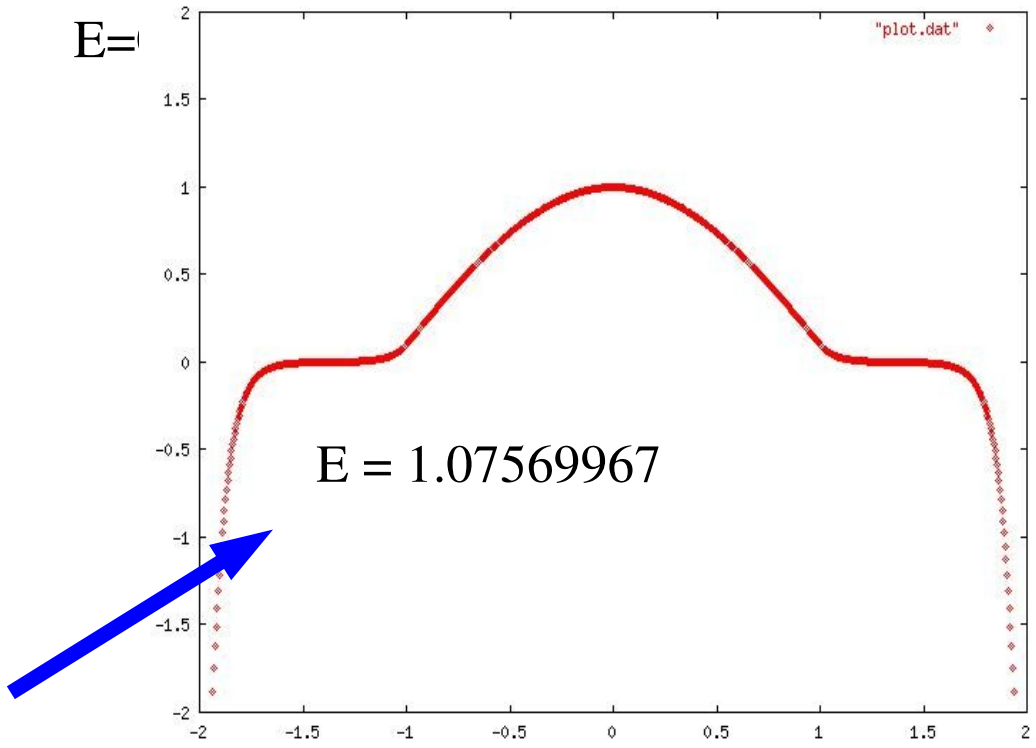
| E | Ψ (at large x) |
|---|---|
| 1 | 2.073079478 |
| 2 | 2.143708206 |
| 1.5 | -2.009692411 |
| ... | |
| 1.075195312 | -2.015385741 |
| 1.075439453 | 2.000674759 |
| 1.075683594 | 2.14053205 |
| 1.075561523 | -2.094250445 |
| 1.075622559 | 2.124420855 |
| 1.075592041 | -2.021076074 |
| 1.0756073 | 2.045394572 |
| 1.07559967 | -2.06349405 |

The eigen energy is 1.075

$V(x) = 100$ for $|x| > 1$

$V(x) = 0$ for $|x| \leq 1$

E=



# Extra Credit Project

## *Energy Eigenvalues & Eigenvectors of Schrödinger's Equation*

Using the procedures illustrated in the previous slides, implement a C++ class object for a WaveFunctionCalculator

Using your WaveFunctionCalculator, numerically solve Schrödinger's equation for the given potential energies.