Computational Physics Lab

Boundary Value Problems

Prof. Paul Eugenio
Department of Physics
Florida State University

Apr 18, 2019

Announcements

No More Homework Exercises!

Exam 2

- No collaborative work allowed
- Due Friday April 26

Energy Eigenvalues & Eigenvectors of Schrödinger's Equation

- 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
  - Similar to other 2nd order ODE with known boundary conditions

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

  - Expand to two 1st order ODE

$$(1)\quad \phi=\frac{d\psi}{dx}$$

$$(2)\quad \frac{d\phi}{dx}=2[V(x)-E]\psi$$

Solving Schrödinger's Equation

- Find Numerical solutions for $\phi(x)$ & $\psi(x)$

$$(1) \quad \frac{d\psi(x)}{dx} = \phi(x)$$

$$\boxed{\psi(x+h) = \psi(x) + h\cdot f_{\psi}(\psi,\phi,x)}$$

$$(2) \quad \frac{d\phi(x)}{dx} = 2[V(x)-E]\psi(x)$$

$$\boxed{\phi(x+h) = \phi(x) + h\cdot f_{\phi}(\psi,\phi,x)}$$

Where $f_{\psi}(\psi,\phi,x)$ and $f_{\phi}(\psi,\phi,x)$ are obtained using the 4th order Runge-Kutta method

Solving Schrödinger's Equation

- Find Numerical solutions for $\phi(x)$ & $\psi(x)$

$$(1) \quad \frac{d\psi(x)}{dx} = \phi(x)$$

$$(2) \quad \frac{d\phi(x)}{dx} = 2[V(x)-E]\psi(x)$$

- Implement standard 4th order Runge-Kutta method

- Similar as before except for:
  - The Energy **E** is unknown
  - Ψ**(x)** must vanish as **x** becomes large
    - Ψ**(x)** must be normalizable

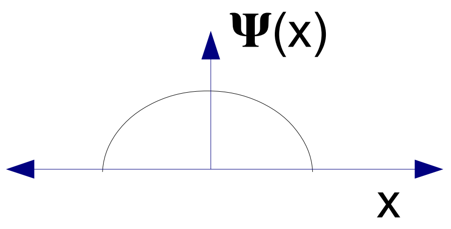
Boundary Conditions $\longrightarrow$

Initial State Conditions

- Exploit Symmetry
  - Symmetric Potential
    - Wave functions are Parity Eigenstates
      - solutions are purely odd or purely even functions

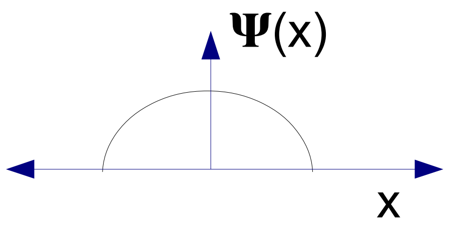
Initial State Conditions

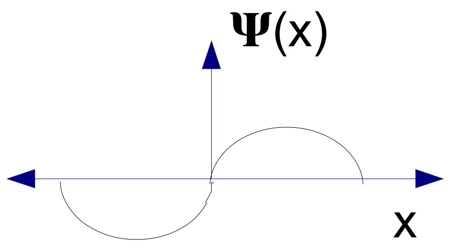
- Exploit Symmetry
  - Symmetric Potential
    - Wave functions are Parity Eigenstates
      - solutions are purely odd or purely even functions
  - Even Parity State Requires
    - **$\Psi(x) = \Psi(-x)$**
    - **$d\Psi(x=0)/dx = 0$**
    - **$\Psi(x=0) \neq 0$**
      - choose Ψ**(x=0) =1** and renormalize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



Initial State Conditions

- Exploit Symmetry
  - Symmetric Potential
    - Wave functions 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 Ψ**(x=0) =1** and renormalize later
  - Odd Parity State Requires
    - **$\Psi(x) = -\Psi(-x)$**
    - **$\Psi(x=0) = 0$**
    - **$d\Psi/dx(x=0) \neq 0$**





Procedure

1) Pick a value of E

2) Solve for the wave function out to large x

- Use 4th Order Runge Kutta method
- Solve for positive x values and use symmetry:

$$\Psi(-x) = \Psi(x) \quad \text{or} \quad \Psi(-x) = -\Psi(x)$$

3) Determine if boundary conditions match

i.e. $\Psi \to 0$ as $x \to \infty$

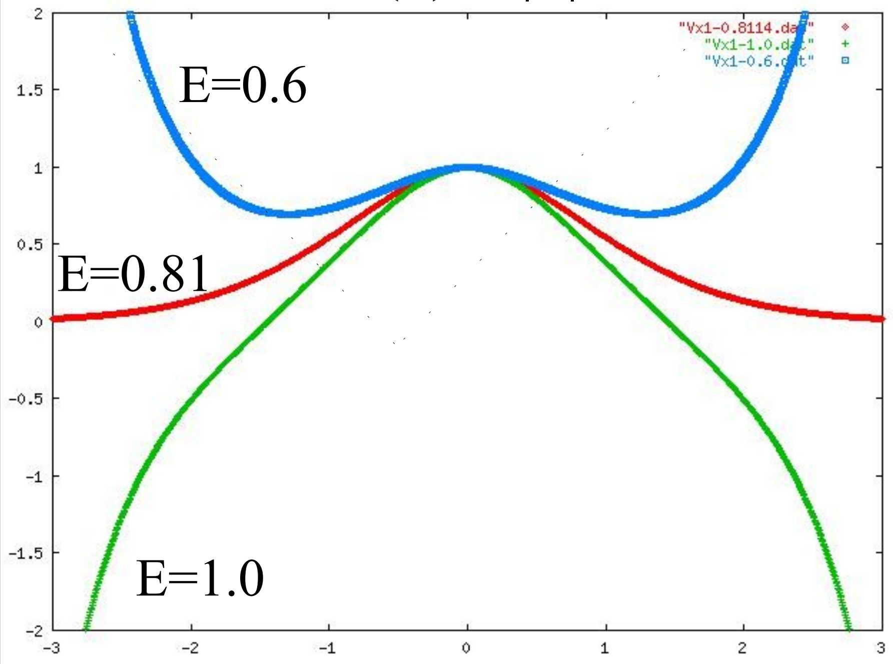
- If they do not
  - Adjust the value of E and try again

Utilize root-finding techniques!

Obtaining a Wave Function

```
def waveFunction(fcn, initialState, xValues, deltaX, E):
    """
     Solve for the wave function

    This routine uses the Runge-Kutta 4th order
    method to solve for the values for the
    wave function for a given value of the energy E

    It returns the wave function as an array of
    values with array length equal to len(xValues)

    Note: The wave function is not necessarily an
    eigenfunction.  This would only be true
    if the provided energy value "E" happens to be
    an eigenvalue.

    Parameters:
              . . .
    """

    # make a copy of the initial values
    # so that this function can be repeatedly
    # called with the same initial values
    s = np.copy( initialState )

    psi = []

    for x in xValues:
        psi += [s[0]]
         # evolve the state psi & dPsi/dX
        s += rungKutta4(fcn, s, x, deltaX, E)

    return np.array(psi,float)
```

$V(x) = |x|$

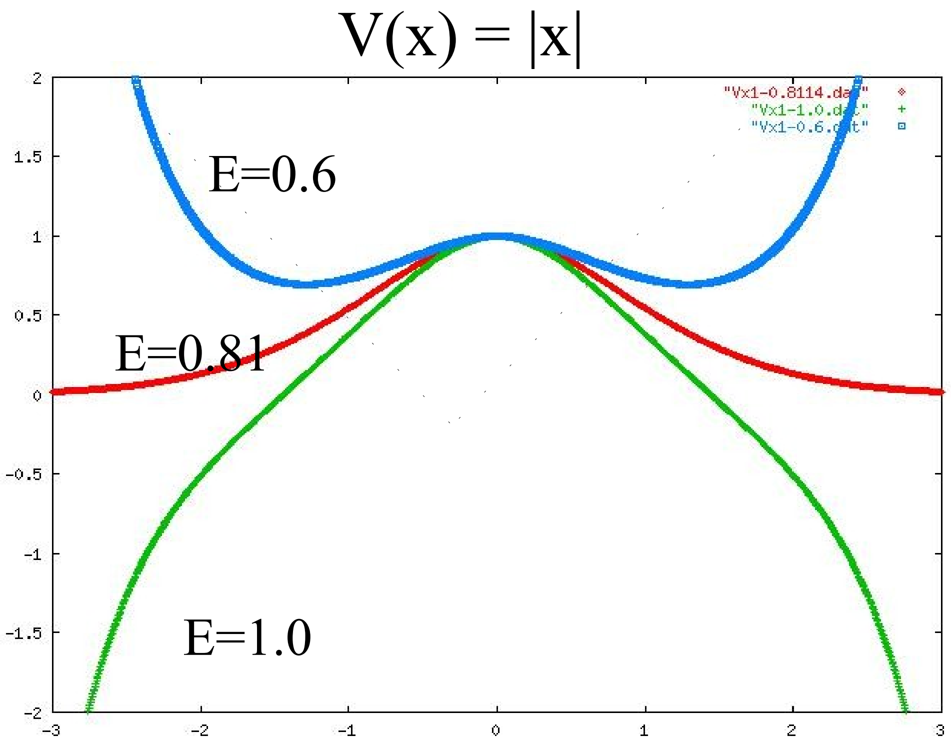


Wave function values at boundary diverges positively or negatively unless the energy is an eigenvalue.

This is like finding the roots of an equation.

Finding the Eigenvalues

Use root finding methods to find the eigenvalues.



```
#
# find eigenvalue using the secant
# root finding method

target = 1e-6

while numpy.abs( E1 - E2 ) > target:
    # obtain wave function for energy for E1
    psi = waveFunction(SchEq, initialState, xArray, deltaX, E1)

    # get the wave function value at the boundary
    psiEnd1 = psi[-1]

   # obtain wave function for energy for E2
    psi = waveFunction(SchEq, initialState, xArray, deltaX, E2)
    psiEnd2 = psi[-1]

   #use secant method to obtain new estimates
   # for the energy eigenvalue
    E1, E2 = E2, E2 - psiEnd2 * (E2 - E1) / (psiEnd2 - psiEnd1)

# We now have an eigenfunction and eigenvalue
#
print("Eigen Energy:",E2)
```

Computational limitations

Divergence & Precision

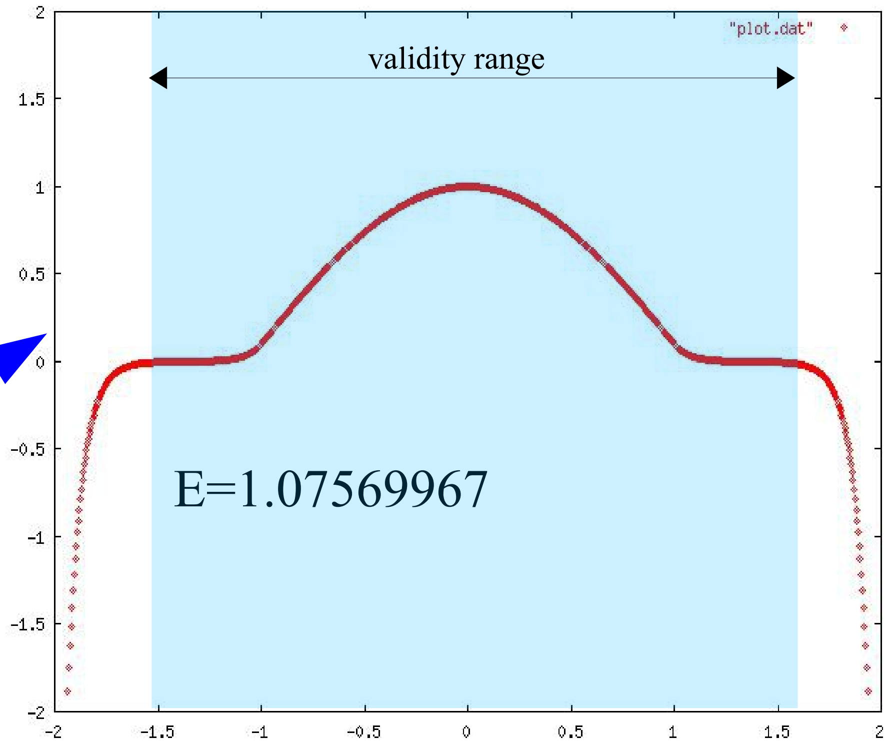
E	Ψ(at large x)
1	2.073079478
2	2.143708206
1.5	-2.009692411
...	
1.075195312	-2.015385741
1.075439453	2.000674759
1.075683594	2.14053205
1.075561523	-2.094250445
1.075622559	2.124420855
1.075592041	-2.021076074
1.0756073	2.045394572
1.07559967	-2.06349405

The energy eigenvalue is
1.075

Finite Square Well Potential

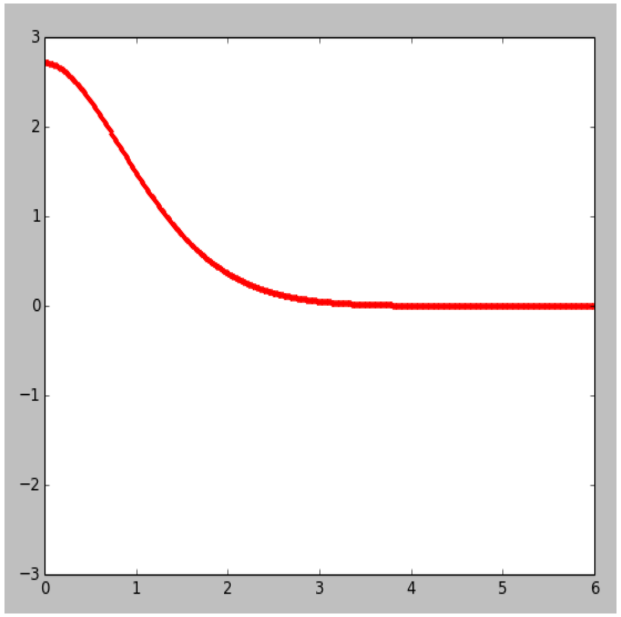
$V(x) = 100 \quad \text{for } |x| > 1$

$V(x) = 0 \quad \text{for } |x| \leq 1$



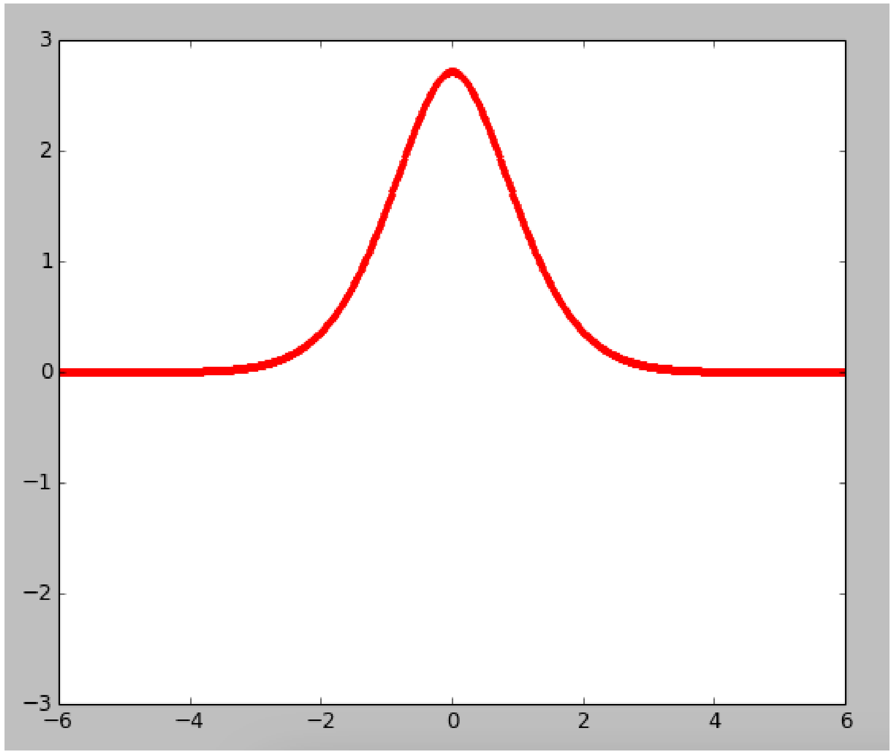
Only use array values in the VALID range

Solve for positive x and extend to negative x



```
# Extend arrays to negative values,
# ordering final array values from
# negative to positive
if parity == even:
    psi = np.append(psi[::-1],psi[1:])
else:
    psi = np.append(-psi[::-1],psi[1:])

x = np.append(-x[::-1],x[1:])
```

plot as points not lines



Normalizations & Expectation Values

$$\int_{-\infty}^{\infty} \psi^*(x)\,\psi(x)\,dx \;\simeq\; \textbf{psi.dot(psi) * deltaX}$$

```
# Normalize wavefunction
print("<psi|psi>:", psi.dot(psi) * deltaX )
Norm = np.sqrt( psi.dot( psi ) * deltaX )
print("Renormalizing by:", Norm)
psi = psi/Norm
print("<psi|psi>:", psi.dot( psi ) * deltaX )
```

```
<psi|psi>: 1.62723311703
Renormalizing by: 1.27563047825
<psi|psi>: 1.0
```

Normalizations & Expectation Values

$$\int_{-\infty}^{\infty} \psi^*(x)\,\psi(x)\,dx \simeq \textbf{psi.dot(psi) * deltaX}$$

```
# Normalize wavefunction
print("<psi|psi>:", psi.dot(psi) * deltaX )
Norm = np.sqrt( psi.dot( psi ) * deltaX )
print("Renormalizing by:", Norm)
psi = psi/Norm
print("<psi|psi>:", psi.dot( psi ) * deltaX )
```

```
<psi|psi>: 1.62723311703
Renormalizing by: 1.27563047825
<psi|psi>: 1.0
```

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} \psi^*(x)\,x^2\,\psi(x)\,dx \simeq \textbf{psi.dot(x*x * psi) * deltaX}$$

```
# Calculate expectation value <x^2>
print("<psi|x^2|psi>:", psi.dot( x*x * psi ) * deltaX )
```

```
<psi|x^2|psi>: 0.469491618962
```

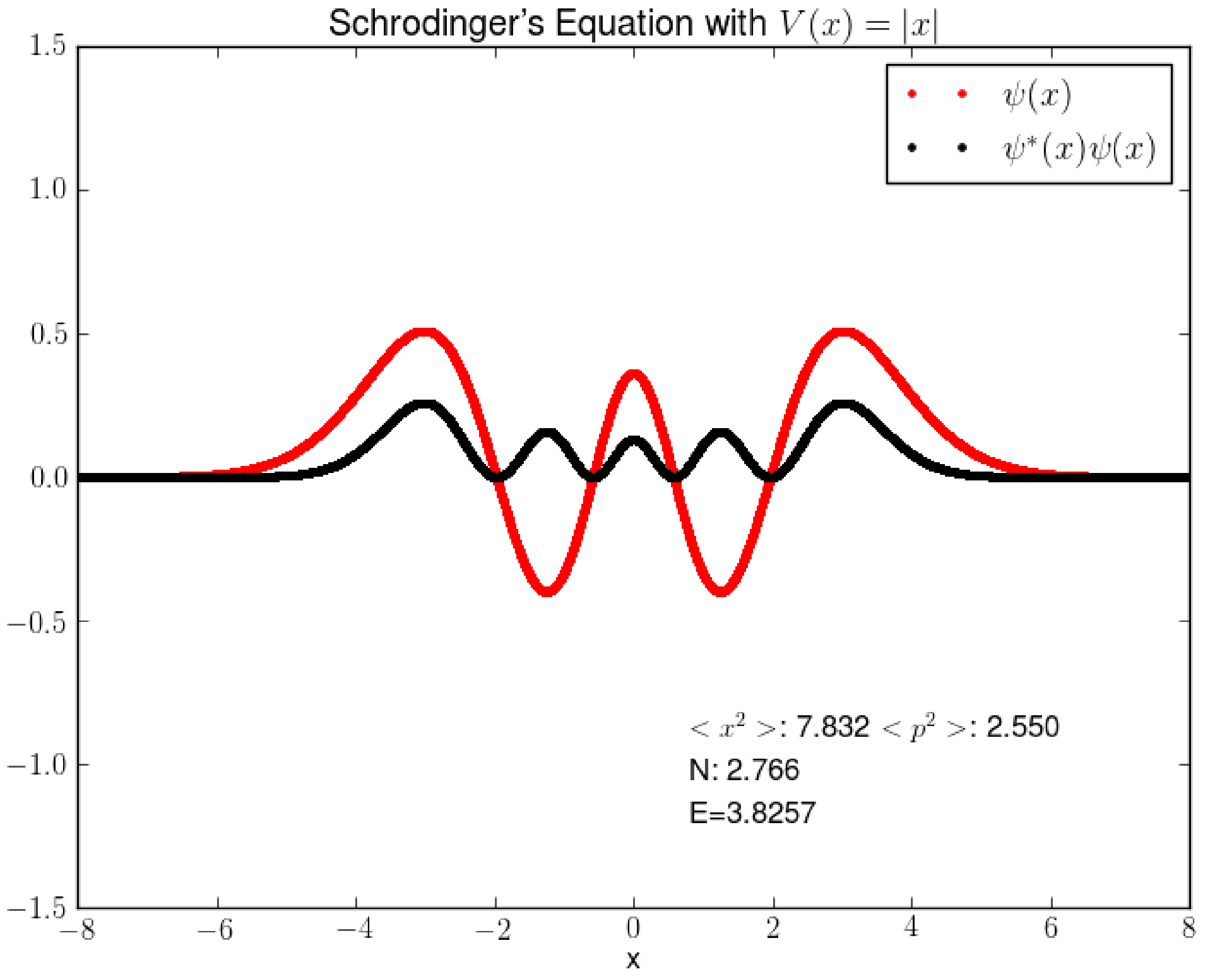
$<p^2>$ Expectation Value

$$\langle p^2 \rangle = \int_{-\infty}^{\infty} \psi^*(x)(-d^2/dx^2)\psi(x)\,dx$$

```
#      Operator[p^2] = - d^2/dx^2  (hbar=1)
#     <p^2> = integral[ psi*(-d^2psi/dx^2), dx]
#
# Use the central difference for 2nd derivative
#      f''(x) = [ f(x+dx) - 2f(x) + f(x-dx)]/dx**2

ppPsi = numpy.zeros( len(psi) - 2 )
for i in range( len(ppPsi) ):
    ppPsi[i] = -(psi[i+2] -2.0*psi[i+1] + psi[i]) / deltaX**2

# trim arrays to match the size of ppPsi
psi = psi[1:-1]
x = x[1:-1]

print("<psi|p^2|psi>:", psi.dot(ppPsi) * deltaX )

# using Schrodinger's Equation:
 p**2 * Psi = -d^2Psi/dX^2 = 2 *(E - V(x))*psi

for i in range( len(x) ):
    ppPsi[i] = 2.0*(E - V(x[i])) * psi[i]

print("<psi|p^2|psi>:", psi.dot(ppPsi) * deltaX )
```

not necessarily the same

```
<psi|p^2|psi>: 3.535532
<psi|p^2|psi>: 3.535533
```



Schrodinger's Equation with $V(x) = |x|$
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−8
−6
−4
−2
0
2
4
6
8
x
$\psi(x)$
$\psi^*(x)\psi(x)$
$< x^2 >$: 7.832 $< p^2 >$: 2.550
N: 2.766
E=3.8257

Final Exercise: Mini-Exam 2

Energy Eigenvalues & Eigenvectors of Schrödinger's Equation

Using the procedures illustrated in the previous slides, implement a program to solve Schrödinger's equation.

Using your WaveFunction program numerically solve Schrödinger's equation for several given potential energies.

Due Friday Apr 26