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 2\textsuperscript{nd} order ODE with known boundary conditions

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

Expand to two 1\textsuperscript{st} order ODE

(1) \[\phi = \frac{d\psi}{dx}\]

(2) \[\frac{d\phi}{dx} = 2[V(x) - E]\psi\]
Solving Schrödinger's Equation

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

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

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

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

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

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 4\textsuperscript{th} order Runge-Kutta method

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

Boundary Conditions
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 $\Psi(x=0) = 1$ and renormalize later
Initially 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 $\Psi(x=0) = 1$ and renormalize later
  - Odd Parity State Requires
    - $\Psi(x) = -\Psi(-x)$
    - $\Psi(x=0) = 0$
    - $d\Psi/dx(x=0) \neq 0$
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. \( \Psi \rightarrow 0 \) as \( x \rightarrow \infty \)
   - If they do not
     - Adjust the value of E and try again

Utilize root-finding techniques!
def waveFunction(fcn, initialState, xValues, deltaX, E):
    
    Solve for the wave function

    This routine uses the Runge-Kutta 4\textsuperscript{th} 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)

This is like finding the
roots of an equation.
# 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)
    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)

---

Finding the Eigenvalues

Use root finding methods to find the eigenvalues.

\[ V(x) = |x| \]

E = 0.6
E = 0.81
E = 1.0
Computational limitations

Divergence & Precision

Finite Square Well Potential

\[ V(x) = 100 \quad \text{for} \quad |x| > 1 \]
\[ V(x) = 0 \quad \text{for} \quad |x| \leq 1 \]

<table>
<thead>
<tr>
<th>( E )</th>
<th>( \Psi(\text{at large } x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.073079478</td>
</tr>
<tr>
<td>2</td>
<td>2.143708206</td>
</tr>
<tr>
<td>1.5</td>
<td>-2.009692411</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1.075195312</td>
<td>-2.015385741</td>
</tr>
<tr>
<td>1.075439453</td>
<td>2.000674759</td>
</tr>
<tr>
<td>1.075683594</td>
<td>2.14053205</td>
</tr>
<tr>
<td>1.075661523</td>
<td>-2.094250445</td>
</tr>
<tr>
<td>1.075622559</td>
<td>2.124420855</td>
</tr>
<tr>
<td>1.075592041</td>
<td>-2.021076074</td>
</tr>
<tr>
<td>1.0756073</td>
<td>2.045394572</td>
</tr>
<tr>
<td>1.07559967</td>
<td>-2.06349405</td>
</tr>
</tbody>
</table>

The energy eigenvalue is 1.075

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
\[ \int_{-\infty}^{\infty} \psi^*(x) \psi(x) \, dx \approx \text{psi.dot( psi )} \times \text{deltaX} \]

# 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)
Normalizations & Expectation

Values

\[ \int_{-\infty}^{\infty} \psi^*(x) \psi(x) dx \approx \text{psi.dot( psi )} \times \text{deltaX} \]

# 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 )

\[ \int_{-\infty}^{\infty} \psi^*(x) x^2 \psi(x) dx \approx \text{psi.dot( x^2 * psi )} \times \text{deltaX} \]

# Calculate expectation value <x^2>
print("<\psi|x^2|\psi>: ", psi.dot( x^2 * 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 \approx \text{psi.dot( x^2 * psi )} \times \text{deltaX} 

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

<\psi|x^2|\psi>: 0.469491618962
\[ \langle p^2 \rangle = \int_{-\infty}^{\infty} \psi^*(x)(-\frac{d^2}{dx^2})\psi(x)\,dx \]

# Operator\[p^2\] = \(-\frac{d^2}{dx^2}\) (hbar=1)
# \(\langle p^2 \rangle\) = integral\[ \psi^*(-\frac{d^2}{dx^2})\psi,\,dx\]
#
# Use the central difference for 2nd derivative
# \(f''(x) = \frac{[f(x+dx) - 2f(x) + f(x-dx)]}{dx^2}\)

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

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

print("\langle psi|p^2|psi\rangle: ", psi.dot(ppPsi) * deltaX)

# using Schrodinger's Equation:
\(p^2 \Psi = -\frac{d^2\Psi}{dX^2} = 2(E - V(x))\psi\)

for \(i\) in \(\text{range}(\text{len}(x))\):
    \(ppPsi[i] = 2.0*(E - V(x[i])) \times psi[i]\)

print("\langle psi|p^2|psi\rangle: ", psi.dot(ppPsi) * deltaX)

\langle psi|p^2|psi\rangle: 3.535532
\langle psi|p^2|psi\rangle: 3.535533

not necessarily the same
Schroedinger’s Equation with $V(x) = |x|$
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