DOING PHYSICS WITH PYTHON

QUANTUM MECHANICS

EIGENSTATES OF A PARTICLE BOUND IN A POTENTIAL WEL

Ian Cooper

matlabvisualphysics@gmail.com

DOWNLOAD DIRECTORY FOR PYTHON SCRIPTS

qm040.py

Solution of the [1D] Schrodinger equation by finding the eigenvalues and eigenvectors for an electron confined to a region of space by a square potential well.

GitHub

Google Drive

References

Operators, expectation values, Heisenberg Uncertainty Principle

Transverse standing waves

First and Second derivative operators

INTRODUCTION

We will consider a system of an electron confined to a [1D] region of space by a potential well. The top of the well corresponds to the zero of the potential energy function and the bottom of well has a negative energy value.

The [1D] Schrodinger equation for our system is

$$\hat{H}\,\psi_n(x) = E_n\ \psi_n(x) \qquad E_n < 0 \qquad n = 1,\ 2,\ 3,\ \ldots$$

where $\hat{H}$ is the Hamiltonian operator, $\psi_n(x)$ is the n^{th} eigenvector (eigenfunction) and E_n is the corresponding eigenvalue. The eigenvectors form a complete set. Any wavefunction can be expressed as a linear combination of the eigenvectors.

The Hamiltonian operator $\hat{H}$ depends upon the kinetic energy operator $\hat{K}$ and the potential energy $\hat{U}$ of the system

$$\hat{H} = \hat{K} + \hat{U} \qquad \hat{K} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}$$

So, the Schrodinger equation can be written as

$$\left(\hat{K} + \hat{U}\right)\psi_n(x) = E_n\ \psi_n(x)$$

This is an eigenvalue problem, and Python can be used to solve the Schrodinger equation by finding the eigenvalues and eigenvectors

for the bound electron. In computing the solution of the Schrodinger equation, operators are represented by matrices and the eigenvectors by vectors. For *N* grid points, the Schrodinger equation in matrix form is

$$(\mathbf{K}+\mathbf{U})\,\psi_n = E_n\ \psi_n$$

where **K** and **U** are *N*x*N* matrices and ψ_n is a column vector (1x*N*).

All the elements of the matrix **U** are zero except that the *N* diagonal elements that equal the *N* elements of the potential energy function *U*(*x*). For a square potential well of depth U0 and width w, the Code for the U matrix is

```
U = zeros(N)
U[x>-w/2] = U0
U[x>w/2] = 0
UM = diag(U)
```

The operator $\hat{K}$ has a term for the second derivative and its matrix can be written as

$$\frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{pmatrix}$$

Warning: the second derivative matrix only operates on the *N* internal x grid points and not as the end points of x domain. The boundary conditions are that the two end eigenvector elements for ψ_n are zero at the end points of the x domain.

For calculations, S.I. units are used: energy [J] and positions [m]. However, for display purposes energies are given in eV and positions in nm

```
se = e          # Energy scaling factor   J <---> ev
sx = 1e-9       # Length scaling factor   m <---> nm
```

The Code for the kinetic energy matrix and Hamiltonian matrix is

```
# AM (second derivative), KM (kinetic energy), HM (Hamiltonian) matrices
    Cse = -hbar**2/(2*me)
    off = ones(N-1)
    AM = (-2*np.eye(N) + np.diag(off,1) + np.diag(off,-1))/(dx**2)
    KM = Cse*AM
    HM = KM + UM
```

The solutions of the Schrodinger equation are the eigenvalues ev and the eigenvectors ef and can be found using the Code

```
    ev, ef = eigsh(HM, which="SM", k = M)
```

k = M returns *M* eigenvalues. *M* has to be larger enough to return the most negative eigenvalues.

which="SM" the eigenvalues are sorted from smallest to largest values.

For the bound electron, only the negative eigenvalues are relevant. The energy eigenvalues in eV are determined from the Code

```
E = ev[ev<0]/se              # negative eigenvalues [eV]
```

The eigenvectors are normalized (probability of finding the electron in the *x* domain must be one).

```
psi = zeros([N,len(E)]); psi2 = zeros([N,len(E)])
for c in range(len(E)):
    psi[:,c] = ef[:,c]
    psi2[:,c] = psi[:,c]**2
    area = simps(psi2[:,c],x)
    psi[:,c] = psi[:,c]/sqrt(area)
probD = psi**2    # probability density [1/m]
```

Once we know the normalized eigenvectors, we can calculate the expectation values for position and its uncertainty, momentum and its uncertainty, total energy, kinetic energy and potential energy. Then we can test the Heisenberg Uncertainty Principle

$$\Delta x\ \Delta p \geq \frac{\hbar}{2} \qquad HCP = \frac{2\,dX\ dY}{\hbar} \geq 1$$

In the Code **qm040.py**, I have not included the eigenvector elements at the boundaries xMin and xMax. This introduces a small error in the values of the expectation values. This can be corrected by including the end points and thus the length of the x and psi arrays would be N+2.

```
#%% EXPECTATION VALUE CALCULATIONS
def firstDer(N,dx):
    v  = ones(N-1)
    M1 = diag(-v,-1)
    M2 = diag(v,1)
    M = M1+M2
    M[0,0] = -2; M[0,1] = 2; M[N-1,N-2] = -2; M[N-1,N-1] = 2
    MF = M/(2*dx)
    return MF

def secondDer(N,dx):
     v = -2*ones(N)
```

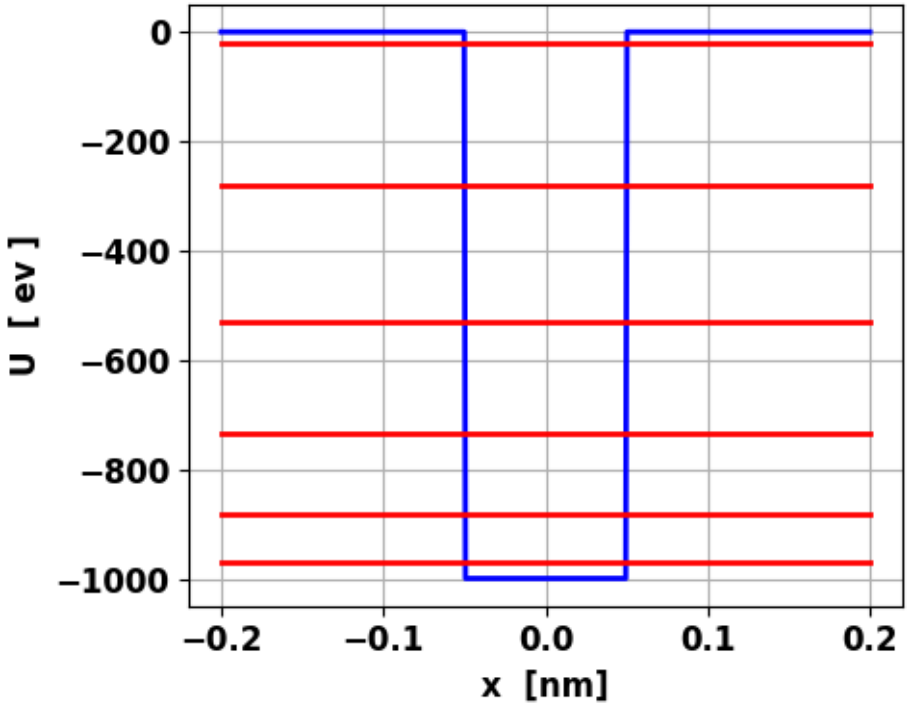
```
    M1 = np.diag(v)
    v = np.ones(N-1)
    M2 = np.diag(v,1)
    M3 = np.diag(v,-1)
    M = M1+M2+M3
    M[0,0] = 1; M[0,1] = -2; M[0,2] = 1
    M[N-1,N-3] = 1; M[N-1,N-2] = -2; M[N-1,N-1]=1
    MS = M/(dx**2)
    return MS

y = psi[:,1]      # eigenfunction n
# Probability
fn = y**2
prob = simps(fn,x)
# Position and its uncertainty
fn = y*x*y
x_avg = simps(fn,x)              # [nm]
fn = y*x**2*y
x2_avg = simps(fn,x)             # [nm*nm]
dX = sqrt(x2_avg - x_avg**2)     #  [m]

# Momentum and its uncertainty
y2 = firstDer(N,dx)@y
fn = y*y2
```

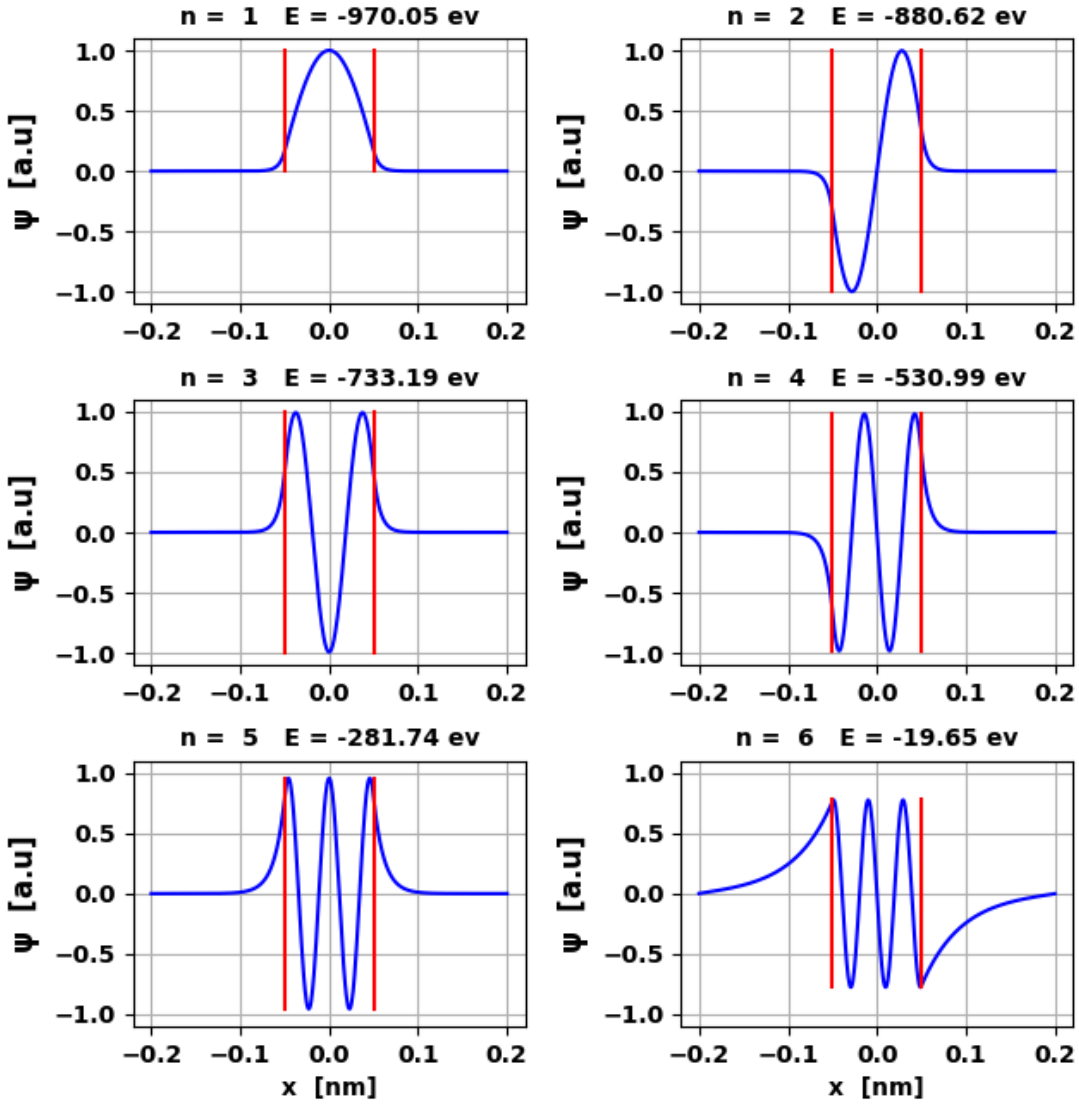
```
p_avg = -1j*hbar*simps(fn,x)                # [N.s]
y2 = secondDer(N,dx)@y     # Second derivative matrix x [m]
fn = y*y2                  # Second derivative of function y
p2_avg = -hbar**2*simps(fn,x)   # [N^2.S^2]
dP = sqrt(p2_avg - imag(p_avg)**2)       # [N.s]

# Heisenberg Uncertainty Principle
HUP = 2*abs(dX*dP/hbar)

# Potential energy  [ev]
fn = y*U*y
U_avg = simps(fn,x)/se
# Kinetic energy [ev]
K_avg = p2_avg/(2*me)/se
# Total energy [ev]
E_avg = K_avg + U_avg
```

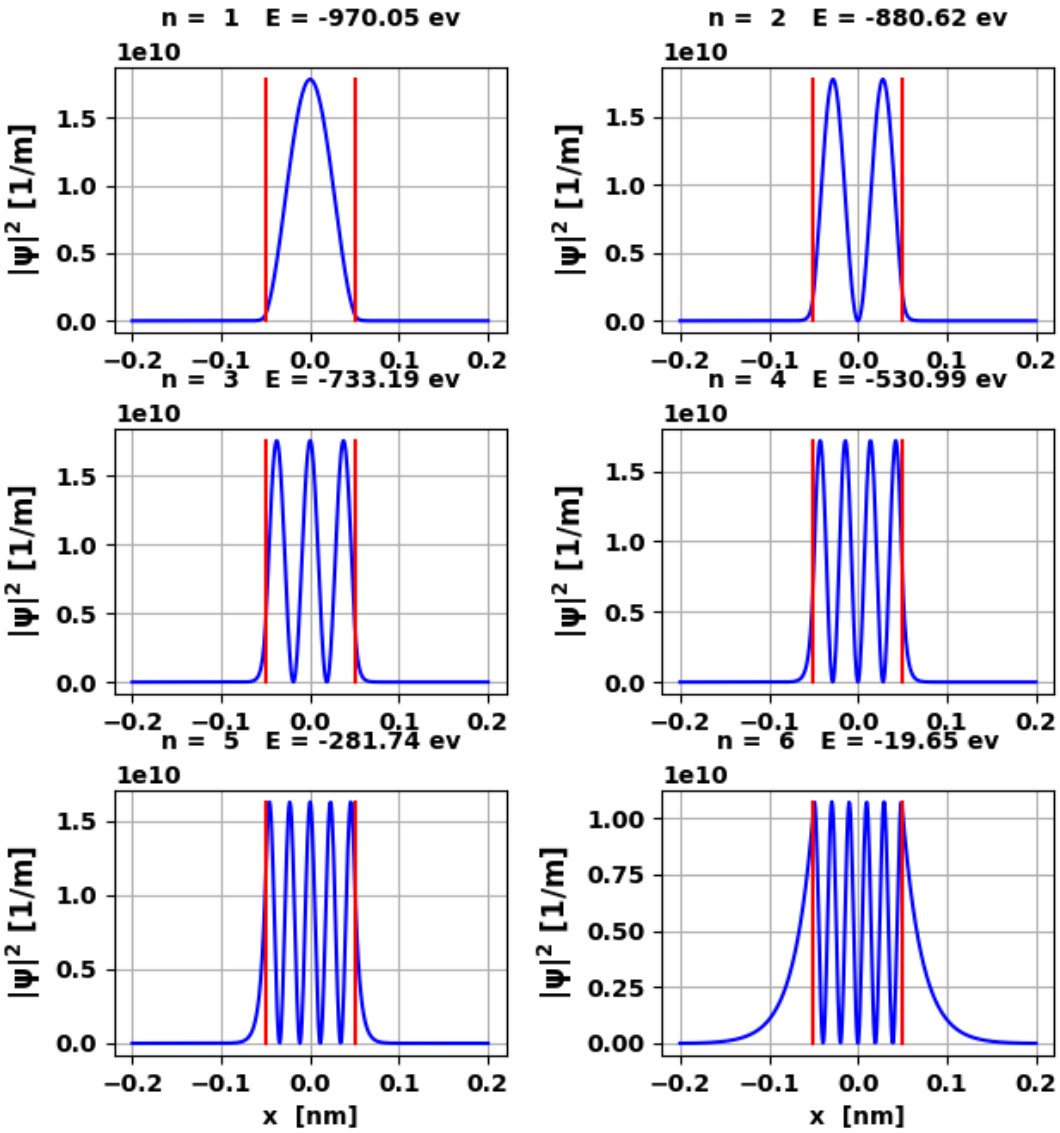
SQUARE WELL

The Code **qm040.py** solves the Schrodinger equation for a square potential well. The results of the simulation are displayed in the Console Window and in Plot Windows.

```
grid point N = 519   eigenvalues returned M = 30
xMin = -0.20 nm   xMax = 0.20 nm
well width w = 0.10 nm
well depth U0 = -1000 ev

Energy eigenvalues [ev]
  E1 = -970.048
  E2 = -880.624
  E3 = -733.192
  E4 = -530.987
  E5 = -281.740
  E6 = -19.645

Eigenstate n =  1
 Expectation values and Uncertainty Principle
   <x> = 0.00 m   deltax dX = 3.00e-11 m
   <p> = 0.00 N.s   deltax dP = 5.56e-24 m
   HUP = 3.17  > 1
  Eigenstate energies
   En = -880.62 eV
   <E> = -880.62 <K> =  106.02 <U> = -986.65
   <K> + <U> = -880.62

Execution time =  4 s
```


0
−200
−400
−600
−800
−1000
U [ev]
−0.2
−0.1
0.0
0.1
0.2
x [nm]


n = 1 E = -970.05 ev
n = 2 E = -880.62 ev
n = 3 E = -733.19 ev
n = 4 E = -530.99 ev
n = 5 E = -281.74 ev
n = 6 E = -19.65 ev
ψ [a.u]
x [nm]
1.0
0.5
0.0
−0.5
−1.0
−0.2
−0.1
0.0
0.1
0.2



It is an easy task to modify the Code to simulate a wide variety of potential energy functions. If you change the well parameters, then it may be necessary to modify the Code for the results displayed in the Console and Plot windows,