In [3]:

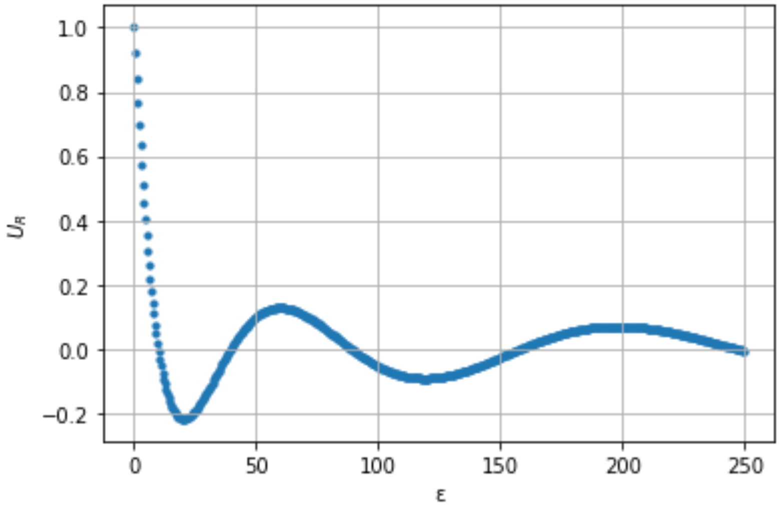
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress
import pandas as pd

# 2(a)


def RK4(func, X0, tmin, tmax, N):
    h = (tmax-tmin)/N
    t = np.linspace(tmin, tmax, N+1)
    X = np.zeros([N+1, len(X0)])
    X[0] = X0
    for i in range(N):
        k1 = func(t[i], X[i])
        k2 = func(t[i] + h/2, X[i] + (h * k1)/2)
        k3 = func(t[i] + h/2, X[i] + h/2 * k2)
        k4 = func(t[i] + h, X[i] + h * k3)
        X[i+1] = X[i] + h / 6. * (k1 + 2*k2 + 2*k3 + k4)
    return X, t


def simps(x, y):
    h = (x[-1]-x[0])/len(x)
    integral = (h/3)*(2*np.sum(y[2:-2:2]) + 4*np.sum(y[1:-1:2]) + y[0] + y
[-1])

    return integral


# 2(a) i.------------------------------------------------------------
e = np.arange(0, 250, 0.5)

index, index_, u_r = [], [], []

for i in e:
    def func(x, Y):
        y, y1 = Y  # y1 is the first order derivative
        f1 = y1
        f2 = -(i)*y
        return np.array([f1, f2])

    x, t = RK4(func, [0, 1], -1/2, 1/2, 100)
    u = x[:, 0][-1]
    u_r.append(u)

for i in range(1, len(e)):
    if u_r[i-1]*u_r[i] < 0:
        index.append(i-1)
        index_.append(i)

plt.scatter(e, u_r, s=10)
plt.xlabel('ε')
plt.ylabel(r'$U_R$')
```

```
plt.grid()
plt.show()

# 2(a) ii.
print('indices of energy vector:', index)
print('the corresponding values of energies:', [e[i] for i in index])

# 2(a) iii.
sec_e = []

for i, j in zip(index, index_):
    # guessing the value of 'e' using secant method
    approx_e = e[j] - u_r[j]*((e[j]-e[i])/(u_r[j]-u_r[i]))
    sec_e.append(approx_e)

print('the final energy eigen values:', sec_e)

n, j = 1, 2
for i in sec_e:

    def func(x, Y):
        y, y1 = Y  # y1 is the first order derivative
        f1 = y1
        f2 = -(i)*y
        return np.array([f1, f2])

    if n % 2 == 0:  # the func is even
        if n == 2:
            inc = -1
        else:
            inc = 1

        x, t = RK4(func, [0, inc], -1/2, 1/2, 100)
        wfn = x[:, 0]  # the wavefunction
        # the normalized wave function
        nwfn = wfn/(np.sqrt(simps(t, wfn**2)))

        ana = np.sin(np.pi*t*n)  # analytical solution
        # normalised analytical solution
        n_ana = ana/(np.sqrt(simps(t, ana**2)))

    else:  # the func is odd
        if n == 3:
            inc = -1
        else:
            inc = 1

        x, t = RK4(func, [0, inc], -1/2, 1/2, 100)
        wfn = x[:, 0]  # the wavefunction
        # the normalized wave function
        nwfn = wfn/(np.sqrt(simps(t, wfn**2)))

        ana = np.cos(np.pi*n*t)  # analytical solution
        # normalized analytical solution
        n_ana = ana/(np.sqrt(simps(t, ana**2)))
```

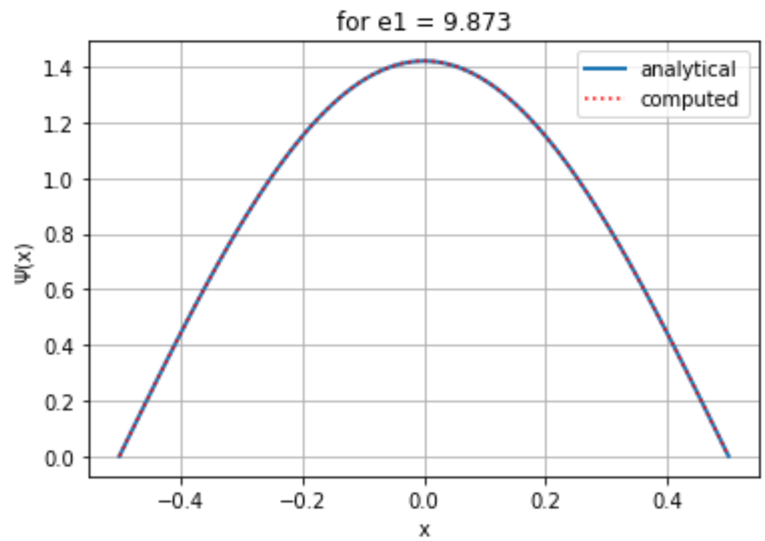
```
    plt.plot(t, n_ana, label='analytical', linewidth='2')
    plt.plot(t, nwfn, label='computed', ls='dotted', color='red')
    plt.xlabel('x')
    plt.ylabel('Ψ(x)')
    plt.title(f'for e{n} = {round(i,3)}')
    plt.grid()
    plt.legend()
    plt.show()

    j += 1
    n += 1

# 2(b) -----------------------------------------------------------------

n = np.array([1, 2, 3, 4, 5])

slope = linregress(n**2, sec_e)[0]
intercept = linregress(n**2, sec_e)[1]

sec_ey = slope*(n**2) + intercept  # fitted points

plt.scatter(n**2, sec_e)
plt.plot(n**2, sec_ey)
plt.xlabel(r'$n^2$')
plt.ylabel(r'$e_n$')
plt.title(r'$e_n$ as a function of $n^2$')
plt.grid()
plt.show()

print('slope:', slope)

# 2(c) -----------------------------------------------------------------

n, j = 1, 8
for i in sec_e:

    def func(x, Y):
        y, y1 = Y  # y1 is the first order derivative
        f1 = y1
        f2 = -(i)*y
        return np.array([f1, f2])

    if n % 2 == 0:  # the func is even
        if n == 2:
            inc = -1
        else:
            inc = 1

        x, t = RK4(func, [0, inc], -1/2, 1/2, 100)
        wfn = x[:, 0]  # the wavefunction
        # the normalized wave function
        nwfn = wfn/(np.sqrt(simps(t, wfn**2)))

        ana = np.sin(np.pi*t*n)  # analytical solution
        # normalised analytical solution
        n_ana = ana/(np.sqrt(simps(t, ana**2)))
```

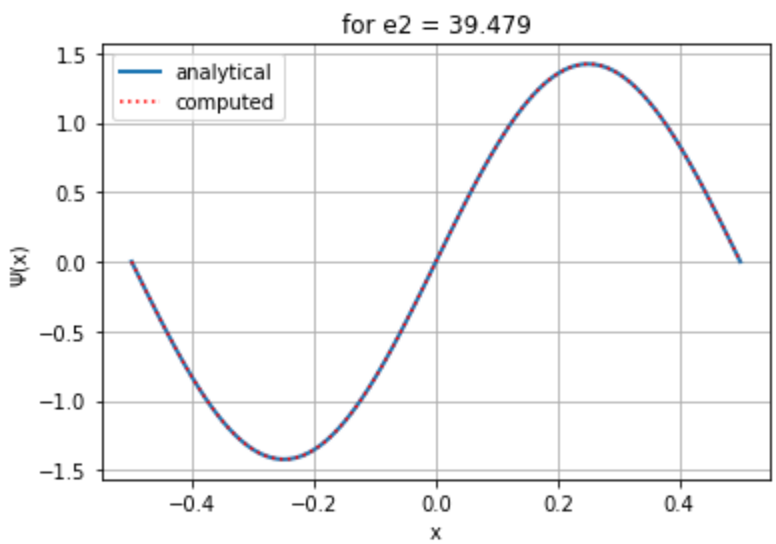
```
    else:  # the func is odd
        if n == 3:
            inc = -1
        else:
            inc = 1

        x, t = RK4(func, [0, inc], -1/2, 1/2, 100)
        wfn = x[:, 0]  # the wavefunction
        # the normalized wave function
        nwfn = wfn/(np.sqrt(simps(t, wfn**2)))

        ana = np.cos(np.pi*n*t)  # analytical solution
        # normalized analytical solution
        n_ana = ana/(np.sqrt(simps(t, ana**2)))

    plt.plot(t, n_ana**2, label='analytical', linewidth='2')
    plt.plot(t, nwfn**2, label='computed', ls='dotted', color='red')
    plt.xlabel('x')
    plt.ylabel(r'$Ψ(x)^2$')
    plt.title(f'probability density plot for e{n} = {round(i,3)}')
    plt.grid()
    plt.legend()
    plt.show()

    j += 1
    n += 1

# 2(d) -----------------------------------------------------------------

n = np.array([1, 2, 3, 4, 5])
e = 1.6*(10**(-19))  # charge on an electron
h_cut = 1.054571817*(10**(-34))  # in Js
m_e = 9.1*(10**(-31))  # mass of electron in kgs
m_p = 1.67262192 * (10**(-27))  # mass of proton in kgs
L1 = 5*(10**(-10))  # width of the well in meters (angstrom to m)
L2 = 10*(10**(-10))
L3 = 5*(10**(-15))  # fermi meter to m
sec_e = np.array(sec_e)


def table(m, L):
    # divided by e for converting into eVs
    cal_E = sec_e*(h_cut**2)/(e*2*m*(L**2))
    ana_e = (n**2)*(np.pi**2)*(h_cut**2)/(e*2*m*(L**2))

    table_ = pd.DataFrame(
        {'computed E': np.round(cal_E, 6), 'analytical E': np.round(ana_e,
6)})

    return table_


print('energies in eV of an electron trapped in a well of 5 Å')
print(table(m_e, L1))
```

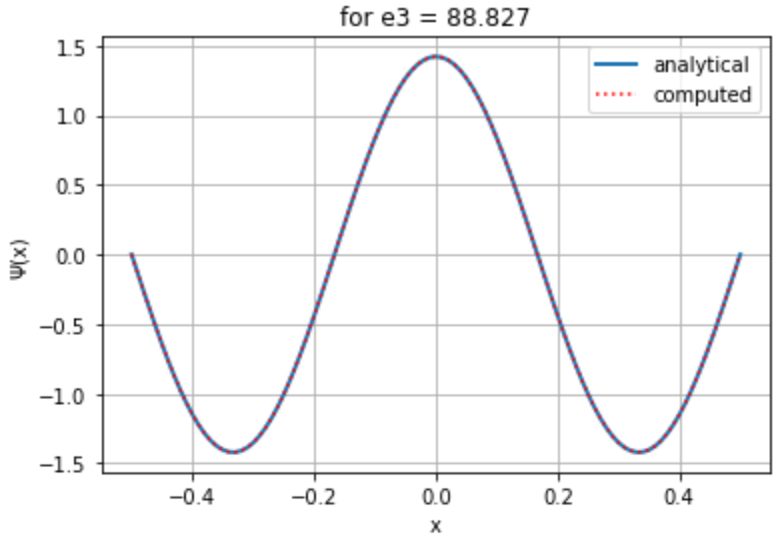
```
# 2(e)-----------------------------------------------------------------------

print('energies in eV of an electron trapped in a well of 10 Å')
print(table(m_e, L2))
print('energies in eV of a proton trapped in a well of 5 fm')
print(table(m_p, L3))

# 2(f) -----------------------------------------------------------------------


def funcn(x, Y):
    y, y1 = Y
    f1 = y1      # y1 is the first order derivative
    f2 = -(1)*y  # e=1 for ground state
    return np.array([f1, f2])


x, t = RK4(funcn, [0, 1], -1/2, 1/2, 100)

u = x[:, 0]  # the wavefunction
u_ = x[:, 1]  # derivative of the wavefunction

exp_x = simps(t, (u**2)*t)  # expectation value of x
exp_x2 = simps(t, (u**2)*(t**2))  # expectation value of x^2

exp_p = simps(t, u*u_)  # expectation value of momentum
exp_p2 = simps(t, u*(u_**2))  # expectation value of momentum^2

var_x = exp_x2 - (exp_x**2)
var_p = exp_p2 - (exp_p**2)

# verifying the uncertainity principle
print('Δx Δp =', var_x*var_p)
print('h/4π =', h_cut/2)
```

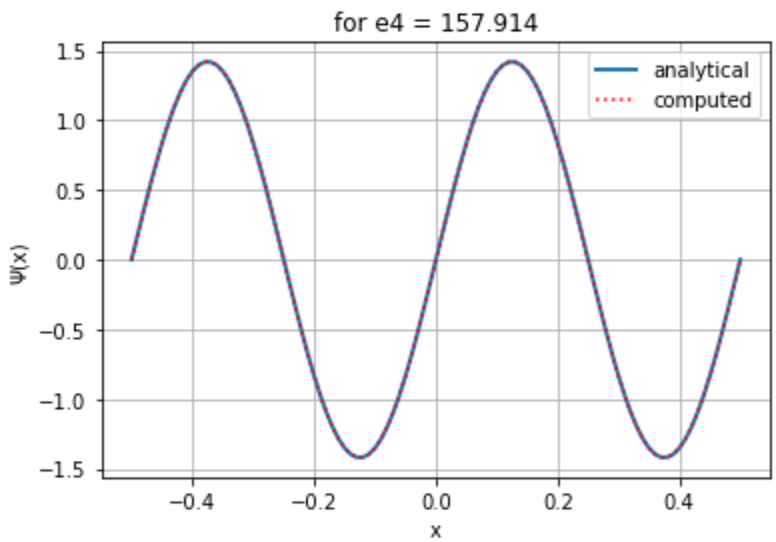


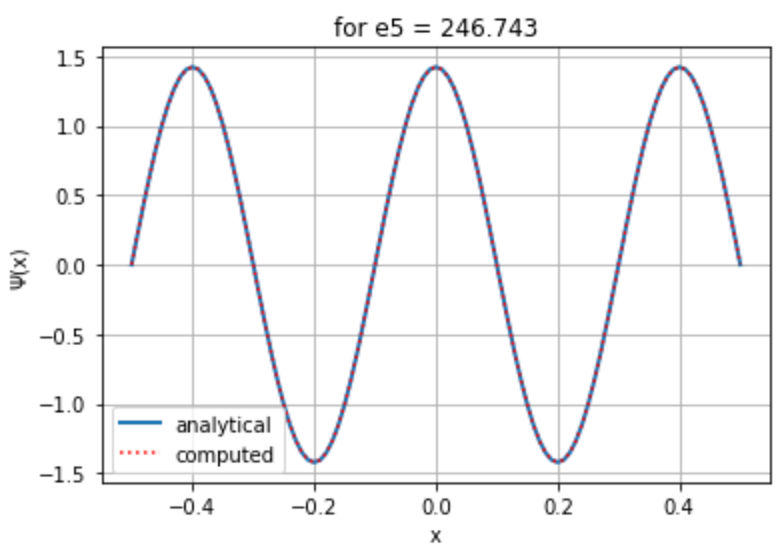

```
indices of energy vector: [19, 78, 177, 315, 493]
the corresponding values of energies: [9.5, 39.0, 88.5, 157.5, 246.5]
the final energy eigen values: [9.873225730254942, 39.47861914121265, 88.82
703027446989, 157.91448858562188, 246.7427816632612]
```

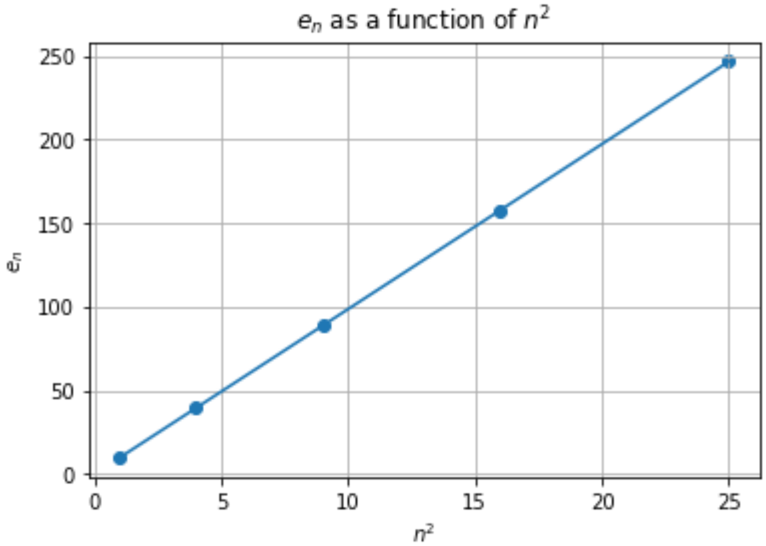




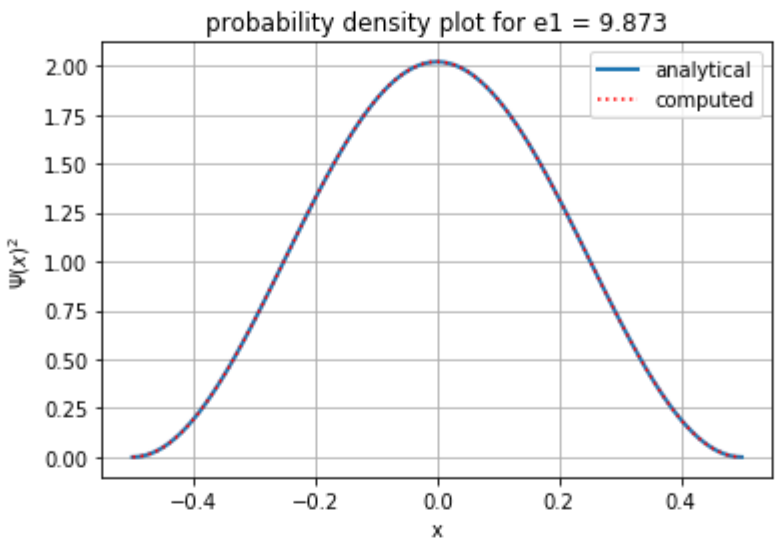

for e3 = 88.827
analytical
computed
Ψ(x)
x

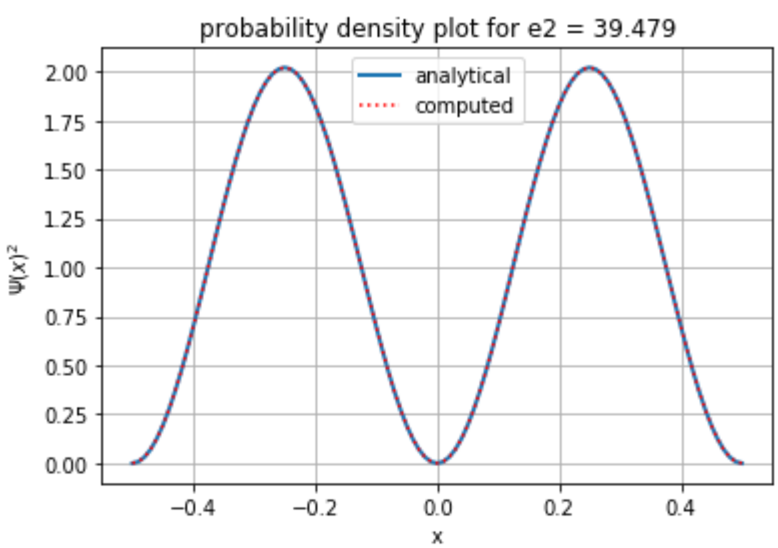

for e4 = 157.914
analytical
computed
Ψ(x)
x

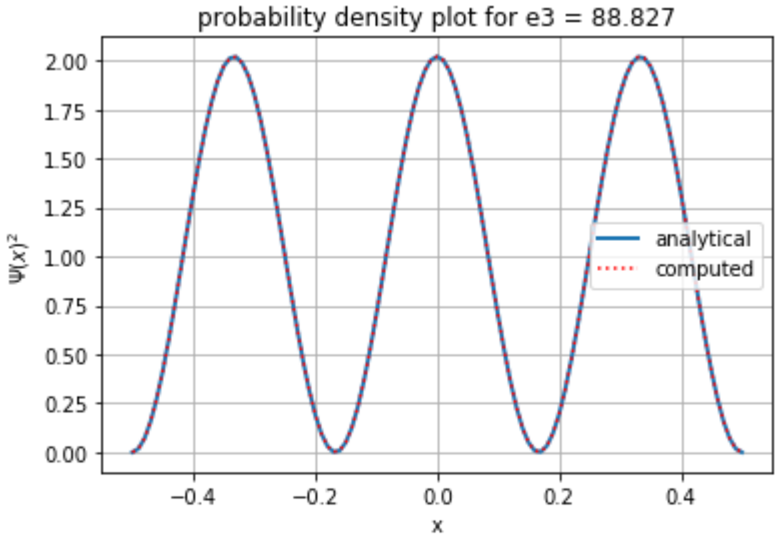

for e5 = 246.743
analytical
computed
Ψ(x)
x

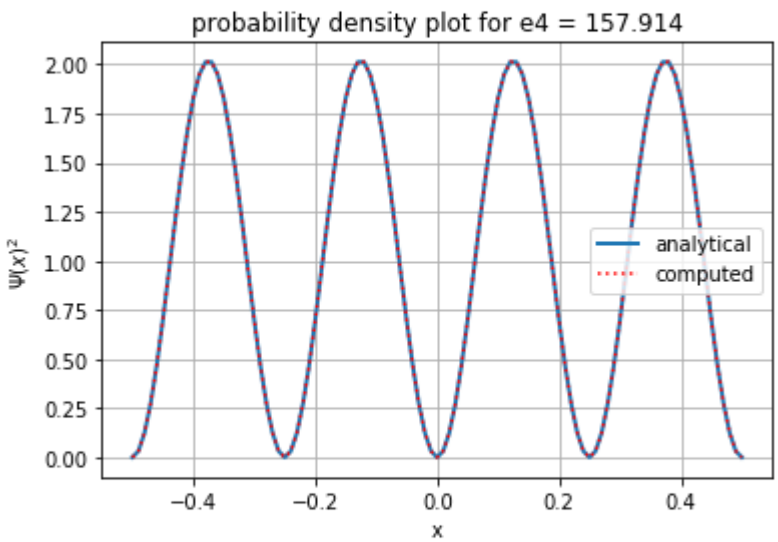


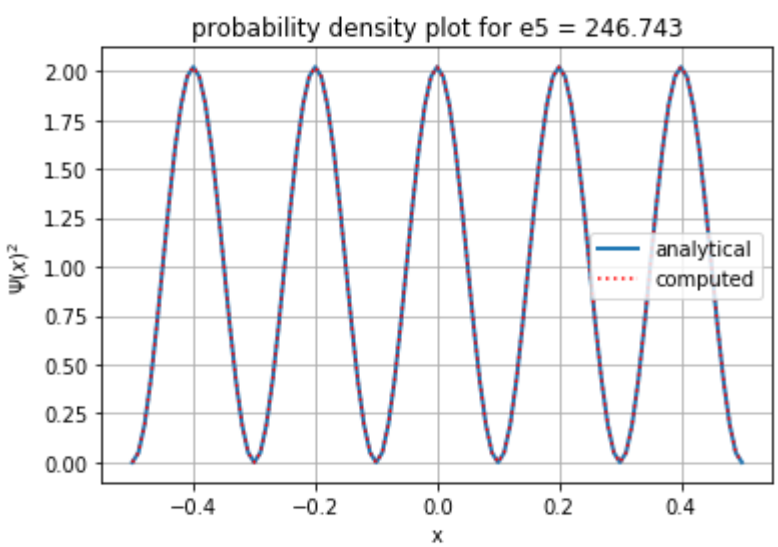

```
slope: 9.86961158923473
```






probability density plot for e3 = 88.827
$\Psi(x)^2$
x
analytical
computed


probability density plot for e4 = 157.914
$\Psi(x)^2$
x
analytical
computed


probability density plot for e5 = 246.743
$\Psi(x)^2$
x
analytical
computed

```
energies in eV of an electron trapped in a well of 5 Å
   computed E  analytical E
0    1.508273      1.507720
1    6.030911      6.030880
2   13.569570     13.569480
3   24.123645     24.123520
4   37.693407     37.692999
energies in eV of an electron trapped in a well of 10 Å
   computed E  analytical E
0    0.377068       0.37693
1    1.507728       1.50772
2    3.392392       3.39237
3    6.030911       6.03088
4    9.423352       9.42325
energies in eV of a proton trapped in a well of 5 fm
     computed E  analytical E
0  8.205851e+06  8.202841e+06
1  3.281153e+07  3.281136e+07
2  7.382606e+07  7.382557e+07
3  1.312461e+08  1.312455e+08
4  2.050732e+08  2.050710e+08
Δx Δp = 0.0033081688651695742
h/4π = 5.272859085e-35
```

In [ ]: