Assignment 11

Ashmit Bathla, 210216

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

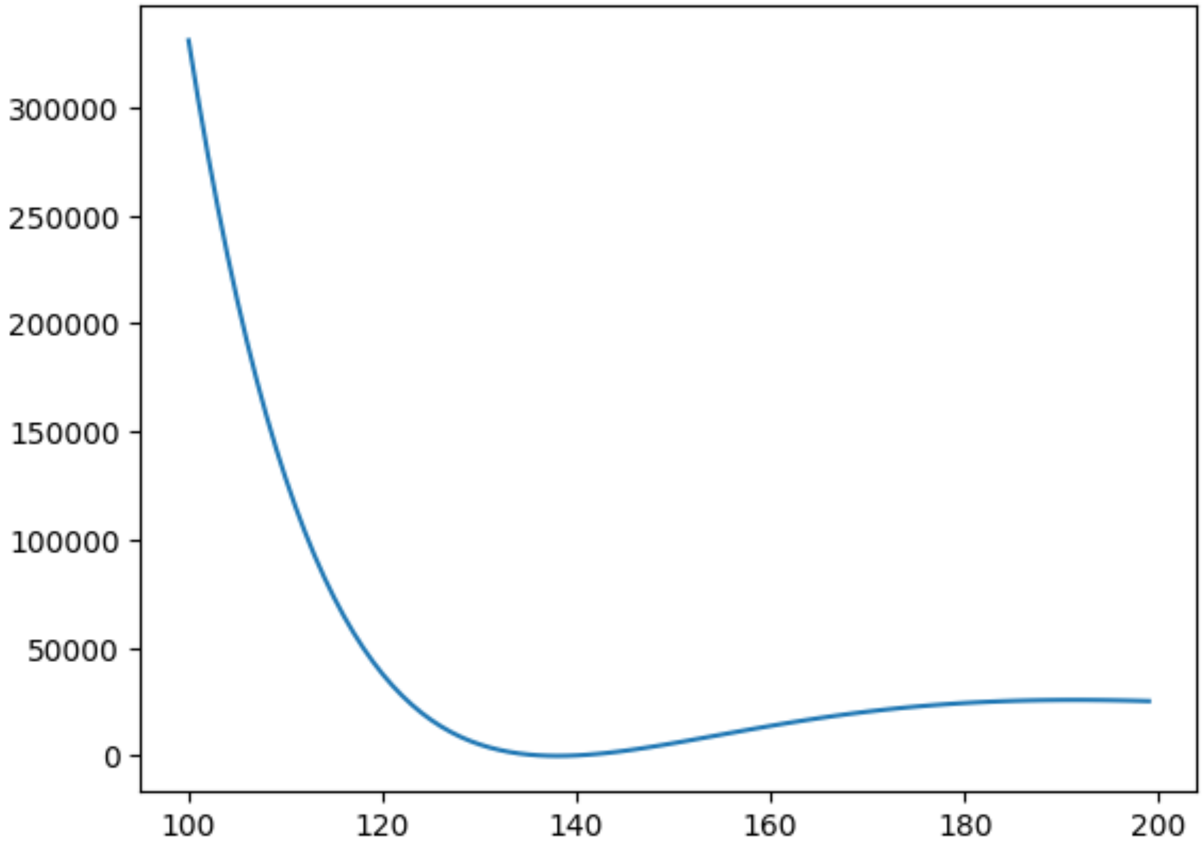
Question 1

In [2]:
```
m = 9.10938356e-31
hbar = 1.0545718e-34
eV=1.602176634e-19
V0=50*eV
a=1e-11
```

In [3]:
```
# define RK4 method !

def RK4( x0 , xn , dx , y0 , f , e ):
    x = np.arange( x0 , xn , dx )
    y = np.zeros( (x.size , y0.size ))
    y[0] = y0
    for i in range( 1 , x.size ):
        rk1 = dx*f( y[i-1] , x[i-1] ,e )
        rk2 = dx*f( y[i-1] + rk1/2 , x[i-1] + dx/2 , e )
        rk3 = dx*f( y[i-1] + rk2/2 , x[i-1] + dx/2 , e )
        rk4 = dx*f( y[i-1] + rk3 , x[i-1] + dx , e )
        y[i] = y[i-1] + ( rk1 + 2*rk2 + 2*rk3 + rk4 )/6
    return y
```

In [4]:
```
# defineie f function :
# y0 -> psi , v

def f( y , x , e ):
    dp = y[1]
    dv = -(2*m/hbar**2)*(e - V0*(x**2)/(a**2) )*y[0]
    return np.array([dp,dv])

y0 = np.array([0 , 1 ])

yend = []
elist = np.arange( 100*eV , 200*eV , 1*eV )
for e in elist :
    y = RK4( -10*a , 10*a , 0.01*a , y0 , f , e )
    yend.append( y[-1 , 0 ]**2  )
    # if( y[ -1 , 0 ] < 0.00001 ) :
    #     print( e/eV )
```

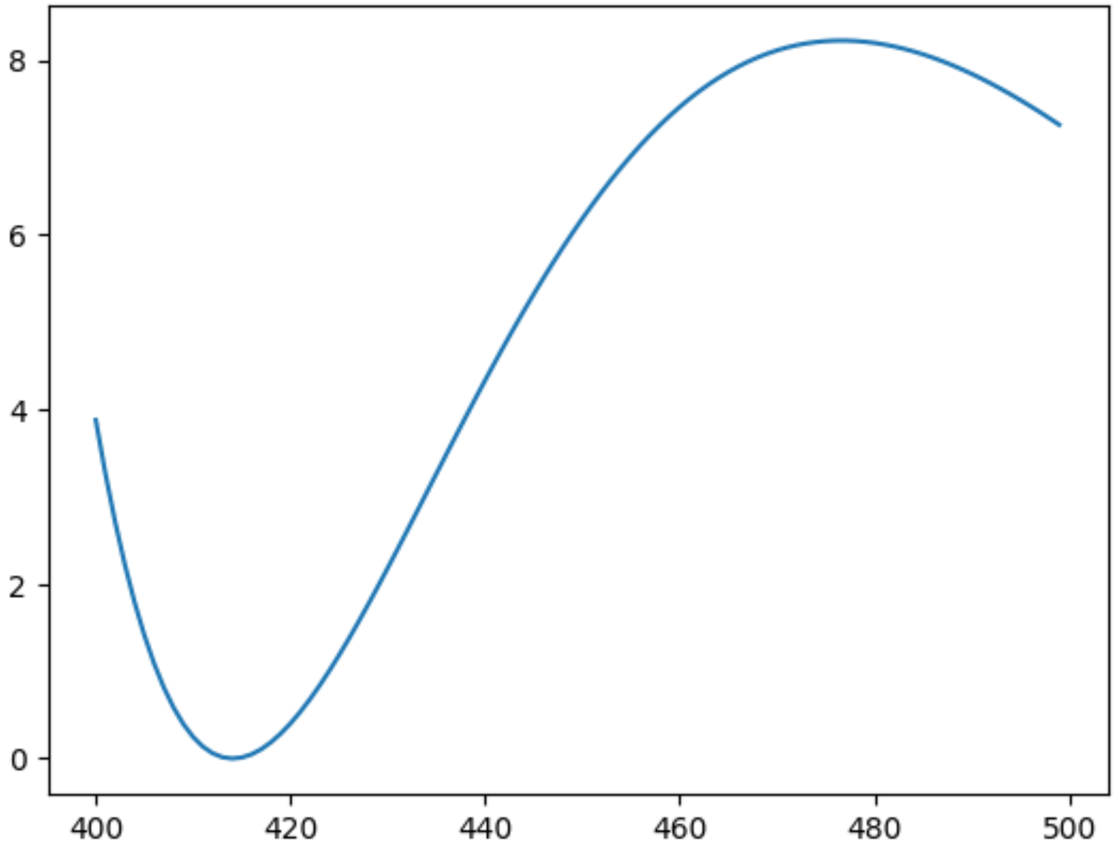
In [5]:
```
plt.plot( elist/eV , yend )
plt.show()

Eg = elist[np.where( yend == np.min( yend  ) )[0][0]]/eV
print( f'ground satate energy = {Eg}')
```




```
ground satate energy = 138.00000000000026
```

In [6]:

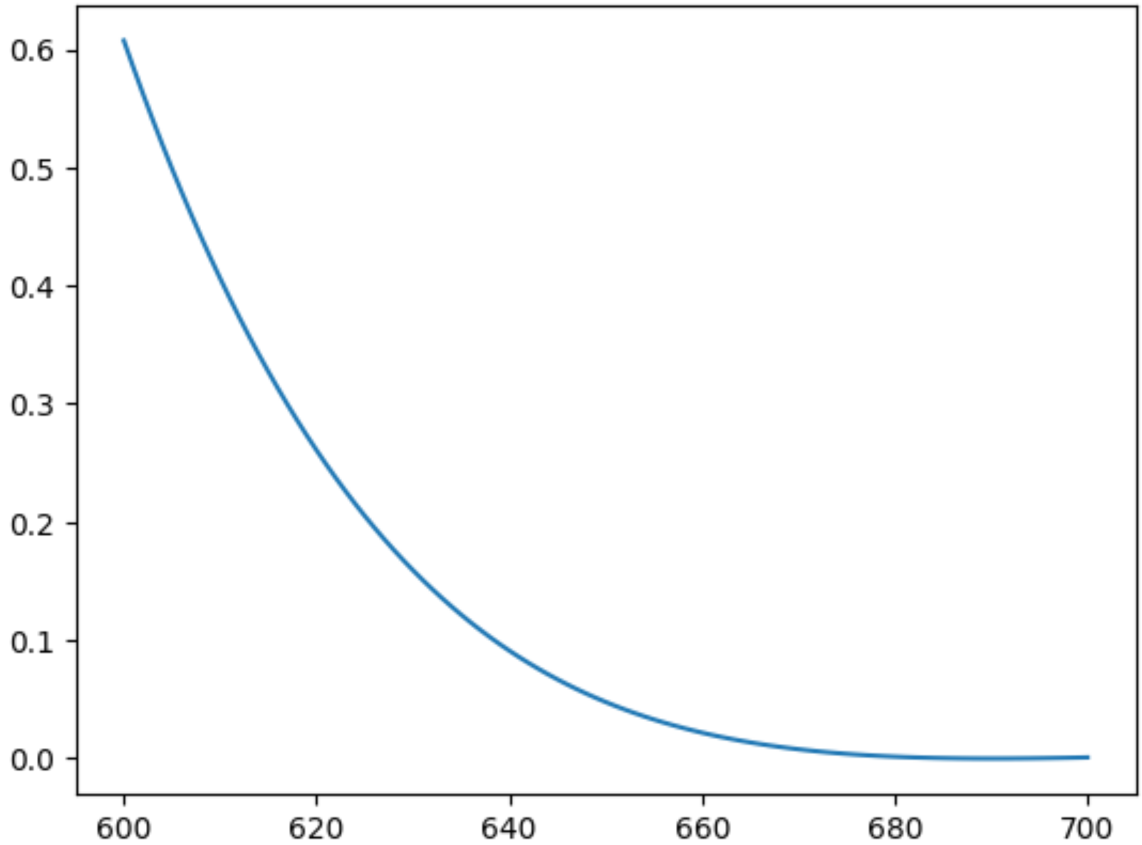
```
# defineie f function :
# y0 -> psi , v

def f( y , x , e ):
    dp = y[1]
    dv = -(2*m/hbar**2)*(e - V0*(x**2)/(a**2) )*y[0]
    return np.array([dp,dv])

y0 = np.array([0 , 1 ])

yend = []
elist = np.arange( 400*eV , 500*eV , 1*eV )
for e in elist :
    y = RK4( -10*a , 10*a , 0.01*a , y0 , f , e )
    yend.append( y[-1 , 0 ]**2  )

plt.plot( elist/eV , yend )
plt.show()

E1 = elist[np.where( yend == np.min( yend  ) )[0][0]]/eV
print( f'1st exicted state  = {E1}')
```




```
1st exicted state  = 413.99999999999983
```

In [7]:

```
# defineie f function :
# y0 -> psi , v

def f( y , x , e ):
    dp = y[1]
    dv = -(2*m/hbar**2)*(e - V0*(x**2)/(a**2) )*y[0]
    return np.array([dp,dv])

y0 = np.array([0 , 1 ])

yend = []
elist = np.arange( 600*eV , 700*eV , 1*eV )
for e in elist :
    y = RK4( -10*a , 10*a , 0.01*a , y0 , f , e )
    yend.append( y[-1 , 0 ]**2  )

plt.plot( elist/eV , yend )
plt.show()

E2 = elist[np.where( yend == np.min( yend  ) )[0][0]]/eV
print( f'2nd exicted state  = {E2}')
```




```
2nd exicted state  = 689.9999999999989
```

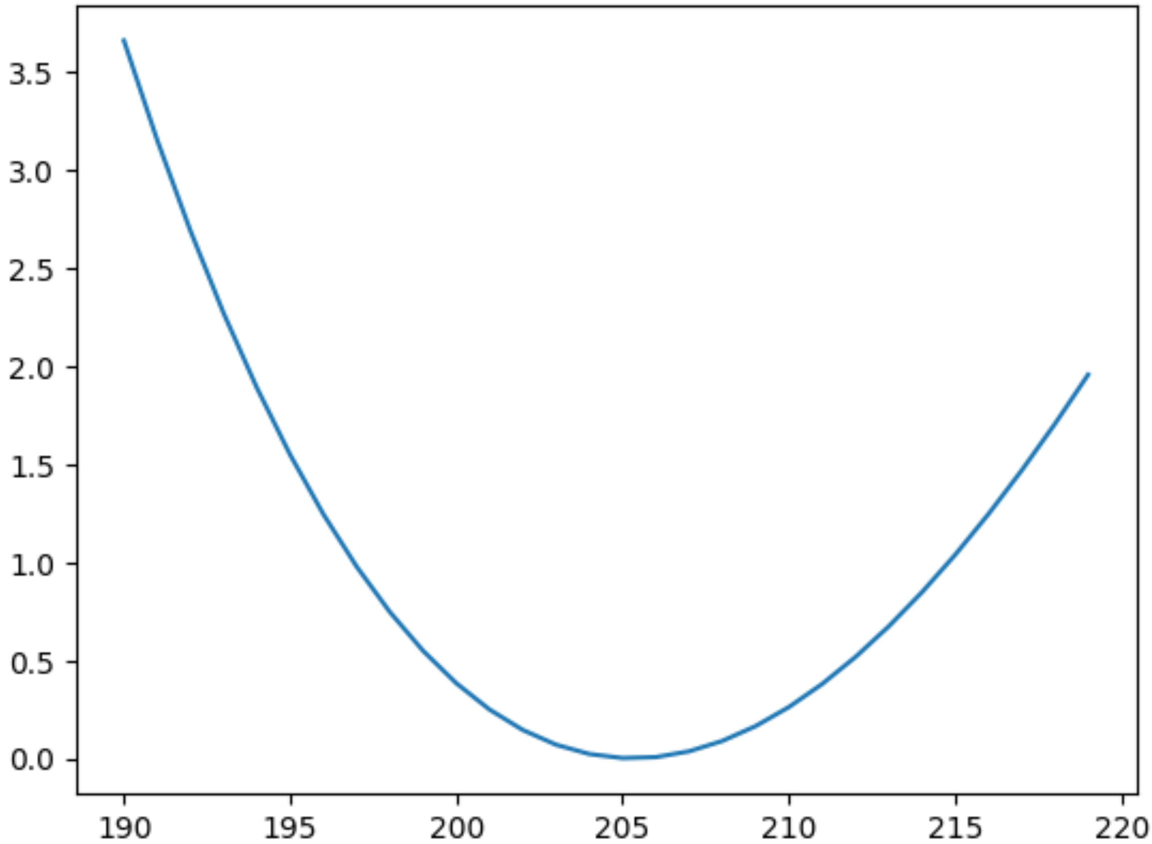
In [8]:
```
print( f'Eg - E1' , Eg - E1 )
print( 'E1 - E2 ' , E1 - E2 )
```

```
Eg - E1 -275.99999999999955
E1 - E2  -275.99999999999903
```

part b

In [9]:
```
def trap( a , dx ):
    return dx*(a[0] + a[-1])/2 + dx*(np.sum( a[1:-1]))
```

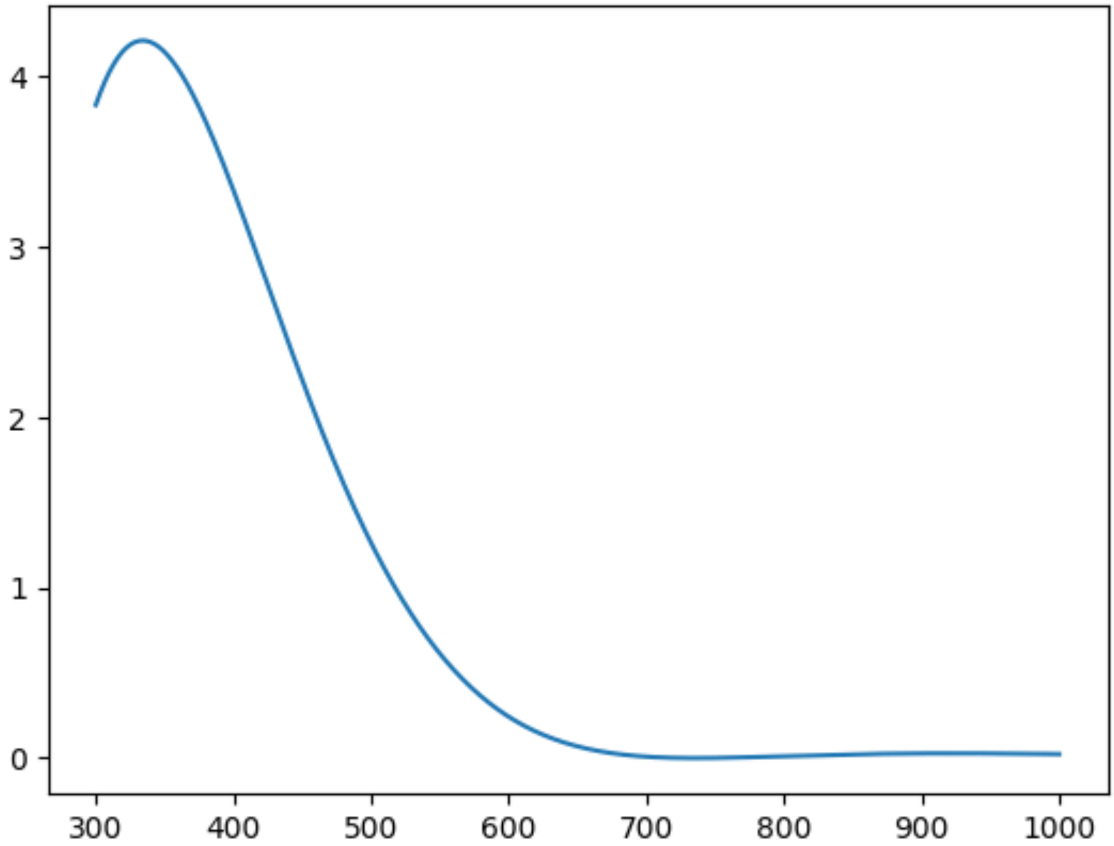
In [10]:
```
# defineie f function :
# y0 -> psi , v

def f( y , x , e ):
    dp = y[1]
    dv = -(2*m/hbar**2)*(e - V0*(x**4)/(a**4) )*y[0]
    return np.array([dp,dv])

y0 = np.array([0 , 1 ])

yend = []
elist = np.arange( 190*eV , 220*eV , 1*eV )
for e in elist :
    y = RK4( -10*a , 10*a , 0.01*a , y0 , f , e )
    psi = y[ : , 0 ]
    psi = psi/np.sqrt(trap( psi**2 , 0.01*a ))
    yend.append( y[-1 , 0   ]**2   )
    # if( y[ -1 , 0 ] < 0.00001 ) :
    #     print( e/eV )

yend = np.array( yend )
yend = yend/np.average( yend )

plt.plot( elist/eV , yend )
plt.show()
```



In [11]:
```
Eg = elist[np.where( yend < 0.01   )]/eV
print( f'ground state   = {Eg} ev')
```

```
ground state   = [205. 206.] ev
```

In [12]:
```
# defineie f function :
# y0 -> psi , v

def f( y , x , e ):
    dp = y[1]
    dv = -(2*m/hbar**2)*(e - V0*(x**4)/(a**4) )*y[0]
    return np.array([dp,dv])

y0 = np.array([0 , 1 ])

yend = []
elist = np.arange( 300*eV , 1000*eV , 0.7*eV )
for e in elist :
    y = RK4( -10*a , 10*a , 0.01*a , y0 , f , e )
    psi = y[ : , 0 ]
    psi = psi/np.sqrt(trap( psi**2 , 0.01*a ))
    yend.append( y[-1 , 0  ]**2  )
    # if( y[ -1 , 0 ] < 0.00001 ) :
    #     print( e/eV )

yend = np.array( yend )
yend = yend/np.average( yend )

plt.plot( elist/eV , yend )
plt.show()
```

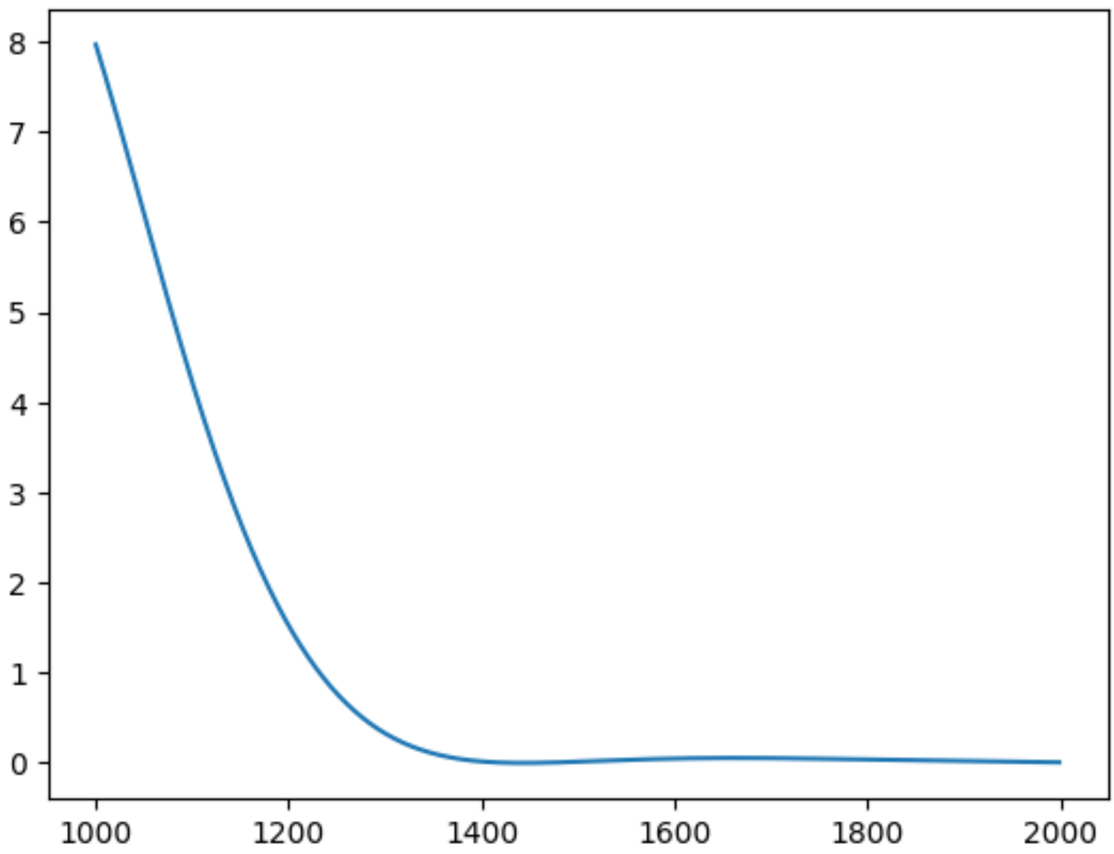


In [13]:
```
E1 = elist[np.where( yend < 0.000001  )]/eV
print( f'first exicted state  = {E1} ev')
```

```
first exicted state  = [735.4 736.1] ev
```

In [14]:

```
# defineie f function :
# y0 -> psi , v

def f( y , x , e ):
    dp = y[1]
    dv = -(2*m/hbar**2)*(e - V0*(x**4)/(a**4) )*y[0]
    return np.array([dp,dv])

y0 = np.array([0 , 1 ])

yend = []
elist = np.arange( 1000*eV , 2000*eV , 1*eV )
for e in elist :
    y = RK4( -10*a , 10*a , 0.01*a , y0 , f , e )
    psi = y[ : , 0 ]
    psi = psi/np.sqrt(trap( psi**2 , 0.01*a ))
    yend.append( y[-1 , 0  ]**2  )
    # if( y[ -1 , 0 ] < 0.00001 ) :
    #     print( e/eV )

yend = np.array( yend )
yend = yend/np.average( yend )

plt.plot( elist/eV , yend )
plt.show()
```



In [15]:

```
E2 = elist[np.where( yend < 0.00001  )]/eV
print( f'second exicted state  = {E2} ev')
```

```
second exicted state  = [1443. 1444.] ev
```

Question 2

In [16]:
```
# define the swap function
def swap( A , i ):
    m = A.shape[0]
    if( i >= m - 1 ):
        return A
    else :
        if( A[ i ,i ] == 0 ):
            j = i
            while A[ j , i ] == 0 :
                j += 1
            A[[i,j]] = A[[j,i]]
        return A
```

In [17]:
```
# define the function that makes the matrix upper triangular
def upper( A  ):
    m = A.shape[0]
    for i in range( m ):
        A = swap( A, i )
        A[i,:]/=A[i,i]
        j = i + 1
        while j < m :
            A[j,:] -= A[i,:]*A[j,i]
            j += 1
    return A
```

In [18]:
```
def back_solve( A ):
    m = A.shape[0]
    result = np.zeros(m)
    for i in range( m - 1, -1 , -1 ):
        result[i] = A[ i , m ]
        for j in range(i + 1 , m ):
            result[i] -= A[i,j]*result[j]
    return result
```

In [19]:
```
A = np.array([
    [0,0,2,1,2],
    [0,1,0,2,-1],
    [1,2,0,-2,0],
    [0,0,0,-1,1],
    [0,1,-1,1,-1]
] , dtype = np.float32 )
print( A , A.shape )
```

```
[[ 0.  0.  2.  1.  2.]
 [ 0.  1.  0.  2. -1.]
 [ 1.  2.  0. -2.  0.]
 [ 0.  0.  0. -1.  1.]
 [ 0.  1. -1.  1. -1.]] (5, 5)
```

In [20]:
```
V = np.array([1 , 1, -4, -2 , -1 ]).reshape((5,1))
print( V , V.shape )
```

```
[[ 1]
 [ 1]
 [-4]
 [-2]
 [-1]] (5, 1)
```

In [21]:

```python
def gaus( A , V ):
    A = np.concatenate( (A , V) , axis = 1 )
    A = upper( A )
    result = back_solve( A  )
    result = result.reshape((result.size , 1 ))
    return result

result = gaus( A , V )
print( f'result = \n {result}' )
```

```
result =
 [[ 2.]
 [-2.]
 [ 1.]
 [ 1.]
 [-1.]]
```

In [22]:

```python
# to ckeck the output !
print(np.matmul( A , result ))
print( 'is equal to V! ')
```

```
[[ 1.]
 [ 1.]
 [-4.]
 [-2.]
 [-1.]]
is equal to V!
```

Question 3

In [23]:

```python
def lu_decomp( A ):
    n = A.shape[0]
    L  = np.zeros( ( n , n ))
    U = np.zeros( (n , n ))
    U[0,:] = A[0,:]
    for i in range( n ):
        L[ i , i ] = 1
    for i in range( 1 , n ):
        for j in range(0 , n ):
            if( i > j ):
                L[i,j] = A[i,j]
                for k in range( j ):
                    L[i,j] -= L[i,k]*U[k,j]
                L[i,j] /= U[j,j]
            else:
                U[i,j] = A[i,j]
                for k in range( i ):
                    U[i,j] -= L[i,k]*U[k,j]
    return L , U
```

In [24]:

```python
A = np.array([
    [4,-3,6],
    [8,-3,10],
    [-4,12,-10]
])
L , U = lu_decomp( A )

print( f'L = {L} \n and \n U = {U}')
```

```
L = [[ 1.  0.  0.]
 [ 2.  1.  0.]
 [-1.  3.  1.]]
 and
 U = [[ 4. -3.  6.]
 [ 0.  3. -2.]
 [ 0.  0.  2.]]
```

In [25]:
```python
np.matmul( L , U )
```

Out[25]:
```
array([[  4.,  -3.,   6.],
       [  8.,  -3.,  10.],
       [ -4.,  12., -10.]])
```

In [26]:
```python
def forward_solve( A , V ):
    m = A.shape[0]
    result = np.zeros( m )
    for i in range( m ):
        result[i] = V[i]
        if i > 0 :
            for k in range( i ):
                result[i] -= result[k]*A[i , k ]
        result[i] /= A[i,i]
    return result
```

In [27]:
```python
def back_solve( A , V ):
    m = A.shape[0]
    result = np.zeros(m)
    for i in range( m - 1, -1 , -1 ):
        result[i] = V[i]
        for j in range(i + 1 , m ):
            result[i] -= A[i,j]*result[j]
        result[i] /= A[i,i]
    return result
```

In [28]:
```python
def lu_solve( A , V ):
    L , U = lu_decomp( A )
    n = V.shape[1]
    result = np.zeros(  V.shape )
    for i in range( n ):
        result1 = forward_solve( L , V[:,i].reshape( - 1, 1 ) )
        result2 = back_solve( U , result1.reshape( -1 , 1 ) )
        result[ : , i ] = result2
    return result

V = np.array([
    [1,0],
    [0,1],
    [0,0]
])
result  = lu_solve( A , V )
print( f'result = \n {result}' )
```

```
result =
 [[-3.75        1.75      ]
 [ 1.66666667 -0.66666667]
 [ 3.5        -1.5       ]]
```

In [29]:
```python
# to check the answer
np.matmul( A  , result )
```

Out[29]:
```
array([[1., 0.],
       [0., 1.],
       [0., 0.]])
```