Resonant tunneling

Álvaro Cauqui

Universidad Carlos III de Madrid

Advanced Quantum Mechanics - Group 111

March 2, 2024

1 Introduction

Aim

- To numerically calculate the transmission coefficient of a wavefunction across a double barrier.

Whenever a free particle approaches a potential barrier, due to the nature of the wavefunction, we find there is always a part of the latter (thus a probability) that goes through the barrier, even if the particle itself has a energy below that of the potential barrier. There is a way to analytically calculate the transmission coefficient analytically, and it is **equation 7** as given in the guide.

2 Results

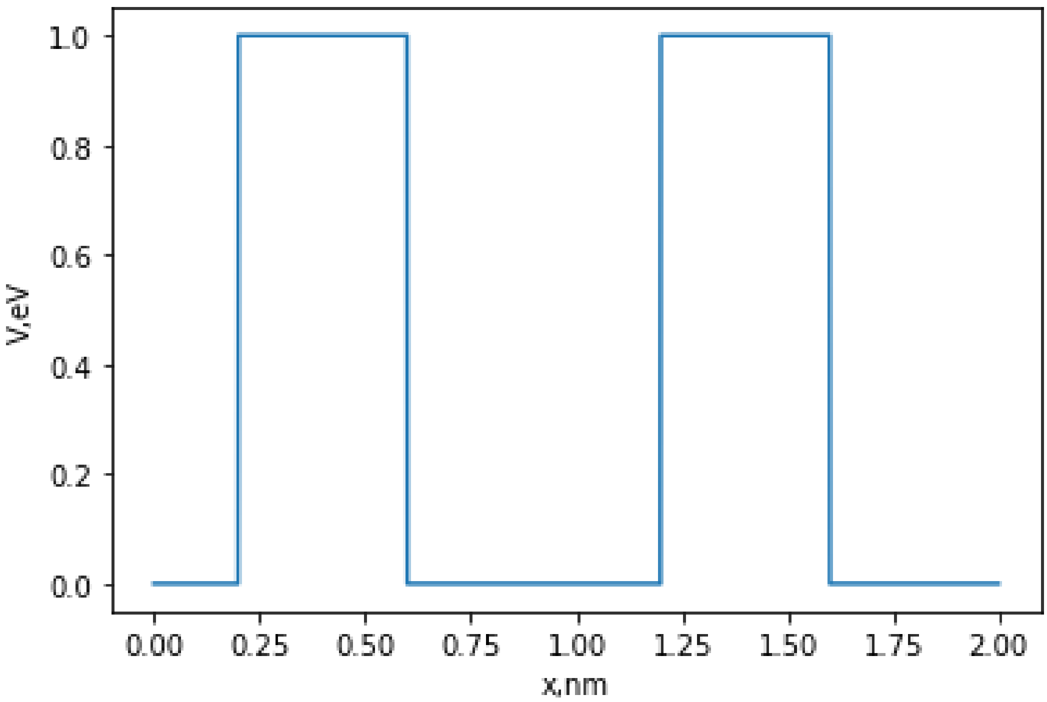
```python
import numpy as np
import math
import matplotlib.pyplot as plt
i=1j
me=9.109e-31
m=0.067*me

def evtoj(ev):
    _=ev*1.602e-19

    return _

def K(x2,x4,k1,k3,k5,w2,w3,w4):

    phi1=k3*w3
    phi2=math.atan(x2/k1)
    phi3=math.atan(x2/k3)
    phi4=math.atan(x4/k3)
    phi5=math.atan(x4/k5)
    return np.exp((x2*w2)+(x4*w4))*(np.exp(i*(-phi1+phi2+phi3+phi4+phi5))-np.exp(i*(phi1+phi2-phi3-phi4+phi5)))\
        +np.exp((x2*w2)-(x4*w4))*(-np.exp(i*(-phi1+phi2+phi3-phi4-phi5))+np.exp(i*(phi1+phi2-phi3+phi4-phi5))) \
            +np.exp((-x2*w2)+(x4*w4))*(-np.exp(i*(-phi1-phi2-phi3+phi4+phi5))+np.exp(i*(phi1-phi2+phi3-phi4+phi5))) \
                +np.exp((-x2*w2)+(-x4*w4))*(np.exp(i*(-phi1-phi2-phi3-phi4-phi5))-np.exp(i*(phi1-phi2+phi3+phi4-phi5)))
def k(Ei,Vx,m=m):
    hbar=1.054571817e-34
    return np.sqrt((2*m*(Ei-Vx)))/hbar
def x(Ei,Vx,m=m):
    hbar=1.054571817e-34
    return np.sqrt((2*m*(Vx-Ei)))/hbar
def T(x2,x4,k1,k3,k5,w2,w3,w4):
    K_=K(x2,x4,k1,k3,k5,w2,w3,w4)
    Ksq=K_.conjugate()*K_
    return (((2**8)*k1*(x2**2)*(k3**2)*(x4**2)*k5)) \
```

```
        /((Ksq.real)*(k1**2+x2**2)*(x2**2+k3**2)*(k3**2+x4**2)*(x4**2+k5**2))

def Tc(E):

    Ej=evtoj(E)
    k1=k3=k5=k(Ej,0,me)
    x4=x2= x(Ej,1.6e-19,me)
    w2=w4=0.4e-9
    w3=0.6e-9
    return T(x2,x4,k1,k3,k5,w2,w3,w4)
```

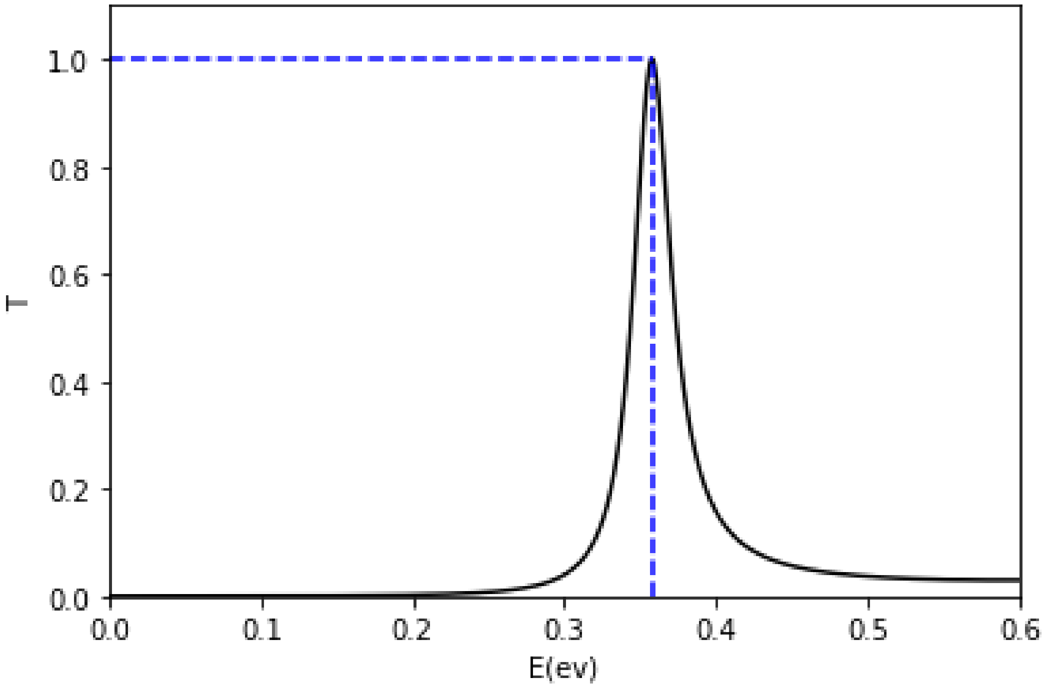
2.1 Potential 1: symmetric double barrier

The first potential studyied is a symmetric double barrier, with barrier width equal to 0.4nm and well width equal to 0.6 nm, with a maximum of 1ev:

```
def V1(x_):
    x= x_ * 1e-9
    if x<0.2e-9:
        return 0
    elif 0.2e-9<=x<0.6e-9:
        return 1
    elif 1.2e-9 <= x < 1.6e-9:

        return 1
    elif 0.6<=x<1.2e-9:
        0
        return 0
    else:
        return 0

x1=np.arange(0,2,0.001)
V10=[V1(_)for _ in x1]
plt.figure()
plt.plot(x1,V10)
plt.xlabel('x,nm')
plt.ylabel('V,eV')
```

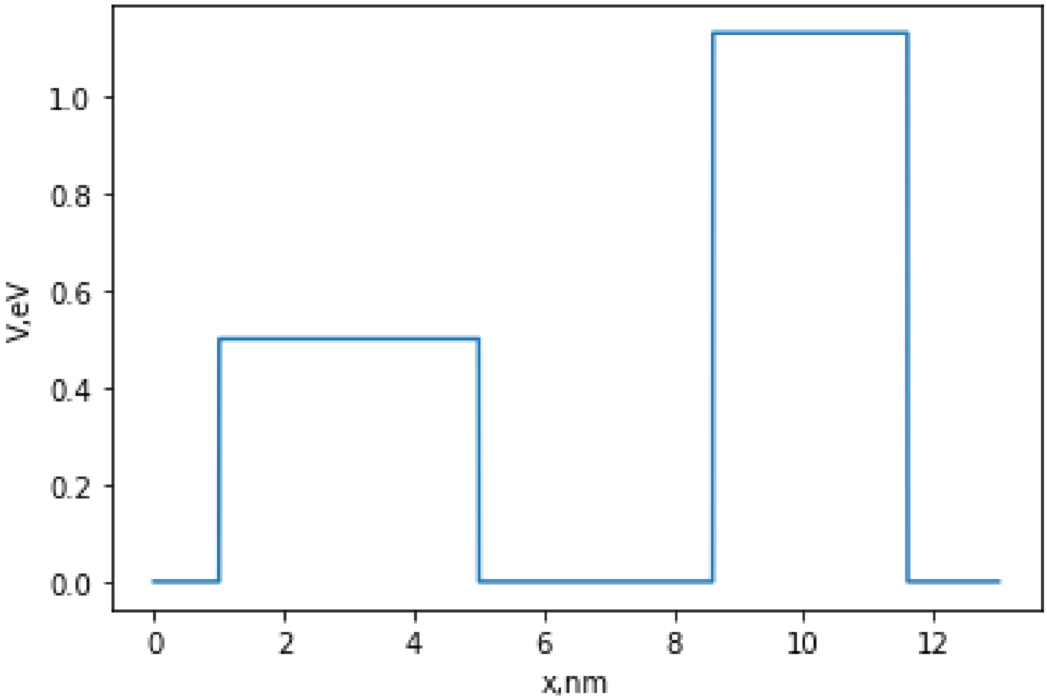



```
Eev=np.arange(0.001,0.6,0.00001)


T_at_E =[Tc(ei) for ei in Eev]

#%%

plt.figure()
plt.plot(Eev,T_at_E,color='black')
plt.hlines(max(T_at_E), 0 ,Eev[T_at_E.index(max(T_at_E))], color='blue', linestyles='--'
    )
plt.vlines(Eev[T_at_E.index(max(T_at_E))], 0, 1, color='blue', linestyles='--')
plt.ylim(bottom=0,top=1.1)
plt.xlim(left=0,right=0.6)
plt.xlabel('E(ev)')
plt.ylabel('T')
```

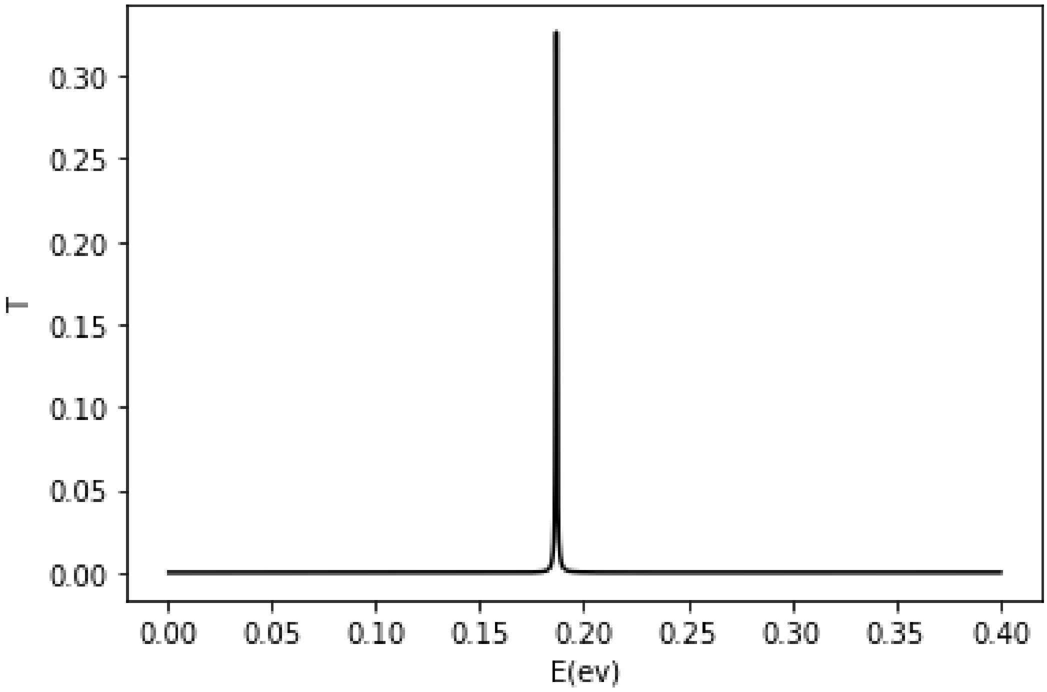


2.2 Potential 2: Asymmetric double barrier

```
def V2(x_):
    x= x_ * 1e-9
    if x<1e-9:
        return 0
    elif 1e-9<=x<5e-9:

        return 0.5

    elif 8.6e-9<=x<11.6e-9:

        return 1.13
    else:
        return 0

x2=np.arange(0,13,0.001)
V20=[V2(_)for _ in x2]
plt.figure()
plt.plot(x2,V20)
plt.xlabel('x,nm')
plt.ylabel('V,eV')
```



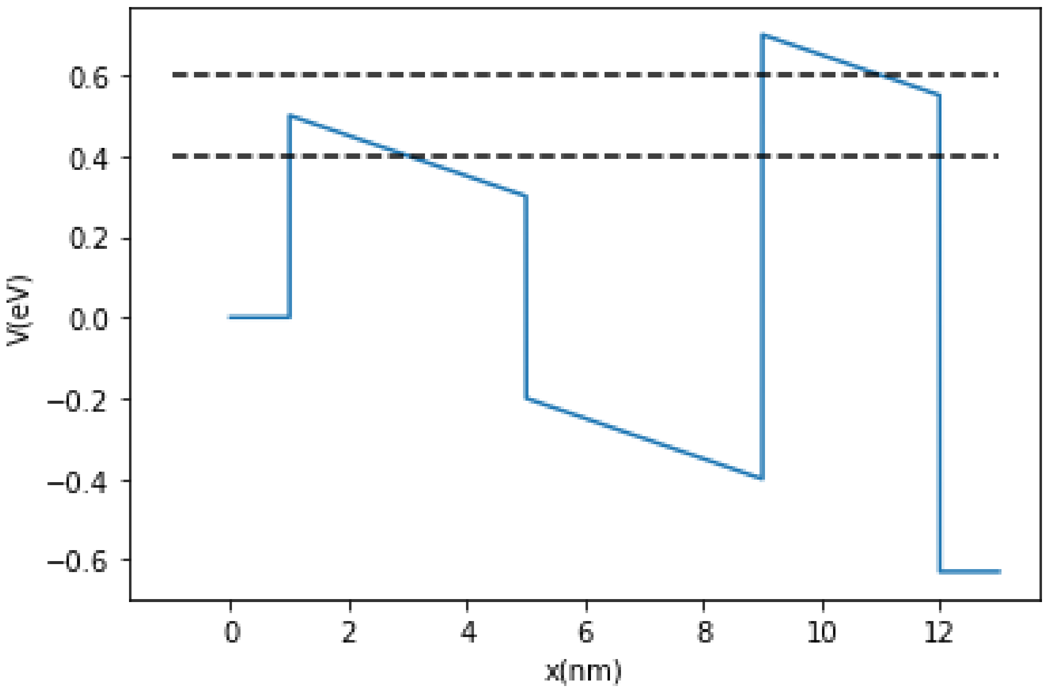

```
Eev2=np.arange(0.001,0.4,0.00001)
def Tc2(E):

    Ej=evtoj(E)
    k1=k3=k5=k(Ej,0,m)
    x2=x(Ej,0.5*1.6e-19,m)
    x4=x(Ej,1.13*1.6e-19,m)
    w2=4e-9
    w4=3e-9
    w3=3.6e-9
    return T(x2,x4,k1,k3,k5,w2,w3,w4)

T_at_E2 = np.array(tuple(Tc2(Ei) for Ei in Eev2))

plt.figure()
plt.plot(Eev2,T_at_E2)
plt.xlabel('E(ev)')
plt.ylabel('T')
```



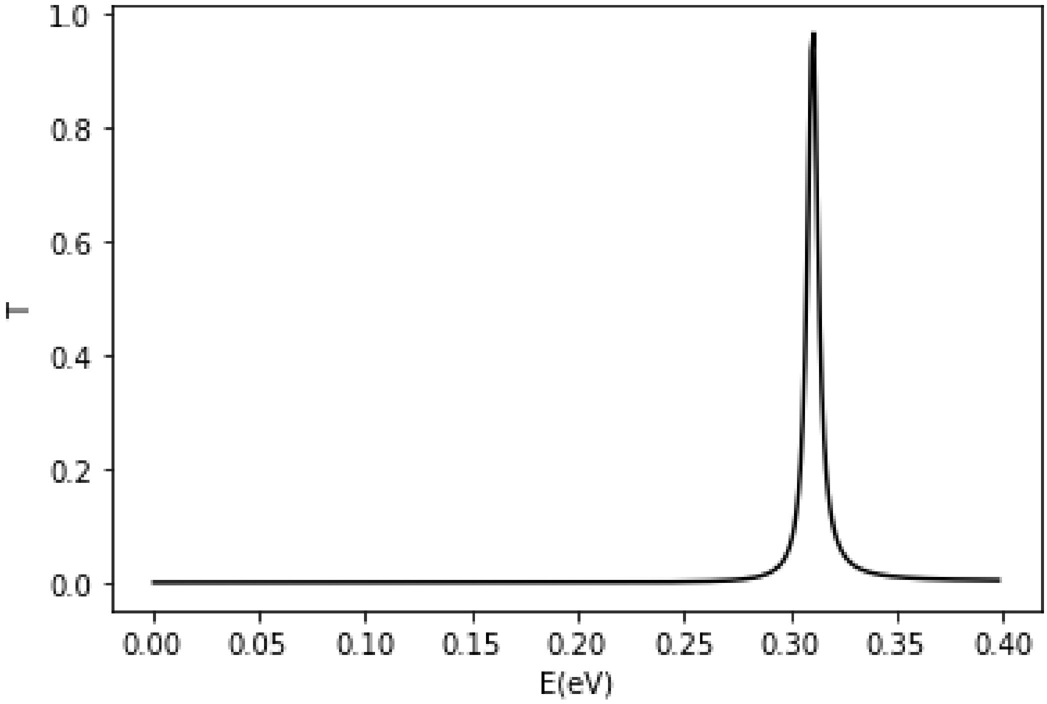
2.3 Potential 3: Asymmetric double barrier with external bias

For simplicity, even though the potential of the barriers changes with position, I used the mean potential of each barrier to calculate the transmission coefficient, shown as the black dashed lines in the diagram of the potential.

```
def V3(x_):
    x= x_ * 1e-9
    if x<1e-9:
        return 0
    elif 1e-9<=x<5e-9:
        m1 = -5e7
        b1=0.55
        return m1*x +b1
    elif 5e-9 <= x < 9e-9:
        m2 = -5e7
        b2=0.05
        return m2*x + b2
    elif 9e-9<=x<12e-9:
        m3=-5e7
        b3=1.15
        return m3*x +b3
    else:
        return -0.63

x0=[x for x in np.arange(0,13,0.001)]
Ve=[V3(f) for f in x0]
plt.plot(x0,Ve)
```




```
def Tc(E):
    Ej=evtoj(E)
    k1=k(Ej,0,me)
    k3=k(Ej,(-0.3)*1.6e-19,m)
    k5=k(Ej,((-0.67)*1.6e-19),m)
    x4=x(Ej,(0.6)*1.6e-19,m)
    x2=x(Ej,(0.4)*1.6e-19,m)
    w2=4e-9
    w3=4e-9
    w4=3e-9
    return T(x2,x4,k1,k3,k5,w2,w3,w4)

Ee=[i for i in np.arange(0.001,0.399,0.001)]

Tg=[Tc(g) for g in Ee]
plt.figure()
plt.plot(Ee,Tg,color='black')
plt.xlabel('E(eV)')
plt.ylabel('T')
```


1.0
0.8
0.6
0.4
0.2
0.0
T
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
E(eV)