Computational quantum mechanics

Status	Started
Assign	Paramesh Chandra
Created time	@April 4, 2022 2:15 PM
Last edited	@May 2, 2022 8:31 PM
Priority	
Subject	Physics

Finding eigenstates solving the transcendental equation

Introduction

We need to make something clear before starting this study of transcendental equations related to quantum mechanics. Ok, let's start with the equation defining quantum mechanics, i.e. **Schrodinger's equation**.

The Schrödinger equation describes the dynamics of a microscopic particle of mass m in
a one-dimensional time-independent potential V(x) is given by

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2}+V(x)\psi(x)=E\psi(x)$$

where E is the total energy of the particle. That's all we need to solve to know about the quantum system. This is the time-independent Schrodinger's equation that we will be dealing with in the upcoming section. Let's talk about the solutions first, we can expect two kinds of solutions from the above equation.

Properties of One-Dimensional Motion

Discrete Spectrum (Bound States)

Bound states occur whenever the particle cannot move to infinity. That is, the particle is confined or bound at all energies to move within a finite and limited region of space which is delimited by two classical turning points. The Schrödinger equation in this region admits only solutions that are discrete. The infinite square well potential and the harmonic oscillator are typical examples that display bound states.

Continuous Spectrum (Unbound States)

Unbound states occur in those cases where the motion of the system is not confined; a typical example is a free particle.

Mixed Spectrum

Potentials that confine the particle for only some energies give rise to mixed spectra; the motion
of the particle for such potentials is confined for some energy values only.

Symmetric Potentials and Parity

- Non-degenerate spectrum
- Degenerate spectrum

Summary

- The energy spectrum of a bound state system is discrete and non-degenerate.

- The bound state wave function has: (a) n nodes if n= 0 corresponds to the ground
  state and (b) (n-1) nodes if n=1 corresponds to the ground state.
- The bound state eigenfunctions in an even potential have definite parity.
- The eigenfunctions of a degenerate spectrum in an even potential do not have definite
  parity.

List of one-dimensional motions we can solve analytically

- The Free Particle: Continuous States
- The Potential Step
- The Potential Barrier and Well
- The Infinite Square Well Potential

We are close...

The Finite Square Well Potential

Consider a particle of mass m moving in the following symmetric potential:

$$V(x) = \begin{cases} V_0, & x < -a/2. \\ 0, & -a/2 < x < a/2. \\ V_0 & x > a/2 \end{cases}$$

The two physically interesting cases are $E > V_0$ and $E > V_0$. We expect the solutions to yield a continuous doubly-degenerate energy spectrum for $E > V_0$ and a discrete nondegenerate spectrum for $0 < E < V_0$.

Two possible solutions are,

- The Scattering Solutions ($E > V_0$)
- The Bound State Solution $(0 < E < V_0)$

Solutions are,

$$-\alpha_n cot\alpha_n = \sqrt{R^2 - \alpha_n^2}, (oddstates)$$

$$-\alpha_n tan\alpha_n = \sqrt{R^2 - \alpha_n^2}, (evenstates)$$

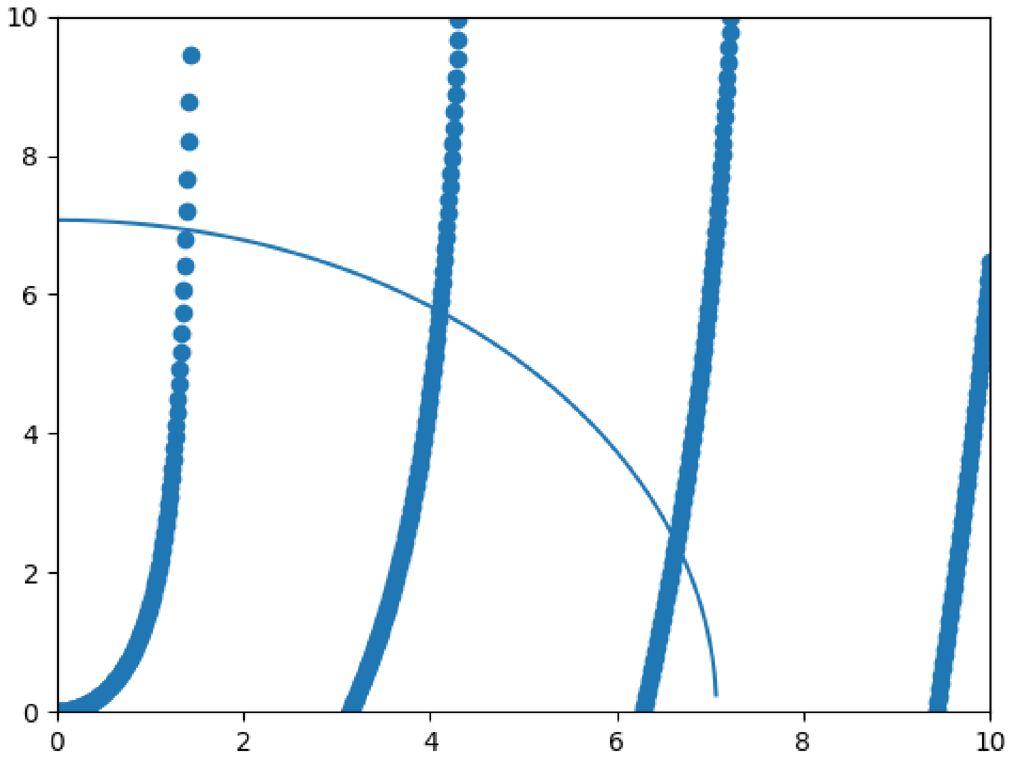
Optimization and root finding (scipy.optimize) - SciPy v1.8.0 Manual

SciPy provides functions for minimizing (or maximizing) objective functions, possibly subject to constraints. It includes solvers for nonlinear problems (with support for both local and global optimization algorithms), linear programing, constrained and nonlinear least-squares, root finding, and curve fitting.

https://docs.scipy.org/doc/scipy/reference/optimize.html

Code

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import newton


def leftfunc(x):
    return(x*np.tan(x))

def rightfunc(x):
    return(np.sqrt(50-x**2))

def your_funcs(x):

    f = leftfunc(x)-rightfunc(x)
    return f

x = np.linspace(0,10,1000)
plt.scatter(x, leftfunc(x))
plt.plot(x, rightfunc(x))
plt.axis([0, 10, 0, 10])
plt.show()

initial_guess = float(input("Input the initial guess: "))
sol2 = newton(your_funcs, initial_guess)
print(sol2)
```



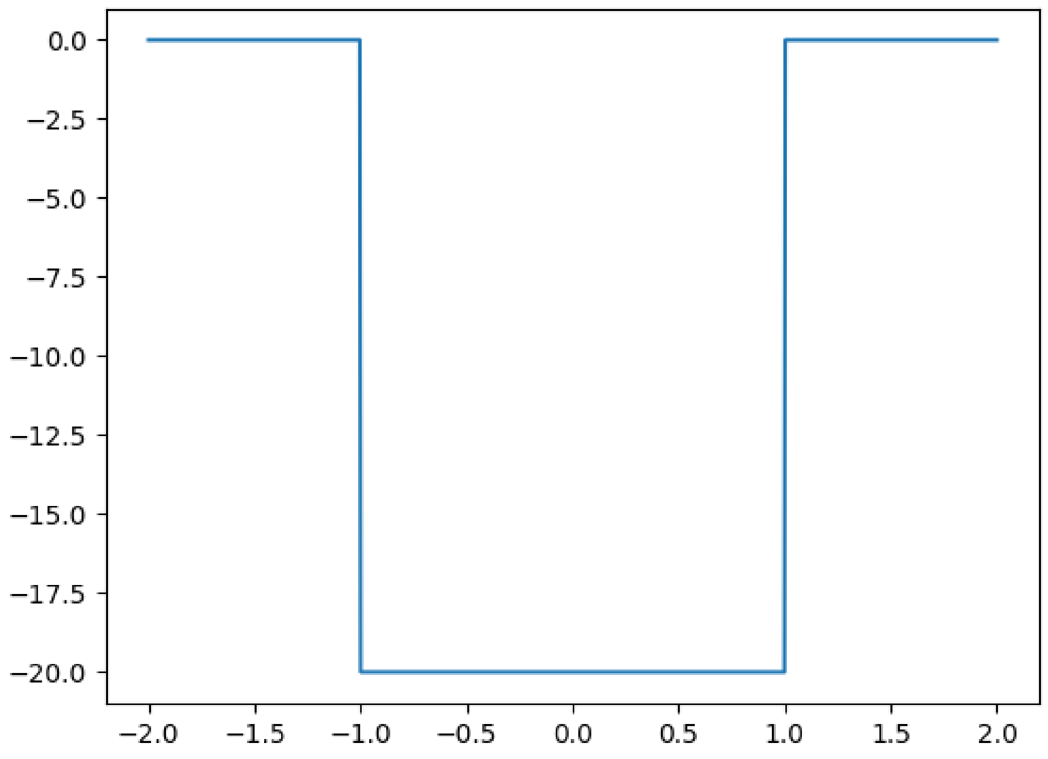
We can predict the value close to the solution and find the root from that value.

Using shooting method

Solution using SciPy module

Finite square well potential

$$V(x) = \begin{cases} 0, & x < -L. \\ -V_0, & -L < x < L. \\ 0 & x > L \end{cases}$$



Schrodinger’s equation

$$\frac{d^2}{dx^2}\psi(x) = \frac{2m_e E}{\hbar}\psi(x)$$

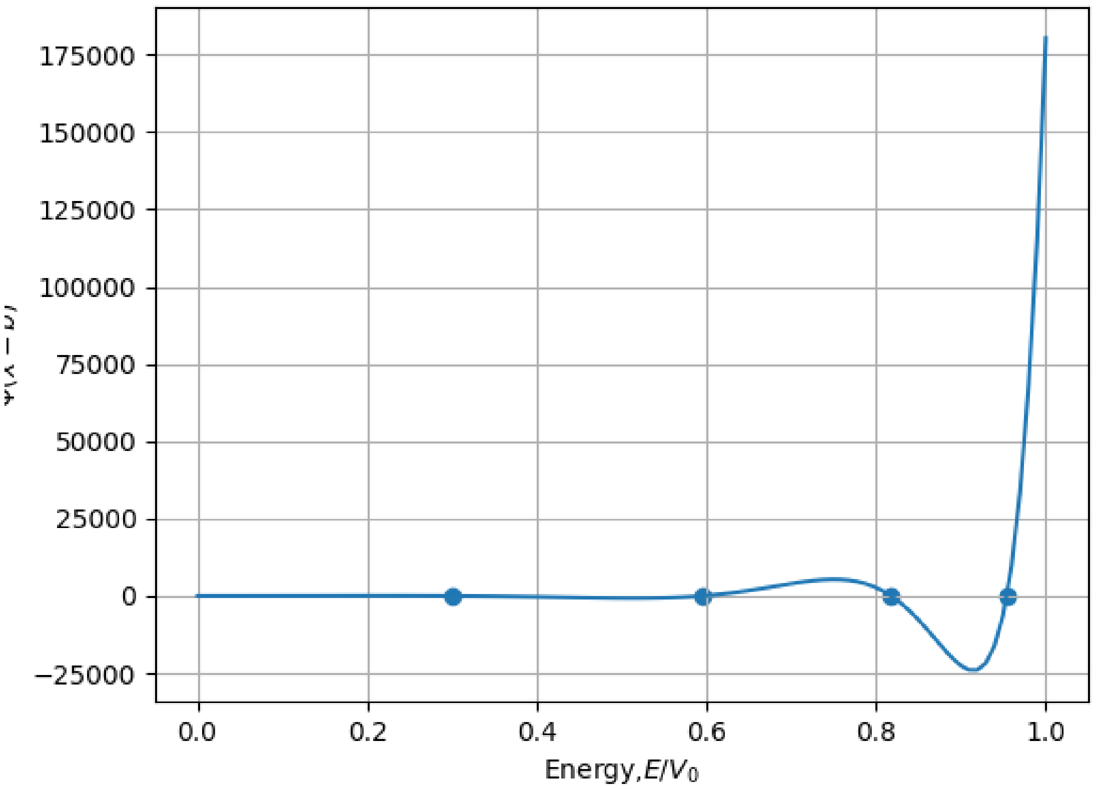
Boundary conditions

$$\psi(x=-L) = 0, \psi(x=L) = 0$$

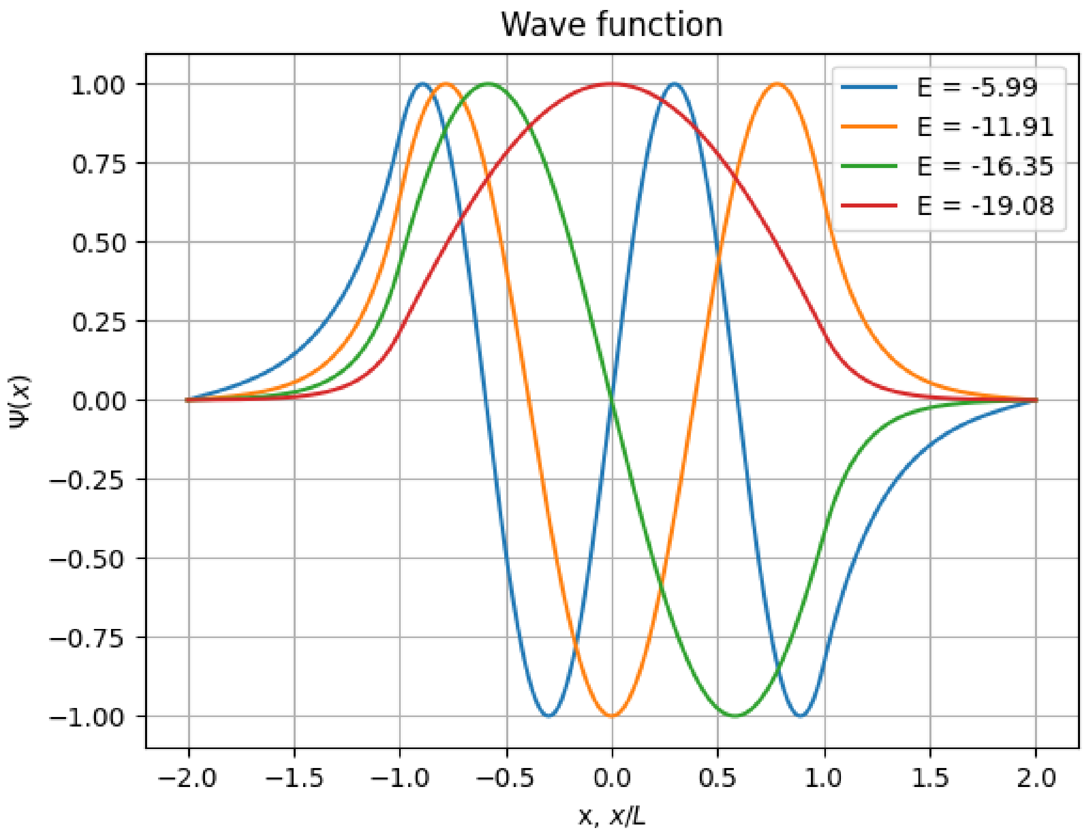
Code

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
from scipy.optimize import brentq


def V(x):

    L = 1
    if abs(x)>L:
        return 0
    else:
        return Vo

def plotpotential(x):

    pot = []
    for i in x:
        pot.append(V(i))
    return(np.array(pot))

def SE(psi,x):
    state0 = psi[1]
    state1 = 2.0*(V(x)-E)*psi[0]
    return np.array([state0,state1])

def Wave_function(energy):

    global psi
    global E
    E = energy
    psi = odeint(SE, psi0,x)
    return psi[-1,0]

def find_all_zeros(x,y):

    all_zeros = []
    s = np.sign(y)
    for i in range(len(y)-1):
        if s[i]+s[i+1]==0:
            zero = brentq(Wave_function,x[i],x[i+1])
            all_zeros.append(zero)
    return all_zeros


N = 1000
psi = np.zeros([N,2])
psi0 = np.array([0,1])
Vo = -20
E = 0.0
b = 2
x = np.linspace(-b,b,N)

def main():

    en = np.linspace(0,Vo,100)

    psi_b = []

    for e in en:
        psi_b.append(Wave_function(e))

    E_zeros = find_all_zeros(en,psi_b)

    print("Energy of the bound states")
    #print(E_zeros, psi_b)
    for E in E_zeros:
        print("Energy = ", E)
    plt.plot(en/Vo,psi_b)
    plt.scatter(np.array(E_zeros)/Vo,np.zeros_like(E_zeros))
    plt.xlabel('Energy,$E/V_0$')
    plt.ylabel('$\Psi(x=b)$')
    plt.grid()
    plt.show()

    for E in E_zeros:
        Wave_function(E)
        plt.plot(x,(psi[:,0]/np.max(psi[:,0])), label="E = %.2f"%E)
    #plt.plot(x,plotpotential(x), label = "Potential")
    plt.grid()
    plt.title("Wave function")
    plt.xlabel('x, $x/L$')
    plt.ylabel('$\Psi(x)$')
    plt.legend()
    plt.show()



#if __name__ =="__main__":
main()
```

Screening for Possible Energy Values for the finite square well potential

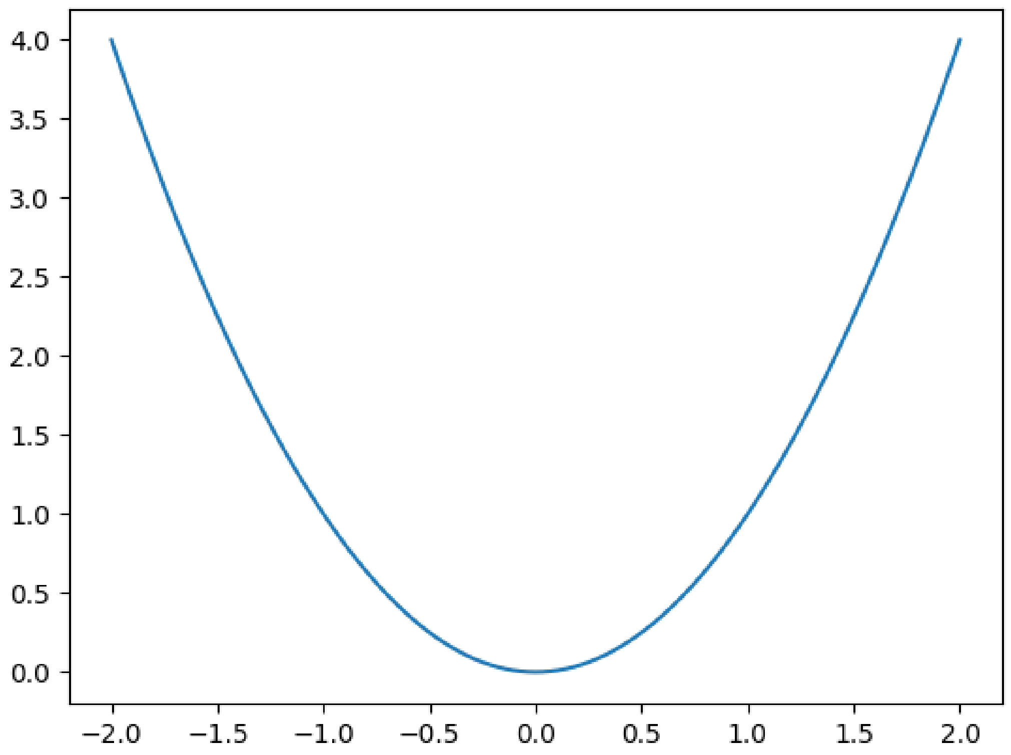


Calculated wave functions plot



Harmonic Oscillator potential

$$V(x) = \frac{1}{2}kx^2 = \frac{1}{2}m\omega x^2$$



Schrodinger's equation

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x) + \frac{m\omega x^2}{2}\psi(x) = E\psi(x)$$

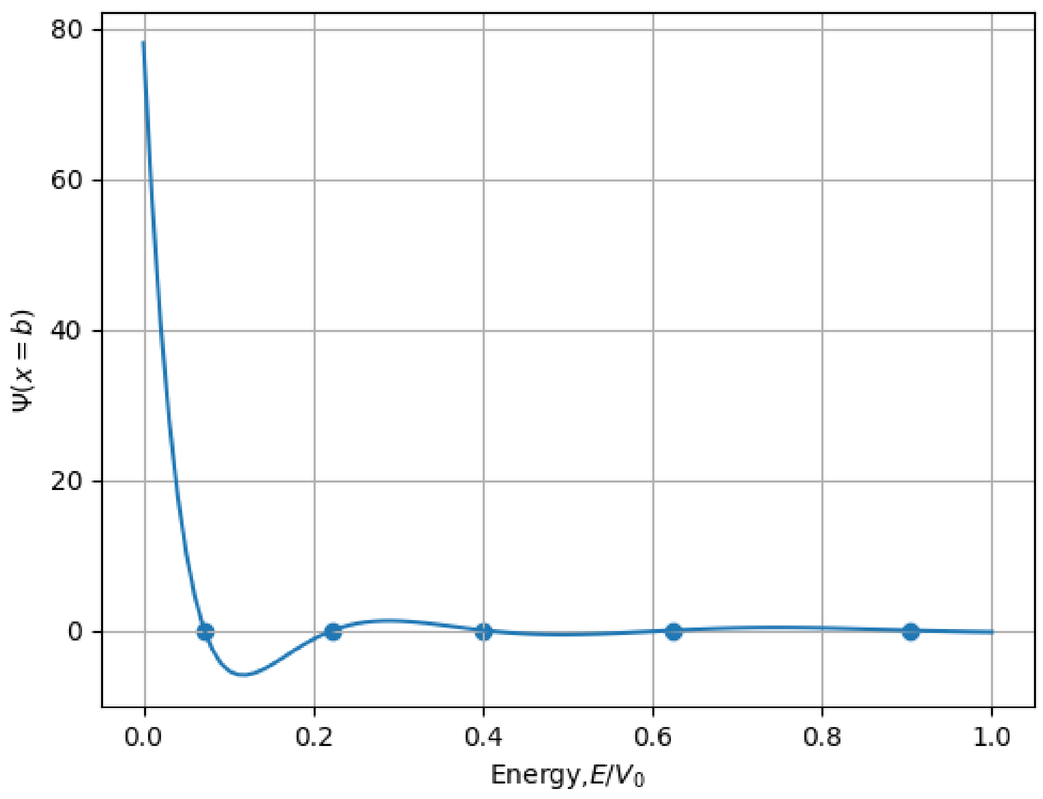
Boundary conditions

$$\psi(x \to -\infty) = 0, \psi(x \to \infty) = 0$$

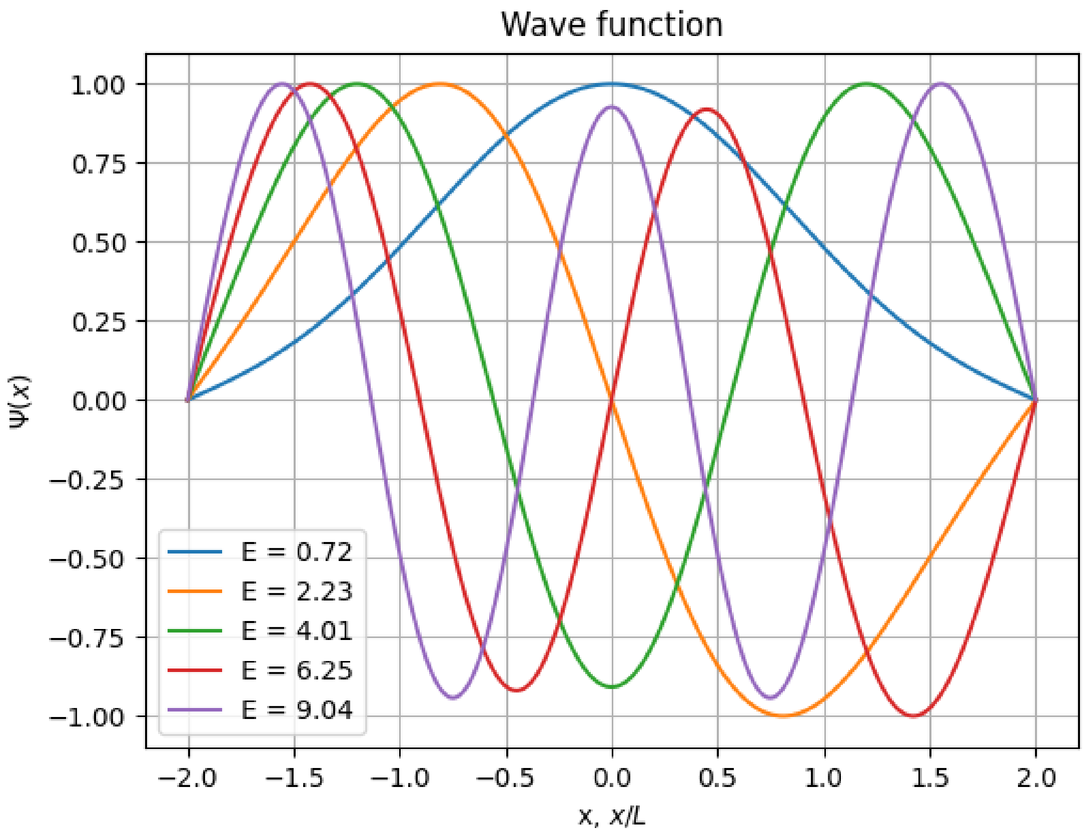
Code

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
from scipy.optimize import brentq


def V(x):

    return x ** 2


def plotpotential(x):
    pot = []
    for i in x:
        pot.append(V(i))
    return np.array(pot)


def plot_potential():
    x = np.linspace(-2, 2, 1000)
    plt.plot(x, plotpotential(x), label="Potential")
    plt.show()


def SE(psi, x):
    state0 = psi[1]
    state1 = 2.0 * (V(x) - E) * psi[0]
    return np.array([state0, state1])


def Wave_function(energy):

    global psi
    global E
    E = energy
    psi = odeint(SE, psi0, x)
    return psi[-1, 0]


def find_all_zeros(x, y):

    all_zeros = []
    s = np.sign(y)
    for i in range(len(y) - 1):
        if s[i] + s[i + 1] == 0:
            zero = brentq(Wave_function, x[i], x[i + 1])
            all_zeros.append(zero)
    return all_zeros


N = 1000
psi = np.zeros([N, 2])
psi0 = np.array([0, 1])
Emax = 10
E = 0.0
b = 2
x = np.linspace(-b, b, N)


def main():

    en = np.linspace(0, Emax, 100)

    psi_b = []

    for e in en:
        psi_b.append(Wave_function(e))

    E_zeros = find_all_zeros(en, psi_b)
    print("Energy of the bound states")
    # print(E_zeros, psi_b)
    for E in E_zeros:
        print("Energy = ", E)
    plt.plot(en / Emax, psi_b)
    plt.scatter(np.array(E_zeros) / Emax, np.zeros_like(E_zeros))
    plt.xlabel("Energy,$E/V_0$")
    plt.ylabel("$\Psi(x=b)$")
    plt.grid()
    plt.show()

    for E in E_zeros:
        Wave_function(E)
        plt.plot(x, (psi[:, 0] / np.max(psi[:, 0])), label="E = %.2f" % E)
    # plt.plot(x,plotpotential(x), label = "Potential")
    plt.grid()
    plt.title("Wave function")
    plt.xlabel("x, $x/L$")
    plt.ylabel("$\Psi(x)$")
    plt.legend()
    plt.show()
    plot_potential()


if __name__ == "__main__":
    main()
```

Screening for Possible Energy Values for the finite square well potential



Calculated wave functions plot

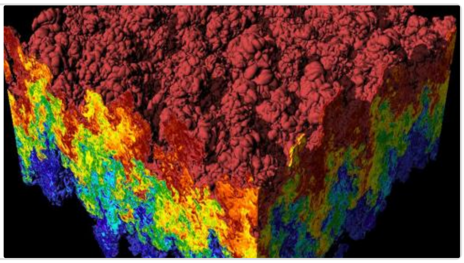


Morse potential

Morse potential - Wikipedia

The Morse potential, named after physicist Philip M. Morse, is a convenient interatomic interaction model for the potential energy of a diatomic molecule. It is a better approximation for the

W https://en.wikipedia.org/wiki/Morse_potential

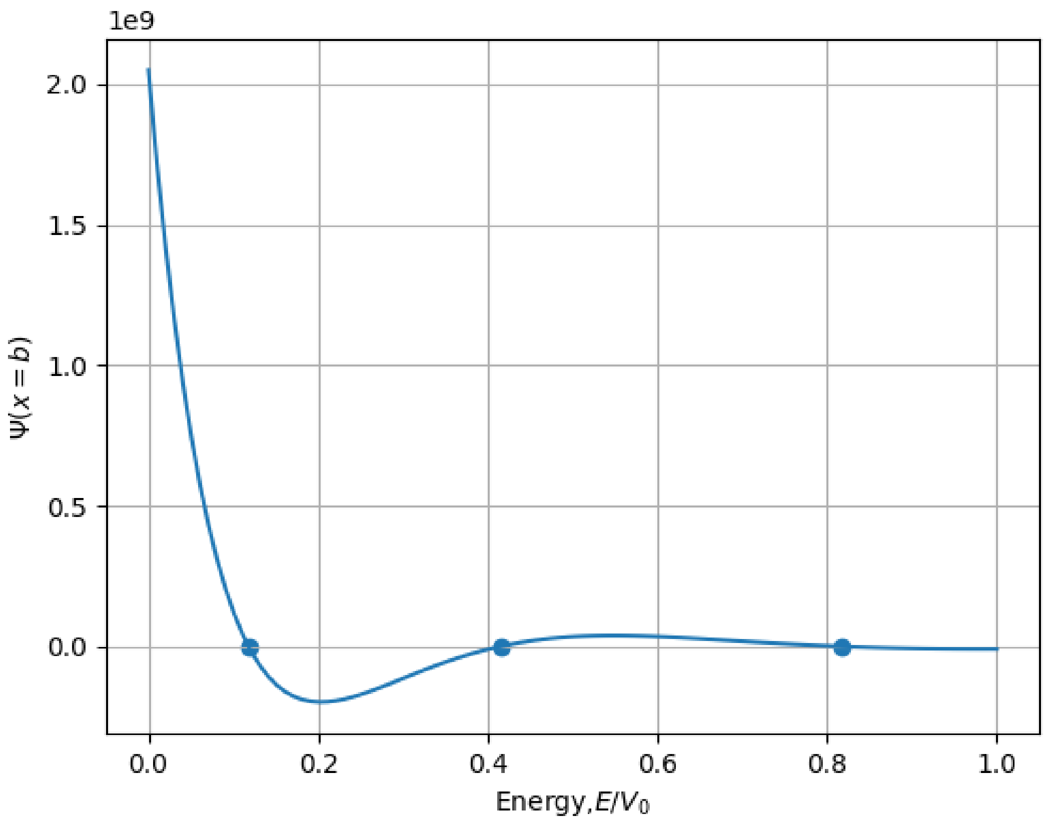


Code

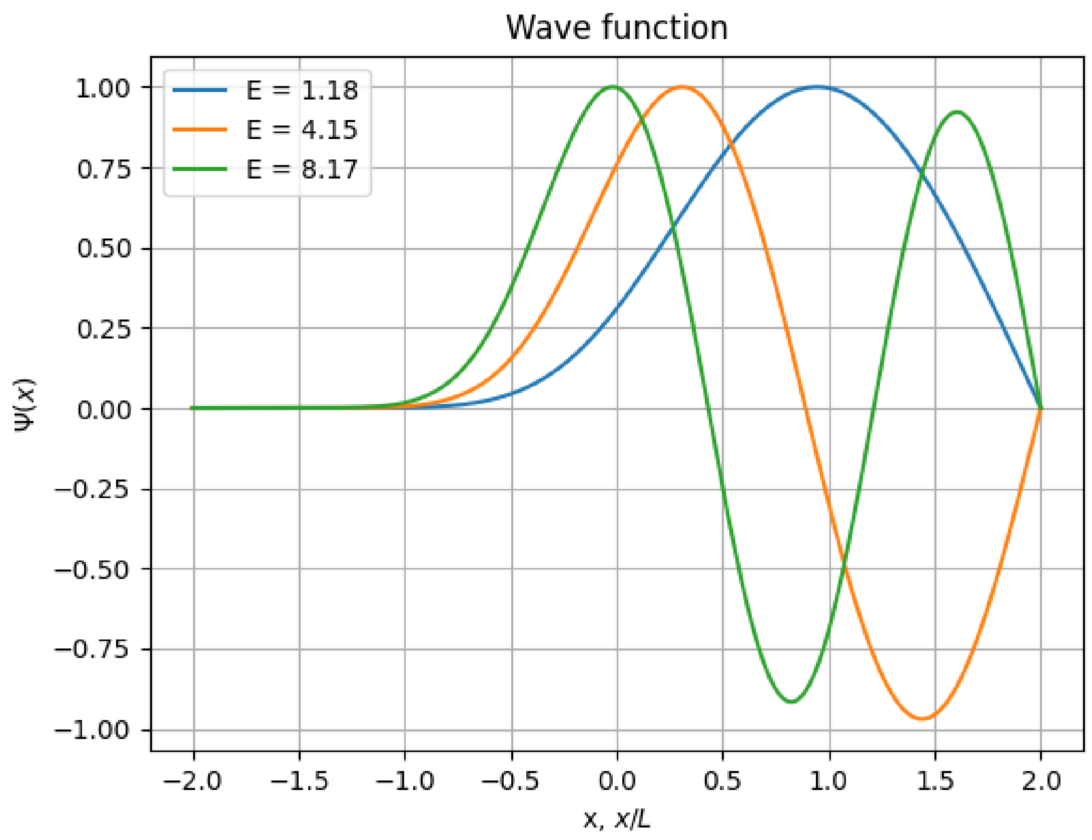
```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
from scipy.optimize import brentq


def V(x):

    return (1 - np.exp(-(x - 1))) ** 2
```

```
def plotpotential(x):
    pot = []
    for i in x:
        pot.append(V(i))
    return np.array(pot)


def SE(psi, x):
    state0 = psi[1]
    state1 = 2.0 * (V(x) - E) * psi[0]
    return np.array([state0, state1])


def Wave_function(energy):

    global psi
    global E
    E = energy
    psi = odeint(SE, psi0, x)
    return psi[-1, 0]


def find_all_zeros(x, y):

    all_zeros = []
    s = np.sign(y)
    for i in range(len(y) - 1):
        if s[i] + s[i + 1] == 0:
            zero = brentq(Wave_function, x[i], x[i + 1])
            all_zeros.append(zero)
    return all_zeros


N = 1000
psi = np.zeros([N, 2])
psi0 = np.array([0, 1])
Emax = 10
E = 0.0
b = 2
x = np.linspace(-b, b, N)


def main():

    en = np.linspace(0, Emax, 100)

    psi_b = []

    for e in en:
        psi_b.append(Wave_function(e))

    E_zeros = find_all_zeros(en, psi_b)

    print("Energy of the bound states")
    # print(E_zeros, psi_b)
    for E in E_zeros:
        print("Energy = ", E)
    plt.plot(en / Emax, psi_b)
    plt.scatter(np.array(E_zeros) / Emax, np.zeros_like(E_zeros))
    plt.xlabel("Energy,$E/V_0$")
    plt.ylabel("$\Psi(x=b)$")
    plt.grid()
    plt.show()

    for E in E_zeros:
        Wave_function(E)
        plt.plot(x, (psi[:, 0] / np.max(psi[:, 0])), label="E = %.2f" % E)
    # plt.plot(x,plotpotential(x), label = "Potential")
    plt.grid()
    plt.title("Wave function")
    plt.xlabel("x, $x/L$")
    plt.ylabel("$\Psi(x)$")
    plt.legend()
    plt.show()


if __name__ == "__main__":
    main()
```

Screening for Possible Energy Values for the finite square well potential



Calculated wave functions plot

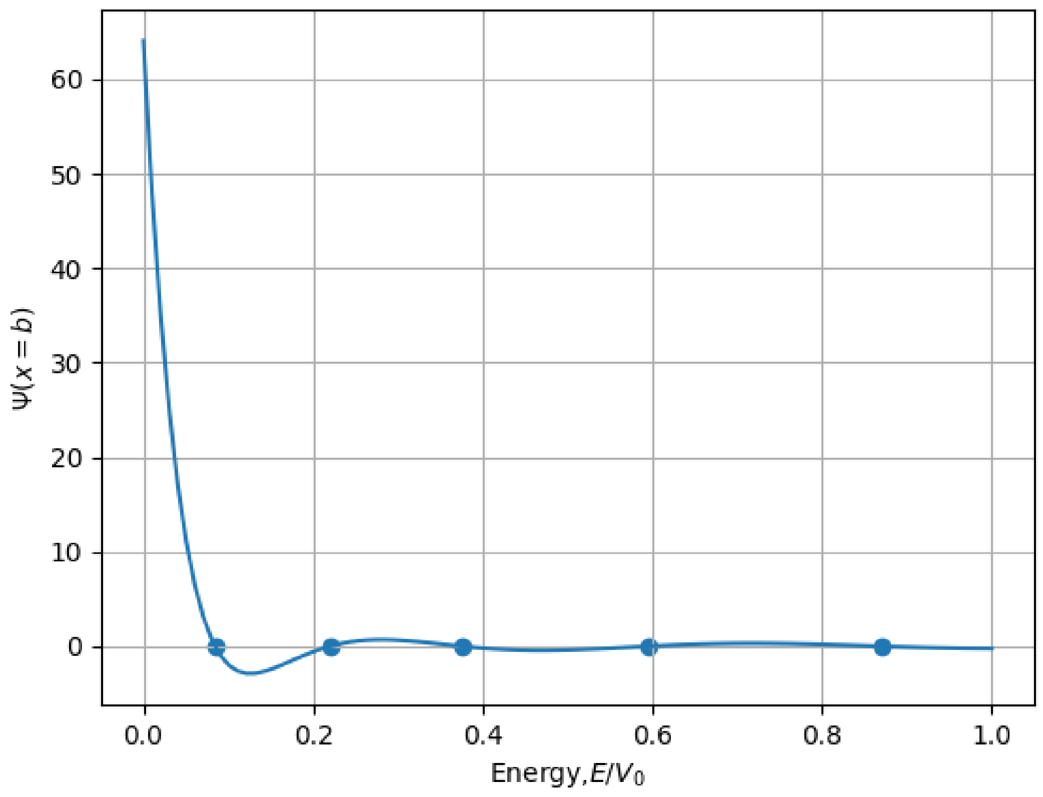


Triangular well potential

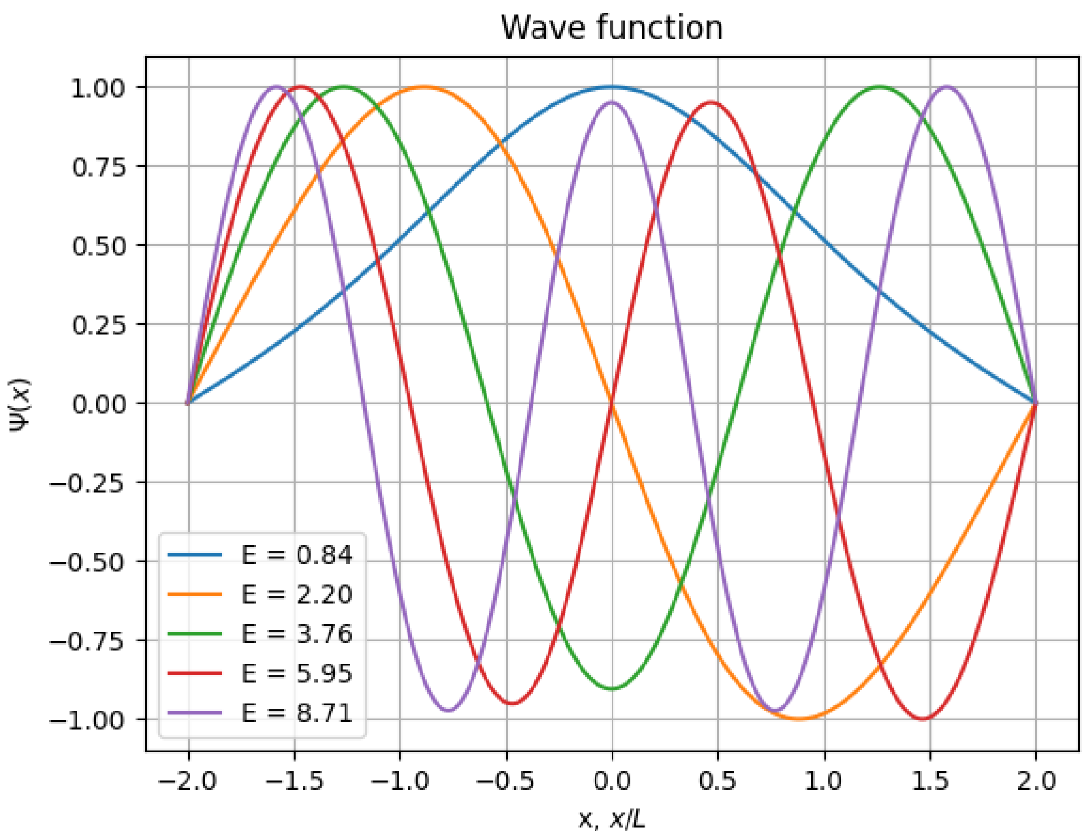
```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
from scipy.optimize import brentq


def V(x):

    return np.abs(x)


def plotpotential(x):
    pot = []
    for i in x:
        pot.append(V(i))
    return np.array(pot)


def SE(psi, x):
    state0 = psi[1]
    state1 = 2.0 * (V(x) - E) * psi[0]
    return np.array([state0, state1])
```

```
def Wave_function(energy):

    global psi
    global E
    E = energy
    psi = odeint(SE, psi0, x)
    return psi[-1, 0]


def find_all_zeros(x, y):

    all_zeros = []
    s = np.sign(y)
    for i in range(len(y) - 1):
        if s[i] + s[i + 1] == 0:
            zero = brentq(Wave_function, x[i], x[i + 1])
            all_zeros.append(zero)
    return all_zeros


N = 1000
psi = np.zeros([N, 2])
psi0 = np.array([0, 1])
Emax = 10
E = 0.0
b = 2
x = np.linspace(-b, b, N)


def main():

    en = np.linspace(0, Emax, 100)

    psi_b = []

    for e in en:
        psi_b.append(Wave_function(e))

    E_zeros = find_all_zeros(en, psi_b)

    print("Energy of the bound states")
    # print(E_zeros, psi_b)
    for E in E_zeros:
        print("Energy = ", E)
    plt.plot(en / Emax, psi_b)
    plt.scatter(np.array(E_zeros) / Emax, np.zeros_like(E_zeros))
    plt.xlabel("Energy,$E/V_0$")
    plt.ylabel("$\Psi(x=b)$")
    plt.grid()
    plt.show()

    for E in E_zeros:
        Wave_function(E)
        plt.plot(x, (psi[:, 0] / np.max(psi[:, 0])), label="E = %.2f" % E)
    # plt.plot(x,plotpotential(x), label = "Potential")
    plt.grid()
    plt.title("Wave function")
    plt.xlabel("x, $x/L$")
    plt.ylabel("$\Psi(x)$")
    plt.legend()
    plt.show()


if __name__ == "__main__":
    main()
```

Screening for Possible Energy Values for the finite square well potential



Calculated wave functions plot



Time Evaluation of Wave Packet