Ramakrishna Mission Vivekananda Centenary College
Rahara, Kolkata, West Bengal - 700118
College with Potential for Excellence (CPE)
Accredited by NAAC with Grade A++

PHSA CC-XI: Quantum Mechanics and Applications

Laboratory Notebook

Rohan Deb Sarkar
Undergraduate Student
Department of Physics
Ramakrishna Mission Vivekananda Centenary College

End Semester Examination

Subject Code: **PHSA CC-XI**
Registration Number: **A01-1112-111-020-2021**
Eamination Roll Number:

December, 2023

Contents

1 Shooting Method

The Shooting Method is a numerical technique used to solve Boundary Value Problems (BVPs) for ordinary differential equations. It works by converting a BVP into a set of Initial Value Problems (IVPs) and then iteratively refining the solution.

The method begins by taking the boundary condition at one end of the interval and making an initial "guess" for the missing initial condition. This guessed initial condition, along with the known initial condition, forms a set of initial conditions for an IVP.

This IVP is then solved using a numerical method such as Euler's method or one of the Runge-Kutta methods. The solution obtained from this process is then checked against the known boundary condition at the other end of the interval.

If the solution does not satisfy the boundary condition, the initial guess is adjusted and the process is repeated. This iterative process continues until the solution sufficiently satisfies the boundary condition.

The Shooting Method is particularly useful when the boundary value problem is difficult to solve analytically.

[1]:
```
import numpy as np
from scipy import integrate, optimize
from matplotlib import pyplot as plt

# Default configuaration for matplotlib
import scienceplots
plt.style.use(['science','ieee'])
plt.rcParams["figure.figsize"] = (10, 6)

# Natural units
hbar = m = 1
```

[2]:
```
# Euler method for solving differential equations
def euler(ypp, xi, xf, yi, ypi, n=1000, args=None):
    x, dx = np.linspace(xi, xf, n, retstep=True)

    yp = np.empty(n)
    y = np.empty(n)

    yp[0] = ypi
    y[0] = yi

    for i in range(1, n):
        yp[i] = yp[i-1] + dx * ypp(x[i], y[i-1], yp[i-1], *args)
        y[i] = y[i-1] + dx * yp[i-1]

    return x, y
```

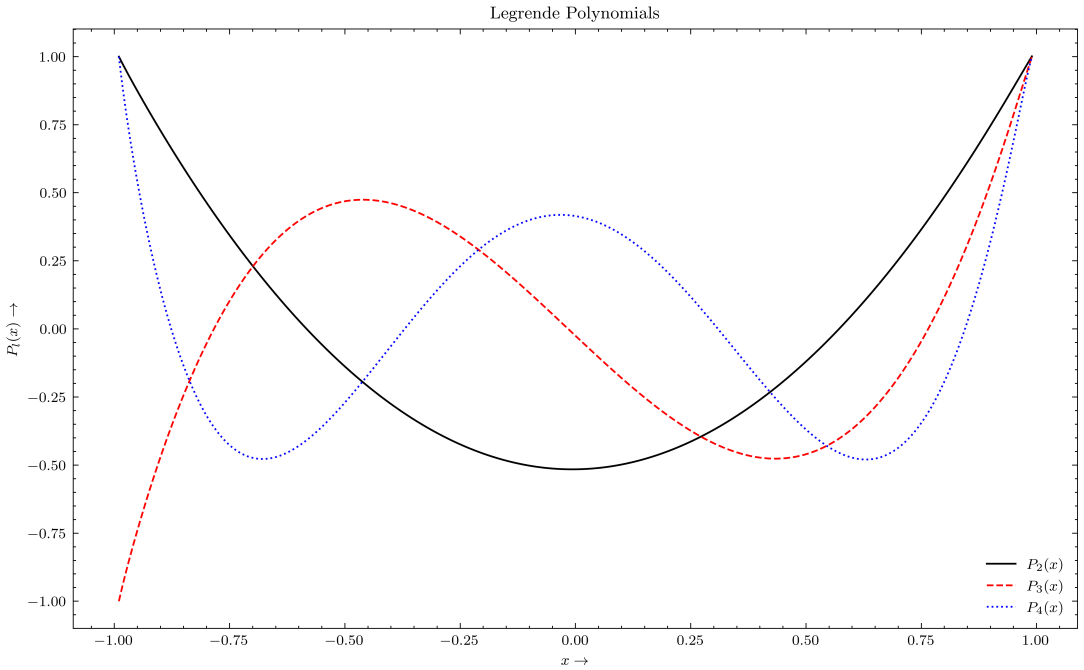
1.1 Boundary-value Problems

[3]:
```
# Function for solving Schrodinger's equation using shooting method
def solve_bvp(ypp, xi, xf, yi, yf, ypi_i, ypi_f, ypi_h, args=None):
    ypi = np.arange(ypi_i, ypi_f, ypi_h)
    Y_f = np.empty(ypi.size)

    for i in range(ypi.size):
        x, y = euler(ypp, xi, xf, yi, ypi[i], args=args)
        Y_f[i] = y[-1]

        if i and (Y_f[i] - yf) * (Y_f[i-1] - yf) < 0:
```

```
            return x, ypi[i], y
```

1.1.1 Legrende Differential Equation

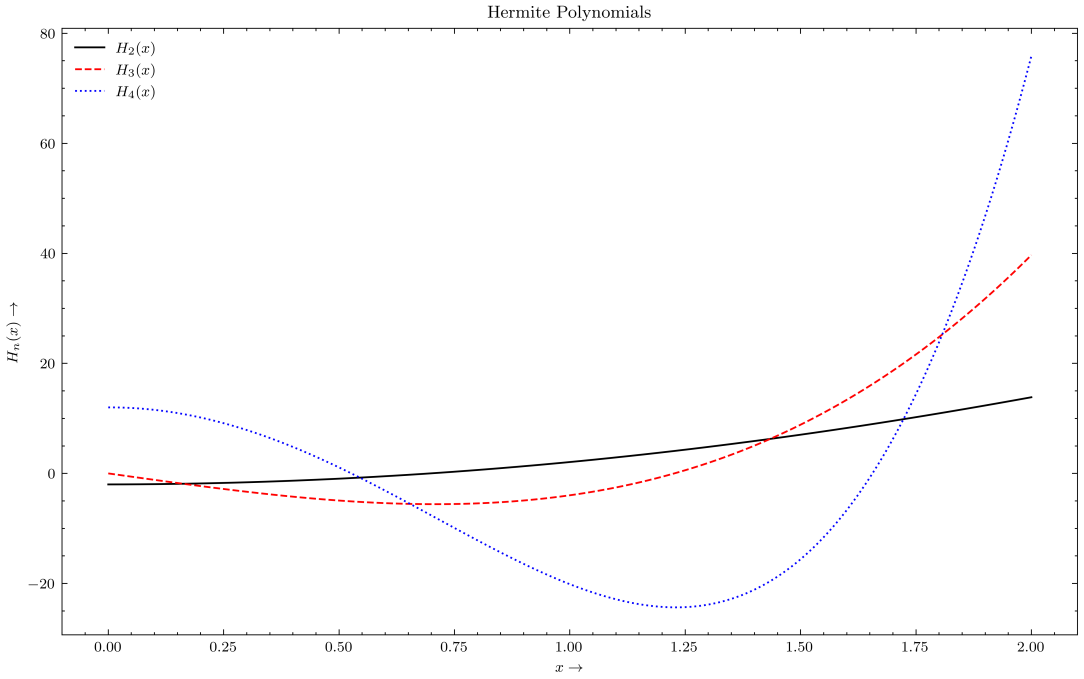
[4]:
```
def ypp(x, y, yp, l):
    return (2 * x * yp - l * (l + 1) * y) / (1 - x**2)

ypi_i, ypi_f, ypi_h = -20, 20, 0.1

l =  2, 3, 4
xi = - 0.99
xf = 0.99
yi = 1, -1, 1
yf = 1, 1, 1

for k in range(len(l)):
    x, ypi, y = solve_bvp(ypp, xi, xf, yi[k], yf[k], ypi_i, ypi_f, ypi_h,␣
 ↪args=(l[k],))
    plt.plot(x, y, label=f"$P_{l[k]}(x)$")

plt.title("Legrende Polynomials")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$P_l(x) \\rightarrow$")
plt.legend()
plt.show()
```



1.1.2 Hermite Differential Equation

[5]:
```
def ypp(x, y, yp, n):
    return 2 * x * yp - 2 * n * y

ypi_i, ypi_f, ypi_h = -20, 20, 0.1
```

```
n =  2, 3, 4
xi = 0
xf = 2
yi = -2, 0, 12
yf = 14, 40, 76

for k in range(len(n)):
    x, ypi, y = solve_bvp(ypp, xi, xf, yi[k], yf[k], ypi_i, ypi_f, ypi_h,␣
 ↪args=(n[k],))
    plt.plot(x, y, label=f"$H_{n[k]}(x)$")

plt.title("Hermite Polynomials")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$H_n(x) \\rightarrow$")
plt.legend()
plt.show()
```



1.2 Eigenvalue Problems

[6]:
```
# Default configuaration for matplotlib
plt.rcParams["figure.figsize"] = (10, 16)
```

[7]:
```
def solve_sch_eqn(ypp, xi, xf, E_n, yi=0, yf=0, ypi=1, n=1000):
    eigenvalues = np.zeros(E_n)
    eigenvectors = []

    eigenvalues[0] = 0

    for i in range(E_n):
        eigenvalues[i] = optimize.newton(shoot, eigenvalues[i], args=(ypp, xi, xf,␣
 ↪yi, yf, ypi, n))
```

```
        x, y = euler(ypp, xi, xf, yi, ypi, n, args=(eigenvalues[i],))
        eigenvectors.append(y.copy())

        if i+1 < E_n:
            eigenvalues[i+1] = 2 * eigenvalues[i] - (eigenvalues[i-1] if i >0 else␣
↪0) + eigenvalues[0] / (i+1)

    return x, eigenvalues, np.array(eigenvectors)
```

[8]:
```
def shoot(E, ypp, xi, xf, yi, yf, ypi, n):
    x, y = euler(ypp, xi, xf, yi, ypi, n, args=(E,))

    return yf - y[-1]
```

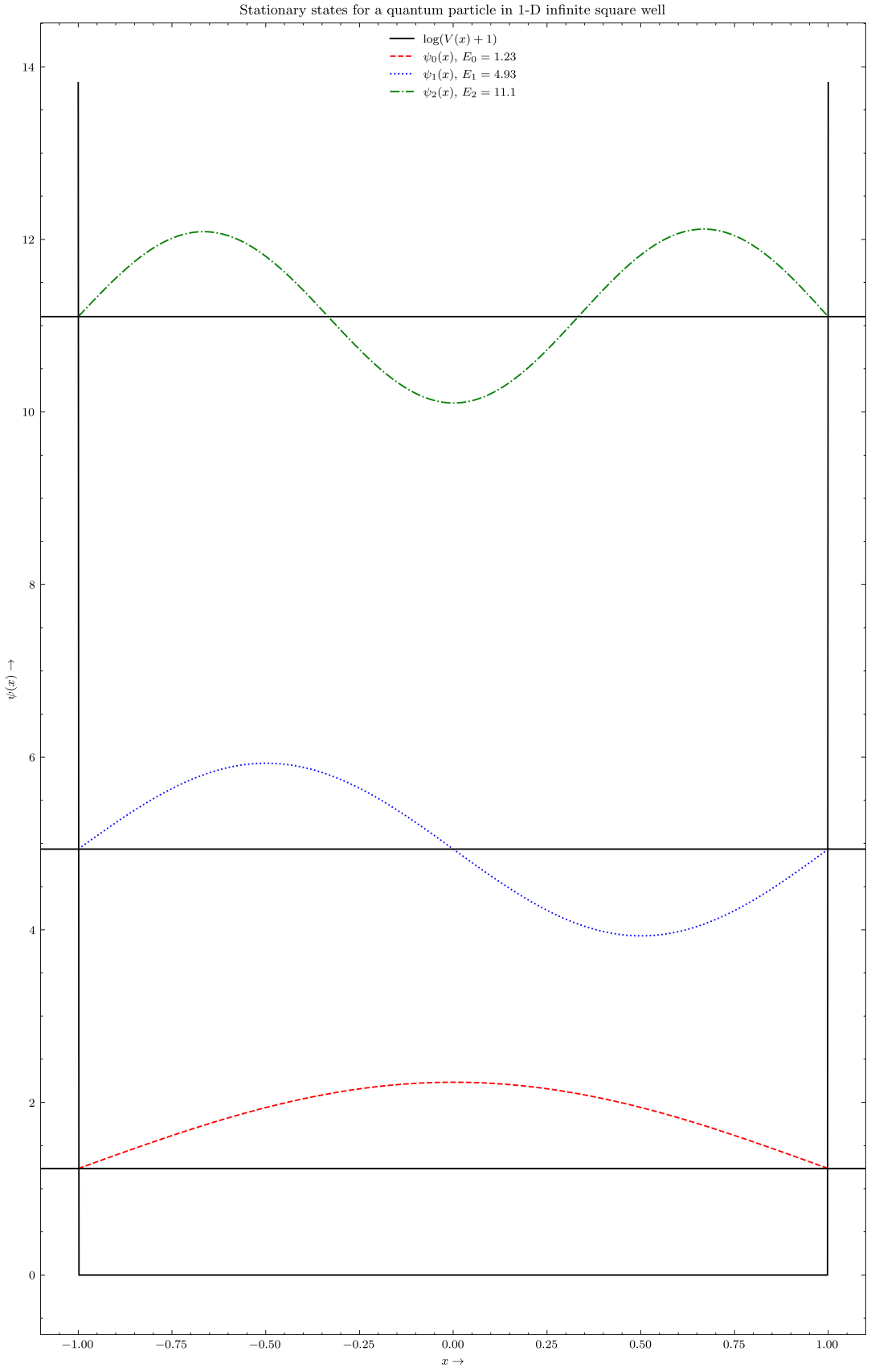
[9]:
```
def sch_eqn(V):
    def ypp(x, y, yp, E):
        return - (2 * m / hbar**2) * (E - V(x)) * y
    return ypp
```

1.2.1 Infinite square potential well

$$V(x) = \begin{cases} 0, & -a < x < a \\ \lim\limits_{V_0 \to \infty} V_0, & \text{otherwise} \end{cases}$$

[10]:
```
a = 1
V_0 = 1e6

@np.vectorize
def V(x):
    if -a < x < a:
        return 0.0
    else:
        return V_0

xi, xf = -1, 1
E_n = 3

x, E, psi = solve_sch_eqn(sch_eqn(V), xi, xf, E_n)
```

[11]:
```
plt.plot(x, np.log(V(x) + 1), label="$\\log(V(x) + 1)$")

for i in range(E.size):
    plt.plot(x , E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D infinite square well")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```

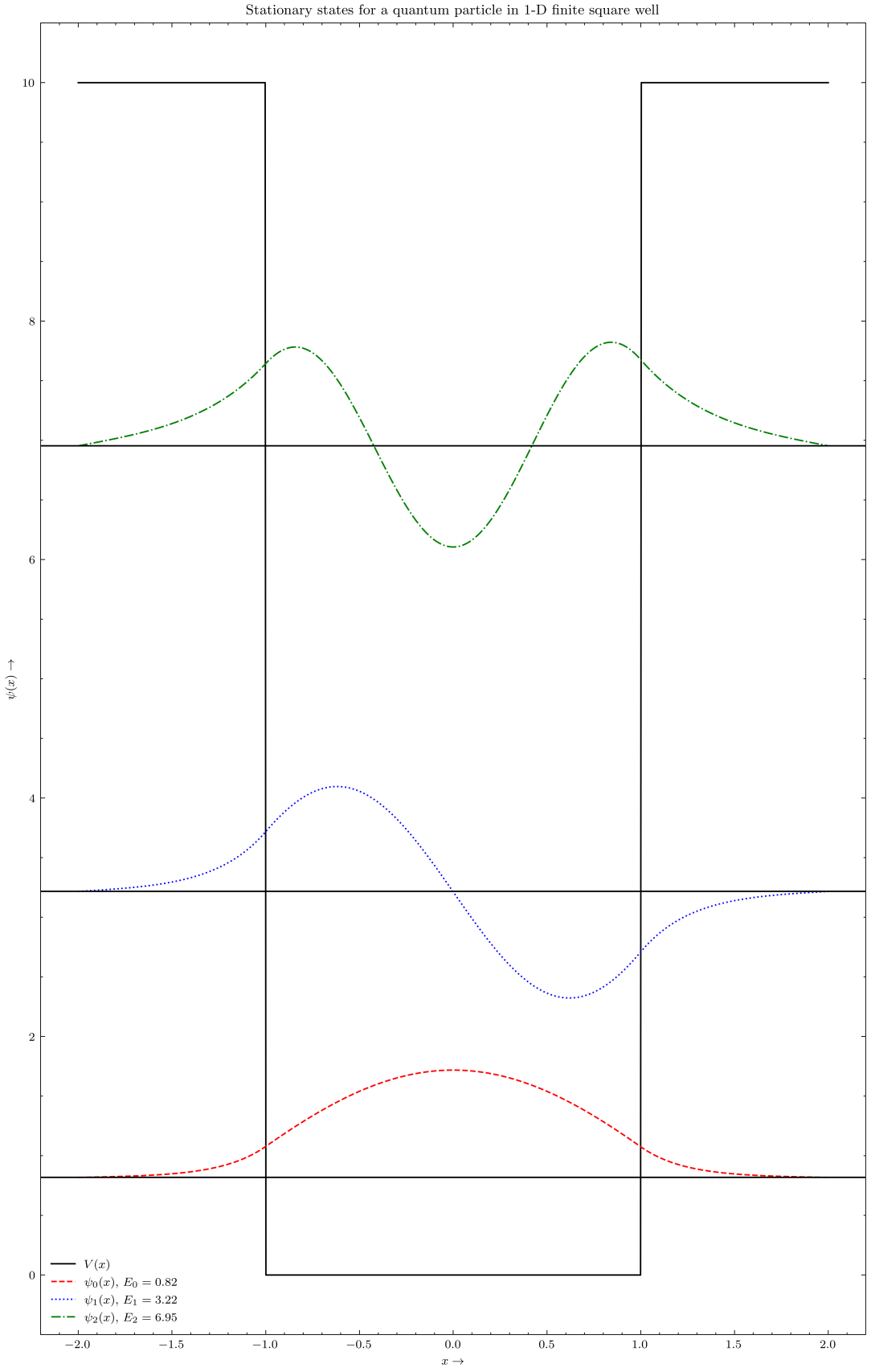

Stationary states for a quantum particle in 1-D infinite square well
log(V(x) + 1)
ψ0(x), E0 = 1.23
ψ1(x), E1 = 4.93
ψ2(x), E2 = 11.1
ψ(x) →
x →
0
2
4
6
8
10
12
14
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

1.2.2 Finite square potential well

$$V(x) = \begin{cases} 0, & -a < x < a \\ V_0, & \text{otherwise} \end{cases}$$

[12]:
```
a = 1
V_0 = 10.0

@np.vectorize
def V(x):
    if -a < x < a:
        return 0
    else:
        return V_0

xi, xf = -2, 2
E_n = 3

x, E, psi = solve_sch_eqn(sch_eqn(V), xi, xf, E_n)
```

[13]:
```
plt.plot(x, V(x), label="$V(x)$")

for i in range(E.size):
    plt.plot(x , E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D finite square well")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```

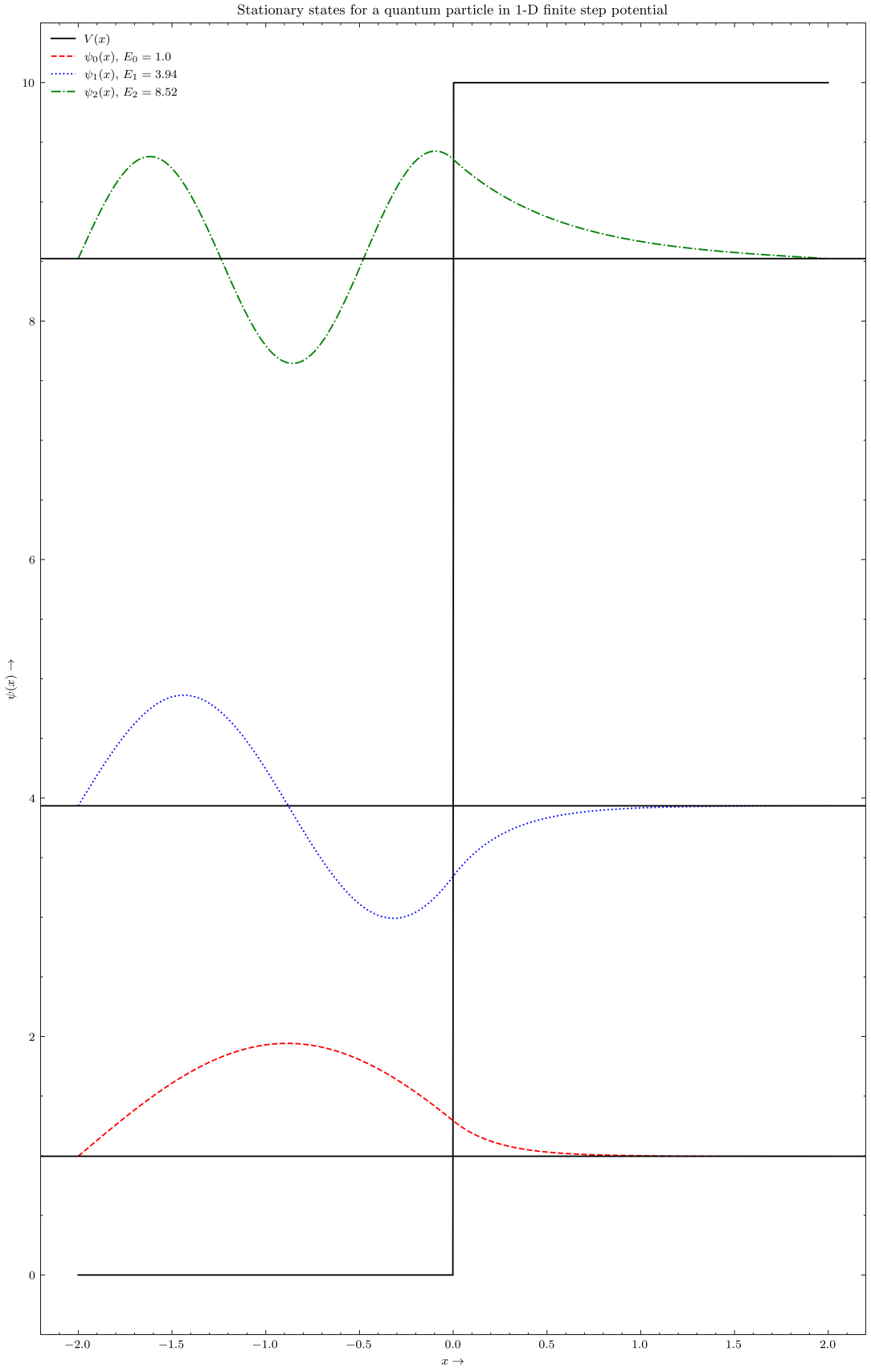

Stationary states for a quantum particle in 1-D finite square well
10
8
6
4
2
0
$\psi(x) \rightarrow$
$V(x)$
$\psi_0(x),\ E_0 = 0.82$
$\psi_1(x),\ E_1 = 3.22$
$\psi_2(x),\ E_2 = 6.95$
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
$x \rightarrow$

1.2.3 Step potential

$$V(x) = \begin{cases} 0, & x < 0 \\ V_0, & x \geq 0 \end{cases}$$

[14]:
```
a = 1
V_0 = 10

@np.vectorize
def V(x):
    if x < 0:
        return 0.0
    else:
        return V_0

xi, xf = -2, 2
E_n = 3

x, E, psi = solve_sch_eqn(sch_eqn(V), xi, xf, E_n)
```

[15]:
```
plt.plot(x, V(x), label="$V(x)$")

for i in range(E_n):
    plt.plot(x, E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D finite step potential")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```

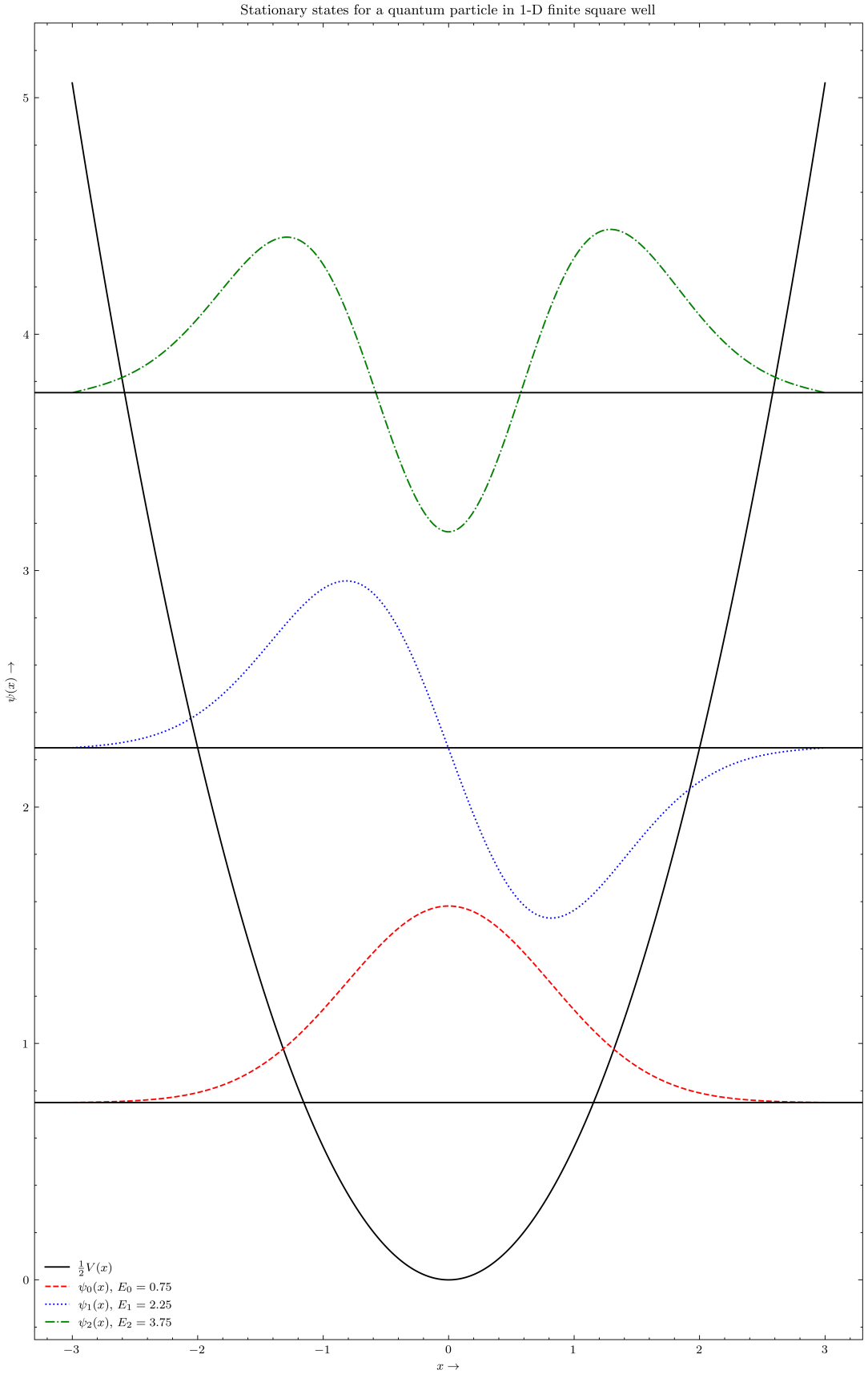

Stationary states for a quantum particle in 1-D finite step potential
$V(x)$
$\psi_0(x)$, $E_0 = 1.0$
$\psi_1(x)$, $E_1 = 3.94$
$\psi_2(x)$, $E_2 = 8.52$
10
8
6
4
2
0
$\psi(x) \rightarrow$
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
$x \rightarrow$

1.2.4 Linear Harmonic Oscillator

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

[16]:
```
omega = 1.5 # angular frequency

@np.vectorize
def V(x):
    return 0.5 * m * omega**2 * x**2

xi, xf = -3, 3
E_n = 3

x, E, psi = solve_sch_eqn(sch_eqn(V), xi, xf, E_n)
```

[17]:
```
plt.plot(x, 0.5*V(x), label="$\\frac{1}{2} V(x)$")

for i in range(E.size):
    plt.plot(x , E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D finite square well")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```

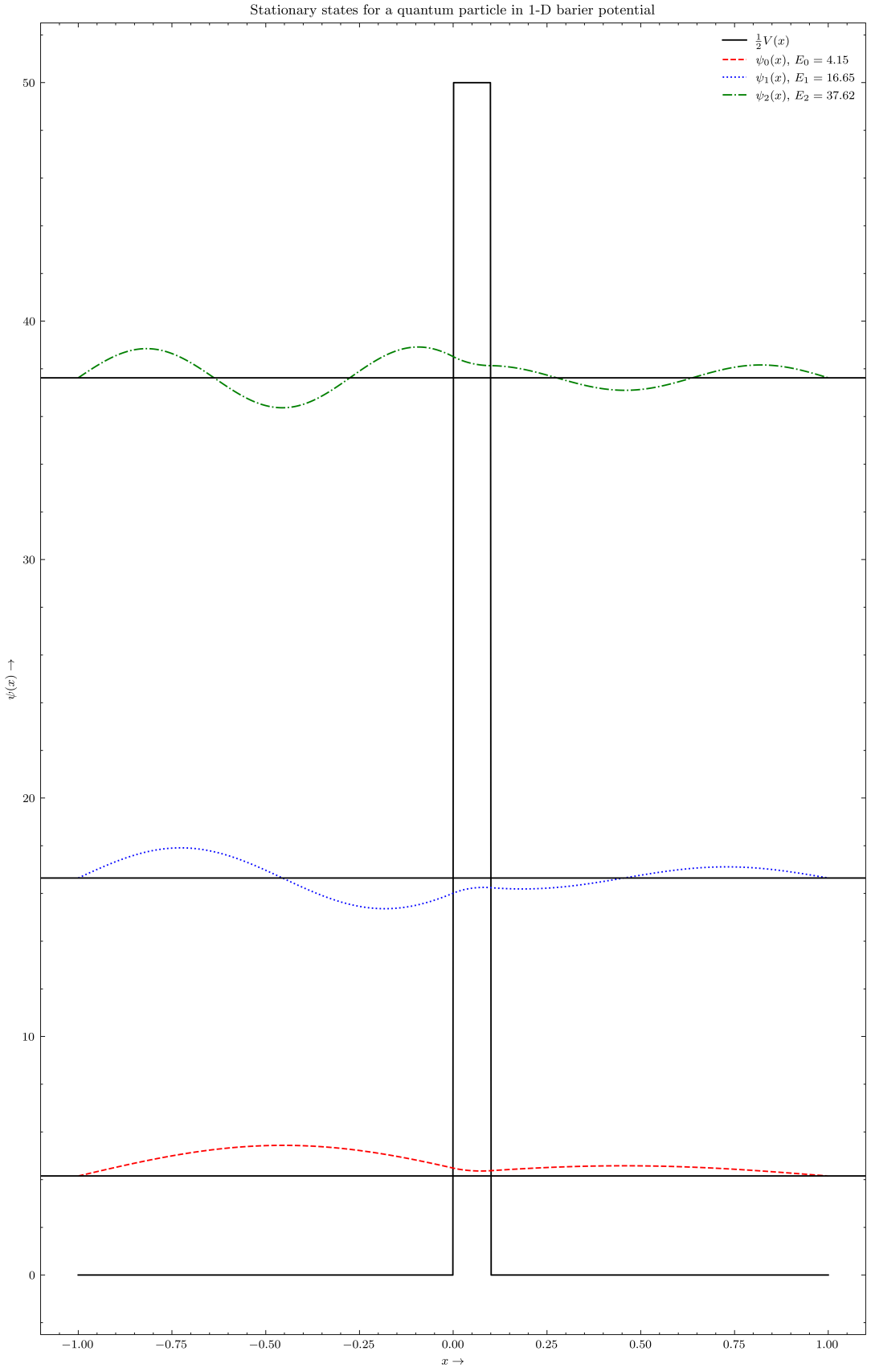

Stationary states for a quantum particle in 1-D finite square well
$\psi(x) \rightarrow$
$x \rightarrow$
$\frac{1}{2}V(x)$
$\psi_0(x)$, $E_0 = 0.75$
$\psi_1(x)$, $E_1 = 2.25$
$\psi_2(x)$, $E_2 = 3.75$

1.2.5 Barier Potential

$$V(x) = \begin{cases} V_0, & 0 < x \leq a \\ 0, & \text{otherwise} \end{cases}$$

[18]:
```
a = 0.1
V_0 = 100.0

@np.vectorize
def V(x):
    if 0 < x <= a:
        return V_0
    else:
        return 0.0

xi, xf = -1, 1
E_n = 3

x, E, psi = solve_sch_eqn(sch_eqn(V), xi, xf, E_n)
```

[19]:
```
plt.plot(x, 0.5 * V(x), label="$\\frac{1}{2}V(x)$")

for i in range(E_n):
    plt.plot(x, E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D barier potential")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```


Stationary states for a quantum particle in 1-D barier potential
$\frac{1}{2}V(x)$
$\psi_0(x),\ E_0 = 4.15$
$\psi_1(x),\ E_1 = 16.65$
$\psi_2(x),\ E_2 = 37.62$
$\psi(x) \rightarrow$
$x \rightarrow$
0
10
20
30
40
50
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

2 Finite difference method

Time Independent Schrödinger Equation

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

where,

ψ is the stationary state wave fuction of a quantum particle,

m is the mass of the quantum paricle,

$V = V(x)$ is the potential applied on the quantum particle, and

E is the energy of the quantum particle.

Let us discritize x by slicing it into n evenly spaced points $\{x_j\}$, with $\Delta x \equiv x_{j+1} - x_j$ and let $\psi_j \equiv \psi(x_j)$ and $V_j \equiv V(x_j)$. $j = 1, 2, 3 \ldots n$

Therefore,

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi_j + V_j\psi_j = E\psi_j$$

Using finite difference,

$$\frac{d^2}{dx^2}\Psi_j \approx \frac{\Psi_{j-1} - 2\Psi_j + \Psi_{j+1}}{(\Delta x)^2}$$

$$-\frac{\hbar^2}{2m}\left(\frac{\Psi_{j-1} - 2\Psi_j + \Psi_{j+1}}{(\Delta x)^2}\right) + V_j\psi_j = E\psi_j$$

let, $u = -\frac{\hbar^2}{2m(\Delta x)^2}$

$$u\psi_{j-1} + (V_j - 2u)\psi_j + u\psi_{j+1} = E\psi_j$$

In matrix form,

$$\hat{H}\psi = E\psi$$

$$\begin{pmatrix} \ddots & & & & & \\ & u & (V_{j-1} - 2u) & u & 0 & 0 \\ & 0 & u & (V_j - 2u) & u & 0 \\ & 0 & 0 & u & (V_{j+1} - 2u) & u \\ & & & & & \ddots \end{pmatrix} \begin{pmatrix} \vdots \\ \psi_{j-1} \\ \psi_j \\ \psi_{j+1} \\ \vdots \end{pmatrix} = E \begin{pmatrix} \vdots \\ \psi_{j-1} \\ \psi_j \\ \psi_{j+1} \\ \vdots \end{pmatrix}$$

This is an eigenvalue equation, solving for the eigenvalues and eigenvectors for the matrix $\hat{H}$ we will get the allowed energies $\{E\}$ and corresponding wave-functions $\{\psi\}$

[1]:
```
import numpy as np
from scipy import linalg, integrate
from matplotlib import pyplot as plt

# Default configuaration for matplotlib
import scienceplots
plt.style.use(['science','ieee'])
plt.rcParams["figure.figsize"] = (10, 16)

# Natural units
hbar = m = 1
```

2.1 Eigenvalue Problems

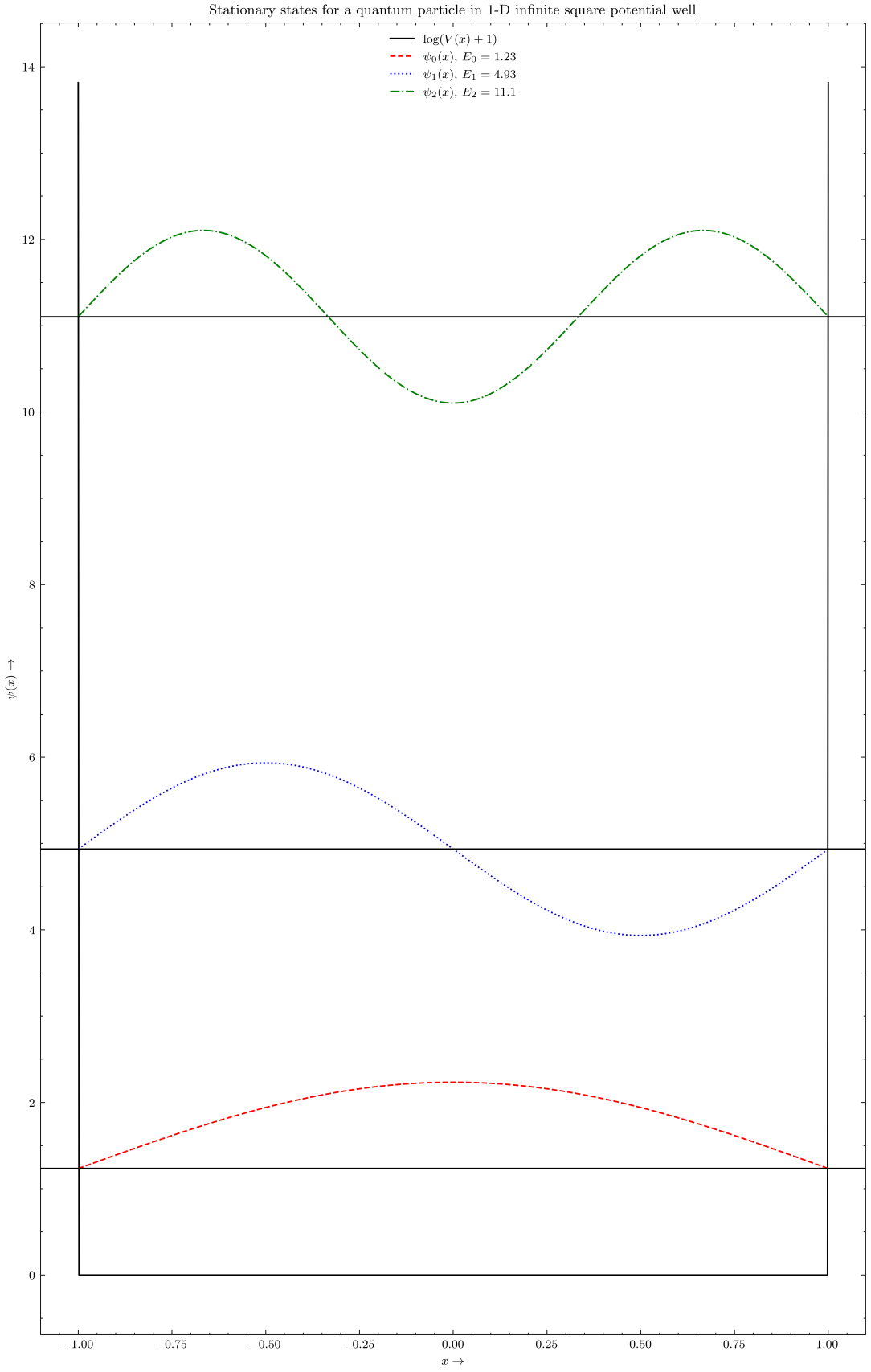
[2]:
```
# Function for solving Schrodinger's equation using finite difference method
def solve_sch_eqn(V, xi, xf, yi=0, yf=0, n=1000):
    x, dx = np.linspace(xi, xf, n, retstep=True)

    u = - hbar**2 / (2 * m * dx**2)

    diagonal = V(x)[1:n-1] - 2 * u
    off_diagonal = np.full(n-3, u)

    E, psi = linalg.eigh_tridiagonal(diagonal, off_diagonal)

    return x, E, np.array([np.full(psi.shape[1], yi), *psi, np.full(psi.shape[1],␣
 ↪yf)]).T
```

2.1.1 Ininite square potential well

$$V(x) = \begin{cases} 0, & -a < x < a \\ \lim\limits_{V_0 \to \infty} V_0, & \text{otherwise} \end{cases}$$

[3]:
```
a = 1
V_0 = 1e6

@np.vectorize
def V(x):
    if -a < x < a:
        return 0.0
    else:
        return V_0

xi, xf = -1, 1
E_n = 3

x, E, psi = solve_sch_eqn(V, xi, xf)
```

[4]:
```
plt.plot(x, np.log(V(x) + 1), label="$\log(V(x) + 1)$")

for i in range(E_n):
    plt.plot(x, E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D infinite square␣
 ↪potential well")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```

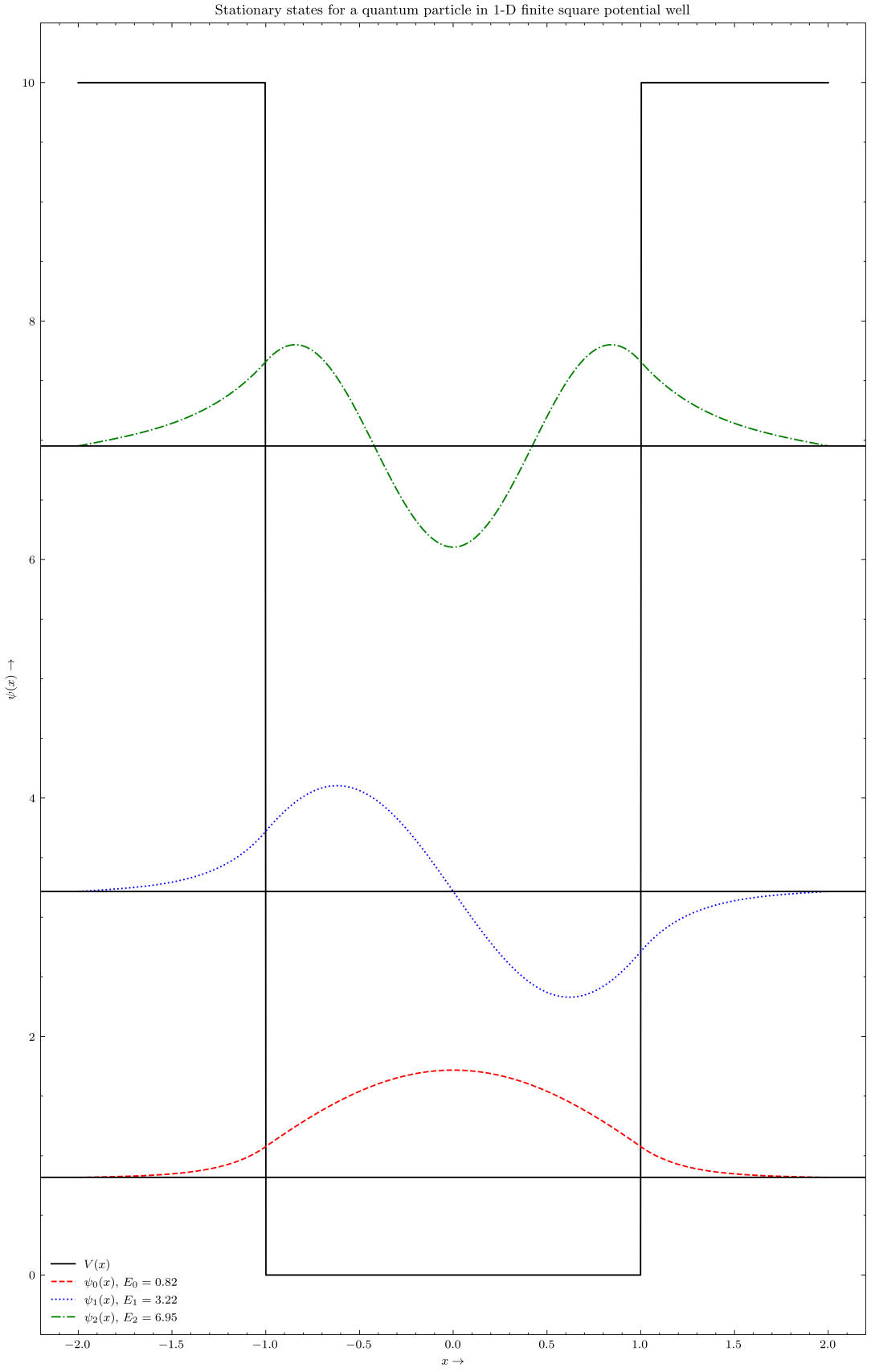

Stationary states for a quantum particle in 1-D infinite square potential well
log(V(x) + 1)
ψ0(x), E0 = 1.23
ψ1(x), E1 = 4.93
ψ2(x), E2 = 11.1
ψ(x) →
x →
0
2
4
6
8
10
12
14
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

2.1.2 Finite square potential well

$$V(x) = \begin{cases} 0, & -a < x < a \\ V_0, & \text{otherwise} \end{cases}$$

[5]:
```
a = 1
V_0 = 10.0

@np.vectorize
def V(x):
    if -a < x < a:
        return 0.0
    else:
        return V_0

xi, xf = -2, 2
E_n = 3

x, E, psi = solve_sch_eqn(V, xi, xf)
```

[6]:
```
plt.plot(x, V(x), label="$V(x)$")

for i in range(E_n):
    plt.plot(x, E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D finite square potential␣
 ↪well")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```

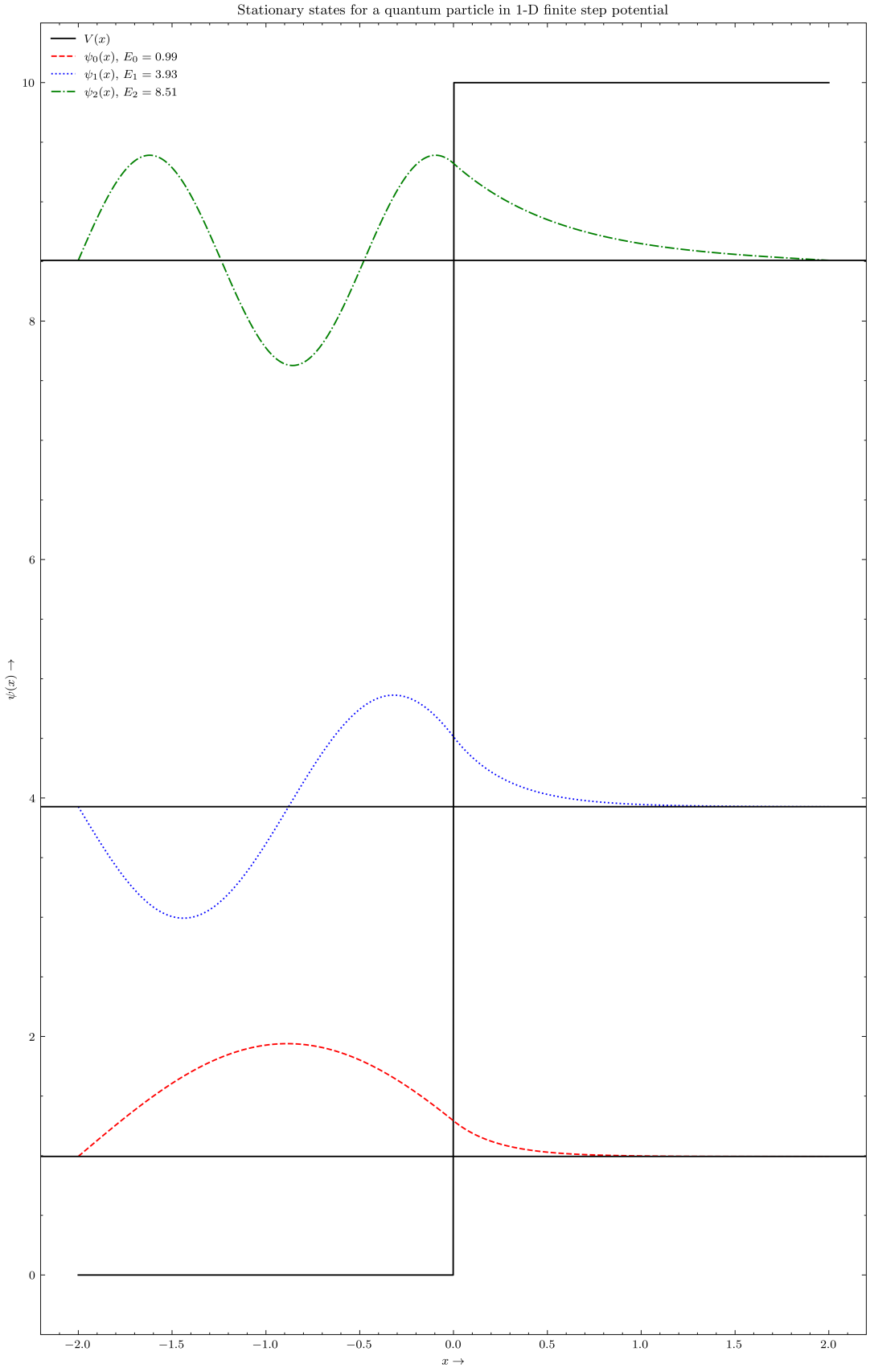

Stationary states for a quantum particle in 1-D finite square potential well
10
8
6
4
2
0
$\psi(x) \rightarrow$
$V(x)$
$\psi_0(x),\ E_0 = 0.82$
$\psi_1(x),\ E_1 = 3.22$
$\psi_2(x),\ E_2 = 6.95$
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
$x \rightarrow$

2.1.3 Step potential

$$V(x) = \begin{cases} 0, & x < 0 \\ V_0, & x \geq 0 \end{cases}$$

[7]:
```
a = 1
V_0 = 10

@np.vectorize
def V(x):
    if x < 0:
        return 0.0
    else:
        return V_0

xi, xf = -2, 2
E_n = 3

x, E, psi = solve_sch_eqn(V, xi, xf)
```

[8]:
```
plt.plot(x, V(x), label="$V(x)$")

for i in range(E_n):
    plt.plot(x, E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D finite step potential")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```

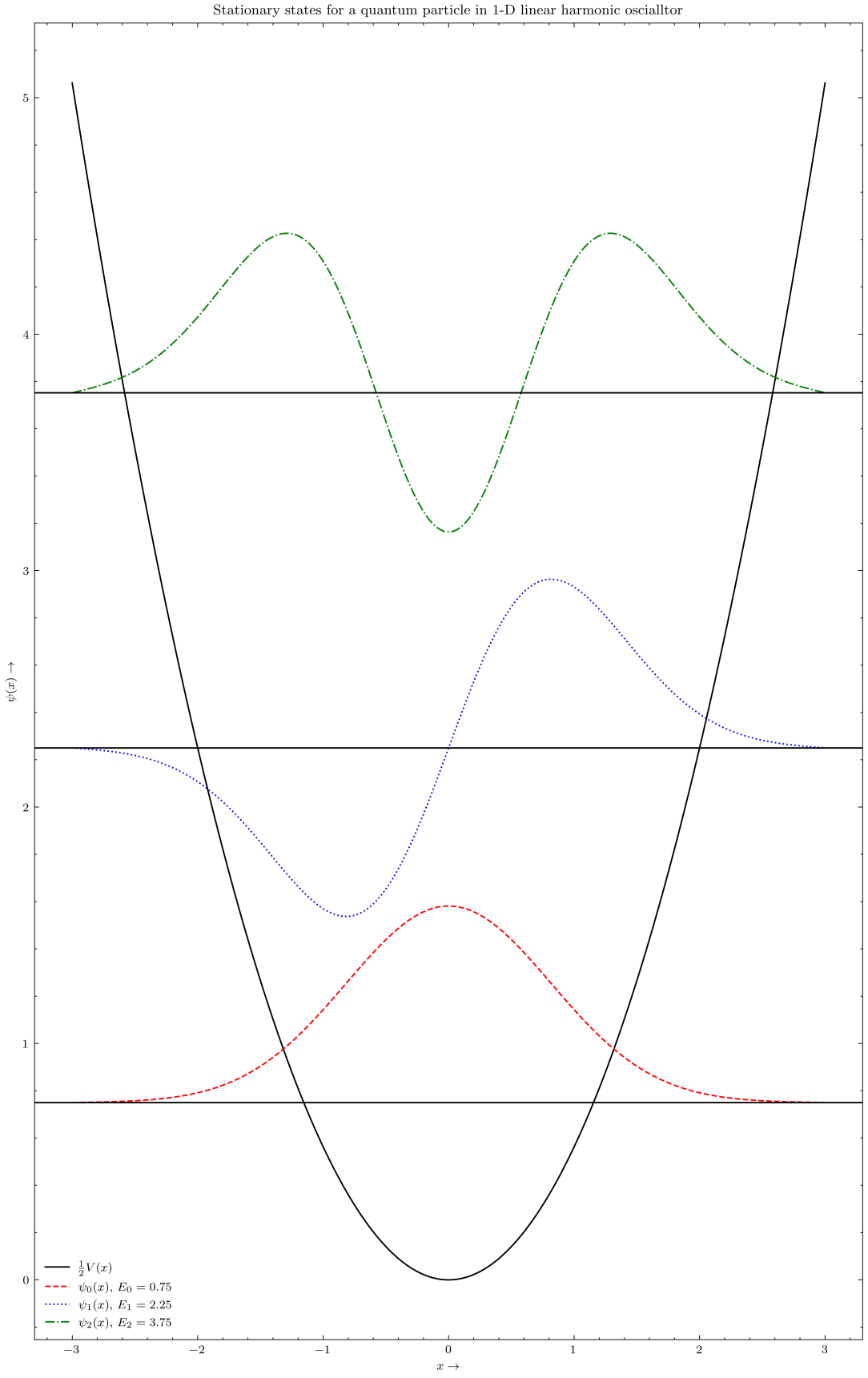

Stationary states for a quantum particle in 1-D finite step potential
$V(x)$
$\psi_0(x)$, $E_0 = 0.99$
$\psi_1(x)$, $E_1 = 3.93$
$\psi_2(x)$, $E_2 = 8.51$
$\psi(x) \rightarrow$
$x \rightarrow$
10
8
6
4
2
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

2.1.4 Linear Harmonic Oscillator

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

[9]:
```
omega = 1.5 # angular frequency

def V(x):
    return 0.5 * m * omega**2 * x**2

xi, xf = -3, 3
E_n = 3

x, E, psi = solve_sch_eqn(V, xi, xf)
```

[10]:
```
plt.plot(x, 0.5*V(x), label="$\\frac{1}{2} V(x)$")

for i in range(E_n):
    plt.plot(x, E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D linear harmonic␣
 ↪oscialltor")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```

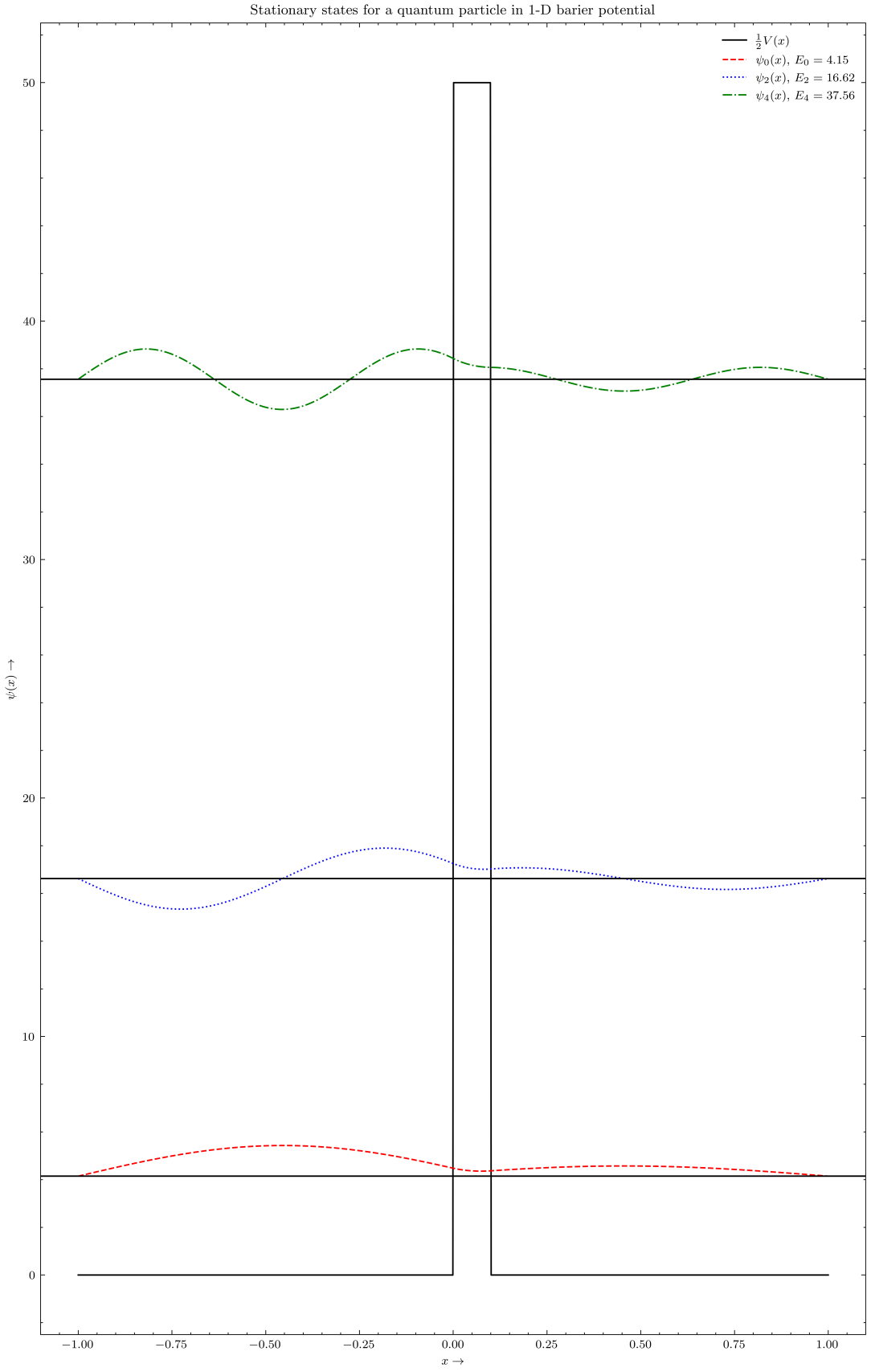

Stationary states for a quantum particle in 1-D linear harmonic oscialltor
$\psi(x) \rightarrow$
$x \rightarrow$
$\frac{1}{2}V(x)$
$\psi_0(x)$, $E_0 = 0.75$
$\psi_1(x)$, $E_1 = 2.25$
$\psi_2(x)$, $E_2 = 3.75$

2.1.5 Barier Potential

$$V(x) = \begin{cases} V_0, & 0 < x \leq a \\ 0, & \text{otherwise} \end{cases}$$

[11]:
```
a = 0.1
V_0 = 100.0

@np.vectorize
def V(x):
    if 0 < x <= a:
        return V_0
    else:
        return 0.0

xi, xf = -1, 1
E_n = 0, 2, 4

x, E, psi = solve_sch_eqn(V, xi, xf)
```

[12]:
```
plt.plot(x, 0.5 * V(x), label="$\\frac{1}{2}V(x)$")

for i in E_n:
    plt.plot(x, E[i] + psi[i]/np.sqrt(integrate.simpson(psi[i]**2, x)),␣
 ↪label=f"$\psi_{i}(x)$, $E_{i} = {round(E[i], 2)}$")
    plt.axhline(E[i])

plt.title("Stationary states for a quantum particle in 1-D barier potential")
plt.xlabel("$x \\rightarrow$")
plt.ylabel("$\psi(x) \\rightarrow$")
plt.legend()
plt.show()
```


Stationary states for a quantum particle in 1-D barier potential
$\frac{1}{2}V(x)$
$\psi_0(x),\ E_0 = 4.15$
$\psi_2(x),\ E_2 = 16.62$
$\psi_4(x),\ E_4 = 37.56$
50
40
30
20
10
0
$\psi(x) \rightarrow$
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
$x \rightarrow$