Implementation of Jacobian Algorithm to obtain energy levels for the Anharmonic Oscillator

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_re
```

```
import proj_2_module as proj2
from scipy.special import eval_hermite
import numpy as np
import math
from matplotlib import pyplot as plt
```

1. Show that the operators $\hat{x}^2$ and $\hat{x}^4$ have the following matrix elements in the harmonic oscillator basis:

$$\langle n|\hat{x}^2|m\rangle = (n+1/2)\delta_{nm} + \tfrac{1}{2}\sqrt{(n+1)(n+2)}\,\delta_{n,m+2} + \tfrac{1}{2}\sqrt{(n-1)n}\,\delta_{n,m-2}$$

$$\langle n|\hat{x}^4|m\rangle = \tfrac{1}{4}\big(6n^2+6n+3\big)\delta_{nm} + \sqrt{(n+1)(n+2)}\,\big(n+\tfrac{3}{2}\big)\delta_{n,m+2} +$$
$$+ \sqrt{(n-1)n}\,\big(n-\tfrac{1}{2}\big)\delta_{n,m-2} + \tfrac{1}{4}\sqrt{(n+1)(n+2)(n+3)(n+4)}\,\delta_{n,m+4} +$$
$$+ \tfrac{1}{4}\sqrt{(n-3)(n-2)(n-1)n}\,\delta_{n,m-4}.$$

We start with rising and lowering operators:

$$a|n>=\sqrt{n}|n-1>$$
$$a^\dagger|n>=\sqrt{n+1}|n+1>$$

With $[a, a^\dagger] = 1$. For $0 < n < N$. In terms of ladder operators we get $\hat{x}$:

$$\hat{x} = \sqrt{\frac{\hbar}{2m\omega}}(a + a^\dagger)$$

So $\hat{x}^2 = \frac{\hbar}{2m\omega}(a + a^\dagger)^2 = \frac{\hbar}{2m\omega}(a^2 + a^\dagger a + aa^\dagger + a^{\dagger 2}) = \frac{\hbar}{2m\omega}(a^2 + a^\dagger a + aa^\dagger + a^{\dagger 2}) = \frac{\hbar}{2m\omega}(a^2 + 2a^\dagger a + 1 + a^{\dagger 2})$.
We can obtain that $a^2|n> = a\sqrt{n}|n-1> = \sqrt{n}\sqrt{n-1}|n-2>$, $a^\dagger a|n> = n|n>$, and
$a^{\dagger 2}|n> = a^\dagger\sqrt{n+1}|n+1> = \sqrt{n+1}\sqrt{n+2}|n+2>$

We consider the eigenvector basis is orthonormal so:

$$< m|\hat{x}^2|n> = \frac{1}{2}[\sqrt{n}\sqrt{n-1}\delta_{mn-2} + 2n\delta_{mn} + \delta_{mn} + \sqrt{n+1}\sqrt{n+2}\delta_{mn+2}]$$
$$= \frac{1}{2}[\sqrt{n}\sqrt{n-1}\delta_{mn-2} + (2n+1)\delta_{mn} + \sqrt{n+1}\sqrt{n+2}\delta_{mn+2}]$$

Now $\hat{x}^4 = \hat{x}^2\hat{x}^2 = (\frac{\hbar}{2m\omega})^2(a^2 + 2a^\dagger a + 1 + a^{\dagger 2})(a^2 + 2a^\dagger a + 1 + a^{\dagger 2})$, which is equal to:

$$(\frac{\hbar}{2m\omega})^2(a^4 + 2a^{\dagger 4} + 2a^2 + a^{\dagger 2} + 2a^2a^\dagger a + a^2a^{\dagger 2} + 2a^\dagger a^3 + 4a^\dagger aa^\dagger a + 2a^\dagger a + a^\dagger aa^{\dagger 2} + 2a^\dagger a + 1 + a^{\dagger 2}a^2 + 2a^{\dagger 3}a$$
$$= (\frac{\hbar}{2m\omega})^2(a^4 + 2a^{\dagger 4} + 2a^2 + a^{\dagger 2} + 2(2a^2 + a^\dagger a^3) + a^2a^{\dagger 2} + 2a^\dagger a^3 + 4a^\dagger aa^\dagger a + 2a^\dagger a + a^\dagger aa^{\dagger 2} + 2a^\dagger a + 1 + a^{\dagger 2}$$
$$+ 2a^{\dagger 3}a^2)$$

Applying the same process as with $\hat{x}^2$ we get:

$$\begin{aligned} < m|\hat{x}^4|n> = \frac{1}{4}(&\sqrt{n}\sqrt{n-1}\sqrt{n-2}\sqrt{n-3}\delta_{mn-4} \\ &+ \sqrt{n+1}\sqrt{n+2}\sqrt{n+3}\sqrt{n+4}\delta_{mn+4} \\ &(4n-2)\sqrt{n}\sqrt{n-1}\delta_{mn-2} \\ &+ (6n^2 + 6n + 3)\delta_{mn} \\ &+ (4n+6)\sqrt{n+1}\sqrt{n+2}\delta_{mn+2} \end{aligned}$$

x. Obtaining the energy levels of the Anharmonic Oscilator

Obtaining the energy levels of the Anharmonic Oscilator

The previous work lets us create a very easy $n \times n$ matrix we can test our work in. Letting $m \in \{1, 2, 3, 4\}$ for i.e the first four energy levels we can obtain all the matrix elements using the δ_{ij} as indicators for the coordinates of our 4×4 matrix.

For $\hat{x}^2$ we get:

$$\hat{x}^2 = \frac{1}{2}\begin{pmatrix} 3 & 0 & \sqrt{6} & 0 \\ 0 & 5 & 0 & \sqrt{12} \\ \sqrt{6} & 0 & 7 & 0 \\ 0 & \sqrt{12} & 0 & 9 \end{pmatrix}$$

For $\hat{x}^4$ we get:

$$\hat{x}^4 = \frac{1}{4}\begin{pmatrix} 15 & 0 & 10\sqrt{6} & 0 \\ 0 & 39 & 0 & 14\sqrt{12} \\ 10\sqrt{6} & 0 & 75 & 0 \\ 0 & 14\sqrt{12} & 0 & 123 \end{pmatrix}$$

Adding the $\hat{p}$ operator definition wrt the raising and lowering operators

$$\hat{p} = i\sqrt{\frac{\hbar m\omega}{2}}(a^\dagger - a)$$

We get that the kinetic energy is

$$\begin{aligned} T &= \frac{\hat{p}^2}{2m} \\ &= -\frac{\hbar\omega}{4}(a^\dagger - a)^2 \\ &= \frac{\hbar\omega}{4}(a^{\dagger 2} + \quad + \quad + \quad a^2) \end{aligned}$$

$$= -\frac{}{4}(a'^{2} - a'a - aa' + a^{2})$$
$$= -\frac{\hbar\omega}{4}(a^{\dagger 2} - 1 - 2a^{\dagger}a + a^{2})$$

So its proyections with the eigenvectors will be proportional to:

$$< m|T|n >= -\frac{1}{2}[\sqrt{n}\sqrt{n-1}\delta_{mn-2} - (2n+1)\delta_{mn} + \sqrt{n+1}\sqrt{n+2}\delta_{mn+2}]$$

```
import numpy as np

def delta(i,j):
  if i == j:
    return 1
  else:
    return 0

def p2(n):
  x_2 = np.zeros((n,n))
  for i in range(n):
    for j in range(n):
      x_2[i,j] = -0.5*(np.sqrt((i+1)*(i))*delta(i-1,j+1) - (2*(i+1) + 1)*delta(i+1,j+1) + np.sqrt((i+2)*(i+3))*delta(i+3
  return x_2

def x2(n):
  x_2 = np.zeros((n,n))
  for i in range(n):
    for j in range(n):
      x_2[i,j] = 0.5*(np.sqrt((i+1)*(i))*delta(i-1,j+1) + (2*(i+1) + 1)*delta(i+1,j+1) + np.sqrt((i+2)*(i+3))*delta(i+3,

  return x_2

def x4(n):
  x_4 = np.zeros((n,n))
  for i in range(n):
    for j in range(n):
      x_4[i,j] = 0.25*(np.sqrt((i+1)*(i)*(i-1)*(i-2))*delta(i-3,j+1) +
```

```
                        np.sqrt((i+2)*(i+3)*(i+4)*(i+5))*delta(i+5,j+1) +
                         (4*(i+1)-2)*np.sqrt((i+1)*(i))*delta(i-1,j+1) +
                         (6*(i+1)**2 + 6*(i+1) + 3)*delta(i+1,j+1) +
                          (4*(i+1)+6)*np.sqrt(i+2)*np.sqrt(i+3)*delta(i+3,j+1))
  return x_4
```

```
def Hamiltonian(n,lmbd):
  return 0.5*p2(n) + 0.5*x2(n) + lmbd*x4(n)
```

```
lmbda = 1

H1 = proj2.Hamiltonian(4,lmbda)

eval, evec = proj2.hermitian_eigensystem(H1,5e-5)
#evalo, evec0 = proj2.hermitian_eigensystem(Ho,1e-5)
# U[:,i] are the eigenvectors
```

Considering the way matrix product is computed, we can conclude that $V \equiv J_1 \dots J_n$, where J_i is the i-th rotational matrix used in the algorithm, will induce a matrix where the columns are the eigenvectors we look for.

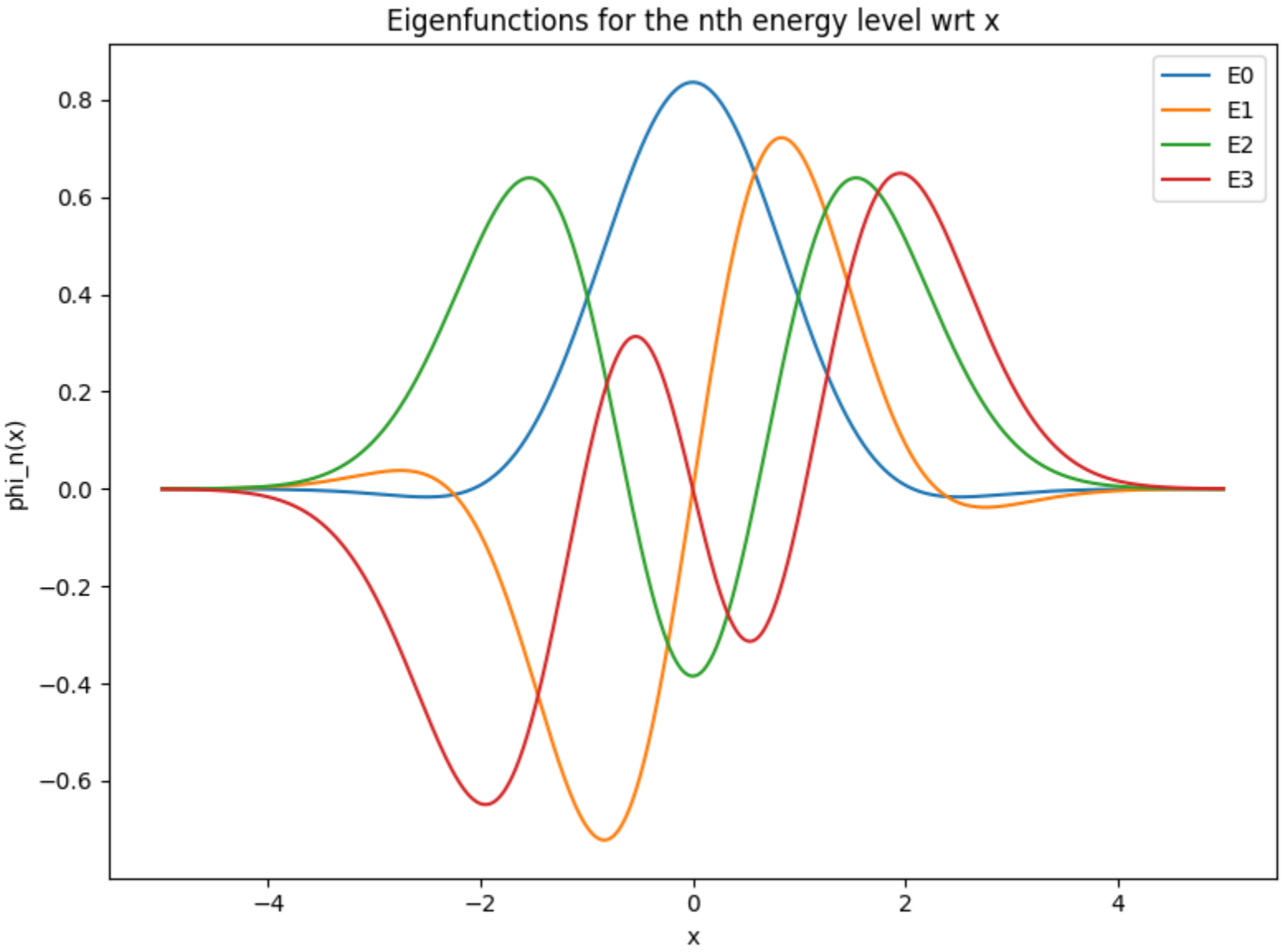
Comienza a programar o generar con IA.

Obtaining the function $\phi_n(x)$

We will obtain the eigenfunctions for the H_1 hamiltonian previously obtained (using $\lambda = 0.1$), on the first four energy levels (including the base level)

(including the base level).

```
from scipy.special import eval_hermite
import numpy as np
import math
from matplotlib import pyplot as plt

def psi(a=-5,b=5,c=1000,n=4):
  x = np.linspace(a,b,c)
  # (2**n*np.sqrt(np.pi)*math.factorial(n))**(-0.5)*np.exp(-i**2/2)*eval_hermite(n,i)
  phi = []
  res = []
  for i in range(n):
    psin = [(2**i*np.sqrt(np.pi)*math.factorial(i))**(-0.5)*np.exp(-t**2/2)*eval_hermite(i,t) for t in x] # obtain the e
    phi.append(psin)
  phi1 = np.array(phi)
  return x, np.real(evec.T @ phi1)

x, resu = psi(n = 4)


fig, ax = plt.subplots(figsize=(8, 6))
n = 4
# Plot multiple lines on the same axis
for i in range(n):
  ax.plot(x, resu[i].T, label="E" + str(i))

# Add a legend
ax.legend()

# Set axis labels and title
ax.set_xlabel('x')
ax.set_ylabel('phi_n(x)')
ax.set_title('Eigenfunctions for the nth energy level wrt x')

# Show the plot
plt.tight_layout()
plt.show()
```

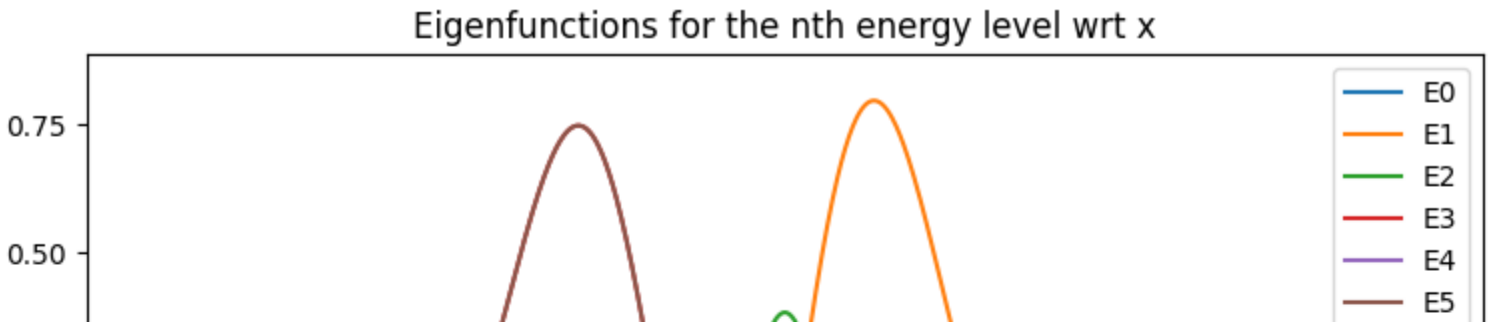


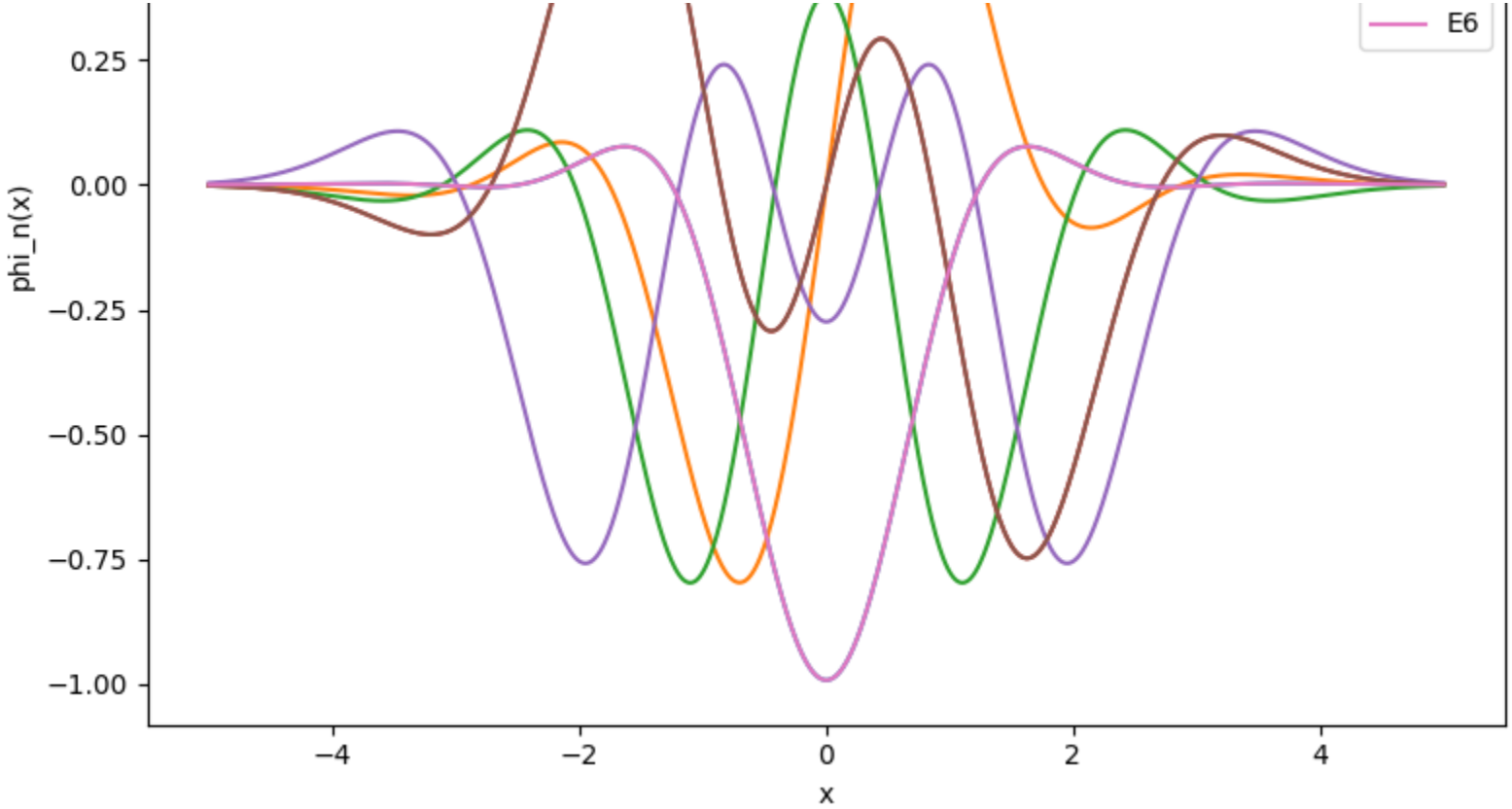

```
H1 = proj2.Hamiltonian(7,lmbda)

eval, evec = proj2.hermitian_eigensystem(H1,5e-5)
```

Tests for bigger N

I will now show the results for $N = 7$.

```
res = psi(n=7)
fig, ax = plt.subplots(figsize=(8, 6))
n = 7
# Plot multiple lines on the same axis
for i in range(n):
  ax.plot(res[0], res[1][i].T, label="E" + str(i))

# Add a legend
ax.legend()

# Set axis labels and title
ax.set_xlabel('x')
ax.set_ylabel('phi_n(x)')
ax.set_title('Eigenfunctions for the nth energy level wrt x')

# Show the plot
plt.tight_layout()
plt.show()
```



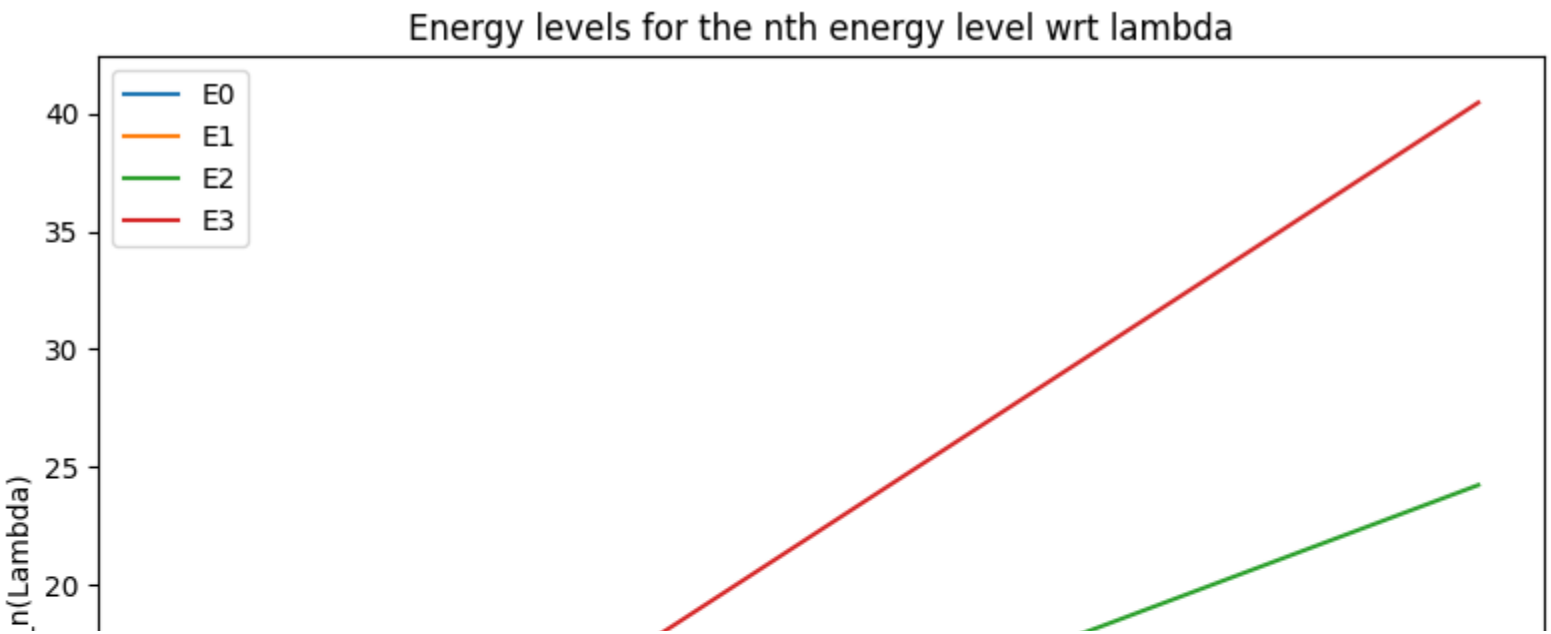


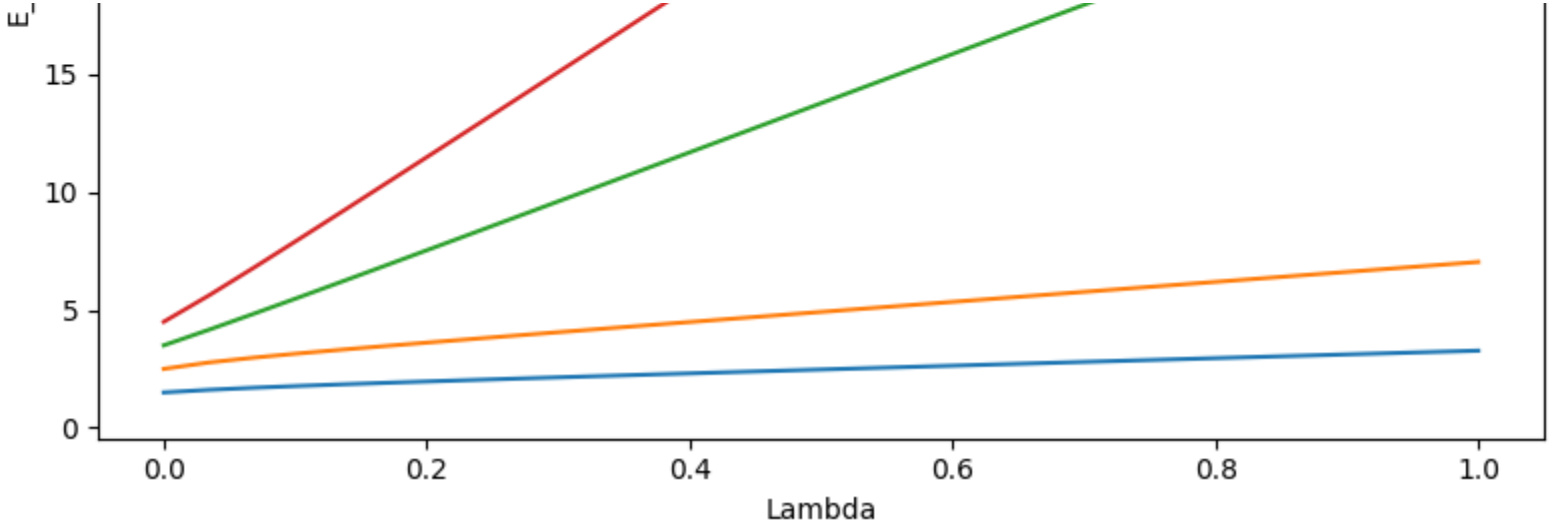
Obtaining the functions $E_n(\lambda)$ for testing

Modifying the λ parameter in the anharmonic Hamiltonian for $\lambda \in [0, 1]$ for first four energy levels will be our next task.

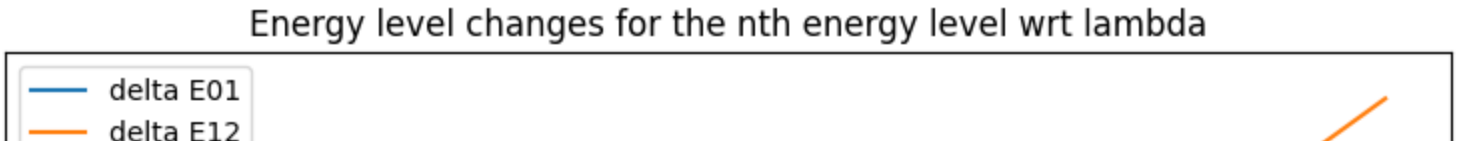
```
def en(a=0, b=1, c=30, n=4):
    x = np.linspace(a, b, c)
    psin = [proj2.hermitian_eigensystem(proj2.Hamiltonian(n, xi), 5e-5)[0] for xi in x]
    return x, np.array(psin)

x, resul = en()
```

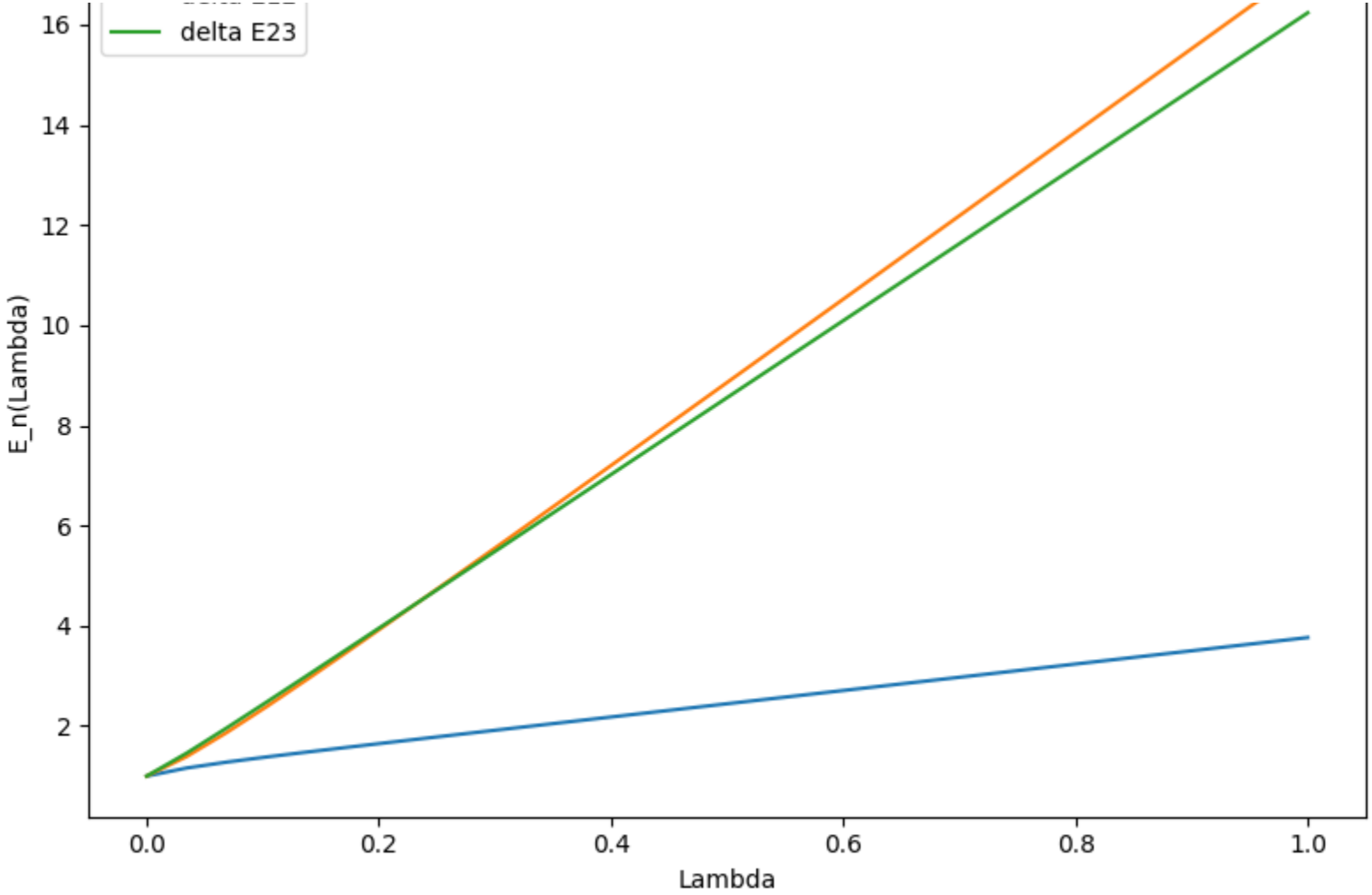
```
fig, ax = plt.subplots(figsize=(8, 6))
n = 4
# Using dimension as a mobile input
for i in range(n):
  ax.plot(x, resul.T[i], label="E" + str(i))


# Add a legend
ax.legend()

# Set axis labels and title
ax.set_xlabel('Lambda')
ax.set_ylabel('E_n(Lambda)')
ax.set_title('Energy levels for the nth energy level wrt lambda')

# Show the plot
plt.tight_layout()
plt.show()
```






```
fig, ax = plt.subplots(figsize=(8, 6))
n = 4
for i in range(n-1):
  ax.plot(x, resul.T[i+1]-resul.T[i], label="delta E" + str(i) + str(i+1))


# Add a legend
ax.legend()

# Set axis labels and title
ax.set_xlabel('Lambda')
ax.set_ylabel('E_n(Lambda)')
ax.set_title('Energy level changes for the nth energy level wrt lambda')

# Show the plot
plt.tight_layout()
plt.show()
```





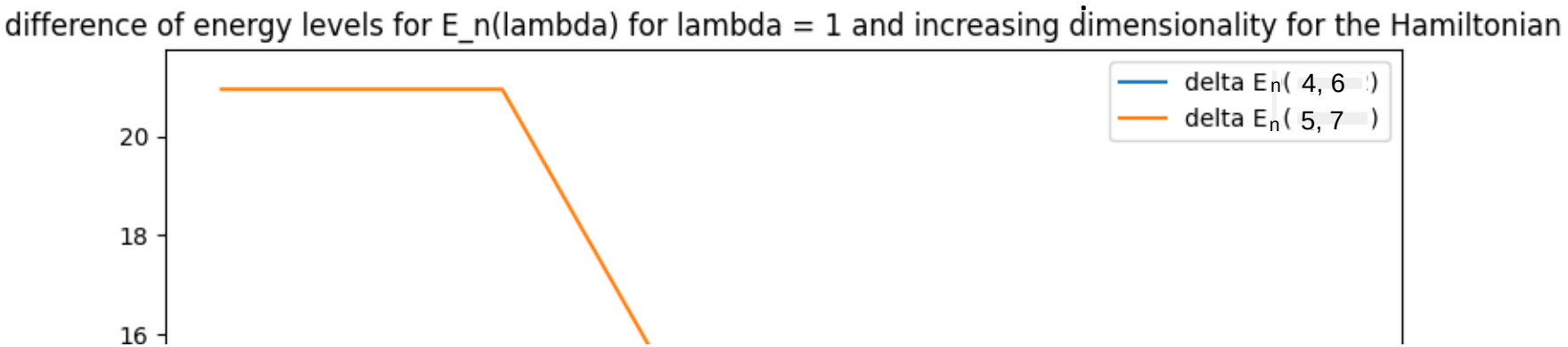
Plotting of $E_n(N)$ with fixed $\lambda = 1$ value

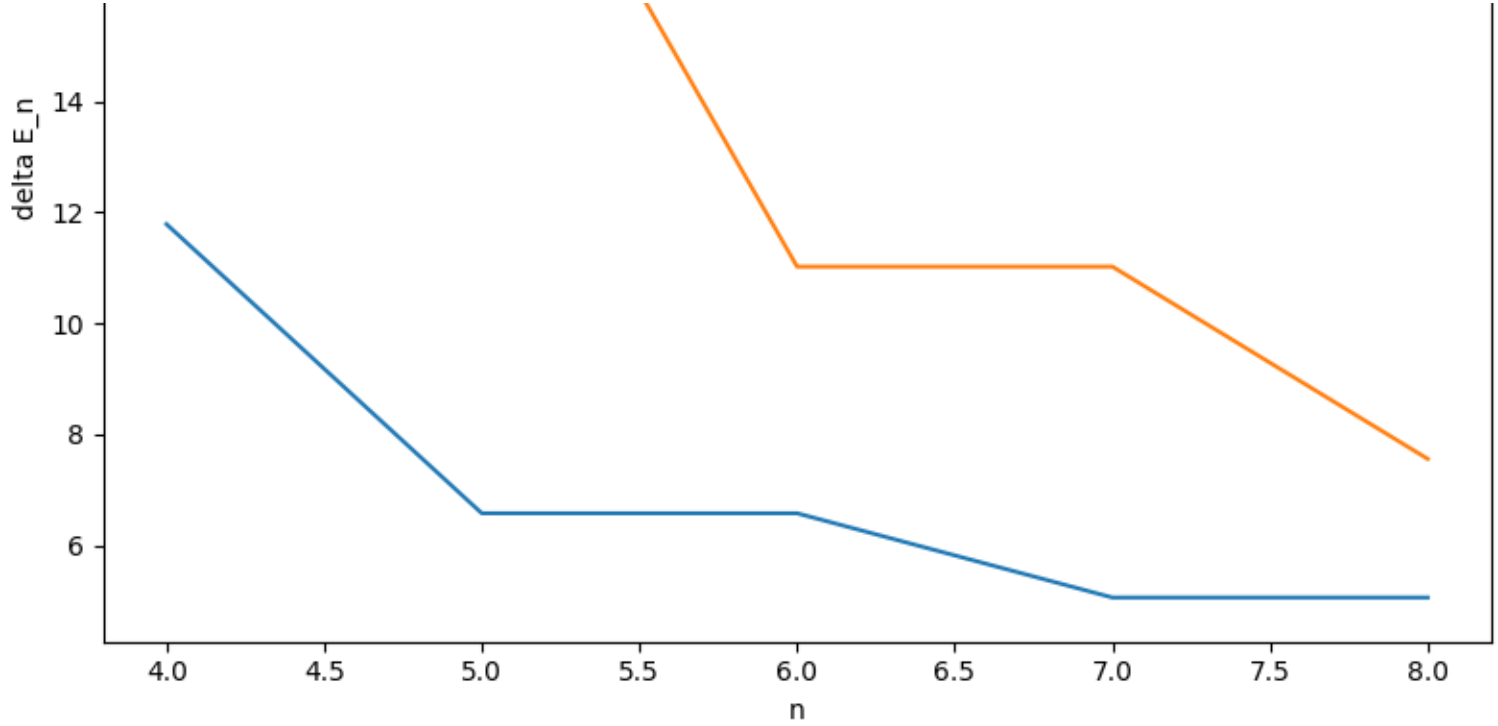
```
def en(n):
  energ = [proj2.hermitian_eigensystem(proj2.Hamiltonian(i,1), 5e-5)[0][0:4] for i in range(4,n)]
  N = [i for i in range(4,n)]
  return N, np.array(energ)

n, resulN = en(n=9)
```

Comienza a programar o generar con IA.

```
array([[ 0.85508695,  3.27383661, 12.64491305, 24.22616339],
       [ 0.80822855,  3.27383661,  7.38282469, 24.22616339],
       [ 0.80822889,  2.84387227,  7.38282435, 13.86748964],
       [ 0.80587076,  2.84387228,  5.86071285, 13.86748964],
       [ 0.80587076,  2.75257652,  5.86071292, 10.30811771]])
```

```
fig, ax = plt.subplots(figsize=(8, 6))
N = 4
for i in range(N-2):
  ax.plot(n, np.abs(resulN.T[i]-resulN.T[i+2]), label="delta E"+str(i)+"(N,N+2)")


# Add a legend
ax.legend()

# Set axis labels and title
ax.set_xlabel('n')
ax.set_ylabel('delta E_n')
ax.set_title('difference of energy levels for E_n(lambda) for lambda = 1 and increasing dimensionality for the Hamiltoni

# Show the plot
plt.tight_layout()
plt.show()
```





This implies that the bigger the dimension for the Hamiltonian, the lower the difference between the calculated values of the same energy level. We can very clearly see a downward tendency for up to $N = 9$ dimensional Hamiltonian matrices (even imrpving my code was not enough for the calculations to be done in a reasonable timeframe for bigger N). However, the tendency for the values of the energy levels obtained algorithmically indicates that the bigger your Hamiltonian matrix, the more self consistent your energy values will be, implying that for a big enough M the Hamiltonian should produce a value that will stay the same even if we use $N > M$ dimensions.

```
eval4, evec4 = proj2.hermitian_eigensystem(proj2.Hamiltonian(4,0), 5e-5)
eval41, evec41 = proj2.hermitian_eigensystem(proj2.Hamiltonian(4,1), 5e-5)

def psi(a=-5,b=5,c=1000,n=4):
  x = np.linspace(a,b,c)
  # (2**n*np.sqrt(np.pi)*math.factorial(n))**(-0.5)*np.exp(-i**2/2)*eval hermite(n,i)
```

```
  phi = []
  res = []
  for i in range(n):
    psin = [(2**i*np.sqrt(np.pi)*math.factorial(i))**(-0.5)*np.exp(-t**2/2)*eval_hermite(i,t) for t in x] # obtain the e
    phi.append(psin)
  phi1 = np.array(phi)
  return x, np.real(evec4.T @ phi1), np.real(evec41.T @ phi1)


x, resu, resu1 = psi(n = 4)
```

```
/content/proj_2_module.py:140: ComplexWarning: Casting complex values to real discards the imaginary part
  B[i, j + dim[0]] = -IM[i,j]
/content/proj_2_module.py:141: ComplexWarning: Casting complex values to real discards the imaginary part
  B[i + dim[0], j] = IM[i,j]
```

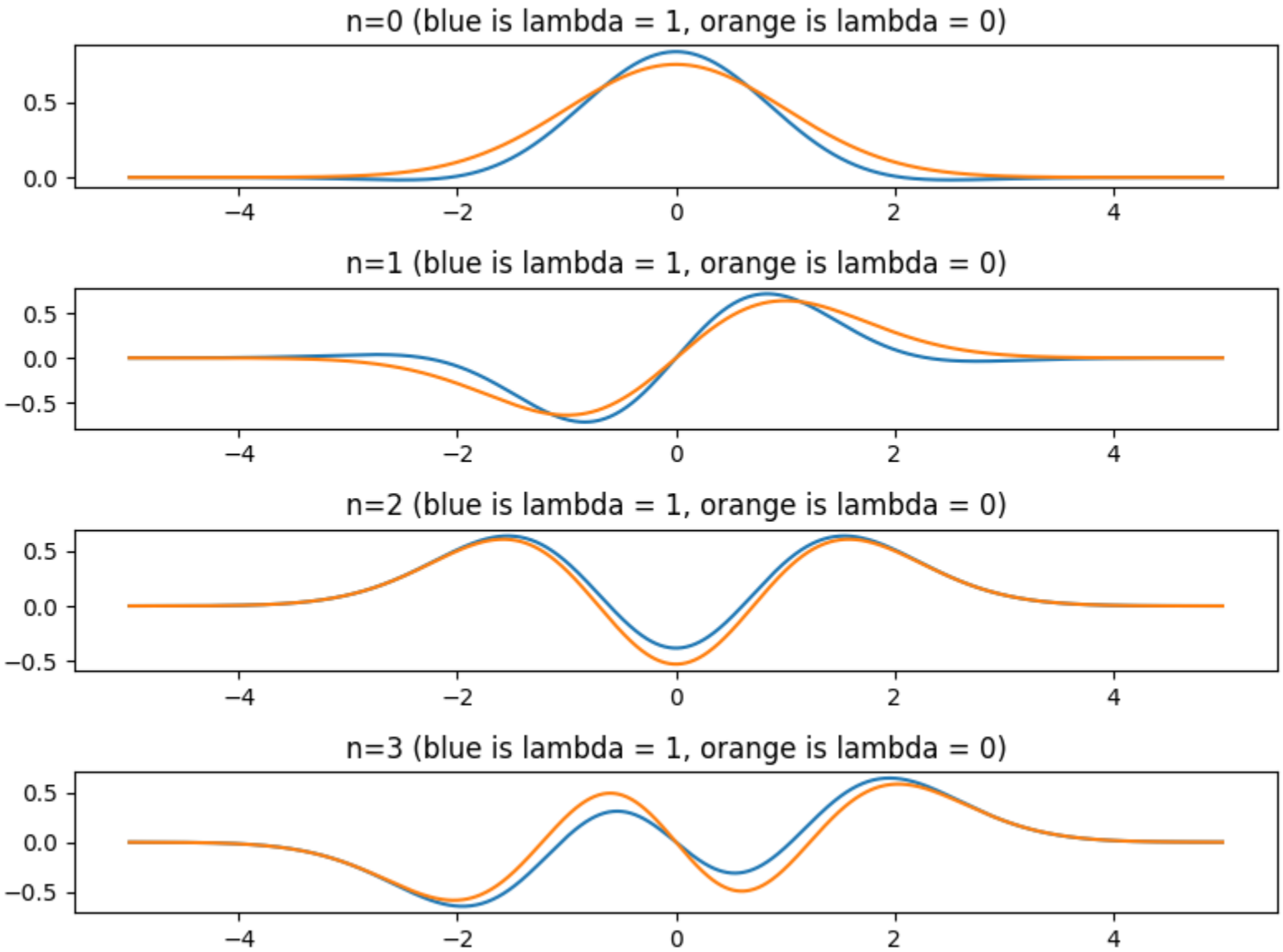
```
resu[0][0].T.shape
```

```
(1000, 1)
```

```
n = 4
fig, ax = plt.subplots(n,figsize=(8, 6))
for i in range(n):
  ax[i].plot(x, resu1[i][0].T)
  ax[i].plot(x, resu[i][0].T)
  ax[i].set_title("n="+ str(i) + " (blue is lambda = 1, orange is lambda = 0)")
```

```
# Set axis labels and title
"""
ax.set_xlabel('x')
ax.set_ylabel('psi_n(x)')
ax.set_title('Resulting eigenfunctions for lambda = 0, lambda = 1')
"""
```

```
# Show the plot
plt.tight_layout()
plt.show()
```



We notice that the eigenfunction for $\lambda = 0$ is more spread out than the eigenfunction for $\lambda = 1$. There definitely has to be room for improvement, but given the current computational capabilites it should suffice the fact that for the first energy levels there were deeper spikes for $\lambda = 1$. Considering $\epsilon \approx 5 \times 10^{-5}$ and the dimension for the Hamiltonian is relatively low there can be some errors for higher energy levels that should be accounted for in order for all the functions to display the tendency the first two did.