Computational Physics: Introductory Course
Spring 2023

Assignment 8:
Eigen-Value Problems

Name: Nikolaos Giannakis
Personummer: 950417921
Hand-in date: 2022-03-31

Exercise 1

Look through the commands for handling matrices so that you master them (need not be reported in writing, but is left to you for your own knowledge)

Reading the commands for the matrices.

Exercise 2

Write a py-file that determines and visualizes the eigen oscillations for a string with three mass points as an animation (see code in section 11.20 in the lecture notes). Include a few 'snapshots' in the report.

```
#### CODE IN SECTION 11.20 ####
# egensvangning.py
import numpy as np
import matplotlib.pyplot as plt
A = np.array([[-2,1],[1,-2]])
w, v = np.linalg.eig(A)
# l = pos. in x-led for mass points and end points
l = np.arange(0,4)

# largest k
k = np.sqrt(-w[1])
fig, ax = plt.subplots()
for t in np.arange(0,10*np.pi,0.1):
    # u = dist. from equilibrium; dist. is 0 for end points
    u = np.sin(k*t)*np.hstack([0,v[:,1],0])
    ax.plot(l,u,l,u,   o   )
    ax.set_ylim([-1,1])
    ax.tick_params(labelsize=14)
    plt.pause(0.1)
    ax.cla() # clear graphical window

# smallest k
k = np.sqrt(-w[0])
fig, ax = plt.subplots()
for t in np.arange(0,10*np.pi,0.1):
    # u = dist. from equilibrium; dist. is 0 for end points
    u = np.sin(k*t)*np.hstack([0,v[:,0],0])
    ax.plot(l,u,l,u,   o   )
    ax.set_ylim([-1,1])
    ax.tick_params(labelsize=14)
    plt.pause(0.1)
    ax.cla() # clear graphical window
```

By checking the code above we can basically modify it in order to solve the corresponding exercise.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
3
4  # Define matrix A for three masses
5  A = np.array([[-2,1,0],[1,-2,1],[0,1,-2]])
6
7  # Find eigenvalues and eigenvectors of A
8  w, v = np.linalg.eig(A)
9
10 # Define positions of mass points and end points
11 l = np.arange(0,5)
12
13 # Set up plot for largest eigenvalue
14 k = np.sqrt(-w[2])
15 fig, ax = plt.subplots()
16 for t in np.arange(0,10*np.pi,0.1):
17     u = np.sin(k*t)*np.hstack([0,v[:,2],0])
18     ax.plot(l,u,l,u,'o')
19     ax.set_ylim([-1,1])
20     ax.tick_params(labelsize=14)
21     plt.pause(0.1)
22     ax.cla()
23
24 # Set up plot for middle eigenvalue
25 k = np.sqrt(-w[1])
26 fig, ax = plt.subplots()
27 for t in np.arange(0,10*np.pi,0.1):
28     u = np.sin(k*t)*np.hstack([0,v[:,1],0])
29     ax.plot(l,u,l,u,'o')
30     ax.set_ylim([-1,1])
31     ax.tick_params(labelsize=14)
32     plt.pause(0.1)
33     ax.cla()
34
35 # Set up plot for smallest eigenvalue
36 k = np.sqrt(-w[0])
37 fig, ax = plt.subplots()
38 for t in np.arange(0,10*np.pi,0.1):
39     u = np.sin(k*t)*np.hstack([0,v[:,0],0])
40     ax.plot(l,u,l,u,'o')
41     ax.set_ylim([-1,1])
42     ax.tick_params(labelsize=14)
43     plt.pause(0.1)
44     ax.cla()
```

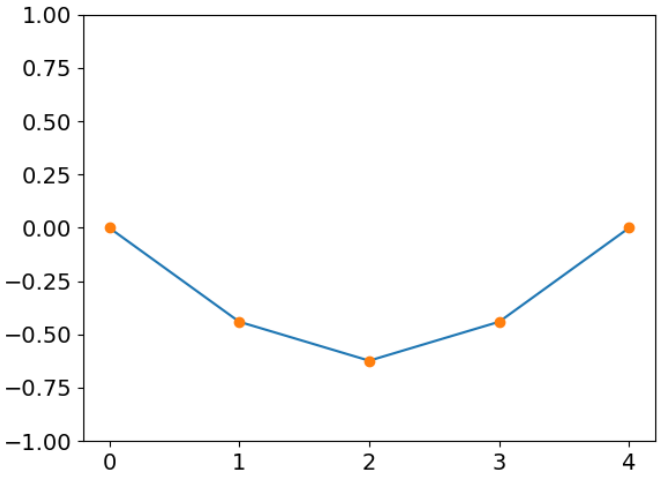

Figure 1: Largest eigenvalue

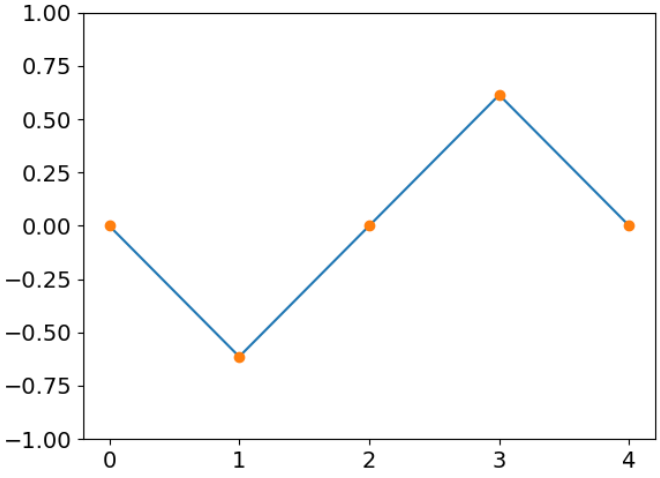

Figure 2: Middle eigenvalue

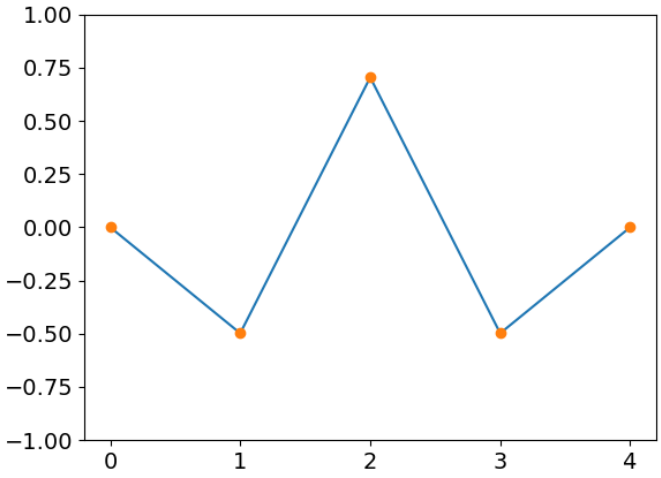

Figure 3: Lowest eigenvalue

Exercise 3

Write a py-file that determines and visualizes the eigen oscillations for a string with 10 mass points as an animation. Also, test to see how the vibrations look like when you take some linear combinations of the eigen oscillations. Again, include a few 'snapshots' in the report.

```
import numpy as np
import matplotlib.pyplot as plt

# Define matrix A for 10 masses
A = np.zeros((10,10))
np.fill_diagonal(A, -2)
np.fill_diagonal(A[1:], 1)
np.fill_diagonal(A[:,1:], 1)

# Find eigenvalues and eigenvectors of A
w, v = np.linalg.eig(A)

# Define positions of mass points and end points
l = np.arange(0,12)
linear = np.arange(0,10)

# Set up plot for each eigenvalue
for i in range(len(w)):
    k = np.sqrt(-w[i])
    fig, ax = plt.subplots()
    for t in np.arange(0,10*np.pi,0.1):
        u = np.sin(k*t)*np.hstack([0,v[:,i],0])
        u_linear = (v[:,0] + v[:,1] + v[:,2]) * np.sin(k*t)
        u_linear_interp = np.interp(l, np.arange(0, 10), u_linear)
```

```
25          ax.plot(l,u,l,u,'o')
26          ax.plot(l, u_linear_interp, '-', label='Linear combination')
27          ax.set_ylim([-1,1])
28          ax.tick_params(labelsize=14)
29          plt.pause(0.1)
30          ax.cla()
```

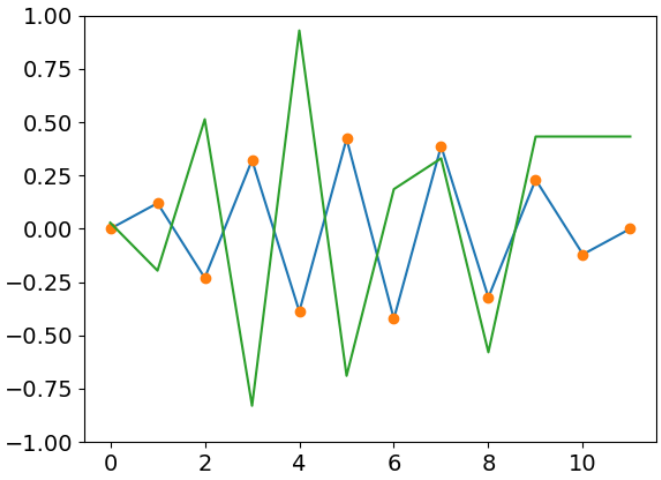

Figure 4: Plot of linear combination 1

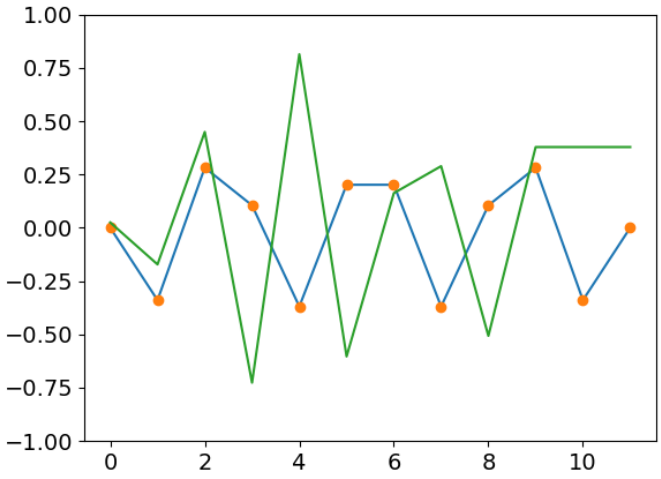

Figure 5: Plot of linear combination 2

Here I got quite confused with the linear combination. I think I got the first part of plotting the 10 mass points correct however, the part of the linear combination seems kinda

wrong. This could be due to the reason that in the plot of eigenvalues we need basically to have 2 extra points in order to plot it correctly. But in the scenario of the linear combination the last two end points are a straight line due to the in-build function I used which basically just connects the points in order to make the plots in the same graph. I do not know if I was supposed to make to two separate plots. In case you need me to adjust it end re-upload just let me know. I would appreciate some feedback to understand that part better no matter if I need to re-upload or not. Thank you in advance!

Exercise 4

Download quantumwell.py (code in section 21.7 available from Canvas) and test the code for solving the wave equation for a particle in a potential well

Basically this is also done.

Exercise 5

```
import numpy as np
import matplotlib.pyplot as plt

n = 5000  # number of grid points
x_min, x_max = -10, 10  # solution interval
x_half = 5  # half width of potential
k = 1  # spring constant
m = 1  # mass of the particle


# Potential and potential matrix
x = np.linspace(x_min, x_max, n + 1)
u = 0.5 * k * x**2
U = np.diag(u[1:n])

# Discretized differential operator
dx = (x_max - x_min) / n
L = (2 * np.eye(n - 1) - np.diag(np.ones(n - 2), 1) - np.diag(np.ones(n -
    2), -1)) / (2 * dx**2)

# Hamilton matrix, H is Hermitian
H = L + U

# Diagonalize and determine E and psi for inner points
w, v = np.linalg.eigh(H)

# Extract and print bound states E < 0
nbound = 10
energies = w[:nbound]
print("Energies:", energies)
```

```
30
31 # Plot wave functions and probability distributions
32 fig, axs = plt.subplots(nbound, 2, figsize=(8, 20))
33 for i in range(nbound):
34     psi = v[:, i]  # wave function psi
35     psi_norm = psi / np.sqrt(dx * np.sum(psi**2))  # normalized wave
       function
36     psi2 = psi_norm**2  # probability density
37
38     # Plot wave function
39     axs[i, 0].plot(x[1:n], psi_norm, 'r')
40     axs[i, 0].set_xlim([-10, 10])
41     title = ["n =", str(i + 1), "E =", str(round(energies[i], 2))]
42     axs[i, 0].set_title(" ".join(title), fontsize=14)
43     axs[i, 0].axis("off")
44
45     # Plot probability distribution
46     axs[i, 1].plot(x[1:n], psi2, 'r--')
47     axs[i, 1].set_xlim([-10, 10])
48     axs[i, 1].set_ylim([0, 0.8])
49     axs[i, 1].set_title("Probability distribution", fontsize=14)
50     axs[i, 1].axis("off")
51
52 plt.show()
53
54 # part (b)
55 analytical_energies = np.array([(n+1/2) for n in range(nbound)]) #
       analytical energies
56 computed_energies = w[:nbound] # computed energies
57
58 # Compare the computed and analytical energies
59 print("Analytical energies: ", analytical_energies)
60 print("Computed energies: ", computed_energies)
61
62
63 # Analytical energies:  [0.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5]
64 # Computed energies:  [0.4999995 1.4999975 2.4999935 3.4999875 4.4999795
       5.4999695 6.4999575
65 # 7.4999435 8.4999275 9.4999095]
```

The former code has both part (a) and (b). As we can see when we increased n to 5000, we basically got the computed energies closer to the analytical ones. In the following figure we can see the corresponding plots of wave functions and their corresponding probability distributions. Additionally, the code is pretty similar except the part where I plot the graphs. Apart this nothing is changed more likely.

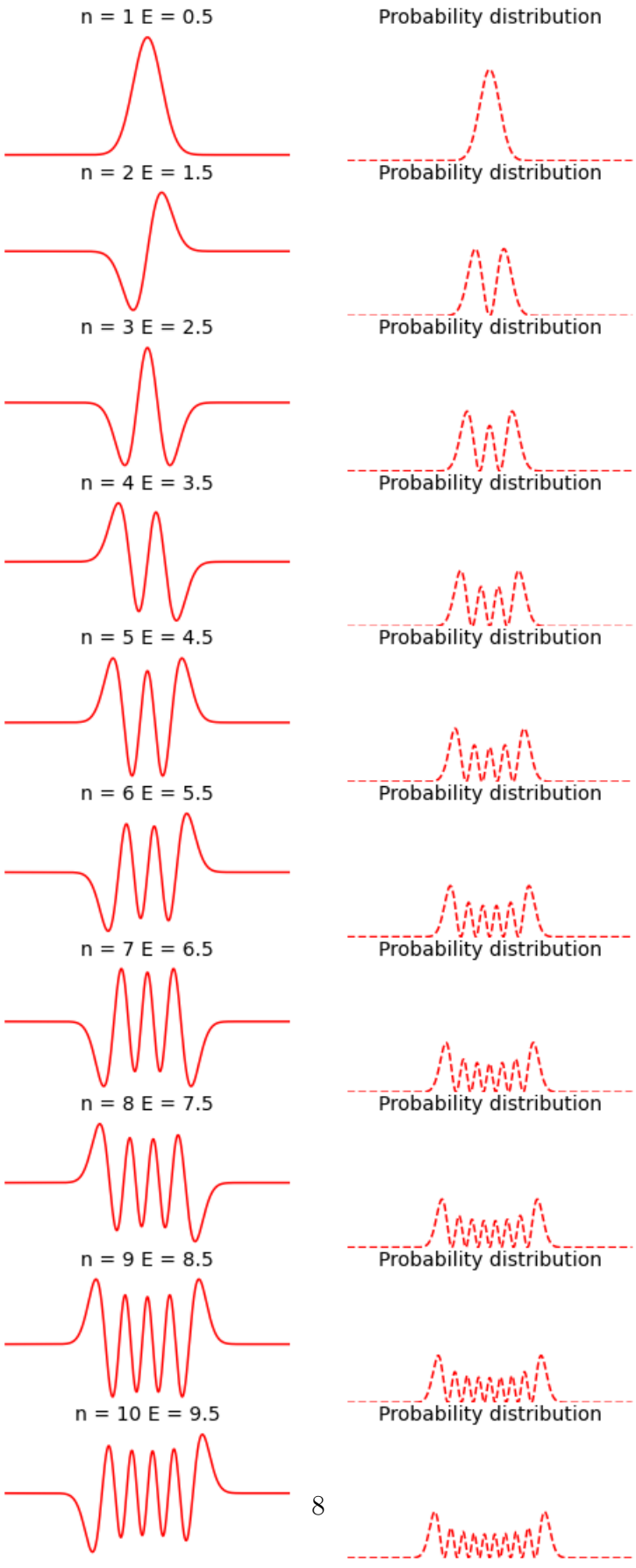

Figure 6: Plots of wave functions and probability distributions

Part (c)

In classical mechanics, the particle oscillates back and forth around the equilibrium position with the highest probability close to the turning points. However, the reason behind this is that the particle spends more time near the turnings points than anywhere else as it reverses its direction. Thus, it is quite clear that there is a higher probability for the particle to be found there.

In contrast, the probability distribution in quantum mechanics is given by the square of the wave function, which is not a smooth curve but has peaks and valleys. The highest probability density is located at regions where the wave function has the highest amplitude, which can be at any point in space. Noteworthy is that as we move to higher energy states, the probability distribution becomes more complex, with additional peaks appearing at the turning points. This happens because the higher energy states have more nodes in the wave functions which are basically correspond to points where the probability of finding the particle is zero.

Consequently, in quantum mechanics, the probability of finding a particle at the turning points is high but not the highest as the probability distribution becomes more complex and has multiple peaks as we have higher energy states.