Chapter 13

Eigenvalues

Chapter Objectives

- Understanding the mathematical definition of eigenvalues and eigenvectors.

- Knowing how to use and interpret the Python NumPy linalg **eigvals** and **eig** functions.

Characteristic Eigenvalue Equation

The characteristic or eigenvalue equation is of the form

$$Ax = \lambda x$$

Here A is a matrix, x is a vector and λ is a scalar.

In this equation both x (eigenvector) and λ (eigenvalue) are unknowns.

For physical systems A is an hermitian matrix and hence λ gives real eigenvalues

Eigenvalues & Eigenvectors with Python: ‘**eig**’

```
A =
    [[10  ,  -5],
    [-5  ,  10]
```

[lam, v] = np. linalg.eig(A)

```
v =
[[ 0.70710678   0.70710678]
 [-0.70710678   0.70710678]]
Lam = [15 , 5]
```

Check eigenvectors are orthogonal

- Diagonal elements of lambda are the eigenvalues
- Columns of V are the normalized eigenvector in the order of eigenvalues: first column is the eigenvector of the first eigenvalue: $V[:,0] = |\lambda_1>$
- To make sure that the eigenvalues are ordered in ascending order together with the corresponding eigenvectors use np.argsort()

Example

Find the eigenvalues of A=magic(3).
Note A is not Hermitian.

Open a function file example_eig

Show the eigenvalues and eigenvectors

```python
import numpy as np
from scipy.linalg import eig

def example_eig():
    A = np.array([[8, 1, 6],
                  [3, 5, 7],
                  [4, 9, 2]])  # 3x3 magic square

    # Compute eigenvalues and eigenvectors
    eigenvalues, eigenvectors = eig(A)

    return eigenvectors, eigenvalues

# Example of how to call the function
V, D = example_eig()
print("Eigenvectors (V):")
print(V)
print("Eigenvalues (D):")
print(D)
```

```
Eigenvectors (V):
[[-0.57735027 -0.81305253 -0.34164801]
 [-0.57735027  0.47140452 -0.47140452]
 [-0.57735027  0.34164801  0.81305253]]

Eigenvalues (D):[15.      +0.j  4.89897949+0.j -4.89897949+0.j]
```

Functions of Matrices

When working with functions of matrices add ‘m’ to the function. For example for matrix A

expm(A) or e^A and not exp(A) is used for e^A

Functions like sqrtm, logm exist.

A = [[1 , 2],
[2 , -1]]

Then np.exp(A) =

2.7183	7.3891
7.3891	0.3679

np.expm(A)= e^A

6.7999	4.1365
4.1365	2.6634

Different values

- Note exp(A) takes the exponentials of the elements. i.e.

$$\exp(A) = \begin{bmatrix} \exp(A_{11}) & \exp(A_{21}) \\ \exp(A_{21}) & \exp(A_{22}) \end{bmatrix}$$

- While expm do the true exponentiation

$$\mathrm{expm}(A) = 1 + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots.$$

- Similarly for other functions with ‘m’

Matrix Representation of Angular Momentum

In the special case of spin-1/2, we have found the spin matrices by considering their action on the basis vectors $|sm\rangle$

$$S^2|sm\rangle = \hbar^2 s(s+1)|sm\rangle$$

$$S_z|sm\rangle = \hbar m|sm\rangle$$

$$S_\pm|sm\rangle = \hbar\sqrt{s(s+1)-m(m\pm 1)}\,|s(m\pm 1)\rangle$$

$$S_x = (S_+ + S_-)/2 \quad \text{and} \quad S_y = (S_+ - S_-)/2i$$

Since there are only two values of m for s=1/2, the matrices are 2x2.

Angular Momentum

For a general value of j, the matrices will be (2j+1)x(2j+1).

The matrix elements are given by:

$$A_{m'm} = \langle jm' | A | jm \rangle$$

where *A* is one of the angular momentum operators: J^2, J_x, J_y, J_z, J_+, J_-

Angular Momentum

Using the orthonormality of the eigenstates $| jm\rangle$

$$\langle jm' | J^2 | jm \rangle = \hbar^2 j(j+1)\ \delta_{m',m}$$

$$\langle jm' | J_z | jm \rangle = \hbar m\ \delta_{m',m}$$

$$\langle jm' | J_\pm | jm \rangle = \hbar \sqrt{j(j+1) - m(m\pm 1)}\ \delta_{m',m\pm 1}$$

Notice that J^2 and J_z are diagonal matrices.

For $\boldsymbol{J_x}$ and $\boldsymbol{J_y}$, we use

$$J_x = (J_+ + J_-)/2$$

$$J_y = (J_+ - J_-)/2i$$

```python
import numpy as np

def Joperator(j):
    N = int(2 * j + 1)

    # Initialize matrices with zeros
    Jp = np.zeros((N, N), dtype=complex)
    Jm = np.zeros((N, N), dtype=complex)

    # Calculate J+ and J- matrices
    for n in range(N):
        for m in range(N):
            Jp[m, n] = np.sqrt(j * (j + 1) - (n - j) * (n - j + 1)) * (m == n + 1)
            Jm[m, n] = np.sqrt(j * (j + 1) - (n - j) * (n - j - 1)) * (m == n - 1)

    # Calculate Jx, Jy, and Jz matrices
    Jx = (Jp + Jm) / 2
    Jy = (1j * (Jm - Jp)) / 2  # `1j` represents the imaginary unit in Python
    Jz = np.diag(np.arange(-j, j + 1))

    return Jx, Jy, Jz, Jp, Jm

# Example usage:
Jx, Jy, Jz, Jp, Jm = Joperator(1)  # Example for j=1
print("Jx:\n", Jx)
print("Jy:\n", Jy)
print("Jz:\n", Jz)
print("Jp:\n", Jp)
print("Jm:\n", Jm)
```

Example1: Spin in a magnetic field

Consider a particle of spin 5/2 in a magnetic field

B = 3 **i** + 2 **j** - 4 **k**.

Its Hamiltonian is given by $H = -\vec{\mu} \cdot \vec{B}$

Here $\vec{\mu} = \gamma \vec{S}$. Take $\gamma = 1,\ \hbar = 1$

Calculate the eigenvalues and eigenvectors. What are the ground state eigenvalues and eigenvectors. Show that ground state is orthogonal to the second excited state.

- Write a function file called magnetic
- Call Joperator with j=5/2
- Write the B vector (array)
- Calculate the Hamiltonian matrix
- Calculate ordered the eigenvalues and eigenvectors.
- Obtain the ground state eigenvalue and eigenvector
- Verify that the second excited state is orthogonal to the ground state.

```python
import numpy as np
from scipy.linalg import eig

def Joperator(j):
    N = int(2 * j + 1)

    # Initialize matrices with zeros
    Jp = np.zeros((N, N), dtype=complex)
    Jm = np.zeros((N, N), dtype=complex)

    # Calculate J+ and J- matrices
    for n in range(N):
        for m in range(N):
            Jp[m, n] = np.sqrt(j * (j + 1) - (n - j) * (n - j + 1)) * (m == n + 1)
            Jm[m, n] = np.sqrt(j * (j + 1) - (n - j) * (n - j - 1)) * (m == n - 1)

    # Calculate Jx, Jy, and Jz matrices
    Jx = (Jp + Jm) / 2
    Jy = (1j * (Jm - Jp)) / 2  # `1j` represents the imaginary unit in Python
    Jz = np.diag(np.arange(-j, j + 1))

    return Jx, Jy, Jz, Jp, Jm
```

```python
def magnetic():
    # Elzain Summer 2014
    j = 5 / 2
    # Call the Joperator function
    Jx, Jy, Jz, Jp, Jm = Joperator(j)

    # Define the magnetic field vector B
    B = np.array([3, 2, -4])

    # Hamiltonian matrix H
    H = -(B[0] * Jx + B[1] * Jy + B[2] * Jz)

    # Eigenvalues and eigenvectors
    D, V = eig(H)

    # Sorting eigenvalues and corresponding eigenvectors
    idx = np.argsort(D)
    D = D[idx]
    V = V[:, idx]

    # Extract ground state energy and eigenvector
    Eg = D[0]  # ground state energy
    Vg = V[:, 0]  # ground state eigenvector
    Ve2 = V[:, 2]  # third column eigenvector
    s = np.vdot(Vg, Ve2)  # Dot product, `vdot` handles conjugate transpose

    # Display results
    print(f'Ground state energy Eg: {Eg.real:.4f}')
    print(f'Ground state eigenvector Vg:\n{Vg}')
    print(f'Orthogonality check s = {s.real:.4f}')
# Call the function
magnetic()
```

Development in Time

The time-dependent Schrodinger equation

$$i\hbar \frac{\partial}{\partial t} |\psi(t) > = \quad \mathrm{H} |\psi(t) >$$

has formal solution

$$|\psi(t) > = \quad \mathrm{e}^{-\frac{\mathrm{iHt}}{\hbar}} |\psi(0) >$$

where $|\psi(0) >$ is the initial state

Example1: Development in Time

For example for an electron of spin ½ in a magnetic field B in the z-directions where values of physical constants are set to 1, the Hamiltonian

$$H = -\frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

has eigenvalues $\pm\frac{1}{2}$ and eigenvectors $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

If the system is initial in the state $\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$, find its state at time t. What is the probability for finding the system in $\frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$?

Example2: Development in Time

Spectral solution

$$|\psi(t) > \quad = \quad \frac{1}{\sqrt{2}} e^{it/2} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{1}{\sqrt{2}} e^{-it/2} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Probability amplitude for particle to be in

$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

$$pA = \frac{1}{2} \left(e^{\frac{it}{2}} - e^{-\frac{it}{2}} \right) = i \sin\left(\frac{t}{2}\right)$$

Probability is then

$$p = |pA|^2 = \sin^2\left(\frac{t}{2}\right)$$

The direct solution can also be analytically calculated since in this case, the Hamiltonian satisfy

$$H^2 = \frac{1}{2^2} I, H^4 =, \; H^3 = \frac{1}{2^2} H$$

Hence

$$e^{-iHt} = \cos\left(\frac{1}{2}t\right) I - i\, 2\sin\left(\frac{1}{2}t\right) H$$

$$= \begin{bmatrix} \cos\left(\frac{t}{2}\right) + i\sin\left(\frac{t}{2}\right) & 0 \\ 0 & \cos\left(\frac{t}{2}\right) - i\sin\left(\frac{t}{2}\right) \end{bmatrix}$$

Hence

$$e^{-iHt} = \begin{bmatrix} e^{i\frac{t}{2}} & 0 \\ 0 & e^{-i\frac{t}{2}} \end{bmatrix}$$

$$|\chi(t) > = \begin{bmatrix} e^{i\frac{t}{2}} & 0 \\ 0 & e^{-i\frac{t}{2}} \end{bmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} e^{i\frac{t}{2}} \\ e^{-i\frac{t}{2}} \end{pmatrix}$$

Probability amplitude $pA = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}' * |\chi(t) >$

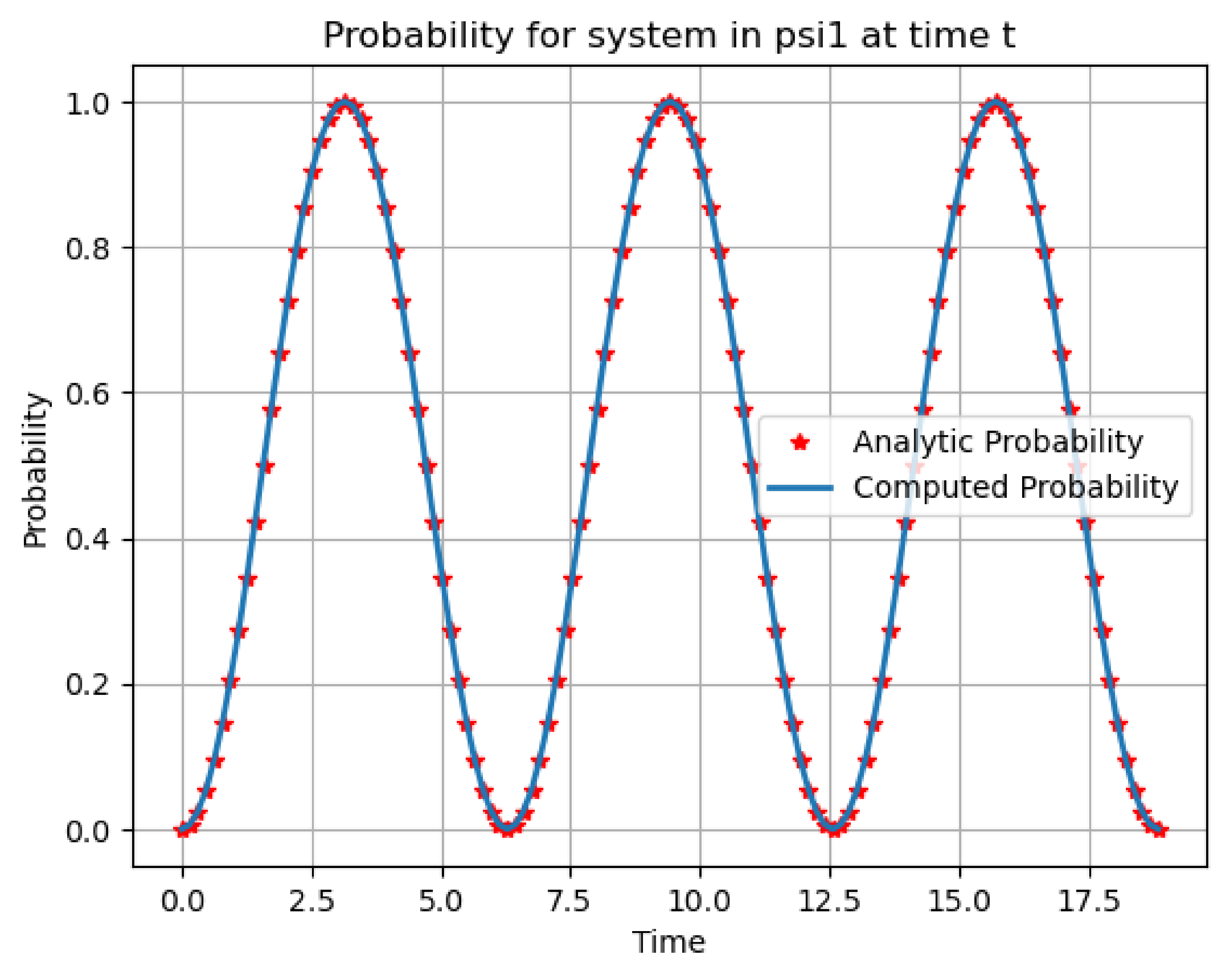
$$pA = i \sin\left(\frac{t}{2}\right)$$

$$p(t) = |pA|^2 = \sin^2 \frac{t}{2}$$

Numerical Solution

- Open a function file called time_development
- Define the Hamiltonian H
- In a loop over time from t = 0 to t = 6*pi in steps of pi/20, calculate the probability amplitude $pA =< \psi_f \left| e^{-iHt} \right| \psi_i >$
- Calculate probability *p* and plot *p* vs. t
- Compare with analytic solution

```python
import numpy as np
from scipy.linalg import expm
import matplotlib.pyplot as plt

def time_development():
    # Hamiltonian matrix H
    H = -0.5 * np.array([[1, 0], [0, -1]], dtype=complex)

    # Initial state psi_i
    psi_i = 1 / np.sqrt(2) * np.array([[1], [1]], dtype=complex)

    # Final state psi_f
    psi_f = 1 / np.sqrt(2) * np.array([[1], [-1]], dtype=complex)

    # Parameters for the time evolution
    T = 6 * np.pi
    step = np.pi / 20

    # Arrays to store results
    p = []
    pa = []
    tp = []

    # Time evolution loop
    t_vals = np.arange(0, T + step, step)  # Generate time values from 0 to T with step size

    for t in t_vals:
        # Probability amplitude
        pA1m1 = np.dot(psi_f.T.conj(), expm(-1j * H * t) @ psi_i)  # Using @ for matrix multiplication
        p.append(np.abs(pA1m1[0, 0])**2)  # Extract scalar value and compute probability
        pa.append(np.sin(t / 2)**2)  # Analytic probability
        tp.append(t)
    # Plotting
    plt.plot(tp, pa, '*r', label='Analytic Probability')
    plt.plot(tp, p, linewidth=2, label='Computed Probability')
    plt.grid(True)
    plt.title('Probability for system in psi1 at time t')
    plt.xlabel('Time')
    plt.ylabel('Probability')
    plt.legend()
    plt.show()
# Call the function
time_development()
```


Probability for system in psi1 at time t
Probability
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Time
Analytic Probability
Computed Probability

Example3

- A system with $j=5/2$ is in magnetic field $\mathbf{B} = 3\,\mathbf{i} + 2\,\mathbf{j} - 4\,\mathbf{k}$.
- Initially the system is in the state A[1;-1;i;1;0;2]
- Find the probability that the system will be at the same state at time t, using

a. Spectral resolution

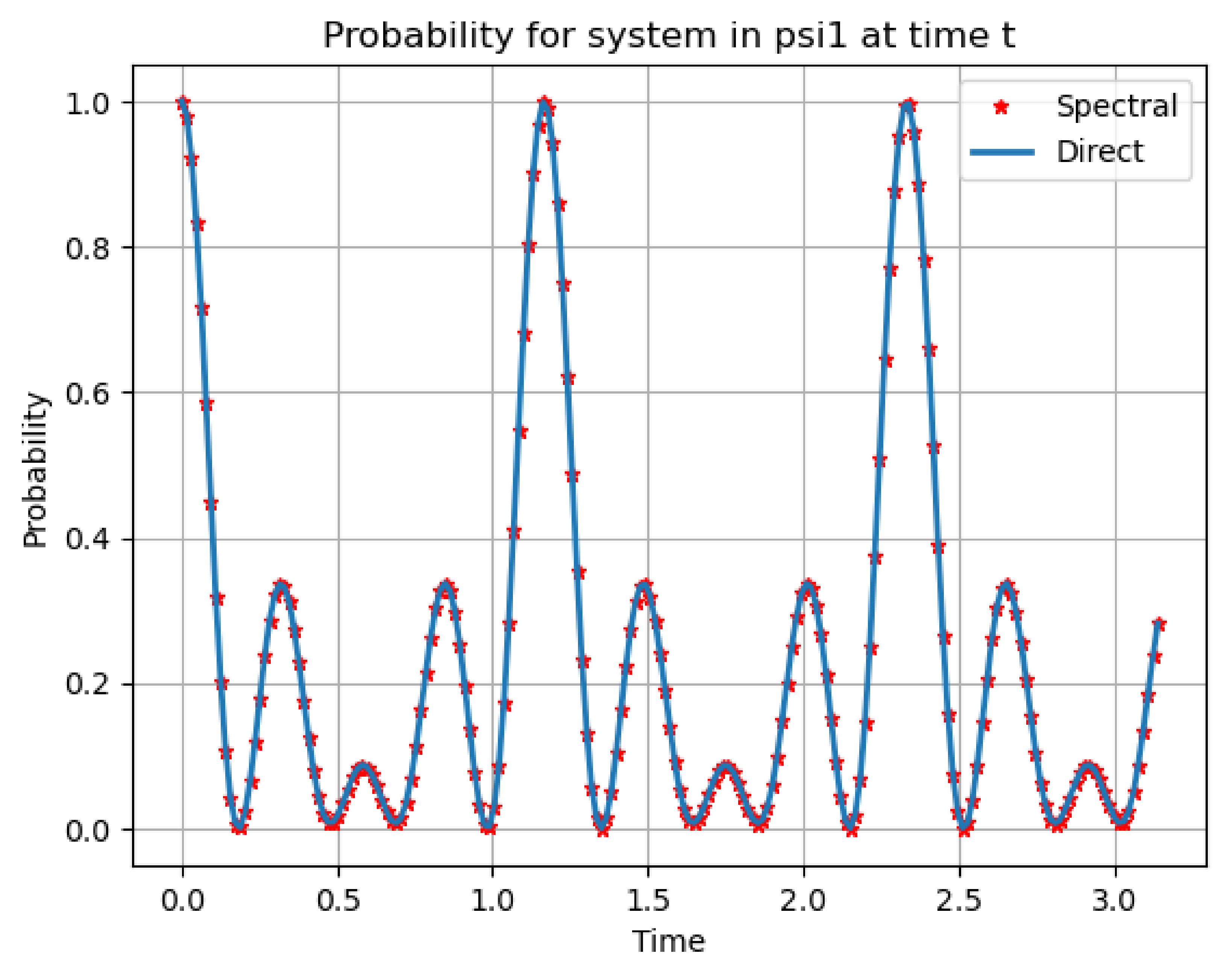
b. Direct time development

- Recall for spectral resolution we have
- $|\chi(t)>=\sum_{n=1}^{N} c_n e^{-\frac{iE_n t}{\hbar}}|n>$
- $c_n =< n|\chi_i >$
- $pA =< \chi_i|\chi(t)>=\sum_{n=1}^{N} |c_n|^2 e^{-\frac{iE_n t}{\hbar}}$
- For direct calculation we have
- $pA =< \chi_i|\chi(t)>= \quad < \chi_i|e^{-\frac{iHt}{\hbar}}|\chi_i >$

- Open function file time_spect_develop
- Call Joperator, Write B array and Hamiltonian
- Calculate Eigenvalues and Eigenvectors
- You need two for loops: One over time and an inner loop for carrying the spectral sum over eigenvalues
- The direct development is calculated within the time loop
- Create arrays to plot the probabilities vs. time

```python
import numpy as np
from scipy.linalg import expm, eig
import matplotlib.pyplot as plt

def Joperator(j):
    N = int(2 * j + 1)

    # Initialize matrices with zeros
    Jp = np.zeros((N, N), dtype=complex)
    Jm = np.zeros((N, N), dtype=complex)

    # Calculate J+ and J- matrices
    for n in range(N):
        for m in range(N):
            Jp[m, n] = np.sqrt(j * (j + 1) - (n - j) * (n - j + 1)) * (m == n + 1)
            Jm[m, n] = np.sqrt(j * (j + 1) - (n - j) * (n - j - 1)) * (m == n - 1)

    # Calculate Jx, Jy, and Jz matrices
    Jx = (Jp + Jm) / 2
    Jy = (1j * (Jm - Jp)) / 2  # `1j` represents the imaginary unit in Python
    Jz = np.diag(np.arange(-j, j + 1))

    return Jx, Jy, Jz, Jp, Jm

def time_spect_develop():
    # Elzain Fall 2014

    j = 5 / 2
    # Call the Joperator function
    Jx, Jy, Jz, Jp, Jm = Joperator(j)

    # Define the magnetic field vector B
    B = np.array([3, 2, -4])

    # Hamiltonian matrix H
    H = -(B[0] * Jx + B[1] * Jy + B[2] * Jz)

    # Eigenvalues and eigenvectors
    D, V = eig(H)

    # Sorting eigenvalues and corresponding eigenvectors
    idx = np.argsort(D)
    D = np.diag(D[idx])  # Sorted eigenvalues in diagonal matrix form
    V = V[:, idx]        # Sorted eigenvectors
```

```python
  # Compute the condition number of H
    condition_number = np.linalg.cond(H)
    print(f"Condition number of H: {condition_number}")

    # Initial state xi
    xi = np.array([1, -1, 1j, 1, 0, 2], dtype=complex)
    xi = xi / np.linalg.norm(xi)  # Normalizing xi

    # Time evolution parameters
    T = np.pi
    step = np.pi / 200

    N = int(2 * j + 1)
    tp = np.arange(0, T + step, step)  # Time points array

    # Arrays to store results
    ps = np.zeros_like(tp, dtype=float)  # Spectral resolution probability
    p = np.zeros_like(tp, dtype=float)   # Direct calculation probability

# Time evolution loop
    for k, t in enumerate(tp):
        # Spectral resolution calculation
        pAspec = 0
        for n in range(N):
            pAspec += np.exp(-1j * D[n, n] * t) * np.abs(np.vdot(V[:, n], xi))**2

        ps[k] = np.abs(pAspec)**2

        # Direct calculation using matrix exponentiation
        pA1m1 = np.vdot(xi, expm(-1j * H * t) @ xi)  # Probability amplitude
        p[k] = np.abs(pA1m1)**2  # Probability

    # Plotting
    plt.plot(tp, ps, '*r', label='Spectral', markersize=5)
    plt.plot(tp, p, linewidth=2, label='Direct')
    plt.grid(True)
    plt.title('Probability for system in psi1 at time t')
    plt.xlabel('Time')
    plt.ylabel('Probability')
    plt.legend()
    plt.show()

# Call the function
time_spect_develop()
```


Probability for system in psi1 at time t
Spectral
Direct
Probability
Time
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Homework

Consider the Hamiltonian $H = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}$

- Find analytically its eigenvalues and eigenvectors
- If the system was initially in the state $\chi_i = \frac{1}{\sqrt{2}}(1; 1)$
- Calculate and plot the probability as function of time for finding the system in the same state using analytic calculation, spectral resolution and direct time development