Lab 2: Computation of Eigenvalues and Eigenvectors

Yan Chen (100496757)

Siro Brotón (100496683)

1. Rayleigh Quotient Iteration

The Rayleigh quotient can be used in conjunction with the Shifted Inverse Power Method (sIPM). The sIPM converges to the eigenvector associated with the eigenvalue closest to the shift s, and convergence is faster when this distance is small. By updating the shift dynamically using the Rayleigh quotient, we obtain the **Rayleigh Quotient Iteration (RQI)**, which accelerates convergence significantly.

Algorithm: Rayleigh Quotient Iteration (RQI)

Given:

- A matrix A
- An initial vector x_0
- Maximum number of iterations k_{max}
- Initial shift s

Steps:

1. Set $\lambda_0 = s$
2. Normalize the initial vector: $u_0 = x_0/||x_0||$
3. For $j = 1, 2, 3, \dots, k_{max}$:
    - Solve $(A - \lambda_{j-1} I)x_j = u_{j-1}$
    - Normalize: $u_j = x_j/||x_j||$
    - Compute the Rayleigh quotient: $\lambda_j = u_j^T A u_j$

RQI converges **quadratically** for simple eigenvalues and **cubically** for symmetric matrices, meaning it requires very few iterations to reach machine precision.

Questions

(a) Implement the RQI algorithm. Run your RQI implementation using the matrix for $k_{max} = 100$ and $s = 100$:

```
A = np.array([[25, -41, 10, -6], [-41, 68, -17, 10], [10, -17, 5, -3], [-6, 10, -3, 2]])
```

At each iteration, compute the condition number of $(A - \lambda_j I)$ using `np.linalg.cond` and plot it on a **semilog-y scale**. What do you observe about the growth of the condition number? Why does the condition number increase significantly as the iteration progresses?

(b) To prevent ill-conditioning, include the following stopping criterions. Run your modified code and check whether it prevents ill-conditioning.

- Check the variations: if $|\lambda_j - \lambda_{j-1}| < \text{tol}$, stop.
- Use a Relative Condition Number Threshold:

$$\text{cond}(A - \lambda_j I) > \frac{1}{\text{machine epsilon}}$$

```
if np.linalg.cond(A - lam_new * np.eye(A.shape[0])) > 1 / np.finfo(float).eps:
  break  # Stop if nearly singular
```

  This ensures the algorithm stops when the matrix is numerically unstable.

- Track the Condition Number Growth: Instead of stopping abruptly when the condition number crosses the threshold, check its growth rate. If it increases exponentially across multiple iterations, stop early:

```
if len(cond_history) > 2 and cond_history[-1] > 10 * cond_history[-2]:
  break  # Stop if condition number increases too fast
```

- Use Residual-Based Stopping: An alternative approach is to stop when the **residual** becomes too large:

$$||(A - \lambda_j I)x_j|| < \text{tol}$$

  This ensures we halt when the computed eigenvector no longer improves:

```
    residual = np.linalg.norm((A - lam_new * np.eye(A.shape[0])) @ x_new)
    if residual < tol:
      break
```

(c) Compare the efficiency of sIPM and RQI when applied to a $N \times N$ random symmetric matrix. Use:

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

N_values = [4, 8, 16, 32, 64, 128, 256]
iterations_sIPM = []
iterations_RQI = []

for N in N_values:
    A = np.random.rand(N, N)
    B = (A @ A.T) / 2  # symmetric matrix

    # Call your sIPM and RQI functions here to get iterations and store in the lists
    # Example:
    # iterations_sIPM.append(run_sIPM(B, ...))
    # iterations_RQI.append(run_RQI(B, ...))

plt.figure()
plt.semilogy(N_values, iterations_sIPM, label='sIPM')
plt.semilogy(N_values, iterations_RQI, label='RQI')
plt.xlabel('Matrix Size N')
plt.ylabel('Number of Iterations')
plt.legend()
plt.show()
```

with

- `k = 1e4` -- maximum number of iterations
- `s = 100` -- initial shift
- `x = np.random.rand(N, 1)` -- initial vector

(d) Compare the convergence of sIPM and RQI as a function of the number of iterations when computing the eigenvalues of a 24×24 random symmetric matrix. For that purpose, you need to modify both scripts so that:

- The computed eigenvalue $\hat{\lambda}$ is stored in an array.
- Use `eig` from `numpy.linalg` to obtain the reference eigenvalue λ_e.
- Plot in the same figure the error $|\hat{\lambda} - \lambda_e|$ as a function of the number of iterations in a `semilogy` scale. Make sure to select the right reference eigenvalue when computing the error.

  Use:

```
from numpy.linalg import eig

N = 24
B = np.random.rand(N, N)
B = (B @ B.T) / 2  # symmetric matrix
x = np.random.rand(N, 1)  # initial vector
k = 20  # maximum number of iterations
s = 100  # initial shift

reference_eigenvalues, _ = eig(B)

computed_eigenvalues_sIPM = []
computed_eigenvalues_RQI = []

for iteration in range(k):
  # Run sIPM and RQI, storing the computed eigenvalue at each iteration
  # Example:
```

```
    # computed_eigenvalues_sIPM.append(run_sIPM(B, x, s, iteration))
    # computed_eigenvalues_RQI.append(run_RQI(B, x, s, iteration))
    pass

error_sIPM = np.abs(np.array(computed_eigenvalues_sIPM) - reference_eigenvalues[0])  # Assuming using first eigenvalue
error_RQI = np.abs(np.array(computed_eigenvalues_RQI) - reference_eigenvalues[0])

plt.figure()
plt.semilogy(range(k), error_sIPM, label='sIPM Error')
plt.semilogy(range(k), error_RQI, label='RQI Error')
plt.xlabel('Iterations')
plt.ylabel('Error (log scale)')
plt.legend()
plt.show()
```

- `B` -- matrix
- `x = np.random.rand(N, 1)` -- initial vector
- `k = 20` -- maximum number of iterations
- `s = 100` -- initial shift

A

```
import numpy as np

def RQI(A, x, s, k, tol=10e-6):
    u = x / np.linalg.norm(x)  # normalize vector
    I = np.eye(A.shape[0])
    lam = np.array([[s]])
    cond_history = np.zeros(k)
    it = 0
    for j in range(k): # power step
        x = np.linalg.solve(A - lam*I, u)
        u = x / np.linalg.norm(x)
        lam = np.dot(u.T, np.dot(A, u))  # Rayleigh quotient
        it = j

    return lam, u, it
    return lam, u
```

B

Generate | randomly select 5 items from a list | Close

```
import numpy as np

def RQI(A, x, s, k, tol=10e-4):
    u = x / np.linalg.norm(x)  # normalize vector
    lam = np.array([[s]])


    # eig_history = [lam]
    # eigenvectors = [u]
    # cond_history = [np.linalg.cond(A - lam * np.eye(A.shape[0]))]
    cond_history = [np.linalg.cond(A - lam * np.eye(A.shape[0]))]
    eig_history = [lam]
    it = 0

    for j in range(k):
        x = np.linalg.solve(A - lam * np.eye(A.shape[0]), u)
        u = x / np.linalg.norm(x)
        lam_new = u.T @ A @ u
        # lam_new = np.dot(u.T, np.dot(A, u))  # Rayleigh quotient
        eig_history.append(lam_new)
        cond_history.append(np.linalg.cond(A - lam_new * np.eye(A.shape[0])))
        # print(f"Iteration {j+1}:")
        # print(f"lambda = {lam}, u^T = ",u.T)

        if np.abs(lam - lam_new) < tol:
          it = j
          break
```

```
        if np.linalg.cond(A - lam_new * np.eye(A.shape[0])) > 1 / np.finfo(float).eps:
          it = j
          break  # Stop if nearly singular
          raise ValueError("Matrix is singular")

        if len(cond_history) > 2 and cond_history[-1] > 10 * cond_history[-2]:
          it = j
          break  # Stop if condition number increases too fast
          raise ValueError("Condition number increases too fast")

        residual = np.linalg.norm((A - lam_new * np.eye(A.shape[0])) @ u)
        if residual < tol:
          it = j
          break

        lam = lam_new
        it = j

    return lam, u, it


A = np.array([[25, -41, 10, -6], [-41, 68, -17, 10], [10, -17, 5, -3], [-6, 10, -3, 2]])

# x0 = np.array([1, 2, 1, 1])
x0 = np.random.rand(A.shape[0], 1)
print(np.linalg.eigvals(A))
RQI(A, x0, 100, 100)
```

```
[9.85216977e+01 1.18608886e+00 2.59197799e-01 3.30156291e-02]
(array([[95.79462226]]),
 array([[-0.49987633],
        [ 0.83164232],
        [-0.20579908],
        [ 0.127049  ]]),
 1)
```

C

We implement sIPM with the stopping criterions from B

```
import numpy as np

def sIPM(A, x, s, k, tol=1e-4):
    S = np.linalg.inv(A - np.eye(A.shape[0]) * s)
    u = x / np.linalg.norm(x)
    lam = 1 / np.dot(u.T, np.dot(S, u)) + s
    cond_history = []
    eig_history = []
    it = 0

    for i in range(k):
        x = np.dot(S, u)
        u = x / np.linalg.norm(x)
        lam_new = 1 / np.dot(u.T, np.dot(S, u)) + s
        cond_history.append(np.linalg.cond(A - lam_new * np.eye(A.shape[0])))
        eig_history.append(lam_new)

        if np.abs(lam_new - lam) < tol:
          it = i
          break

        if np.linalg.cond(A - lam_new * np.eye(A.shape[0])) > 1 / np.finfo(float).eps:
          it = i
          break  # Stop if nearly singular
          raise ValueError("Matrix is singular")

        if len(cond_history) > 2 and cond_history[-1] > 10 * cond_history[-2]:
          it = i
          break  # Stop if condition number increases too fast
          raise ValueError("Condition number increases too fast")

        residual = np.linalg.norm((A - lam_new * np.eye(A.shape[0])) @ u)
        if residual < tol:
          it = i
          break

        it = i
        lam = lam_new
```

```
    lam = 1 / np.dot(u.T, np.dot(S, u)) + s
    return lam, u, it
```

And compare the number of iteratins needed for each matrix

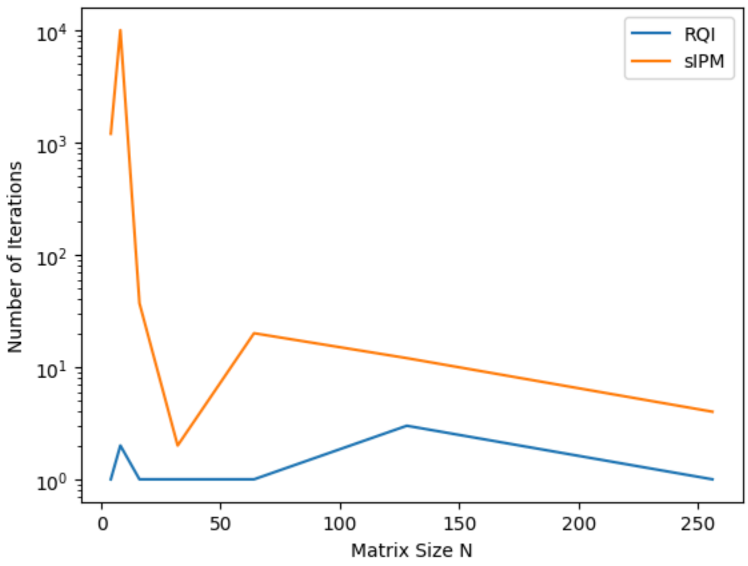
```
import numpy as np
import matplotlib.pyplot as plt

N_values = [4, 8, 16, 32, 64, 128, 256]
iterations_sIPM = []
iterations_RQI = []

for N in N_values:
    A = np.random.rand(N, N)
    B = (A @ A.T) / 2  # symmetric matrix

    # Call your sIPM and RQI functions here to get iterations and store in the lists
    # Example:
    x0 = np.random.rand(N, 1)
    iterations_RQI.append(RQI(B, x0, 100, int(1e4), 0)[2])
    iterations_sIPM.append(sIPM(B, x0, 100, int(1e4), 0)[2])

plt.figure()
plt.semilogy(N_values, iterations_RQI, label='RQI')
plt.semilogy(N_values, iterations_sIPM, label='sIPM')
plt.xlabel('Matrix Size N')
plt.ylabel('Number of Iterations')
plt.legend()
plt.show()
```



D

We compare the error from each method

```
from numpy.linalg import eig

N = 24
B = np.random.rand(N, N)
B = (B @ B.T) / 2  # symmetric matrix
x = np.random.rand(N, 1)  # initial vector
k = 20  # maximum number of iterations
s = 100  # initial shift

reference_eigenvalues, _ = eig(B)

computed_eigenvalues_sIPM = []
computed_eigenvalues_RQI = []

for iteration in range(k):
  # Run sIPM and RQI, storing the computed eigenvalue at each iteration
  # Example:
  computed_eigenvalues_sIPM.append(sIPM(B, x, s, iteration, 0)[0][0])
  computed_eigenvalues_RQI.append(RQI(B, x, s, iteration, 0)[0][0])
```

```python
error_sIPM = np.abs(np.array(computed_eigenvalues_sIPM) - reference_eigenvalues[0])  # Assuming using first eigenvalue
error_RQI = np.abs(np.array(computed_eigenvalues_RQI) - reference_eigenvalues[0])

plt.figure()
plt.semilogy(range(k), error_sIPM, label='sIPM Error')
plt.semilogy(range(k), error_RQI, label='RQI Error')
plt.xlabel('Iterations')
plt.ylabel('Error (log scale)')
plt.legend()
plt.show()
```