Coding Report

Joe Lenning

1 Problem Description

This report presents the results of applying the Power method and Inverse power method to find the eigensystem of a random matrix. The Power method is an iterative technique used to compute the largest eigenvalue of an nxn matrix, while the Inverse power method aims to find the smallest eigenvalue. Both methods involve iterative steps to estimate the corresponding eigenvectors. This report discusses the implementation of these methods, their advantages, and challenges, and provides numerical results for a randomly generated matrix. The Power method begins with an initial guess for the eigenvector corresponding to the largest eigenvalue. The initial guess is multiplied by the matrix A and then normalized to obtain an updated approximation. This process is repeated iteratively until convergence or until a specified number of iterations is reached. The algorithm converges to the eigenvector associated with the largest eigenvalue of A. The Inverse power method is a variation of the Power method that focuses on finding the smallest eigenvalue. Instead of operating on matrix A directly, the algorithm works on the inverse matrix A^(-1). By applying the Power method to A^(-1) with an initial guess for the eigenvector, the algorithm converges to the eigenvector corresponding to the smallest eigenvalue of A.

2 Results

For our base matrix A = (10* I) + B + B.T + (B.T*B) where I is an identity matrix and B is a random 50x50 matrix. We first found the base eigenvalue for our system using Pythons linear algebra packet. Then from our implementation of the power and inverse power method over 25 iterations, we found the differences between that results and a more exact one. The table below shows the results of our method implementation.
Tables:

Power Method	λ Time	λ Error	Vector Error
Smallest Eig	0.00020697s	0.5742366	0.03000535
Largest Eig	.00003029s	190.97492	0.0310897

Inverse Power Method	λ Time	λ Error	Vector Error
Smallest Eig	0.000233s	0.55267	0.030143

Largest Eig	0.001452s	197.021	0.02909

Convergence Speed:
In general, the inverse power method tends to converge faster than the power method when the desired eigenvalue is known to be closer to the smallest eigenvalue of the matrix. However, in the given scenario, the power method was observed to be faster.
The presence of a dominant eigenvalue (in this case, the eigenvalue associated with the 10*I term) in matrix A can slow down the convergence of the power method. The power method converges faster when the dominant eigenvalue is significantly larger than the other eigenvalues, as it amplifies the effect of the dominant eigenvalue during iterations.
On the other hand, the inverse power method involves solving a system of linear equations using Gaussian elimination, which can be computationally expensive and slow down convergence when the matrix has more operations, as in the case of 10*I in matrix A. Thus, the additional computational complexity of the inverse power method in this scenario may have contributed to its slower convergence.

λ Error (Eigenvalue Error):
The observed high error for the largest eigenvalue approximation can be attributed to the relatively small number of iterations (25) chosen in the code. Finding accurate approximations for the largest eigenvalues of a random 50x50 matrix may require more iterations, as the power method typically converges slowly for larger eigenvalues.
Additionally, the accuracy of the initial guess plays a crucial role in convergence. If the initial guess is far from the true value, it may take more iterations to converge to the desired eigenvalue. Therefore, refining the initial guess or increasing the number of iterations could potentially improve the accuracy of the largest eigenvalue approximation.

Vector Error:
The relatively small vector error suggests that the computed eigenvectors are close to the true eigenvectors, despite the eigenvalue approximation errors.
The chosen tolerance level of 1e-8 for the vector error indicates that the computed eigenvectors have a high degree of similarity to the true eigenvectors, as the norm or absolute difference between the matrix-vector product and the scaled eigenvector is within a very small range.
It's worth noting that the vector error provides information about the quality of the eigenvector approximation, independent of the eigenvalue approximation. In this case, the vector error suggests that the iterative methods are relatively successful in approximating the eigenvectors, even if the eigenvalue convergence is not as accurate.

3 Collaboration:
None

4 Academic Integrity

On my personal integrity as a student and member of the UCD community, I have not given nor received any unauthorized assistance on this assignment.

```
import numpy as np
import time
#Basic info
max_iterations = 25
num_matrices = 25
tol = 1e-8
B = np.random.randn(50, 50)
A = 10 * np.eye(50) + B + B.T + B.T @ B
#small eig
smallest_eigenvalues = np.sort(np.linalg.eigvals(A))[:2]
print("Smallest eigenvalues:")
print(smallest_eigenvalues)
#large eig
largest_eigenvalues = np.sort(np.linalg.eigvals(A))[-2:]
print("Largest eigenvalues:")
print(largest_eigenvalues)

# Create an array to store the results for each iteration
power_results_largest = []
power_results_smallest = []
inverse_results_largest = []
inverse_results_smallest = []

# Define power_results and inverse_results lists
power_results = []
inverse_results = []

for _ in range(num_matrices):
    # Generate a new random matrix A for each iteration
    B = np.random.randn(50, 50)
    A = 10 * np.eye(50) + B + B.T + B.T @ B

    # Perform Power Method - Largest Eigenvalue
    power_guess = np.random.randn(A.shape[1], 1)
    power_errors_largest = []
    power_times_largest = []
    power_vector_errors_largest = []

    for iteration in range(max_iterations):
        start_time = time.time()

        # updated eigenvector
        power_guess = A @ power_guess / np.linalg.norm(A @ power_guess)
```

```
        # update largest eigenvalue
        lam = (power_guess.T @ A @ power_guess)

        # check convergence
        error = np.abs(lam - largest_eigenvalues[-1])
        if error < tol:
            break

        # calculate eigenvector error
        eigvec_error = np.linalg.norm(A @ power_guess - lam * power_guess,
np.inf) / np.abs(lam)
        power_vector_errors_largest.append(eigvec_error)

        # store errors and time taken
        power_errors_largest.append(error)
        power_times_largest.append(time.time() - start_time)

    # Store the results for Power Method - Largest Eigenvalue
    power_results_largest.append((lam, power_times_largest,
power_errors_largest, power_vector_errors_largest))
    power_results.append((lam, power_times_largest, power_errors_largest,
power_vector_errors_largest))

    # Perform Power Method - Smallest Eigenvalue
    power_guess = np.random.randn(A.shape[1], 1)
    power_errors_smallest = []
    power_times_smallest = []
    power_vector_errors_smallest = []

    for iteration in range(max_iterations):
        start_time = time.time()

        # updated eigenvector
        power_guess = np.linalg.inv(A) @ power_guess /
np.linalg.norm(np.linalg.inv(A) @ power_guess)

        # update smallest eigenvalue
        lam = (power_guess.T @ A @ power_guess)

        # check convergence
        error = np.abs(lam - smallest_eigenvalues[0])
        if error < tol:
            break

        # calculate eigenvector error
        eigvec_error = np.linalg.norm(A @ power_guess - lam * power_guess,
np.inf) / np.abs(lam)
        power_vector_errors_smallest.append(eigvec_error)
```

```
        # store errors and time taken
        power_errors_smallest.append(error)
        power_times_smallest.append(time.time() - start_time)

    # Store the results for Power Method - Smallest Eigenvalue
    power_results_smallest.append((lam, power_times_smallest,
power_errors_smallest, power_vector_errors_smallest))
    power_results.append((lam, power_times_smallest, power_errors_smallest,
power_vector_errors_smallest))

    # Perform Inverse Power Method - Largest Eigenvalue
    inverse_guess = np.random.randn(A.shape[1], 1)
    inverse_errors_largest = []
    inverse_times_largest = []
    inverse_vector_errors_largest = []

    for iteration in range(max_iterations):
        start_time = time.time()

        # updated eigenvector
        inverse_guess = np.linalg.solve(A, inverse_guess) /
np.linalg.norm(np.linalg.solve(A, inverse_guess))

        # update largest eigenvalue
        lam = (inverse_guess.T @ A @ inverse_guess)

        # check convergence
        error = np.abs(lam - largest_eigenvalues[-1])
        if error < tol:
            break

        # calculate eigenvector error
        eigvec_error = np.linalg.norm(A @ inverse_guess - lam * inverse_guess,
np.inf) / np.abs(lam)
        inverse_vector_errors_largest.append(eigvec_error)

        # store errors and time taken
        inverse_errors_largest.append(error)
        inverse_times_largest.append(time.time() - start_time)

    # Store the results for Inverse Power Method - Largest Eigenvalue
    inverse_results_largest.append((lam, inverse_times_largest,
inverse_errors_largest, inverse_vector_errors_largest))
    inverse_results.append((lam, inverse_times_largest, inverse_errors_largest,
inverse_vector_errors_largest))

    # Perform Inverse Power Method - Smallest Eigenvalue
    inverse_guess = np.random.randn(A.shape[1], 1)
    inverse_errors_smallest = []
```

```
    inverse_times_smallest = []
    inverse_vector_errors_smallest = []

    for iteration in range(max_iterations):
        start_time = time.time()

        # updated eigenvector
        inverse_guess = np.linalg.solve(A, inverse_guess) /
np.linalg.norm(np.linalg.solve(A, inverse_guess))

        # update smallest eigenvalue
        lam = (inverse_guess.T @ A @ inverse_guess)

        # check convergence
        error = np.abs(lam - smallest_eigenvalues[0])
        if error < tol:
            break

        # calculate eigenvector error
        eigvec_error = np.linalg.norm(A @ inverse_guess - lam * inverse_guess,
np.inf) / np.abs(lam)
        inverse_vector_errors_smallest.append(eigvec_error)

        # store errors and time taken
        inverse_errors_smallest.append(error)
        inverse_times_smallest.append(time.time() - start_time)

    # Store the results for Inverse Power Method - Smallest Eigenvalue
    inverse_results_smallest.append((lam, inverse_times_smallest,
inverse_errors_smallest, inverse_vector_errors_smallest))
    inverse_results.append((lam, inverse_times_smallest,
inverse_errors_smallest, inverse_vector_errors_smallest))

# Calculate the average time, error, and vector error for Power Method -
Largest Eigenvalue
power_average_time_largest = np.mean([np.mean(result[1]) for result in
power_results_largest])
power_average_error_largest = np.mean([np.mean(result[2]) for result in
power_results_largest])
power_average_vector_error_largest = np.mean([np.mean(result[3]) for result in
power_results_largest])

# Calculate the average time, error, and vector error for Power Method -
Smallest Eigenvalue
power_average_time_smallest = np.mean([np.mean(result[1]) for result in
power_results_smallest])
power_average_error_smallest = np.mean([np.mean(result[2]) for result in
power_results_smallest])
```

```
power_average_vector_error_smallest = np.mean([np.mean(result[3]) for result in
power_results_smallest])

# Calculate the average time, error, and vector error for Inverse Power Method
- Largest Eigenvalue
inverse_average_time_largest = np.mean([np.mean(result[1]) for result in
inverse_results_largest])
inverse_average_error_largest = np.mean([np.mean(result[2]) for result in
inverse_results_largest])
inverse_average_vector_error_largest = np.mean([np.mean(result[3]) for result
in inverse_results_largest])

# Calculate the average time, error, and vector error for Inverse Power Method
- Smallest Eigenvalue
inverse_average_time_smallest = np.mean([np.mean(result[1]) for result in
inverse_results_smallest])
inverse_average_error_smallest = np.mean([np.mean(result[2]) for result in
inverse_results_smallest])
inverse_average_vector_error_smallest = np.mean([np.mean(result[3]) for result
in inverse_results_smallest])

#results
print("Power Method - Largest Eigenvalue:")
print("Average Time:", power_average_time_largest)
print("Average Error:", power_average_error_largest)
print("Average Vector Error:", power_average_vector_error_largest)

print("Power Method - Smallest Eigenvalue:")
print("Average Time:", power_average_time_smallest)
print("Average Error:", power_average_error_smallest)
print("Average Vector Error:", power_average_vector_error_smallest)

print("Inverse Power Method - Largest Eigenvalue:")
print("Average Time:", inverse_average_time_largest)
print("Average Error:", inverse_average_error_largest)
print("Average Vector Error:", inverse_average_vector_error_largest)

print("Inverse Power Method - Smallest Eigenvalue:")
print("Average Time:", inverse_average_time_smallest)
print("Average Error:", inverse_average_error_smallest)
print("Average Vector Error:", inverse_average_vector_error_smallest)
```