Linear Algebra and Applications Homework #08

Submitted By: Rifat Bin Rashid

RUID: 237000174

Date: Apr 29, 2025

Problem 1:

(a) $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$

power method: let $x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$$x_1 = \frac{Ax_0}{\|Ax_0\|} = \frac{1}{\sqrt{5}}\begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$Ax_0 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$x_2 = \frac{Ax_1}{\|Ax_1\|} = \frac{1}{\sqrt{10}}\begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$Ax_1 = \frac{1}{\sqrt{5}}\begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$x_3 = \frac{Ax_2}{\|Ax_2\|} = \frac{1}{\sqrt{17}}\begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$Ax_2 = \frac{1}{\sqrt{10}}\begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$\vdots$

$$x_{n-1} = \frac{Ax_{n-2}}{\|Ax_{n-2}\|}$$

$$= \frac{1}{\sqrt{n^2+1}}\begin{bmatrix} n \\ 1 \end{bmatrix} = \frac{1}{\sqrt{1+\frac{1}{n^2}}}\begin{bmatrix} 1 \\ 1/n \end{bmatrix}$$

So, $n \to \infty$

$$x_\infty \longrightarrow \frac{1}{\sqrt{1+0}}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

So, the power method converges to $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ which is an eigenvector with eigen values 1. Here the power method converges to the eigenvector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ because A is not diagonalizable with a dominant eigenvalue. The lack of dominat eigenvalue here $(\lambda_1 = \lambda_2 = 1)$ means the method converges to only avaliable eigenvector

QR Algorithm:

$$\left\{ \begin{array}{l} A_0 = A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \\ A_0 = Q_0 R_0 \\ Q_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_0 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \end{array} \right.$$

(Because A is alread upper triangular)

$$\left\{ \begin{array}{l} A_1 = R_0 Q_0 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \\ A_1 = Q_1 R_1 \\ Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \end{array} \right.$$

$$A_2 = R_1 Q_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

So here the matrix A remains unchanged in each iteration. The algorithm converges to A itself, which is already in upper triangular form with eigenvalues $\lambda = 1$ (double)

So, it fails to converge in a diagonal form and QR method can't improve A, but it reveals its eigenvalues ($\lambda_1 = \lambda_2 = 1$) in the diagonal.

b

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

power method:

Let $x_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$$x_1 = \frac{Ax_0}{\|Ax_0\|} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

$$Ax_0 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$x_2 = \frac{Ax_1}{\|Ax_1\|} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$Ax_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$x_3 = \frac{Ax_2}{\|Ax_2\|} = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

$$Ax_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

$x_4 = \frac{Ax_3}{\|Ax_3\|} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ | $Ax_3 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$

$x_5 = \frac{Ax_4}{\|Ax_4\|} = \begin{bmatrix} -1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$ | $Ax_4 = \begin{bmatrix} -1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$

Here the sequence does not converge, but oscillates between vectors. Because A is a matrix with complex eigenvalues here. A is a rotation matrix with eigenvalues $\lambda = e^{\pm i\pi/4}$ here & their equal magnitudes cause oscillations. Here, the power method fails due to equal magnitude complex eigenvalues (no dominance)

QR Algorithm:

$A_0 = A.$

$A_0 = Q_0 R_0$

Here, A is already orthogonal

So, $Q_0 = A_0$, $R_0 = I$

$A_1 = R_0 Q_0 = IA = A$

$A_2 = A$

$A_3 = A$

So, the matrix A remains unchanged in each iteration. The algorithm converges to A itself, which is orthogonal with eigenvalues $\lambda = e^{\pm i\pi/4}$. Here in the QR algorithm, the eigenvalues are not revealed ~~in~~ on the diagonal (because they are complex). So, the QR Algorithm can't simplify A further, it is already orthogonal.

Problem 2 :

Problem 2.

(a) Write a small program that implements the Power Method.

(b) Write a small program that implements the QR Algorithm. For this task you may call a system routine that generates the QR decomposition of an $n \times n$ matrix.

(c) Apply the programs you wrote in (a) and (b) to the symmetric, tridiagonal, 10×10 matrix A, given by $a_{i\,i} = 2$, $i = 1, \ldots, 10$ and $a_{i\,i-1} = -1$, $i = 2, \ldots, 10$.

(d) Apply the programs you wrote in (a) and (b) to the symmetric, full, 10×10 matrix A, given by $a_{i\,i} = 2$, $i = 1, \ldots, 10$ and $a_{i\,j} = -1/(i + j)$ for all entries with $j \neq i$.

(e) Briefly describe the differences and similarities between the results in (c) and (d).

2(a): Program to implement Power Method :

```
def power_method(A, max_iter=1000, tol=1e-6):
   n = A.shape[0]
   x = np.random.rand(n)
   x = x / np.linalg.norm(x)
   iterations = 0

   for i in range(max_iter):
       Ax = A @ x
       x_new = Ax / np.linalg.norm(Ax)
       iterations += 1

       if np.linalg.norm(x_new - x) < tol:
           break
       x = x_new

   eigenvalue = x.T @ A @ x
   return eigenvalue, x, iterations
```

Steps followed:

1. At first, I initialized a random vector x_0 and normalized it.

2. Then the matrix A is repeatedly multiplied with the current vector $\mathbf{x_k}$ to compute $\mathbf{Ax_k}$, which is then normalized to obtain $\mathbf{x_{k+1}}$.

3. The process is terminated when the difference between them falls below a predefined tolerance.

4. Finally, the dominant eigenvalue is estimated using the Rayleigh quotient.

2(b): Program that implements the QR Algorithm:

```python
def qr_algorithm(A, max_iter=1000, tol=1e-6):
   A_k = np.copy(A)
   iterations = 0

   for i in range(max_iter):
       Q, R = qr(A_k)
       A_k = R @ Q
       iterations += 1

       if np.max(np.abs(np.tril(A_k, -1))) < tol:
           break

   eigenvalues = np.diag(A_k)
   return eigenvalues, iterations
```

Steps followed:

1. The matrix A is copied to initialize A_0.
2. In each iteration, A_k is decomposed into an orthogonal matrix Q_k and an upper triangular matrix R_k using ***scipy.linalg.qr.***
3. A_{k+1} is computed as R_kQ_k, preserving eigenvalues through similarity transformations.
4. The algorithm stops when the subdiagonal entries of A_k become negligible.
5. Finally, the eigenvalues are read from the diagonal of the final A_k.

2(c): Applying the programs in (a) and (b) to the symmetric, tridiagonal, 10*10 matrix A:

```python
def create_tridiagonal(n=10):
   A = np.zeros((n, n))
   np.fill_diagonal(A, 2)
   np.fill_diagonal(A[1:], -1)
   np.fill_diagonal(A[:, 1:], -1)
   return A
```

```python
A_tri = create_tridiagonal()
print("\n=== Tridiagonal Matrix Results ===")

# Power Method
eigval_tri, eigvec_tri, iter_power_tri = power_method(A_tri)
print(f"\nPower Method:")
print(f"Dominant eigenvalue: {eigval_tri:.6f}")
print(f"Dominant eigenvector:\n{eigvec_tri}")
print(f"Iterations: {iter_power_tri}")

# QR Algorithm
eigvals_tri, iter_qr_tri = qr_algorithm(A_tri)
print(f"\nQR Algorithm:")
print(f"All eigenvalues:\n{np.sort(eigvals_tri)}")
print(f"Iterations: {iter_qr_tri}")
```

Steps followed:

1. A 10×10 tridiagonal matrix with diagonal entries 2 and off-diagonal entries -1.

2. Power method defined in 2(a) is applied to A.

3. The QR method defined in 2(b) is applied to A.

Results:

```
=== Tridiagonal Matrix Results ===
```

Power Method:

```
Dominant eigenvalue: 3.918986
Dominant eigenvector:
[ 0.12013495 -0.23053639  0.32225963 -0.38787368  0.42206325 -0.42205931
  0.3878631  -0.32224577  0.23052365 -0.12012738]
Iterations: 176
```

QR Algorithm:

```
All eigenvalues:
[0.08101405 0.31749293 0.69027853 1.16916997 1.71537032 2.28462968
 2.83083003 3.30972147 3.68250707 3.91898595]
Iterations: 210
```

2(d) : Applying the programs in (a) and (b) to the symmetric, full, 10*10 matrix A:

```python
def create_full_matrix(n=10):
   A = np.zeros((n, n))
   for i in range(n):
       for j in range(n):
           A[i,j] = 2 if i == j else -1/(i+j+2)
   return A
```

```python
A_full = create_full_matrix()
print("\n=== Full Matrix Results ===")

# Power Method
eigval_full, eigvec_full, iter_power_full = power_method(A_full)
print(f"\nPower Method:")
print(f"Dominant eigenvalue: {eigval_full:.6f}")
print(f"Dominant eigenvector:\n{eigvec_full}")
print(f"Iterations: {iter_power_full}")

# QR Algorithm
eigvals_full, iter_qr_full = qr_algorithm(A_full)
print(f"\nQR Algorithm:")
print(f"All eigenvalues:\n{np.sort(eigvals_full)}")
print(f"Iterations: {iter_qr_full}")
```

Steps followed:

1. A 10×10 full matrix with diagonal entries 2 and off-diagonal entries -1/(i+j).

2. Power method defined in 2(a) is applied to A.

3. The QR method defined in 2(b) is applied to A.

Results:

=== Full Matrix Results ===

Power Method:

```
Dominant eigenvalue: 2.342040
Dominant eigenvector:
[-0.77735034  0.60113948  0.16447079  0.07388177  0.03731598  0.01887837
  0.00845087  0.00212862 -0.00188152 -0.00450102]
Iterations: 178
```

QR Algorithm:

```
All eigenvalues:
[0.95164007 2.02077428 2.05284991 2.0602297  2.06970147 2.08256609
 2.10125536 2.13118591 2.18775695 2.34204025]
Iterations: 1000
```

2(e): The differences and similarities between the results in (c) and (d):

=== Comparison of Results ===
Tridiagonal Matrix:
- Power Method converged in 176 iterations
- QR Algorithm converged in 210 iterations

Full Matrix:
- Power Method converged in 178 iterations
- QR Algorithm converged in 1000 iterations

Here, for the tridiagonal matrix, the tridiagonal structure provides nice eigenvalue separation, so the eigenvalues are well-spaced, allowing faster convergence for both of the methods. But, the power method only finds the largest eigenvalue, while the QR Algorithm computes the entire spectrum of eigenvalues. So, QR is more comprehensive but computationally heavier, which is reflected in more iterations for QR here.

For the full matrix, dense coupling creates eigenvalue clustering. Here, off-diagonal entries decay slowly, maintaining strong interactions. QR Algorithm sensitivity needs clear eigenvalue separation so it hits max iterations (1000) indicating eigenvalues are too close. On the other hand, the Power Method depends mainly on the dominant eigenvalue gap (the ratio of the two largest eigenvalues), not directly on the matrix structure.

Tridiagonal: $\lambda_1 \approx 3.91898595$, $\lambda_2 \approx 3.68250707$ $\rightarrow$ **ratio ≈ 1.0625**

Full matrix: $\lambda_1 \approx 2.34204025$, $\lambda_2 \approx 2.18775695$ → **ratio ≈ 1.0733**

So , power method needed similar iterations for both matrices because λ_1 was similarly dominant in both matrices.

Overall, the observed results affirm that structured sparsity, as in tridiagonal matrices, can lead to faster convergence behavior for both approaches compared to the full matrix case. However, QR method is much more dependent on matrix structure than power method.