Assignment 6

Page No.
Date

10) $A \in \mathbb{C}^{m \times m}$ tridiagonal and hermitian
subdiagonal and superdiagonal nonzero.

$Av = \lambda v \Rightarrow \lambda \in \mathbb{C}$ eigenvalue, $v \in \mathbb{C}^m$ eigenvector

$B = A - \lambda I$

Assume $u \in \mathbb{C}^{m-1}$ removing the last row of v.

$(A - \lambda I)u \Rightarrow$ The last row of $(A-\lambda I)u$ is zero.

$\therefore u$ can be eigenvector for the corresponding value λ.

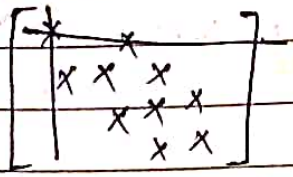
Therefore rank $(A - \lambda I)$ must be atleast $(m-1)$

if λ has algebraic multiplicity of k then rank $(A-\lambda I)$ must be atmost $(m-k)$

Since rank is atleast $(m-1)$ here $k=1$. for all eigenvalue
Each eigenvalue has algebraic multiplicity of 1 then, A has m distinct eigenvalues.

Another way

let $B = A - \lambda I$ Remove 1st row and column.



$(B = A - \lambda I)_{2:m, 2:m}$

$B_{ii} = (A - \lambda I)_{i+1,i}$

$= A_{i+1,i} \quad (1 \le i \le m-1)$

$B_{ij} = (A - \lambda I)_{i+1,j}$

$= A_{i+1,j} = 0 \quad \forall\, i \ge j$

$(i+1 > j)$.

Rank (B) = full rank.

if rank $(A - \lambda I) \le m-2$. There exist linearly dependent rows in $(A - \lambda I)$ Contradicts with full rank of B.

rank $(A - \lambda I) \ge m-1$

$A = A^*$ nondefective.

$A = X^* \Delta X$ Diagonalization

$A = \Delta$ (Both have same eigenvalues).

$A - \lambda I = X^* \Delta X - \lambda I$

$X^* (\Delta - \lambda I) X$

rank $(\Delta - \lambda I)$ = rank $(A - \lambda I) \ge m-1$

b) $A \in q^{m \times m}$ upper hessenberg $a_{ij} \neq 0$ if $j = i-1$ or $j > i$

$Av = \lambda v$

$Av_1 = \lambda_1 v_1 \quad Av_2 = \lambda_2 v_2$

if $\lambda_1 = \lambda_2$

$A(v_1 - v_2) = \lambda_1 (v_1 - v_2) = 0.$

$v_1 - v_2 = 0.$

Suppose $A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$ $m = 4.$

$\det(A - \lambda I) = \begin{bmatrix} 1-\lambda & 1 & 1 & 1 \\ 1 & 1-\lambda & 1 & 1 \\ 0 & 1 & 1-\lambda & 1 \\ 0 & 0 & 1 & 1-\lambda \end{bmatrix}$

$\lambda^2 (\lambda - 1)(\lambda - 3) = 0$

$\lambda_1 = 3, \lambda_2 = 1 \quad \lambda_3 = \lambda_4 = 0.$

Algebraic multiplicity of the eigenvector 0 is 2.
Therefore eigenvalue of A are not necessarily distinct.

2a) $A \in \mathbb{C}^{m \times m}$ be nonsingular $b \in \mathbb{C}^m \setminus \{0\}$

$\mathcal{K}_n = \{ b, Ab, A^2b, \ldots A^{n-1}b \}$ $n \in \{1, 2, \ldots m\}$.

dim $(\mathcal{K}_n)$ is the no. of linearly Independent vectors in the set $(b, Ab, A^2b \ldots A^{n-1}b)$

Consider A as nonsingular idempotent matrix $(A^2 = A)$

A = nonsingular rank $(A) = m$.

A has eigenvector v that can be expressed as

$$v = A^{k-1} b \quad k \leq n$$

$$A^k v = A^k (A^{k-1} b) = A^{2k-1} b.$$

Also, $A^2 = A \Rightarrow A^2 b = Ab$.

Thus the vector $(b, Ab, A^2b, \ldots A^{k n-1}b)$ are all not linearly Independent. dim $(\mathcal{K}_n) < n$ as there are less than n linearly Independent vectors.

b) det $(A) \neq 0$ nonsingular.

$Ax = b$. Assume $x \notin \{0\}$.

$A^2 = A$.

$AAx = Ab$.

$Ax = Ab$.

$A^{-1} Ax = A^{-1} Ab$

$x = b$

$b \in$ span of $\mathcal{K}_n$.

$\therefore x \in \mathcal{K}_n$. if dim $(\mathcal{K}_n) < n$.

Exercise 3 (a), (b), (c), (d)

```python
def gershgorin(A):
    λ_min, λ_max = 0,0
    r = []
    c_max = A[0][0]
    c_min = A[0][0]
    m = A.shape[0]
    for i in range (m):
      r.append(0)
      for j in range (m):
        if(i!=j):
            r[i] += np.absolute(A[i][j])

        if(i==j):
          if (A[i][i]) > c_max:
            c_max = A[i][i]
        if(A[i][i] == c_max):
            λ_max = c_max + r[i]

        if(i==j):
          if (A[i][i]) < c_min:
            c_min = A[i][i]
        if(A[i][i] == c_min):
            λ_min = c_min - r[i]

    return λ_min, λ_max
```

Figure 1. Code for Gershgorin's Theorem

```python
def power(A, v0):
    v = v0.copy()
    λ = 0
    err = []

    k=1
    for k in iter(int, 1):
        w = np.matmul(A, np.transpose(v))
        v = w / np.linalg.norm(w)
        λ = np.matmul(np.matmul(np.transpose(v), A), v)
        err.append(np.linalg.norm((np.matmul(A, np.transpose(v)))
                                  - (λ * np.transpose(v)), np.inf))

        if (err[k-1] <= 10**-13):
            break

    return v, λ, err
```

Figure 2. Code for Power iteration method

```python
def inverse(A, v0, μ):
    v = v0.copy()
    λ = 0
    err = []

    k=1
    m = A.shape[0]
    for k in iter(int, 1):
        w = np.matmul(np.linalg.inv(A - μ * np.identity(m)), np.transpose(v))
        v = w / np.linalg.norm(w)
        λ = np.matmul(np.matmul(np.transpose(v), A), v)
        err.append(np.linalg.norm(np.matmul(A, np.transpose(v))
                                  - (λ * np.transpose(v)), np.inf))

        if (err[k-1] <= 10**-13):
            break

    return v, λ, err
```

Figure 3. Code for Inverse Power iteration method

```python
def rayleigh(A, v0):
    v = v0.copy()
    λ = 0
    err = []

    m = A.shape[0]
    λ = np.matmul(np.matmul(np.transpose(v), A), v)
    for k in iter(int, 1):
        w = np.matmul(np.linalg.inv(A - λ * np.identity(m)), np.transpose(v))
        v = w / np.linalg.norm(w)
        λ = np.matmul(np.matmul(np.transpose(v), A), v)
        err.append(np.linalg.norm(np.matmul(A, np.transpose(v))
                                  - (λ * np.transpose(v)), np.inf))

        if (err[k-1] <= 10**-13):
            break

    return v, λ, err
```

Figure 4. Code for Rayleigh Quotient iteration method

Exercise 3 (e)

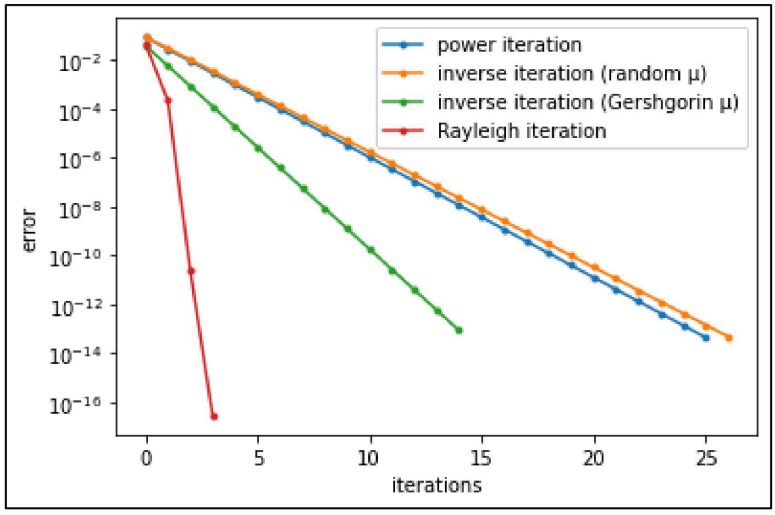


Figure 6. Plot of various iteration algorithms

The plot shows the error rates of four different iteration methods: Power Iteration, Inverse Iteration (random μ), Inverse Iteration (Gershgorin μ), and Rayleigh Iteration. As the number of iterations increases, all methods show a decrease in error, but at different rates.

- **Power Iteration:** In the above plot, the blue line represents Power iteration that has convergence with linearly decreasing slope.
  - It starts with the highest error but decreases rapidly in the initial iterations. However, it seems to plateau around 10^-6 after about five iterations. This is typical for power iteration, which is known for its simplicity but not necessarily for its speed of convergence.
  - It is preferred on the diagonalizing matrix A to determine the largest absolute eigenvalue λ and its corresponding eigenvector.
  - One can see the convergence is linear. The error is reduced by a factor of λ_2/λ_1 at every iteration.
  - If the largest two eigenvalues are very close in magnitude, then the convergence is very slow.
  - Hence, we apply this method for large sparse matrix with proper care.

- **Inverse Iteration (random μ):**
  - This method also starts with a high error but decreases at a much slower rate compared to power iteration. This suggests that using an inverse iteration with a random initial eigenvalue estimate can lead to faster convergence than power iteration.
  - It approximates the eigenvector by taking an approximation to the corresponding eigenvalues.
  - We take $\mu \in \mathbb{C}$ as the approximate eigenvalue of the matrix A.
  - If μ is closer to the actual eigenvalue λ then the algorithm will converge rapidly.
  - Similar to Power iteration, it exhibits linear convergence. The linear convergence is controlled by the value of μ.
  - This algorithm is used if good approximation of eigenvalue is used.

- **Inverse Iteration (Gershgorin μ):**
  - It has an intermediate starting error and decreases steadily over iterations. This indicates that using Gershgorin's circle theorem to provide a better initial estimate for the eigenvalue can improve the efficiency of the inverse iteration.
  - It converges linearly at each step.
  - Here, μ is estimated based on the Gershgorin theorem which returns λ_{max} with the algorithm ($\mu = \lambda_{max}$).

- **Rayleigh Iteration:**
  - This method starts with an extremely high error but drops dramatically and continuously over iterations, reaching the lowest errors among all methods.
  - The Rayleigh iteration's rapid convergence is expected due to its adaptive nature, adjusting itself according to eigenvalue approximations at each step.
  - Similar to the inverse iteration but here we replace the updated eigenvalue at the end of each iteration.
  - As it is continuously improving the eigenvalue therefore the rate of convergence is at a higher rate about each iteration tripling the amount of accuracy.
  - It converges cubically for Hermitian matrix given initially a vector that is close to the eigenvector.

In conclusion, the plots demonstrate the effectiveness of these iterative methods in approximating eigenvalues, with the Rayleigh iteration showing the fastest convergence. The results align with the theoretical expectations of these methods. However, the choice of method in practice would depend on the specific requirements of the problem, such as the size of the matrix, the need for speed or accuracy, and the availability of good initial estimates.

Exercise 3 (f)

```python
if __name__ == '__main__':
    pass
    A,v0=randomInput(5)
    A=np.diag(np.full(6,3))+np.diag(np.ones(5),1)+np.diag(np.ones(5),-1)
    v0 = np.zeros(np.shape(A)[0])
    v0[4] = 1
    μ=10
    print("A = ",A)
    print("V =",v0)
    print("μ =",μ)

    λ_min, λ_max = gershgorin(A)
    print("\nGershgorin circle :\nλ_min = {}, \tλ_max = {}".format(λ_min, λ_max))

    print("\nPower Iteration Method :")
    v, λ, err=power(A,v0)
    print("λ = {}, \terr = {}".format(λ,err[-1]))

    print("\nInverse Iteration Method :")
    v, λ, err=inverse(A, v0, μ)
    print("λ = {}, \terr = {}".format(λ,err[-1]))

    print("\nRayleigh Quotient Iteration Method :")
    v, λ, err=rayleigh(A, v0)
    print("λ = {}, \terr = {}".format(λ,err[-1]))
```

Figure 7. Code for Implementation for different values

```
A =  [[14  0  1]
 [-3  2 -2]
 [ 5 -3  3]]
V = [1 1 1]
μ = 10

Gershgorin circle :
λ_min = -3,     λ_max = 15

Power Iteration Method :
λ = 14.518079350254576,     err = 6.750155989720952e-14

Inverse Iteration Method :
λ = 14.518079350254471,     err = 9.50350909079134e-14

Rayleigh Quotient Iteration Method :
λ = 14.518079350254528,     err = 1.7763568394002505e-15
```

```
A =  [[-3.5 -0.5  0. ]
 [-0.5 -1.5  0.5]
 [ 0.   0.5  1. ]]
V = [0 0 1]
μ = 5

Gershgorin circle :
λ_min = -4.0,   λ_max = 1.5

Power Iteration Method :
λ = -3.620956698902627,     err = 6.072919944699606e-14

Inverse Iteration Method :
λ = 1.0982740771618063,     err = 8.20732370954147e-14

Rayleigh Quotient Iteration Method :
λ = 1.0982740771618063,     err = 1.1102230246251565e-16
```

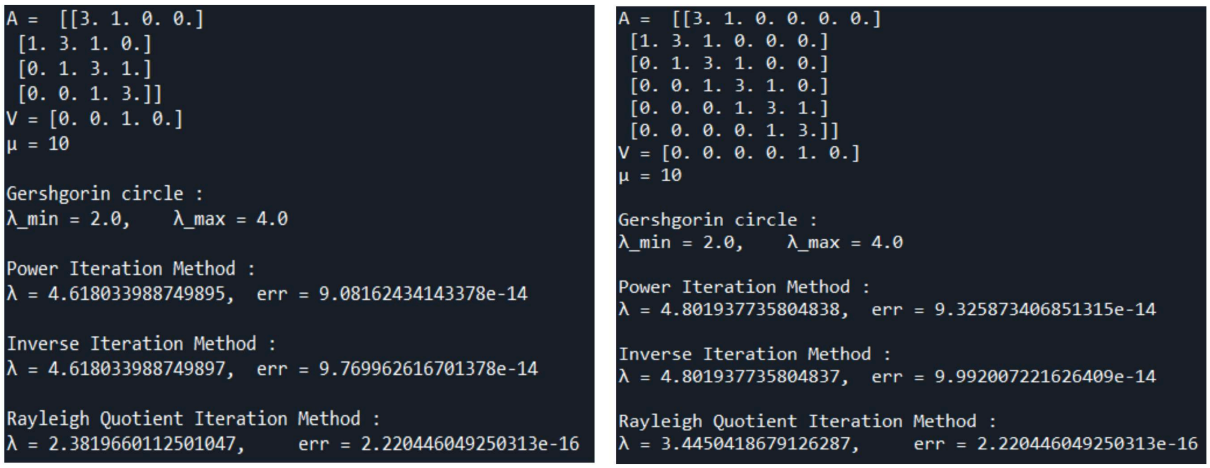


Figure 8. Results after Implementation of different values

This code tests the functions **gershgorin, power, inverse**, and **rayleigh** with random input matrices of different sizes and initial eigenvector guesses. It also plots the error rates of each method over iterations.

The code will print the results of each function and show a plot of the error rates of each method. The **err** output is a list of the error in each iteration, which should decrease with each iteration until the stopping criterion is met.

From these results it is pretty evident that the Rayleigh Quotient iteration is the most efficient at minimizing errors, followed by inverse iteration with Gershgorin's estimate. Power iteration is the simplest but slowest method, and inverse iteration with a random estimate is the least reliable method. The choice of method depends on the problem characteristics and the desired level of precision.

Also, the point to be noted is that the results depend on our choice of the input vector. The choice of input vectors can affect the speed and accuracy of the convergence, as well as the eigenvalues that are approximated. For example, if the input vector is orthogonal to the dominant eigenvector, the power iteration method will fail to converge. Therefore, it is important to choose input vectors that are close to the true eigenvectors, or at least not orthogonal to them.