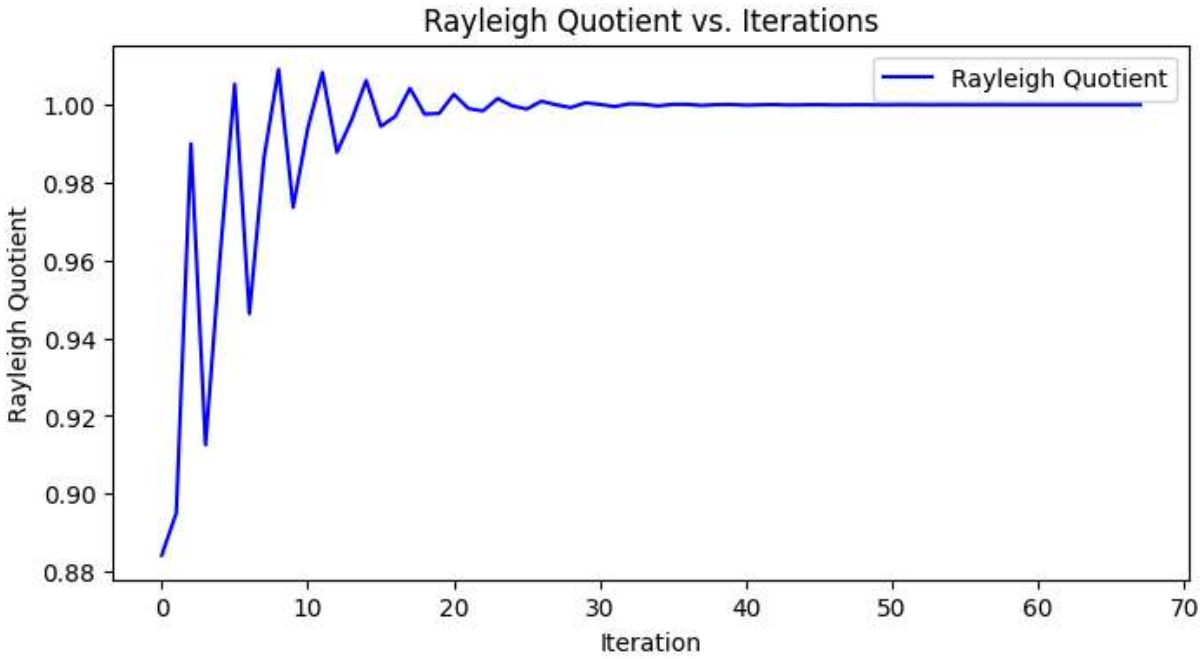
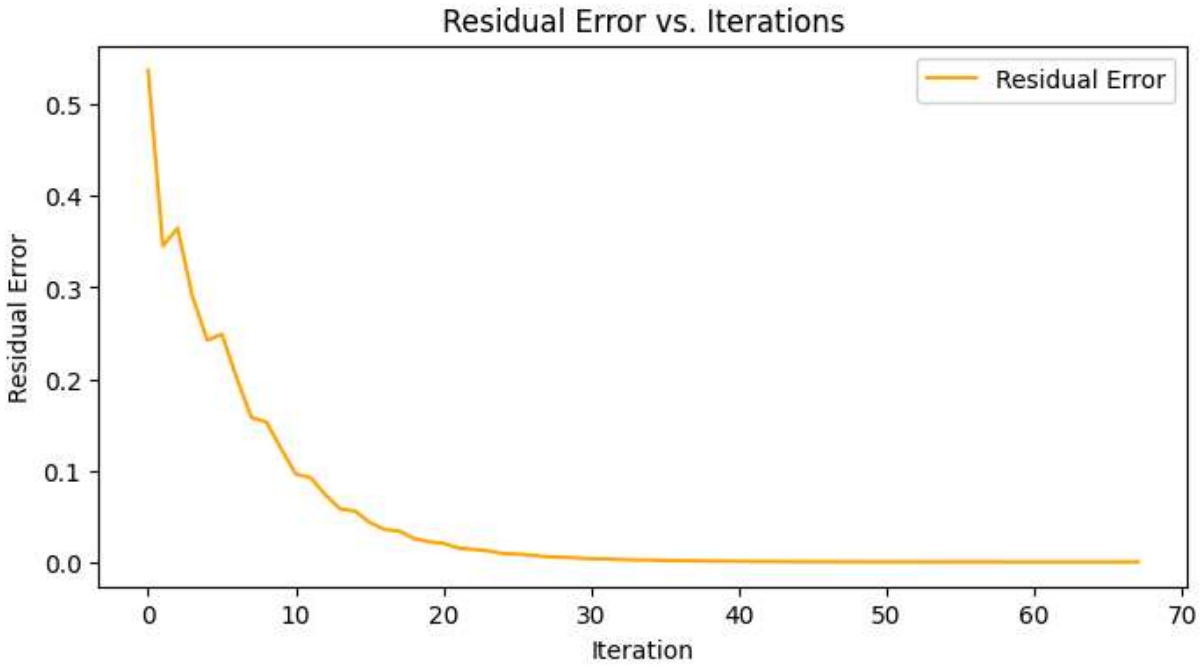
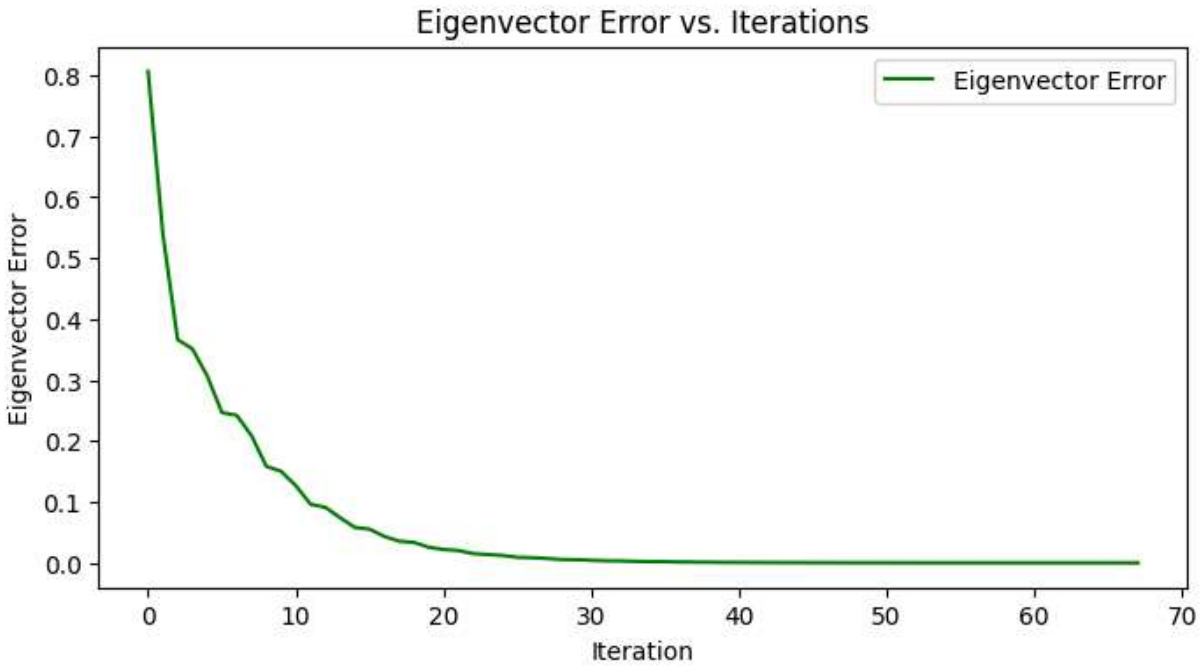
In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the Markov transition matrix
Markov_transition_Matrix = np.array([
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    [0.5, 0, 0, 0, 0, 0, 0, 0, 0.5, 0],
    [0, 0.5, 0, 0, 0, 0, 0.5, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0.5, 0, 0, 0.5, 0, 0],
    [0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0],
    [0, 0.5, 0, 0, 0.5, 0, 0, 0, 0, 1],
    [0.5, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0],
    [0, 0, 0, 0, 0, 0.5, 0, 0, 0.5, 0]
])

# Copy the matrix
A = Markov_transition_Matrix.copy()
v = np.random.rand(10).reshape(10, 1)

# Normalize v
v = v / np.linalg.norm(v, ord=2)

error_max = 1
lamda_max = 0
iteration = 0
rayleigh_quotient = []
residual_error = []
eigenvector_error = []

while error_max > 1e-6:
    v_old = v
    w = np.dot(A, v)
    v = w / np.linalg.norm(w, ord=2)
    lamda_old_max = lamda_max
    lamda_max = np.dot(v.T, np.dot(A, v))
    rayleigh_quotient.append(lamda_max[0, 0])  # Convert matrix to scalar
    residual_error.append(np.linalg.norm(np.dot(A, v) - lamda_max * v, ord=2))
    eigenvector_error.append(np.linalg.norm(v - v_old, ord=2))
    error_max = abs(lamda_old_max - lamda_max)

    iteration += 1

# Plot Rayleigh Quotient
plt.figure(figsize=(8, 4))
plt.plot(rayleigh_quotient, label="Rayleigh Quotient", color="blue")
plt.xlabel("Iteration")
plt.ylabel("Rayleigh Quotient")
plt.title("Rayleigh Quotient vs. Iterations")
plt.legend()
plt.show()

# Plot Residual Error
```

```
plt.figure(figsize=(8, 4))
plt.plot(residual_error, label="Residual Error", color="orange")
plt.xlabel("Iteration")
plt.ylabel("Residual Error")
plt.title("Residual Error vs. Iterations")
plt.legend()
plt.show()

# Plot Eigenvector Error
plt.figure(figsize=(8, 4))
plt.plot(eigenvector_error, label="Eigenvector Error", color="green")
plt.xlabel("Iteration")
plt.ylabel("Eigenvector Error")
plt.title("Eigenvector Error vs. Iterations")
plt.legend()
plt.show()

# Print the results
print("Eigenvalue (lambda):", lamda_max)
print("Eigenvector v:", v)
print("iteration: ", iteration)

print("Converged eigenvalue of v[k]:")
for i in range(len(v)):
    print("Page rank of Node {}: {}".format( i, v[i]))

print("Node with the highest page rank: {}".format(np.argmax(v)))
print("Node with the lowest page rank: {}".format(np.argmin(v)))
```




Residual Error vs. Iterations
Residual Error
Residual Error
Iteration
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
50
60
70


Eigenvector Error vs. Iterations
Eigenvector Error
Eigenvector Error
Iteration
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50
60
70

```
Eigenvalue (lambda): [[1.0000004]]
Eigenvector v: [[0.24703695]
 [0.35683654]
 [0.24703956]
 [0.        ]
 [0.        ]
 [0.27448607]
 [0.13724515]
 [0.54897633]
 [0.46662901]
 [0.37056094]]
iteration:  68
Converged eigenvalue of v[k]:
Page rank of Node 0: [0.24703695]
Page rank of Node 1: [0.35683654]
Page rank of Node 2: [0.24703956]
Page rank of Node 3: [0.]
Page rank of Node 4: [0.]
Page rank of Node 5: [0.27448607]
Page rank of Node 6: [0.13724515]
Page rank of Node 7: [0.54897633]
Page rank of Node 8: [0.46662901]
Page rank of Node 9: [0.37056094]
Node with the highest page rank: 7
Node with the lowest page rank: 3
```

Node Numbers with Least and Highest page ranks

- Min page rank for page 3 and 4 both having value 0
- Max page rank is for the page 7 having value 0.489763

NLA Assignment 5:

2 (a) Any eigenvalue problem can be expressed as a polynomial root finding problem, we know that for polynomial of degree $\geq 5$, there doesn't exist a direct method to find roots of that polynomial thus computing eigenvalues & eigenvectors for any random matrix $A \in \mathbb{R}^{m \times m}$ $(m \geq 5)$ cannot be solved in a finite number of steps.

False.

(b) without the preliminary step to reduce it to its tridiagonal form then phase-2 will involve a full matrix, requiring $O(m^2)$ work which will bring the overall work to $O(m^3)$ or higher as convergence might also need more than $O(m)$ iterations. False

(c) If the initial guess $v^{(0)}$ is $\perp$ to $q_1$ (eigenvector corresponding to largest eigenvalue), then the power iteration will fail. False

(d) since $F$ is real and orthogonal (Householder reflector)

$$FF^T = F^TF = I = F^{-1} = F^T$$

$\Rightarrow FAF^T$ is a similar transformation & hence

$$\Lambda(A) = \Lambda(FAF^T)$$ true

Q3. $A, B$ are real large symmetric positive definite sparse matrices

$$Au_i = \lambda_i B u_i$$

(a) Standard eigenvalue problem $Hv_i = \lambda_i v_i$

where $H \rightarrow$ symmetric matrix.

Express $u_i$ in terms of $v_i$.

$B$ is SPD $\Rightarrow B = B^T$, $B$ is invertible

Cholesky decomposition of $B$: $B = R^TR$ where $R^T$ is upper triangular

$$Au_i = \lambda_i B u_i$$
$$Au_i = \lambda_i R^T R u_i \qquad \text{let's assume } w_i = Ru_i$$
$$AR^{-1}w_i = \lambda_i R^T w_i$$
$$\Rightarrow \quad R^{-T} A R^{-1} w_i = \lambda_i w_i$$

Made with Goodnotes

$R^{-T}AR^{-1}$ is symmetric

$H = R^{-T}AR^{-1}$

$H^T = R^{-T}AR^{-1} = H$

$w_i = R^{-1}v_i$

$$Hw_i = \lambda_i w_i$$

(b) Find an iterative algorithm to find eigenvalue $\lambda_i$ closest to 2.0 ( 2.0 is not the eigenvalue) & eigenvector.

Algo: Inverse iteration modified.

1. compute $R^T, R$ by cholesky decomposition of B
2. compute $H = R^{-T}AR^{-1}$
3. $v^{(0)}$ = some random vector (i.e. $\|v^{(0)}\|_2 = 1$
4. for $k = 1, 2, \ldots$
5. solve $(H - 2I)u = v^{(k-1)}$ for $u$
6. $v^{(k)} = u / \|u\|_2$
7. $\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$
8. some stopping condition to stop the loop
9. $w_i = R^{-1} v_i$
10. Return $\lambda_i, w_i$

Above iteration have following properties:

suppose $\lambda_J$ is the closest eigenvalues to $u$ and $\lambda_K$ is second closest

$$|u - \lambda_J| < |u - \lambda_K| \leq |u - \lambda_j| \text{ for } j \neq J \text{ such that } q_J^T v^{(0)} \neq 0,$$

where $q_J$ is the corresponding eigenvector to $\lambda_J$ is the corresponding eigenvector to $\lambda_J$ for matrix $H$, then the iterate satisfy

Made with Goodnotes

$$\|v^{(k)} - (\pm q_J)\| = O\left(\left|\frac{\mu-\lambda_J}{\mu-\lambda_K}\right|^k\right), \quad |\lambda^{(k)} - \lambda_J| = O\left(\left|\frac{\mu-\lambda_J}{\mu-\lambda_K}\right|^{2k}\right)$$

iteration will fail if $q_J^T v^{(0)} = 0$ i.e $v^{(0)} \perp$ to $q_J^T$ satisfying $H q_J^T = \lambda_J q_J^T$

(c) Most computationally dominant term is step (5), inverting $(H-\mu I)$, which takes $O(m^3)$ steps.

if we solve the step (5) using QR or LU decomposition. this reduce the complexity from $O(m^3)$ to $O(m^2)$ flops.

→ in the first iteration, we can find and store $H-\mu I = QR$

then we can solve the problem

| | | |
|---|---|---|
| ① | $QRu = v^{(k)}$ | $O(m^3)$ (only 1st iteration) |
| ② | $Ru = Q^T v^{(k)}$ | $O(m^2)$ flops |
| ② | solve $u$ using back subs. | $O(m^2)$ flops. |

in general except step ① first time:

flops ~ $O(m^2)$

(d) As eigenvalue closest to $\mu = 20$ is the smallest eigenvalue, we can use inverse iteration without shifts.

change step (5) of above written algo to:

– solve $Hu = v^{(k-1)}$ for $u$

in doing so we will get the following advantages

– since $H$ is SPD we can calculate $H = R^T R$ and then use two sets of back substitution. this will be better for first iteration as cholesky decomposition is better than QR.

Made with Goodnotes

— only need to store only R matrix (compared to QR) for subsequent iterations. this will reduce space complexity.

④ $A \in \mathbb{R}^{m \times m}$ is symmetric invertible $\in \mathbb{R}^{m \times m}$, $K(A)$ is very large.

$|\lambda_1| << |\lambda_2| \leq |\lambda_3| \ldots \leq |\lambda_m|$

A is ill-conditioned as $K(A) = O(\varepsilon_m^{-1})$

$q_1, q_2, \ldots, q_m$ are corresponding eigenvectors.

ⓐ $Aw = v$ is solved using some backward stable algorithm.

$v \in \mathbb{R}^m$, yields $\tilde{w} = w + \delta w$

show that $\delta w = -(A+\delta A)^{-1}(\delta A)w$ such that $\|\delta A\| = O(\varepsilon_m)\|A\|$

$K(A) = \|A\|\|A\|^{-1} = O(\varepsilon_m^{-1})$

Input: $A$
output: $w$

$\Rightarrow$ $Aw = b$

$(A+\delta A)(w+\delta w) = b$

$Aw + A\delta w + \delta A w + \delta A \delta w = b$

$(A+\delta A)(\delta w) = -(\delta A)w$

$$\boxed{(\delta w) = -(A+\delta A)^{-1}(\delta A)w}$$

ⓑ $v$ is a vector with components in the all directions of all eigenvectors $\langle q_1, q_2, \ldots, q_m \rangle$ of A

show that $\frac{w}{\|w\|_2} = q_1$ where $w = A^{-1}v$ is the exact solution of equation given in ⓐ.

$v = \alpha_1 q_1 + \alpha_2 q_2 + \ldots + \alpha_m q_m$

Made with Goodnotes

$$A^{-1}v = \alpha_1 A^{-1} q_1 + \alpha_2 A^{-1} q_2 + \cdots + \alpha_m A^{-1} q_m$$

$$= \frac{\alpha_1 q_1}{\lambda_1} + \frac{\alpha_2 q_2}{\lambda_2} + \cdots + \frac{\alpha_m q_m}{\lambda_m}$$

$$\frac{1}{\lambda_1}\left(\alpha_1 q_1 + \alpha_2 q_2 \frac{\lambda_1}{\lambda_2} + \cdots + \alpha_2 q_m \frac{\lambda_1}{\lambda_m}\right) \quad \text{as } |\lambda_1| << |\lambda_i|$$

$$w = A^{-1}v \approx \frac{\alpha_1}{\lambda_1}(q_1)$$

$$\boxed{\frac{w}{\|w\|_2} \simeq (q_1)}$$

(c) proved in (a) $\quad w + \delta w = w - (A+\delta A)^{-1}\delta A\, w$

$$\delta w = -A^{-1}(\delta A)w + A^{-1}(\delta A)A^{-1}(\delta A)w + O(\varepsilon_m^2)$$

$$\delta w \approx -(A^{-1}\delta A)\frac{\alpha_1 q_1}{\lambda_1}$$

$$\delta A = \sum_i \sum_j \beta_{ij}\, q_i q_j^T$$

$$\delta w = -A^{-1}\left[\sum_i \sum_j \beta_{ij}\, q_i q_j^T\right]\frac{q_1 \alpha_1}{\lambda_1}$$

$$= -A^{-1}\left[\beta_{11} q_1 + \beta_{21} q_1 + \cdots + \beta_{m1} q_1\right]\frac{\alpha_1}{\lambda_1}$$

$$= -A^{-1}\left[\sum_i \beta_{i1}\right] q_1 \frac{\alpha_1}{\lambda_1}$$

$q_1$ is the eigenvector of $A^{-1}$ with $\lambda' = \frac{1}{\lambda_1}$

$$\boxed{\delta w = -\left[\sum_i \beta_{i2}\right]\frac{\alpha_1}{\lambda_1^2} v_1}$$

Both $\delta w$ & $w$ are in the same direction ($q_1$)

$$\boxed{\frac{w}{\|w\|} \simeq q_1}$$

Made with Goodnotes

Q5 $A \in \mathbb{R}^{m \times m}$ large sparse and diagonally dominant symmetric matrix.

we want to solve $A\tilde{x}_i = \lambda_i \tilde{x}_i$ where $i = 1, 2, \ldots, n$.

smallest eigenvectors & eigenvalues pairs of A ($n \ll m$)

(a) $V_{(0)}^{n} = \{\tilde{x}_1^{(0)}, \tilde{x}_2^{(0)}, \ldots, \tilde{x}_n^{(0)}\}$

$t_i = x_i - \tilde{x}_i^{(0)}$

show that $(A - \epsilon_i I)t_i = (\epsilon_i I - A)\tilde{x}_i^{(0)}$ can be solved to get the exact value of $\epsilon_i$

$$Ax_i = \epsilon_i x_i$$

$$(A - \epsilon_i I)x_i = 0$$

$$(A - \epsilon_i I)x_i + (\epsilon_i I - A)\tilde{x}_i^{(0)} = (\epsilon_i I - A)\tilde{x}_i^{(0)}$$

$$(A - \epsilon_i I)(x_i - \tilde{x}_i^{(0)}) = (\epsilon_i I - A)\tilde{x}_i^{(0)}$$

$$(A - \epsilon_i I)t_i = (\epsilon_i I - A)\tilde{x}_i^{(0)}$$

if exact $\epsilon_i$ is known, then

compute $b = (\epsilon_i I - A)\tilde{x}_i^{(0)}$ then solve

$(A - \epsilon_i I)\epsilon_i = b$ using any decomposition/ gauss elimination.

(b) we can use rayleigh quotient to approximate the eigenvalue

$$\tilde{\epsilon}_i^{(0)} = \frac{\tilde{x}_i^{(0)T} A \tilde{x}_i^{(0)}}{\tilde{x}_i^{(0)T} \tilde{x}_i^{(0)}}$$

New equation becomes:

$$(D - \tilde{\epsilon}_i^{(0)} I)t_i = (\tilde{\epsilon}_i^{(0)} I - A)\tilde{x}_i^{(0)}$$

$$\boxed{t_i = D}$$

Made with Goodnotes

(c) By including correction vectors, we are correcting initial estimates and thus accelerating convergence. Higher dimension space also implies a richer space to search for eigenvectors and eigenvalues.

(d) $r_i = A\tilde{x}_i^{(1)} - \tilde{\lambda}_i^{(1)}\tilde{x}_i^{(1)}$ $\tilde{\lambda}_i^{(1)}$ - best approximation to eigenvalue corresponding to eigenvector approximation $\tilde{x}_i^{(1)}$

$r_i$ is orthogonal to space

let $v \in V_0^{2m}$

$v^T r = 0$

$v^T(A\tilde{x}_i^{(1)} - \tilde{\lambda}_i^{(1)}\tilde{x}_i^{(1)}) = 0$

$\tilde{x}_i^{(1)} = Vy_i$

$\Rightarrow r_i = AVy_i - \tilde{\lambda}_i^{(1)}Vy_i$

$v^TAVy_i = \tilde{\lambda}_i^{(1)}v^TVy_i$

$\tilde{A}_p y_i = \tilde{\lambda}_i^{(1)} y_i$ this solves the EVP in lower dimensional space to get $\tilde{x}^{(1)}$ back

$\tilde{x}_i^{(1)} = Vy_i$

Made with Goodnotes