Ma423: Matrix Computation Lab 8

```
format long e;
```

Question 1

```
% Question 1
fprintf(' ---- Question 1 ---- \n');
```

```
 ---- Question 1 ----
```

```
A = [2 -12; 1 -5];
x = [1; 1];
k = 8;
[iter, lambda] = PowerMethod(A, x, k);
disp('Iterates matrix:');
```

```
Iterates matrix:
```

```
disp(iter);
```

```
    1.000000000000000e+00     1.000000000000000e+00     1.000000000000000e+00     1.000000000000000e+00     1.00000
    4.000000000000000e-01     3.571428571428570e-01     3.437499999999999e-01     3.382352941176471e-01     3.35714
```

```
disp('Dominant Eigenvalue:');
```

```
Dominant Eigenvalue:
```

```
disp(lambda);
```

```
    -2.006993006993008e+00
```

Observation: It can be seen that the iterates seem to converge to the dominant eigenvector [3 1]' and lambda is close to the dominant eigenvalue -2.

Question 2

```
fprintf(' ---- Question 2 ---- ');
```

```
 ---- Question 2 ----
```

```
A1 = [1 1 1; -1 9 2; 0 -1 2];
A2 = [1 1 1; -1 9 2; -4 -1 2];
A3 = [1 1 1; -1 3 2; -4 -1 2];
x = [1 1 1]';
k = 20;

matrices = {A1, A2, A3};
matrix_names = {'A1', 'A2', 'A3'};

for idx = 1:length(matrices)
    A = matrices{idx};
```

```
    fprintf('  - Part (%c):\n', 'a' + idx - 1);
    [iter, lambda] = PowerMethod(A, x, k);
    [v1, lam] = eig_simplified(A);
    fprintf('lambda1 = %e, lambda2 = %e, lambda3 = %e\n', lam(1), lam(2), lam(3));
    fprintf('|lambda2| / |lambda1| = %e\n', abs(lam(2) / lam(1)));

    diff_norms = zeros(k, 1);
    for i = 1:k
        diff = iter(:,i) - v1;
        diff_norms(i) = norm(diff);
    end

    ratios = zeros(k-1, 1);
    for i = 1:k-1
        ratios(i) = diff_norms(i+1) / diff_norms(i);
    end

    theoretical_rate = abs(lam(2) / lam(1));

    fprintf('Iteration\tRatio of Norms\tTheoretical Rate\n');
    for i = k-6:k-1
        fprintf('%d\t\t%e\t%e\n', i, ratios(i), theoretical_rate);
    end
    fprintf('\n');
end
```

```
  - Part (a):
lambda1 = 8.584428e+00, lambda2 = 2.194882e+00, lambda3 = 1.220690e+00
|lambda2| / |lambda1| = 2.556818e-01
Iteration     Ratio of Norms     Theoretical Rate
14          2.556807e-01     2.556818e-01
15          2.556812e-01     2.556818e-01
16          2.556814e-01     2.556818e-01
17          2.556813e-01     2.556818e-01
18          2.556809e-01     2.556818e-01
19          2.556802e-01     2.556818e-01
  - Part (b):
lambda1 = 8.455873e+00, lambda2 = 1.772064e+00, lambda3 = 1.772064e+00
|lambda2| / |lambda1| = 2.904340e-01
Iteration     Ratio of Norms     Theoretical Rate
14          1.000000e+00     2.904340e-01
15          1.000000e+00     2.904340e-01
16          1.000000e+00     2.904340e-01
17          1.000000e+00     2.904340e-01
18          1.000000e+00     2.904340e-01
19          1.000000e+00     2.904340e-01
  - Part (c):
lambda1 = 2.380066e+00, lambda2 = 2.380066e+00, lambda3 = 1.239868e+00
|lambda2| / |lambda1| = 1.000000e+00
Iteration     Ratio of Norms     Theoretical Rate
14          1.602716e+00     1.000000e+00
15          1.344530e+00     1.000000e+00
16          5.091967e-01     1.000000e+00
17          9.453771e-01     1.000000e+00
18          1.599008e+00     1.000000e+00
19          1.376617e+00     1.000000e+00
```

Observations:

- **Part (a):** All 3 eigen values are different, thus the ratio of norms are also very close to |lambda2| / |lambda1| which means the iterations converge linearly.
- **Part (b):** Here |lambda2| = |lambda3|, hence there is no definite convergence to the ratios, and the iterations do not converge linearly.
- **Part (c):** Here |lambda1| = |lambda2|, hence there is no definite convergence to the ratios, and the iterations do not converge linearly. $\rightarrow$ check lambda ≈ lambda1

Q3?

Question 4

```
fprintf(' ---- Question 4 ---- ');
```

```
 ---- Question 4 ----
```

```
A = [4, 1, 0; 1, 3, 1; 0, 1, 2];
x = rand(3, 1);
s = 2.5;
k = 10;
[iter, lambda] = Shiftinv(A, x, s, k);
fprintf('Approximate eigenvalue closest to s: %.6f\n', lambda);
```

```
Approximate eigenvalue closest to s: 3.000000
```

```
disp('Approximate eigenvector:');
```

```
Approximate eigenvector:
```

```
disp(iter(:, end));
```

```
  -9.993845912336913e-01
   9.991653939304213e-01
   1.000000000000000e+00
```

Question 5

```
fprintf(' ---- Question 5 ---- ');
```

```
 ---- Question 5 ----
```

```
A = [4, -1, 0,  0;
     -2, 4, -1, 0;
      0, -2, 4, -1;
      0,  0, -2, 4];

[L, U, p] = gepphess(A);

disp('Permutation vector p:');
```

```
Permutation vector p:
```

```
disp(p);
```

```
     1     2     3     4
```

```
disp('Lower triangular matrix L:');
```

```
Lower triangular matrix L:
```

```
disp(L);
```

```
     1.000000000000000e+00                         0                         0                         0
    -5.000000000000000e-01     1.000000000000000e+00                         0                         0
                         0    -5.714285714285714e-01     1.000000000000000e+00                         0
                         0                         0    -5.833333333333333e-01     1.000000000000000e+00
```

```
disp('Upper triangular matrix U:');
```

```
Upper triangular matrix U:
```

```
disp(U);
```

```
     4.000000000000000e+00    -1.000000000000000e+00                         0                         0
                         0     3.500000000000000e+00    -1.000000000000000e+00                         0
                         0                         0     3.428571428571429e+00    -1.000000000000000e+00
                         0                         0                         0     3.416666666666667e+00
```

```


A_permuted = A(p, :);
A_reconstructed = L * U;

disp('A(p,:) - L * U (should be zero matrix):');
```

```
A(p,:) - L * U (should be zero matrix):
```

```
disp(A_permuted - A_reconstructed);
```

```
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Question 7

```
fprintf(' ---- Question 7 ---- ');
```

```
 ---- Question 7 ----
```

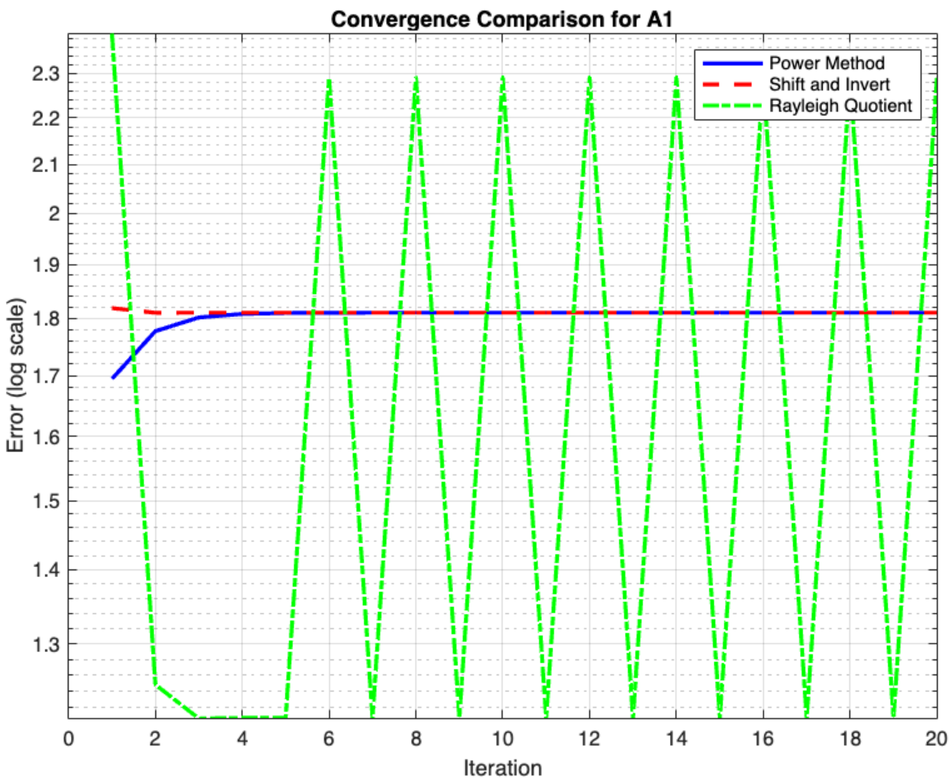
```
A1 = [1 1 1; -1 9 2; 0 -1 2];
A2 = [1 1 1; -1 9 2; -4 -1 2];
A3 = [1 1 1; -1 3 2; -4 -1 2];

x = [1; 1; 1];
k = 20;

matrices = {A1, A2, A3};
matrix_names = {'A1', 'A2', 'A3'};
```

```matlab
for m = 1:length(matrices)
    A = matrices{m};
    fprintf('--- Matrix %s ---\n', matrix_names{m});

    [v, lam] = eig_simplified(A);
    lam1 = lam(1);
    lam2 = lam(2);

    rate_power = abs(lam2 / lam1);

    % Power Method
    [iter_power, lambda_power] = PowerMethod(A, x, k);
    errors_power = zeros(k,1);
    for i = 1:k
        errors_power(i) = norm(iter_power(:,i) - v1);
    end

    % Shift and Invert Method
    s = lam1 - 0.1;
    [iter_shiftinv, lambda_shiftinv] = Shiftinv(A, x, s, k);
    errors_shiftinv = zeros(k,1);
    for i = 1:k
        errors_shiftinv(i) = norm(iter_shiftinv(:,i) - v1);
    end

    % Rayleigh Quotient Iteration
    [iter_rayleigh, lambda_rayleigh] = Rayleigh(A, x, k);
    errors_rayleigh = zeros(k,1);
    for i = 1:k
        errors_rayleigh(i) = norm(iter_rayleigh(:,i) - v1);
    end

    % Plotting the Convergence
    figure;
    semilogy(1:k, errors_power, 'b-', 'LineWidth', 2);
    hold on;
    semilogy(1:k, errors_shiftinv, 'r--', 'LineWidth', 2);
    semilogy(1:k, errors_rayleigh, 'g-.', 'LineWidth', 2);
    xlabel('Iteration');
    ylabel('Error (log scale)');
    title(['Convergence Comparison for ', matrix_names{m}]);
    legend('Power Method', 'Shift and Invert', 'Rayleigh Quotient');
    grid on;

    fprintf('Power Method Eigenvalue: %.6f\n', lambda_power);
    fprintf('Shift and Invert Eigenvalue: %.6f\n', lambda_shiftinv);
    fprintf('Rayleigh Quotient Eigenvalue: %.6f\n', lambda_rayleigh);
    fprintf('True Dominant Eigenvalue: %.6f\n', lam1);
```

where do you define v1?

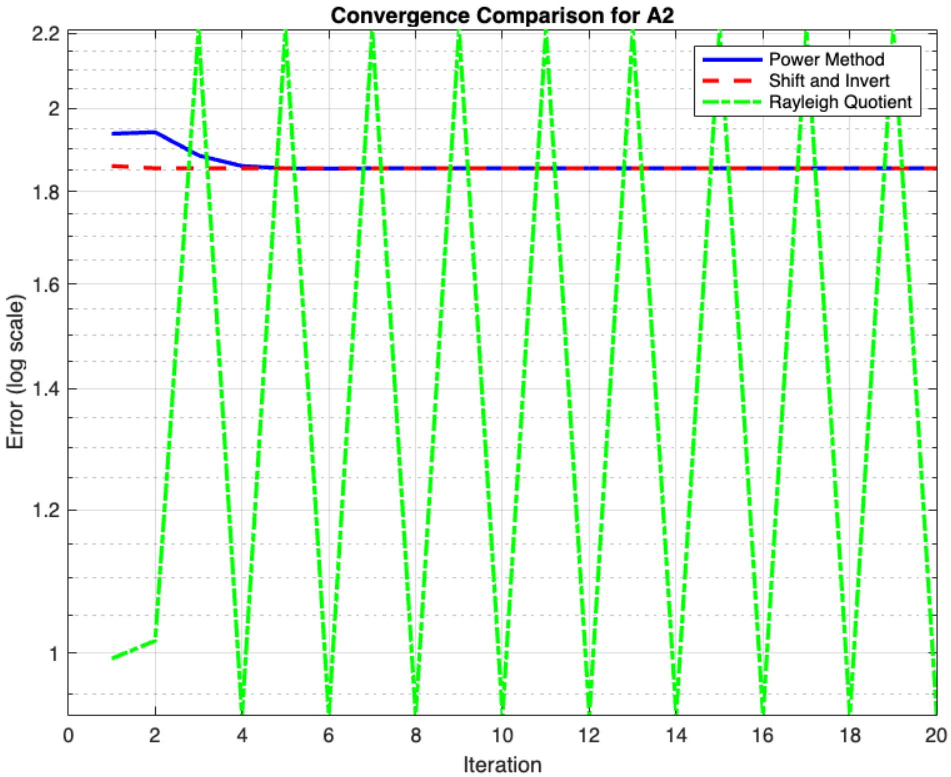
```
    fprintf('Power Method Convergence Rate: %.6f\n', rate_power);
    fprintf('\n');
end
```

```
--- Matrix A1 ---
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.395170e-17.
```

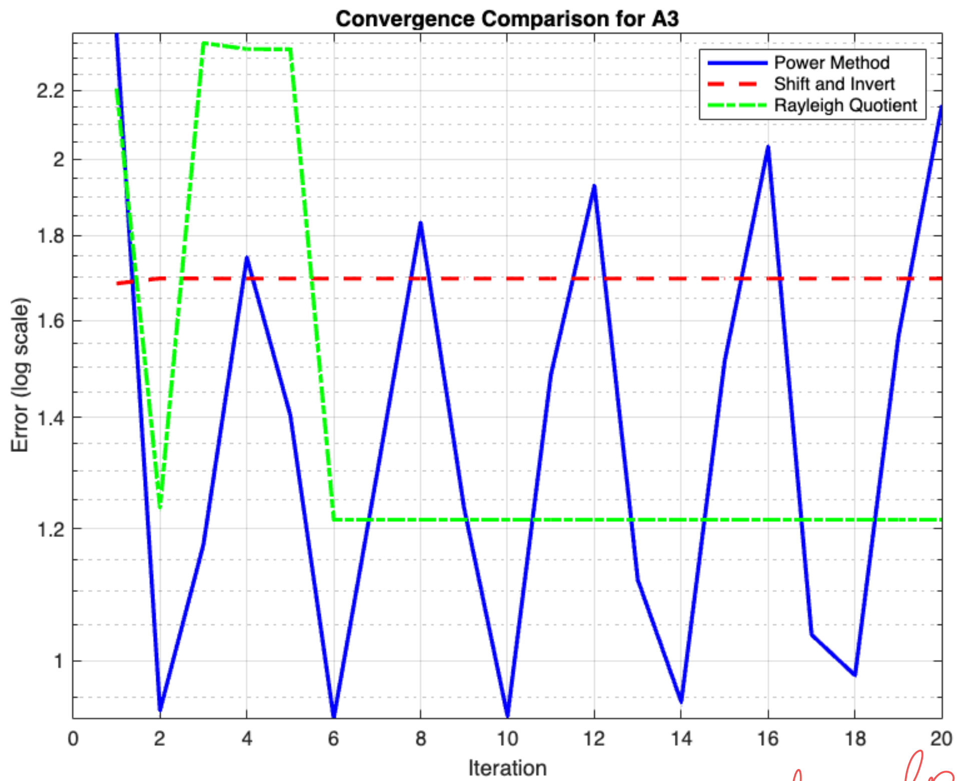



```
Power Method Eigenvalue: 8.584428
Shift and Invert Eigenvalue: 8.584428
Rayleigh Quotient Eigenvalue: 2.194882
True Dominant Eigenvalue: 8.584428
Power Method Convergence Rate: 0.255682
--- Matrix A2 ---
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.247086e-16.
```




```
Power Method Eigenvalue: 8.455873
Shift and Invert Eigenvalue: 8.455873
Rayleigh Quotient Eigenvalue: 8.455873
True Dominant Eigenvalue: 8.455873
Power Method Convergence Rate: 0.290434
--- Matrix A3 ---
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  2.692686e-17.
```



what do you solve?

```
Power Method Eigenvalue: 3.593669
Shift and Invert Eigenvalue: 2.380066
Rayleigh Quotient Eigenvalue: 1.239868
True Dominant Eigenvalue: 2.380066
Power Method Convergence Rate: 1.000000
```

Functions used:

Question 1

need to scale x before computing Ax.

```
function [iter, lambda] = PowerMethod(A, x, k)
    % Initialize variables
    n = length(x);
    iter = zeros(n, k);

    % Power iteration loop
    for i = 1:k
        x = A * x;                      % Matrix-vector multiplication
        [lambda, x] = normalize(x); % Normalize x and extract scaling factor as
lambda
        iter(:, i) = x;                 % Store iteration result
    end
end

function [lambda, x] = normalize(x)
    % Normalize x and return the scaling factor (lambda)
    [~, idx] = max(abs(x));         % Find index with max magnitude
```

```
    lambda = x(idx);            % Scaling factor as eigenvalue approximation
    x = x / lambda;             % Normalize vector x
end
```

Question 2

```
function [v, lam] = eig_simplified(A)
    % Compute eigenvalues and eigenvectors
    [V, eigenvalues] = eig(A, 'vector');

    % Sort eigenvalues by absolute value in descending order
    [~, idx] = sort(abs(eigenvalues), 'descend');
    lam = eigenvalues(idx);   % Sorted eigenvalues
    v = V(:, idx(1));         % Dominant eigenvector

    % Normalize the dominant eigenvector
    v = v / max(abs(v));
end
```

the scaling factor is wrong.

Question 3

```
function [v1, v2, v3, lam1, lam2, lam3] = eig_simplified_full(A)
    % Compute eigenvalues and eigenvectors
    [V, eigenvalues] = eig(A, 'vector');

    % Sort eigenvalues by absolute magnitude in descending order
    [~, idx] = sort(abs(eigenvalues), 'descend');

    % Extract top three eigenvalues and corresponding eigenvectors
    lam1 = eigenvalues(idx(1));
    lam2 = eigenvalues(idx(2));
    lam3 = eigenvalues(idx(3));
    v1 = normalize_vector(V(:, idx(1)));
    v2 = normalize_vector(V(:, idx(2)));
    v3 = normalize_vector(V(:, idx(3)));
end

function v = normalize_vector(v)
    % Normalize vector by its maximum absolute component
    v = v / max(abs(v));
end
```

Question 4

```
function [iter, lambda] = Shiftinv(A, x, s, k)
    n = length(x);
    iter = zeros(n, k);
```

```
    I = eye(n);
    [L, U] = lu(A - s * I);

    % Normalize x with a different technique
    x = x / norm(x, Inf);

    for j = 1:k
        % Solve Ly = x, then Uz = y in one step
        y = L \ x;
        x = U \ y;

        % Renormalize to prevent overflow or underflow
        x = x / norm(x, Inf);

        iter(:, j) = x;
    end

    % Compute the Rayleigh quotient for lambda
    lambda = (x' * A * x) / (x' * x);
end
```

[L, U, P] = lu(A - sI)

The scaling factor is wrong

Question 5

```
function [L, U, p] = gepphess(A)
    n = size(A, 1);
    p = 1:n;

    for i = 1:n-1
        [~, max_idx] = max(abs(A(i:i+1, i)));
        if max_idx == 2
            p([i, i + 1]) = p([i + 1, i]);
            A([i, i + 1], :) = A([i + 1, i], :);
        end

        if A(i, i) == 0
            continue
        end

        A(i + 1, i) = A(i + 1, i) / A(i, i);
        A(i + 1, i + 1 : end) = A(i + 1, i + 1 : end) - A(i + 1, i) .* A(i, i + 1 :
end);
    end
    L = eye(n) + tril(A, -1);
    U = triu(A);
end
```

Question 6

```
function [iter, lambda] = Rayleigh(A, x, k)
```

```
    n = length(x);
    iter = zeros(n, k);

    % Check if A is nearly upper Hessenberg; if not, convert
    if ~isequal(A, triu(A, -1))
        H = hess(A);
    else
        H = A;
    end

    % Initial normalization
    x = x / norm(x, Inf);

    for j = 1:k
        % Rayleigh quotient to approximate the eigenvalue
        lambda = (x' * H * x) / (x' * x);
        ShiftedMatrix = H - lambda * eye(n);

        % Factorize shifted matrix with partial pivoting
        [L, U, p] = gepphess(ShiftedMatrix); % Updated to match the function name
used previously
        x_perm = x(p);

        % Solve Ly = x_perm, Uz = y
        y = L \ x_perm;
        x = U \ y;

        % Renormalize x to maintain numerical stability
        x = x / norm(x, Inf);

        % Store the current iteration vector
        iter(:, j) = x;
    end
end
```