MAT128B Coding Report 4

Jun 5th, 2023

1 Problem Description

This report aims to analyze three methods for determining the eigensystem of matrix A: the power method, inverse power method, and deflation method. Matrix A is defined as follows: $A = 10I + B + B^T + B^TB$. Here, B is a 50×50 matrix with entries randomly generated from a normal distribution with a mean of 0 and a standard deviation of 1. The matrix I represents a 50×50 identity matrix. To evaluate these methods, we conducted experiments on 25 different instances of matrix A. We measured the average computation time of each method for computing the eigenvalues and also calculated the average errors of the eigenvalues and eigenvectors obtained from these methods. To determine the two largest and two smallest eigenvalues, we used the "eig" command in MATLAB. We then applied the power method to find the largest eigenvalue, the inverse power method to find the smallest eigenvalue, and used the deflation method to find the second largest and second smallest eigenvalues. The results are presented in two separate tables: Table 1 displays the results obtained using the power method and inverse power method, while Table 2 showcases the results obtained using the deflation method.

1.1 Method Descriptions

Built-in Method: We applied the built-in method "**eig(A)**" in MATLAB to compute the two smallest and two largest eigenvalues of matrix A. The eigenvalues are sorted in ascending order, where the first two values correspond to the smallest eigenvalues, and the last two values represent the largest eigenvalues. These computed eigenvalues serve as the "ground truth" (λ^*).

Power Method: The power method approximates the largest eigenvalue of matrix A. Our implementation, the function "$Power\_Method\_fun\_modified$," takes inputs including the matrix dimension (n), matrix A, an initial approximation vector (x), a tolerance value (TOL = 1e-8), and the maximum number of iterations (N = 4000) allowed for convergence. The function computes an estimate of the largest eigenvalue (μ) and its corresponding eigenvector (x). Starting with an initial approximation vector of $x_0 = 1$, the iteration stops when $||x^n - x^{n+1}|| < 1e-8$. The eigenvector is normalized to have a maximum absolute value of 1 ($||x||_\infty = 1$). If the maximum number of iterations is exceeded before convergence, a message indicating "Maximum number of iterations exceeded in the power method" is returned. We repeated this experiment 25 times, for different matrices A and determine the average time it takes to find the eigenvalue and the average error in the vector. The eigenvalue error is defined as $|\lambda - \lambda^*|$, and the eigenvector error is calculated as $\frac{||Ax-\lambda x||_\infty}{|\lambda^*|}$, where λ^* is the eigenvalue obtained from the built-in method.

Inverse Power Method: The inverse power method calculates the smallest eigenvalue of matrix A. Implemented as the function "$inv\_power\_method\_modified$," it requires inputs such as the matrix dimension (n), matrix A, an initial approximation vector (x), a tolerance value (tol = 1e-8), and the maximum number of iterations (N = 4000) allowed for convergence. By solving the linear system $(A-qI)y = x$ using LU factorization, with the functions like "LU_Fac_Fun" for LU factorization, "$Backward\_Sub\_Fun$" for backward substitution, and "$Forward\_Sub\_Fun$" for forward substitution, the method iteratively approximates the smallest eigenvalue (μ) and its corresponding eigenvector (x). Beginning with an initial approximation vector of $x_0 = 1$, the iteration stops when $||x^n - x^{n+1}|| < 1e - 8$. The computed eigenvector is normalized to have a maximum absolute value of 1 ($||x||_\infty = 1$). If the maximum number of iterations is exceeded before convergence, a message indicating that the maximum number of iterations was exceeded in the inverse power method is returned. The method starts with an initialization value of q = 0. We repeated this experiment 25 times, for different matrices A and determine the average time it takes to find the eigenvalue and the average error in the vector. The eigenvalue error is defined as $|\lambda - \lambda^*|$, and the eigenvector error is calculated as $\frac{||Ax-\lambda x||_\infty}{|\lambda^*|}$, where λ^* is the eigenvalue obtained from the built-in method.

Deflation Method: The deflation method (based on Wielandt Deflation Theorem) is employed to determine the second largest and second smallest eigenvalues of matrix A through an iterative procedure. Using functions like the power method or inverse power method, the largest eigenvalue and its corresponding eigenvector of matrix B are computed and stored. We then repeated this process to calculate the second largest eigenvalue and its eigenvector of the deflated matrix B. The obtained values are stored, and the deflation step is applied again. By iterating this process, the second smallest eigenvalue and its eigenvector can be obtained.

2 Results

We used the build-in method "**eig(A)**" in Matlab to obtain the first two largest and smallest eigenvalues, denoted as the "ground truth" (λ^*). We then applied the power method to find the largest eigenvalue, the inverse power method to find the smallest eigenvalue, and used the deflation method to find the second largest and second smallest eigenvalues. The experiment was repeated 25 times using different matrices A, and for each iteration, we recorded the computation time of each method for approximating the eigenvalues. The eigenvalue error was calculated as $|\lambda - \lambda^*|$, and the eigenvector error was evaluated as $\frac{||Ax-\lambda x||_\infty}{|\lambda^*|}$, where λ^* represents the eigenvalue obtained from the built-in method. The results are presented below in two distinct tables.

In Table 1, we applied the power method to find the largest eigenvalue and the inverse power method to find the smallest eigenvalue. The obtained results demonstrate small average errors in both eigenvalues and eigenvectors, indicating a high level of accuracy in the approximated values. The computation time for the largest eigenvalue is shorter than that of the smallest eigenvalue. This difference in computation time arises from the additional steps involved in the inverse power method, particularly the LU decomposition performed at each iteration. These additional computations result in a relatively slow execution of the inverse power method compared to the power method. Therefore, the calculation cost of inverse power method is higher and the calculation time is longer.

Table 2 provides an overview of the results obtained from the deflation method, specifically showcasing the computations for the second largest and second smallest eigenvalues. Firstly, the computation time for the second largest eigenvalue is shorter compared to that of the second smallest eigenvalue. This difference in computation time arises from the additional steps involved in the inverse power method, particularly the LU decomposition performed at each iteration. Furthermore, all the average errors in eigenvalues and eigenvectors are very small, indicating accurate approximations overall.

Table 1: Using power method and inverse power method			
	λtime	λerror	Vector Error
Smallest Eigenvalue	0.056231	1.3281e-08	9.9582e-09
Largest Eigenvalue	0.00043816	4.6421e-07	8.8926e-09

Table 1: A table of the largest and smallest eigenvalues computed using power method and inverse power method. The table includes the dimension of matrix A (n), a n x n matrix (A), the initial vector (x), the tolerance value (tol), and the maximum number of iterations (N) used for convergence. Note: we set tol=1e-8 and N=4000.

Table 2: Using deflation method			
	λtime	λerror	Vector Error
Second Smallest Eigenvalue	0.075216	7.1878e-08	1.9623e-09
Second Largest Eigenvalue	0.00034957	2.3896e-07	1.8052e-07

Table 2: A table of the second largest eigenvalue and second smallest eigenvalue computed using deflation method. The table includes the dimension of matrix A (n), a n x n matrix (A) for which the largest eigenvalue is to be computed, The eigenvalue to be deflated (removed) from the matrix A (lambda), the tolerance value (TOL), The corresponding eigenvector to the eigenvalue lambda (v) and the maximum number of iterations (N) used for convergence. Note: we set tol=1e-8 and N=4000.

3 Collaboration

Yirong Xu and Sabrina Zhu

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.

5 Appendix

```
clc; clear; close all;

% times
power_method_times = [];
inverse_power_method_times = [];
deflation_largest_times = [];
deflation_smallest_times = [];

% eigenvalues error
power_method_val_errors = [];
inverse_power_method_val_errors = [];
deflation_largest_val_errors = [];
deflation_smallest_val_errors = [];

% eigenvectors error
power_method_vec_errors = [];
inverse_power_method_vec_errors = [];
deflation_largest_vec_errors = [];
deflation_smallest_vec_errors = [];

for i = 1:25
    tol = 1e-8;
    N = 4000;
    n = 50;
    B = randn(n, n);
    x_0 = ones(n, 1);
    I = eye(n);
    A = 10 * I + B' + B' * B;

    eigenvalues = eig(A);
    sorted_eigenvalues = sort(abs(eigenvalues));

    % Power method
    tic;
    [largest_eigenvalue, largest_eigenvector] =
       Power_Method_fun_modified(n, A, x_0, tol, N);
    power_method_times(i) = toc;
    power_method_val_errors(i) = abs(largest_eigenvalue -
       sorted_eigenvalues(end));
    power_method_vec_errors(i) = norm((A *
       largest_eigenvector - largest_eigenvalue *
       largest_eigenvector), Inf) / abs(sorted_eigenvalues(
       end));

    % Inverse power method
```

```
    tic;
    [smallest_eigenvalue, smallest_eigenvector] =
       inv_power_method_modified(n, A, x_0, tol, N);
    inverse_power_method_times(i) = toc;
    inverse_power_method_val_errors(i) = abs(
       smallest_eigenvalue - sorted_eigenvalues(1));
    inverse_power_method_vec_errors(i) = norm((A *
       smallest_eigenvector - smallest_eigenvalue *
       smallest_eigenvector), Inf) / abs(sorted_eigenvalues
       (1));

    % Deflation method for largest eigenvalue
    tic;
    [second_largest_eigenvalue, second_largest_eigenvector] =
        deflation_largest(n, A, largest_eigenvalue,
       largest_eigenvector, tol, N);
    deflation_largest_times(i) = toc;
    if second_largest_eigenvector(1) < 0
        second_largest_eigenvector = -
           second_largest_eigenvector;
    end
    deflation_largest_val_errors(i) = abs(
       second_largest_eigenvalue - sorted_eigenvalues(end -
       1));
    deflation_largest_vec_errors(i) = norm((A *
       second_largest_eigenvector - second_largest_eigenvalue
        * second_largest_eigenvector), Inf) / abs(
       sorted_eigenvalues(end - 1));

    % Deflation method for smallest eigenvalue
    tic;
    [second_smallest_eigenvalue, second_smallest_eigenvector]
        = deflation_second_smallest(n, A, smallest_eigenvalue
       , smallest_eigenvector, tol, N);
    deflation_smallest_times(i) = toc;
    if second_smallest_eigenvector(1) < 0
        second_smallest_eigenvector= -
           second_smallest_eigenvector;
    end
    deflation_smallest_val_errors(i) = abs(
       second_smallest_eigenvalue - sorted_eigenvalues(2));
    deflation_smallest_vec_errors(i) = norm((A *
       second_smallest_eigenvector -
       second_smallest_eigenvalue *
       second_smallest_eigenvector), Inf) / abs(
       sorted_eigenvalues(2));
end
```

```
% Initialize variables
table1_data = cell(2, 4);
table2_data = cell(2, 4);

% Compute average values for Table 1
for i = 1:2
    eigenvalue_Name = '';
    times = [];
    val_errors = [];
    vec_errors = [];

    if i == 1
        eigenvalue_Name = 'Largest Eigenvalue';
        times = power_method_times;
        val_errors = power_method_val_errors;
        vec_errors = power_method_vec_errors;
    else
        eigenvalue_Name = 'Smallest Eigenvalue';
        times = inverse_power_method_times;
        val_errors = inverse_power_method_val_errors;
        vec_errors = inverse_power_method_vec_errors;
    end

    average_time = mean(times);
    average_val_error = mean(val_errors);
    average_vec_error = mean(vec_errors);

    table1_data{i, 1} = eigenvalue_Name;
    table1_data{i, 2} = average_time;
    table1_data{i, 3} = average_val_error;
    table1_data{i, 4} = average_vec_error;
end

% Compute average values for Table 2
for i = 1:2
    eigenvalue_Name = '';
    times = [];
    val_errors = [];
    vec_errors = [];

    if i == 1
        eigenvalue_Name = '2nd Largest Eigenvalue';
        times = deflation_largest_times;
        val_errors = deflation_largest_val_errors;
        vec_errors = deflation_largest_vec_errors;
    else
        eigenvalue_Name = '2nd Smallest Eigenvalue';
```

```
        times = deflation_smallest_times;
        val_errors = deflation_smallest_val_errors;
        vec_errors = deflation_smallest_vec_errors;
    end

    average_time = mean(times);
    average_val_error = mean(val_errors);
    average_vec_error = mean(vec_errors);

    table2_data{i, 1} = eigenvalue_Name;
    table2_data{i, 2} = average_time;
    table2_data{i, 3} = average_val_error;
    table2_data{i, 4} = average_vec_error;
end

% Create Table 1
power_inverse_table = table(table1_data(:, 1), table1_data(:,
    2), table1_data(:, 3), table1_data(:, 4), ...
    'VariableNames', {'Eigenvalue Types', 'Time', 'Value
       Error', 'Vector Error'});

% Display Table 1
disp("Table 1:");
disp(power_inverse_table);

% Create Table 2
deflation_table = table(table2_data(:, 1), table2_data(:, 2),
    table2_data(:, 3), table2_data(:, 4), ...
    'VariableNames', {'Eigenvalue Types', 'Time', 'Value
       Error', 'Vector Error'});

% Display Table 2
disp("Table 2:");
disp(deflation_table);



% ----- power method ----------
function [u, x] = Power_Method_fun_modified(n, A, x, TOL, N)
    % Power Method for computing dominant eigenvalue and
       eigenvector
    %   n: Dimension of matrix A
    %   A: n x n matrix
    %   x: Initial vector
    %   TOL: tolerance value for convergence
    %   N: Maximum number of iterations
    %   u: Dominant eigenvalue, NaN if maximum iterations
       exceeded
```

```
    %   x: Corresponding eigenvector, NaN if maximum
       iterations exceeded
    k = 1;
    for p = 1:n
        if abs(x(p)) == norm(x, Inf)
            x_p = x(p);
            break;
        end
    end
    x = x / x_p;

    while k <= N
        y = A * x;

        for p = 1:n
            if abs(y(p)) == norm(y, Inf)
                y_p = y(p);
                break;
            end
        end
        u = y_p;
        if u == 0
            fprintf('Eigenvector:\n');
            disp(x);
            fprintf('A has the eigenvalue 0, select a new
               vector x and restart.\n');
            return;
        end

        ERR = norm((x - y / u), Inf);
        x = y / u;

        if ERR < TOL
            disp(u);
            return;
        end
        k = k + 1;
    end
    fprintf('The maximum number of iterations exceeded in the
        power method.\n');
end

% ----- Inverse power method ----------

function [mu, x] = inv_power_method_modified(n, A, x, tol, N)
    % Inverse Power Method for computing smallest eigenvalue
       and eigenvector
    %   n: Dimension of matrix A
```

```
    %   A: n x n matrix
    %   x: Initial vector
    %   tol: Tolerance value for convergence
    %   N: Maximum number of iterations
    %   mu: Smallest eigenvalue, NaN if maximum iterations
       exceeded
    %   x: Corresponding eigenvector, NaN if maximum
       iterations exceeded
    n = size(A,1);
    q = 0;
    k = 1;
    x_p = x(find(abs(x) == norm(x, inf), 1));
    x = x / x_p;

    while k <= N
        [L, U] = LU_Fac_Fun(A - q * eye(n));
        LU_Solver = Forward_Sub_Fun(L, x);
        y = Backward_Sub_Fun(U, LU_Solver);

        y_p = y(find(abs(y) == norm(y, inf), 1));
        mu = y_p;

        ERR = norm((x - y / mu), inf);
        x = y / mu;

        if ERR < tol
            mu = (1 / mu) + q;
            disp(mu);
            return;
        end

        k = k + 1;
    end
    mu = (1/mu)+ q;
    fprintf('Maximum number of iterations exceeded in inverse
        power method\n');
end

%--------------- Back Substitution--------------
% this function is to solve the system Ux = b for x, where
% U: the upper triangular matrix
% b: the right-hand side vector
% x: the solution vector
function x = Backward_Sub_Fun(U,b)

    n = size(U,1);
    if size(b, 1) ~= n
        error('Matrix dimensions are inconsistent.')
```

```
    end
    x = zeros(n,1);

    for i = n:-1:1
        x(i) = b(i);
        for j = i+1:n
            x(i) = x(i) - U(i,j)*x(j);
        end
        x(i) = x(i)/U(i,i);
    end

end

%---------------- Forward Substitution---------------
% this function is to solve the system Ly = b for y, where
% L: the lower-triangular matrix
% b: the right-hand side vector
% y: the solution vector

function x = Forward_Sub_Fun(L, b)
    n = size(L,1);
    x = zeros(n,1);
    x(1,1) = b(1,1)/L(1,1);
    for i = 2:n
        sum = 0;
        for j = 1:i-1
            sum = sum + L(i,j)*x(j,1);
        end
        x(i,1) = (b(i,1)-sum)/L(i,i);
    end
end


%--------LU factorization-------------------
function [L, U] = LU_Fac_Fun(A)
% this function is to solve the LU factorization of a nxn
   matrix A
% A: the nxn matrix
% L: the lower-triangular matrix
% U: the upper-triangular matrix
    n = size(A,1);
    % Initialize L and U matrices
    L = eye(n);
    U = A;
    % Gaussian elimination without pivoting
    for k = 1:n-1
        if U(k,k) == 0
           disp('Factorization impossible');
```

```
        end
        for i = k+1:n
            factor = U(i,k) / U(k,k);
            L(i,k) = factor;
            U(i,k:n) = U(i,k:n) - factor * U(k,k:n);
        end
    end
end

%--------deflation method (largest)--------------------
function [mu, u] = deflation_largest(n, A, lambda, v, TOL, N)
    B = zeros(n-1, n-1);
    V = sort(v);
    idx = 1;

    for p = 1:n
        if v(p) == V(end)
            break;
        end
        idx = idx + 1;
    end

    if idx ~= 1
        for k = 1:idx-1
            for j = 1:idx-1
                B(k, j) = A(k, j) - v(k) / v(idx) * A(idx, j)
                    ;
            end
        end
    end

    if (idx ~= 1) && (idx ~= n)
        for k = idx:n-1
            for j = 1:idx-1
                B(k, j) = A(k+1, j) - v(k+1) / v(idx) * A(idx
                    , j);
                B(j, k) = A(j, k+1) - v(j) / v(idx) * A(idx,
                    k+1);
            end
        end
    end

    if idx ~= n
        for k = idx:n-1
            for j = idx:n-1
                B(k, j) = A(k+1, j+1) - v(k+1) / v(idx) * A(
                    idx, j+1);
            end
```

```
        end
    end

    x = ones(n-1, 1);
    [mu, w_prime] = Power_Method_fun_modified(n-1, B, x, TOL,
        N);

    w = zeros(n, 1);

    if idx ~= 1
        for k = 1:idx-1
            w(k) = w_prime(k);
        end
    end

    w(idx) = 0;

    if idx ~= n
        for k = idx+1:n
            w(k) = w_prime(k-1);
        end
    end

    u = zeros(n, 1);

    for k = 1:n
        sum_val = 0;
        for j = 1:n
            sum_val = sum_val + A(idx, j) * w(j);
        end

        u(k) = (mu - lambda) * w(k) + sum_val * v(k) / v(idx)
           ;
    end
end


% ----- deflation 2nd smallest method ----------

function [mu, u] = deflation_second_smallest(n, A, lambda, v,
    TOL, N)
    B = zeros(n-1, n-1);
    V = sort(v);
    idx = 1;

    for p = 1:n
        if v(p) == V(end)
            break;
```

```
        end
        idx = idx + 1;
    end

    if idx ~= 1
        for k = 1:idx-1
            for j = 1:idx-1
                B(k, j) = A(k, j) - v(k) / v(idx) * A(idx, j)
                    ;
            end
        end
    end

    if (idx ~= 1) && (idx ~= n)
        for k = idx:n-1
            for j = 1:idx-1
                B(k, j) = A(k+1, j) - v(k+1) / v(idx) * A(idx
                    , j);
                B(j, k) = A(j, k+1) - v(j) / v(idx) * A(idx,
                    k+1);
            end
        end
    end

    if idx ~= n
        for k = idx:n-1
            for j = idx:n-1
                B(k, j) = A(k+1, j+1) - v(k+1) / v(idx) * A(
                    idx, j+1);
            end
        end
    end

    x = ones(n-1, 1);
    [mu, w_prime] = inv_power_method_modified(n-1, B, x, TOL,
        N);

    w = zeros(n, 1);

    if idx ~= 1
        for k = 1:idx-1
            w(k) = w_prime(k);
        end
    end

    w(idx) = 0;

    if idx ~= n
```

```
        for k = idx+1:n
            w(k) = w_prime(k-1);
        end
    end

    u = zeros(n, 1);

    for k = 1:n
        sum_val = 0;
        for j = 1:n
            sum_val = sum_val + A(idx, j) * w(j);
        end

        u(k) = (mu - lambda) * w(k) + sum_val * v(k) / v(idx)
           ;
    end
end
```