Numerical Analysis

LU Decomposition and Power Method

Aristotle University of Thessaloniki

Computational Physics

Professor of Subject: Dr. Kosmas Kosmidis

$$\begin{bmatrix} 4 & 1 & 2 & 3 & 5 \\ 1 & 3 & 1 & 4 & 2 \\ 2 & 1 & 5 & 2 & 3 \\ 3 & 4 & 2 & 4 & 1 \\ 5 & 2 & 3 & 1 & 5 \end{bmatrix}$$

Papadopoulou Martha

AEM 4418

20/12/2023

Contents

1 Introduction

1.1 The Problem Statement

This assignment explores the analysis of the following square matrix:

$$\begin{bmatrix} 4 & 1 & 2 & 3 & 5 \\ 1 & 3 & 1 & 4 & 2 \\ 2 & 1 & 5 & 2 & 3 \\ 3 & 4 & 2 & 4 & 1 \\ 5 & 2 & 3 & 1 & 5 \end{bmatrix} \tag{1}$$

This analysis includes the evaluation of the inverse matrix, through the LU decomposition method, as well as the estimation of the largest eigenvalue and its corresponding eigenvector. For both processes, there is the theoretical explanation and the implementation through a MATLAB code. At the end of the LU decomposition, there is an extra verification step to ensure the accuracy of the obtained results. For accurate results in the power method we rely on the approximation error.

Before implementing both LU decomposition and the power method, the nature of the matrix needs to be examined. One requirement is that the target matrix need to be square, which is evident in the current 5×5 matrix. The other requirement is that the matrix determinant must be non-zero, which indicates that the matrix is invertible. Both criteria are essential for implementing of these processes.

2 LU Decomposition

2.1 Theoretical Background of LU Decomposition

2.1.1 Solutions with LU Decomposition

The LU decomposition is a more effective version of the Gaussian elimination method. In Gaussian elimination, if we have different right-hand-side b vectors for the same A matrix in the equation $A\,x = b$, the calculations must restart from the beginning. Thus, it is more efficient to use the LU decomposition method, which preserves the results of Gaussian elimination for subsequent utilization. (ref. Pang [2006])

As briefly mentioned in Introduction, in order to apply LU decomposition, the A matrix must be invertible, meaning a non-zero determinant and square. Consequently, the A matrix can be expressed as the multiplication of a lower-triangular matrix L and an upper-triangular matrix U.

$$A = L\,U \tag{2}$$

For a 5×5 matrix, this expression takes the following form.

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} & \alpha_{15} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} & \alpha_{25} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} & \alpha_{35} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} & \alpha_{45} \\ \alpha_{51} & \alpha_{52} & \alpha_{53} & \alpha_{54} & \alpha_{55} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix} \cdot \begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & 1 & u_{23} & u_{24} & u_{25} \\ 0 & 0 & 1 & u_{34} & u_{35} \\ 0 & 0 & 0 & 1 & u_{45} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Whether the diagonal values of matrix A go to L or U, and respectively, the entries with a value of one go to U or L, is arbitrary. In this assignment the L matrix contains the diagonal values of the A matrix. (ref. Kokkotas [2008])

The initial elements of the two matrices are straightforward to determine. The first column of L is equal to the first column of A, while the first row of U is equal to the values of the first row of A divided by its first element.

$$l_{i1} = \alpha_{i1} \qquad\qquad u_{1j} = \frac{\alpha_{1j}}{l_{11}} = \frac{\alpha_{1j}}{\alpha_{11}} \tag{4}$$

The remaining elements are determined by the following formulas, for $j \leq i$.

$$\begin{aligned} l_{ij} &= \alpha_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \qquad for\ i = 1, 2, ..., N \\ u_{ji} &= \frac{\alpha_{ji} - \sum_{k=1}^{j-1} l_{jk} u_{ki}}{l_{jj}} \qquad for\ j = 2, 3, ..., N \end{aligned} \tag{5}$$

By finding L and U, the linear equations $A\,x = b$ can be transformed into two equations.

$$L\,y = b \qquad\qquad U\,x = y \tag{6}$$

The first equation can be solved, since L and b are known. The vector y can be calculated using forward substitution. The formula used to calculate its elements is the following.

$$y_1 = \frac{b_1}{l_{11}}, \qquad y_i = \frac{b_i - \sum_{k=1}^{i-1} l_{ik} y_k}{l_{ii}} \qquad for\ i = 2, 3, ..., N \tag{7}$$

Having determined U and y, we can proceed to calculate the solution vector x using backward substitution. Similarly, the formula to compute its elements is as follows.

$$x_N = \frac{y_N}{u_{NN}}, \qquad x_i = \frac{y_i - \sum_{k=i+1}^{N} u_{ik} x_k}{u_{ii}} \qquad for\ i = N-1, N-2, ..., 1 \tag{8}$$

The formulas above are given in ref. Pang [2006].

2.1.2 Inversion with LU Decomposition

As mentioned before, the LU decomposition is ideal for solving equations sets for different b vectors. This is why it can be utilized for matrix inversion. To do this, a set of vectors b_{ij} has to be defined such that $b_{ij} = \delta_{ij}$, where $i, j = 1, 2, ..., N$. This corresponds to a zero matrix with ones on the diagonal. By solving $A\,x_j = b_j$ for each column of b_{ij} yields the different x_j, which represents the columns of the inverse A^{-1} matrix. Hence, in the end we obtain $A^{-1}_{ij} = x_{ij}$. (ref. Pang [2006])

2.1.3 Pivoting Criteria

Pivoting is typically employed to prevent division by very small values, comparatively to the other elements of the matrix. Applying the LU decomposition or Gaussian elimination to a matrix without checking if it requires pivoting can lead to significant errors due to this division. To avoid such issues, pivoting is employed, where certain rows or columns of the matrix are swapped to ensure a more reliable computation. (ref. Kokkotas [2008])

There are two criteria that determine if a matrix requires pivoting, if either of these is met, it doesn't need pivoting, and the matrix can be used as is. That happens when the matrix is either diagonally dominant, as expressed in relation (9), or symmetric with positive-definite, as depicted in relation (10). (ref. Kosmidis [2023])

$$|\alpha_{ii}| \geq \sum_{j=1,\, j\neq i}^{N} |\alpha_{ij}|, \qquad for\ i = 1, 2, ..., N \tag{9}$$

$$A^T = A \qquad and \qquad x^T\, A\, x > 0 \qquad for\ all\ x \neq 0 \tag{10}$$

2.2 Implementation of LU Decomposition in Code

2.2.1 LU Decomposition Code Overview

The first thing we do in the code is define the A matrix as is shown in (1). The aim is to compute the inverse matrix A^{-1}, which is executed within the custom function "LU_inversion". The function takes as input the designated matrix A and it returns the inverse A^{-1}.

Initially, the "LU_inversion" function verifies that the target matrix fulfills the requirements for employing LU decomposition. It checks if the determinant is non-zero and if the A matrix is square. Additionally, it examines if the matrix needs pivoting, by checking whether either of the conditions (9) or (10) are satisfied. If all above requirements are fulfilled, the function displays a message noting that the A matrix is amenable to the LU decomposition without pivoting, and then proceeds to the calculations. If the conditions aren't met the function displays a message indicating that the method cannot be used.

The size of L and U matrices are initialized according to the dimensions of the target matrix A. For the L matrix the "zeros" function is employed, but for U the "eye" function is used, as per theoretical

knowledge indicating that its diagonal values are ones. The L and U matrices are computed using the formulas of equations (4) and (5). The sums are independently calculated and then incorporated into the calculation of each element. When the computation is complete the final matrices L and U are displayed. To validate whether they are correct, a confirmation matrix A_{LU} is created, by multiplying $L \cdot U$. If the the A_{LU} matrix is equal to the original A, then the calculations were successful and an appropriate message is displayed. Alternatively, the message states that the calculations were unsuccessful.

Afterwards, the code advances to the computation of the inverse A^{-1} matrix. At first, the inverse matrix is initialized with zero values and the same size as the original A matrix. The b vector is likewise initialized with zero values and size corresponding to that of matrix A. It is updated to have a value of 1 in the corresponding row element, progressing sequentially with each iteration.

As was stated in the theoretical section, the first step is to find the y vector. Prior to this, the y vector is initialized as a column with zero values, its number of rows matching those of matrix A. The zero values are replaced by its elements that are computed using the equations (7), where the first value is calculated individually and the subsequent values are calculated within two nested "for" loops that iterate over changing indices. Once this procedure is complete, the solution vector x is computed. This computation is nearly identical to that of the y vector. The x vector is initialized with zero values and then each element is replaced with the corresponding value calculated using equation (8).

For every b vector, a corresponding solution x vector is obtained, which is a column in the inverse matrix A^{-1}. Therefore, by solving for all b, the matrix A^{-1} is acquired. To verify if the computation is correct, we assess whether the product of A and A^{-1} equals to the identity matrix I and an appropriate message is displayed as an output. Due to the fact that computers use finite precision to represent numbers, there are rounding and truncation errors, that may lead to small differences in the results, after a decimal point. To mitigate this issue in the confirmation process, the "round" function is employed to achieve precision up to 13 decimal places, ensuring accuracy.

The function returns the inverse matrix A^{-1} and if the process is successful, the resulting matrix is presented.

2.2.2 LU Decomposition Code Execution Results

For the target matrix (1), the specified conditions are fulfilled and the output displays that the matrix A is invertible and amenable to the LU decomposition, without the need for pivoting.

The computed L and U matrices are displayed as follows.

$$L = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 1 & 2.75 & 0 & 0 & 0 \\ 2 & 0.5 & 3.9091 & 0 & 0 \\ 3 & 3.25 & -0.0909 & -2.093 & 0 \\ 5 & 0.75 & 0.3636 & -3.6279 & 4.8 \end{bmatrix} \quad U = \begin{bmatrix} 1 & 0.25 & 0.5 & 0.75 & 1.25 \\ 0 & 1 & 0.1818 & 1.1818 & 0.2727 \\ 0 & 0 & 1 & -0.0233 & 0.093 \\ 0 & 0 & 0 & 1 & 1.7333 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

The confirmation matrix A_{LU} is indeed equal to the original A and a message is presented to the user confirming the correctness of the calculations for L and U. The functions concludes by confirming the

inverse matrix A^{-1}, by multiplying it with the original A matrix and confirming that the product is the identity I matrix. Subsequently, it presents a message affirming the correctness of the calculations for A^{-1}.

Once all processes are completed the function returns the A^{-1} matrix, which is displayed in the main program as follows.

$$A^{-1} = \begin{bmatrix} 0.2454 & -0.4583 & -0.1019 & 0.3426 & -0.0694 \\ -0.4583 & 0.3750 & -0.0833 & -0.0833 & 0.3750 \\ -0.1019 & -0.0833 & 0.2593 & 0.0370 & -0.0278 \\ 0.3426 & -0.0833 & 0.0370 & 0.1481 & -0.3611 \\ -0.0694 & 0.3750 & -0.0278 & -0.3611 & 0.2083 \end{bmatrix} \tag{12}$$

3 Power Method

3.1 Theoretical Background of Power Method

3.1.1 Determination of Dominant Eigenvalue and Corresponding Eigenvector

The power method is an iterative technique used to find the largest, or smallest eigenvalue of the matrix, and additionally it produces the corresponding eigenvector. For the purposes of this assignment, the power method is only used to compute the largest eigenvalue and its eigenvector. (ref. Kiusalaas [2010])

It is established in the field of linear algebra that the system at hand can be analyzed using eigenvalues and eigenvectors. In that case it has to take the following form.

$$A \cdot x = \lambda\, x \tag{13}$$

This method is utilized for square matrices A with a distinct dominant eigenvalue, which is contributing to the convergence of the method.

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq ... \geq |\lambda_N| \tag{14}$$

Furthermore, each vector x has to be expressible as a linear combination of the N eigenvectors. For $1 \leq i \leq N$, it can be represented as follows.

$$Au^{(i)} = \lambda_i\, u^{(i)}, \qquad x = \alpha_1\, u^{(1)} + \alpha_2\, u^{(2)} + ... + \alpha_N\, u^{(N)} \tag{15}$$

If we multiply the equation (15) with the A matrix k times, the resulting form is the following.

$$\begin{aligned} x^{(k)} \equiv A^k\, x &= \alpha_1\, \lambda_1^k\, u^{(1)} + \alpha_2\, \lambda_2^k\, u^{(2)} + ... + \alpha_N\, \lambda_N^k\, u^{(N)} \\ &= \lambda_1^k \left(\alpha_1\, u^{(1)} + \alpha_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k u^{(2)} + ... + \alpha_N \left(\frac{\lambda_N}{\lambda_1}\right)^k u^{(N)} \right) \end{aligned} \tag{16}$$

As k approaches infinity, the fractions $(\lambda_j/\lambda_1)^k$ tend toward zero, given that λ_1 is the dominant eigenvalue and the fractions are values less than 1. Therefore, for large values of k, the following approximation holds well.

$$x^{(k)} = A^k\, x \approx \lambda_1^k\, \alpha_1\, u^{(1)} \tag{17}$$

It is evident that we can approximate the largest eigenvalue using the following fraction.

$$\frac{x^{(k+1)}}{x^{(k)}} = \frac{A^{k+1}\, x}{A^k\, x} \approx \frac{\lambda_1^{k+1}\, \alpha_1\, u^{(1)}}{\lambda_1^k\, \alpha_1\, u^{(1)}} \to \lambda_1 \tag{18}$$

And the corresponding eigenvector is the normalized $x^{(k+1)}$. Only the largest element of vector $x^{(k+1)}$ and the corresponding element of $x^{(k)}$ are divided for the determination of the eigenvalue. The formulas above are given in ref. Kokkotas [2008].

In order to utilize the equations above, we initiate the process with a randomly chosen vector x, which is then successively multiplied by matrix A in each iteration. This iterative multiplication terminates when the calculation of the dominant eigenvalue has reached a desirable satisfactory level.

3.1.2 Error Assessment

As mentioned earlier, the power method is an iterative approach. Consequently, it requires a termination condition, that once met, the iterations cease, ensuring the attainment of a satisfactory level of accuracy. This condition is defined by the evaluating the approximate percent relative error ϵ_a and the prespecified percent tolerance ϵ_s. Both of these are computed using the following formulas, with n representing the desired number of significant figures for accuracy.

$$\epsilon_a = \frac{\textit{current approximation} - \textit{previous approximation}}{\textit{current approximation}} \times 100\% \tag{19}$$

$$\epsilon_s = (0.5 \times 10^{2-n})\% \tag{20}$$

In order to obtain a result that is accurate for n significant figures the approximate error must be lower than the prespecified tolerance $|\epsilon_a| < \epsilon_s$. This is the condition that terminates the iterations, assuring the desirable accuracy for the eigenvalue. (ref. Kosmidis [2023])

3.2 Implementation of Power Method

3.2.1 Power Method Code Overview

The main program calls the "power_method" function, which takes as input the target matrix A and returns the dominant eigenvalue l and its corresponding eigenvector v. Initially the function examines the nature of the A matrix. If its determinant is non-zero and it's a square matrix, the computation proceeds, otherwise, the user is presented with an appropriate message.

The function specifies a precision level of 10 significant figures for the accuracy of the calculations, and defines the percent tolerance error es that is needed for the termination condition, using the equation (20). In reference Kiusalaas [2010] it is mentioned that any random vector of unit magnitude is suffices for the convergence process. If the selection of x is random each time the function runs, the number of iterations may vary for the same target matrix A. To mitigate this randomness, the starting random vector x is initialized with values of one, except for the first element, which is set to two, and afterwards normalized.

Before entering the iteration loop, certain initial values must be set. The initial approximate error is set to $ea = 100$ to allow entry into the "while" loop. However, it is not added to the $ea\ values$ list, which is initially empty. This list is subsequently populated with ea values within the loop. The loop tracks the iterations with the i variable that initiates at zero. The final variable defined before the loop is the $x^{(1)}$, which represents the product of matrix A and the normalized initial vector x. This is set to be the vector from the previous iteration $(i-1)$ allowing the calculation of the vector for the current iteration i within the loop .

The computation of the power method occurs within a "while" loop that runs until the approximate error is less that the tolerance $ea < es$, or until the number of iterations exceed a maximum value of 300. The first process within the loop is determining the vector $x^{(i)}$, which is achieved by multiplying the A matrix with the vector of the previous iteration, $x^{(i-1)}$. Subsequently, the "max" function is employed to find the largest value of $x^{(i)}$ and its corresponding index. This approach facilitates the calculation of the the eigenvalue $lambda$ through the fraction of equation (18). To determine whether the accuracy is satisfactory, the ea error is calculated using the equation (19) and the result is appended to the $ea\ values$ list. However, the error ea requires a previous value of $lambda$ for computation, leading to the exclusion of the calculation during the initial run of the loop. Lastly, the values are updated for the next iteration, where the eigenvalue $lambda$ and vector $x^{(i)}$ now assume the roles of the variables from the preceding iteration and the count i is increased by one.

Once the loop concludes, the corresponding eigenvector v is normalized by dividing each element with its largest value. If the power method computation fails to achieve the desired accuracy within the maximum allowed iterations, an appropriate message is displayed. Alternatively, a message is presented to the user indicating the number of iterations that required to achieve accuracy of k significant figures, which in this case is $k = 10$.

The final step of the function involves plotting the convergence of the power method, by graphing the ea values for each iteration. Note that the iterations start at 2, as the first ea value cannot be calculated during the first iteration. Afterwards, the function returns the eigenvalue l and the corresponding eigenvector v to the main program. There, if the computation was successful the eigenvalue and eigenvector are displayed.

3.2.2 Power Method Code Execution Results

The target matrix A of (1) is square with a non-zero determinant, thus the functions outputs a message indicating that the matrix is amenable to the power method. The computations proceeds to the calculation of the dominant eigenvalue and corresponding eigenvector. As the calculations were successful, the message presented to the user indicates that for accuracy of 10 significant figures, the number if required iterations is 20.

The function also provides the plot for the convergence of the power method, which is presented in Figure 1. As observed, the approximate error starts from a very low value, yet it still needs 20

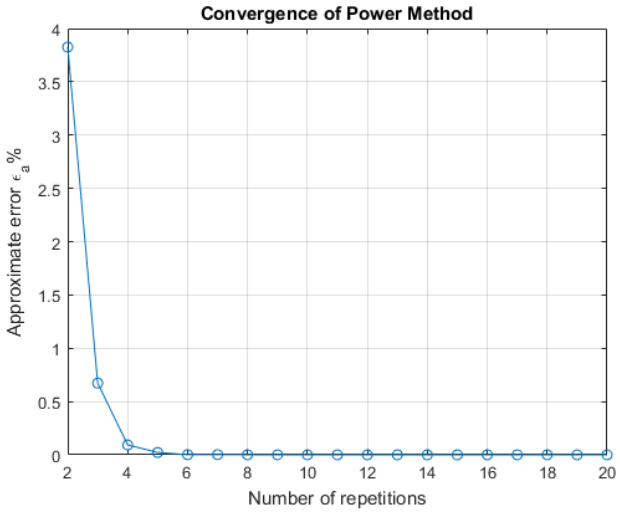

Figure 1: Plot of the convergence of the Power Method. The x-axis represents the number of iterations, until the satisfactory value of the approximate error was achieved. The iterations start at 2, since the calculation of ϵ_a is impossible for the first iteration. The y-axis represents the values of the approximate percent error. The values are represented by circles and the lines connect them. The convergence of the method depends on the initial random vector x.

iterations to achieve accuracy up to 10 significant figures. It is important to note that the convergence of the method varies depending on the initial random vector x. This figure and its convergence are corresponding to the normalized version of $x = [2\ 1\ 1\ 1\ 1]^T$. If different normalized random vector were chosen, the number of iterations needed would vary from 19 to 26, which is not a significant difference. In summary, the power method demonstrates an adequate rate of convergence, efficiently achieving satisfactory results.

When the function successfully returns the results to the main program, the dominant eigenvalue and its corresponding eigenvector are revealed to be the following.

$$\lambda = 14.07561378 \qquad v = [0.94455\ \ 0.61803\ \ 0.77879\ \ 0.78043\ \ 1]^T \tag{21}$$

4 Conclusions

In conclusion, the nature of the target matrix A is such that it did not pose any challenges in the computation of both the LU decomposition and the power method. The matrix is square, possesses a non-zero determinant and satisfies at least one of the two conditions that obviate the necessity for pivoting.

The LU decomposition is a more efficient alternative to Gaussian elimination, since it stores information within the L and U matrices. This feature enables the system to be solved for various right-hand side vectors b. Utilizing this, the LU decomposition can be employed for the inversion of a matrix. For the target matrix (1), the inverse matrix is given by relation (12). In order to validate the correctness of the computation, the target matrix A and its inverse A^{-1} are multiplied,

and the product is an identity matrix I, as anticipated. This outcome serves as a confirmation for the accuracy of the calculations.

The power method is a numerical technique for approximating the dominant eigenvalue and its corresponding eigenvector. Given a target matrix and an initial random vector, it starts the iterative process, which concludes when the desired level of accuracy is achieved. For this analysis the accuracy is configured to be up to 10 significant figures of the eigenvalue. For the target matrix (1) and the normalized version of the random vector $x = [2\ 1\ 1\ 1\ 1]^T$, the method converges in 20 iterations. The rate of convergence is influenced by the choice of the initial random vector. The calculations are successful and the numerical results for the eigenvalue and eigenvector are $\lambda = 14.07561378$ and $v = [0.94455\ \ 0.61803\ \ 0.77879\ \ 0.78043\ \ 1]^T$

Appendix: Code Listings

```
% Ypologistika Mathhmatika I
% Project 2
% Martha Papadopoulou
% AEM 4438

clear; clc;

A = [4 1 2 3 5; 1 3 1 4 2; 2 1 5 2 3; 3 4 2 4 1; 5 2 3 1 5]; %target matrix

fprintf('Task 1\n ------------------------------------------------------\n');

A_inv = LU_inversion(A); %returns the inverted A
disp('The inverse A matrix using the LU decomposition inversion is:');

%if the calculations were successful displays the inverted A
if A_inv
    disp(A_inv);
end

fprintf('Task 2\n ------------------------------------------------------\n');

[l, v] = power_method(A); %returns the dominant eigenvalue l and corresponding
    eigenvector v

%if the calculations were successful displays l and v
if l
    fprintf('The largest eigenvalue of matrix A using the power method is: %.8f\n', l
    );
    fprintf('\n');
    disp('The corresponding eigenvector is:');
    disp(['[', num2str(v'), ']^T']);
end
```

Listing 1: Main Program

```
function A_inv = LU_inversion(A)
    %input: A (target matrix)
    %output: A_inv (inverse of target matrix)

    D = det(A); %determinant of A
    [m, n] = size(A); %check if A is a square matrix

    if (D ~= 0) && (m == n) %proceeds if A is invertible and square
        disp('Matrix A is invertible and amenable to the LU decomposition.')

        fprintf('\n');

        %check conditions for pivoting
        condition_1a = (A == A');
        rand_x = ones(n,1);
        ran_index = randi(n);
        rand_x(ran_index) = 2;
        condition_1b = (rand_x' * A * rand_x >0);

        condition_2 = 0;
        for i = 1:n
            sum_row = sum(A(i,:)) - A(i,i);
            if (abs(A(i,i)) >= sum_row)
                condition_2 = condition_2 +1;
```

```
            end
        end

        if ((all(condition_1a(:)) && condition_1b) || (condition_2 == n)) %proceeds
    if A doesn't need pivoting
            disp('Matrix A does not need pivoting.')

            fprintf('\n');

            %calculation of L and U matrices
            L = zeros(n); %initialize dimensions of L and U
            U = eye(n); %the diagonal of U is ones, so we use eye() instead of zero()

            %populate L and U matrices
            for i = 1:n
                L(i,1) = A(i,1);
                U(1,i) = A(1,i)./L(1,1);
            end

            for i = 2:n
                for j = 2:i
                    sum_1 = 0;
                    for k = 1:j-1
                        sum_1 = sum_1 + L(i,k) * U(k,j);
                    end
                    L(i,j) = A(i,j) - sum_1;
                end
                for j = i:n
                    sum_2 = 0;
                    for k = 1:j-1
                        sum_2 = sum_2 + (L(i,k) * U(k,j));
                    end
                    U(i, j) = (A(i, j) - sum_2)/L(i,i);
                end
            end

            %display L and U matrices
            disp('L matrix is:');
            disp(L);
            disp('U matrix is:');
            disp(U);

            A_LU = L*U; %confirmation matrix to check LU calculations
            confirmation_1 = isequal(A, A_LU); %if true returns 1, if faulse 0
            if confirmation_1
                disp('LU calculations are correct.')
            else
                disp('LU calculations are incorrect.')
            end

            fprintf('\n');

            A_inv = zeros(n,n); %initialize inverse matrix A_inv

            %solving each column of the inverse matrix
            for j = 1:n
                b = zeros(n,1); %initialize b
                b(j) = 1; %each iteration value 1 is in the next row

                y = zeros(n,1); %initialize y

```

```
                %populate y vector
                y(1) = b(1) / L(1,1);
                for i = 2:n
                    sum_3 = 0;
                    for k = 1:i-1
                        sum_3 = sum_3 + L(i,k) * y(k);
                    end
                    y(i) = (b(i) - sum_3) / L(i,i);
                end

                x = zeros(n,1); %initialize x

                %populate x vector
                x(n) = y(n) / U(n,n);
                for i = n-1:-1:1
                    sum_4 = 0;
                    for k = i+1:n
                        sum_4 = sum_4 + U(i,k) * x(k);
                    end

                    x(i) = (y(i) - sum_4) / U(i,i);
                end
                A_inv(:,j) = x;
            end

            I = eye(n); %identity matrix
            AA = A*A_inv; %confirmation matrix to check A_inv calculation
            confirmation_2 = isequal(round(AA, 13), I); %checks if AA == I
            if confirmation_2
                disp('A^(-1) calculations are correct.');
            else
                disp('A^(-1) calculations are incorrect.');
            end

            fprintf('\n');

        else
            disp('Matrix A needs pivoting.');
        end

    elseif D == 0
        disp('Matrix A is not invertible.')
    elseif m ~= n
        disp('Matrix A cannot be inverted using the LU decomposition.')
    end
end
```

Listing 2: LU Decomposition

```
function [lambda, v] = power_method(A)
    %input: A (target matrix)
    %output: lambda(largest eigenvalue), v (corresponding eigenvector)

    D = det(A); %determinant of A
    [m, n] = size(A); %check if A is a square matrix

    if (D ~= 0) && (m == n) %proceeds if A is invertible and square
        disp('Matrix A is amenable to the power method.')
        fprintf('\n');

        k = 10; %significant figures
```

```
        es = 0.5*10^(2-k); %prespecified percent tolerance

        x = ones(n, 1); %initialize random vector
        x(1) = 2;
        x_norm = x/norm(x);

        ea = 100; %starting value to enter loop, not added to ea_values
        i = 0; %number of iterations
        ea_values = []; %empty list to enter ea values in loop
        x_prev_i = A*x_norm; %first x^(i)

        %power method iteration
        %runs until ea<es or i>300
        while ea >= es && i <= 300

            x_i = A * x_prev_i;
            [max_xi, max_index] = max(x_i); %find max element in x_i and index

            lambda = max_xi/x_prev_i(max_index); %eigenvalue

            %calculation of approximate percent error
            if i > 0
                ea = abs((lambda - lambda_prev)/lambda)*100;
                ea_values = [ea_values, ea];
            end

            %update values for next iteration
            lambda_prev = lambda;
            x_prev_i = x_i;
            i = i+1;
        end

        v = x_i./max_xi; %corresponding eigenvector

        if i > 300
            disp('Maximum number of iterations reached.');
        else
            fprintf('Iterations needed for accuracy of %d significant figures is: %d\
    n', k,i);
            fprintf('\n');
        end

        %plot of convergence of the power method
        figure(1);
        plot(2:(length(ea_values)+1),ea_values, '-o');
        xlabel('Number of repetitions')
        ylabel('Approximate error \epsilon_{a}%')
        title('Convergence of Power Method')
        grid on;


    elseif D == 0
        disp('Matrix A is not invertible.')
    elseif m ~= n
        disp('Matrix A cannot be inverted using the LU decomposition.')
    end

end
```

Listing 3: Power Method

References

Jaan Kiusalaas. *Numerical Methods in Engineering with Python*. Cambridge University Press, 2 edition, 2010. doi: 10.1017/CBO9780511812224.

Konstantinos D. Kokkotas. Introduction to numerical nnalysis with applications in physics, 2008. URL https://elearning.auth.gr/mod/resource/view.php?id=425538. 2023.

Kosmas Kosmidis. Numerical analysis i, 2023. URL https://elearning.auth.gr/course/view.php?id=12327.

Tao Pang. *An Introduction to Computational Physics*. Cambridge University Press, 2 edition, 2006. doi: 10.1017/CBO9780511800870.