Software Project Lab-01

PROJECT NAME: ALGORITHMS FOR THE SOLUTION OF SYSTEM OF LINEAR EQUATIONS

Presented by:
Mohammed Yasin
Roll: 1406

Supervised by:
Professor Dr. Md. Shariful Islam

Overview

Tasks that are completed on my project-

- Motivation
- Dot Product of Matrices
- Multiplication of Matrices
- Matrix Norm
- SVD-
  - Eigenvalue
  - Eigenvector
  - Orthonormal Matrix Calculation
- Implemented the Kaczmarz Method, the OLS Method and the Coordinate Descent Method
- Solved system of linear equations using the Kaczmarz Method and the OLS Method

System of Linear Equation

A linear equation system represents a collection of linear equations that collectively define relationships among a set of variables. The solution to a linear equation system involves finding values for the variables that satisfy all the equations simultaneously.

A linear equation system is written in the form of Ax = b

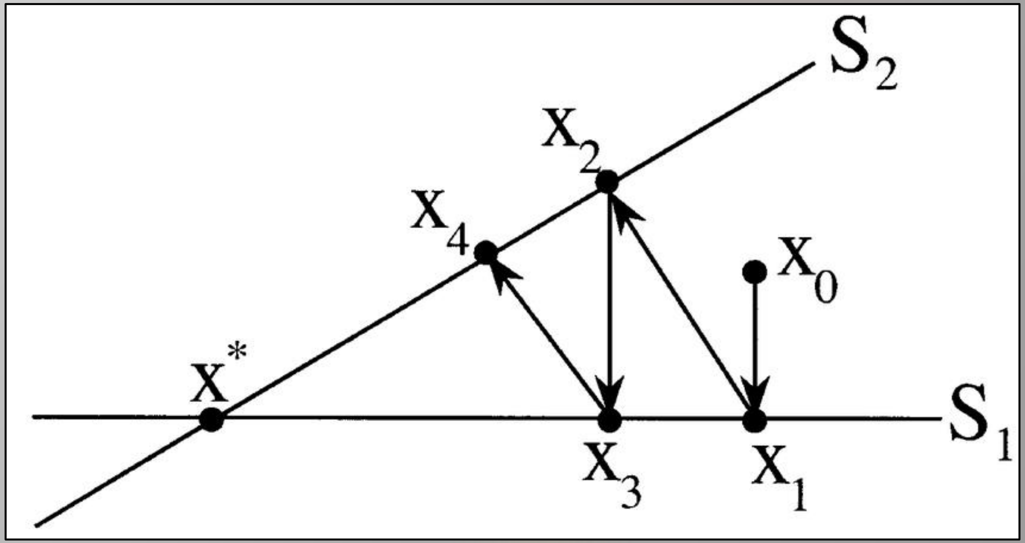
$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Kaczmarz Method

Kaczmarz method is an iterative method to solve linear equation systems.

Initially a solution is guessed. Then in each iteration, using the solution in the last iteration, we head closer to the accurate solution.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots\dots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b \end{pmatrix} \qquad A \cdot x = b$$



Solution using Kaczmarz Method

For a linear equation system Ax=b the Kaczmarz method follows:

- The initial guess of the solution is x^0
- After $(k+1)^{th}$iteration, the solution is –

$$x^{k+1} = x^k + \frac{b_i - <a_i, x^k>}{||a_i||^2} \overline{a_i}$$

- If the process is converged, I printed the solution and stopped the program
- To converge, I have run the process for a number of times

Here,

- $< a_i , x^k >$ is the dot product of the i^{th} row of A and the solution after k iterations
- $||a_i||^2$ is the Euclidean norm of the i^{th} row of A
- $\overline{a_i}$ is the complex conjugation of a_i. For real system, $\overline{a_i} = a_i$
- i is selected using i = k mod m where m is the number of equations

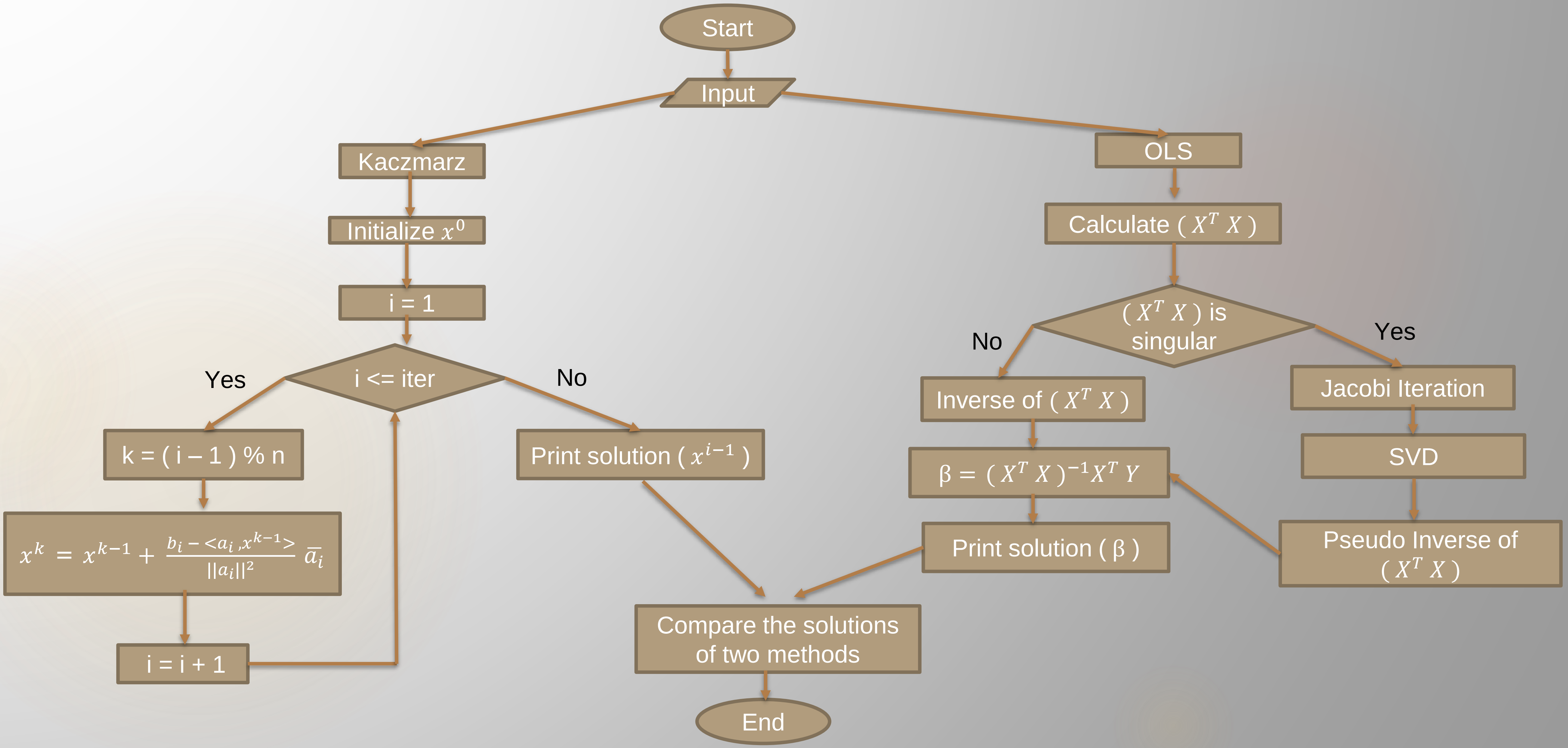
Ordinary Least Squares

In OLS, the coefficient matrix is X, the constant column is Y and the estimates or the solutions are β.

The solution for using OLS is-

$$\beta = (X^T X)^{-1} X^T Y$$

Flow of The Program



Finding the Eigenvalues

At first we need a matrix. Then the transpose of the matrix and the original matrix are to be multiplied.

That means if I have a matrix *mat*, I have to find the transpose matrix mat^T. Then I have to multiply *mat* and mat^T.

A = mat^T x mat or A = mat x mat^T

Now the matrix *A* is a symmetric matrix

Finding the Eigenvalues (continued)

Algorithm to find the eigenvalues and eigenvectors of the matrix A matrix using the Jacobian method:

1. Initialize an identity matrix P of size n x n.
2. Find the index (p, q) of the maximum off diagonal value in the matrix A.
3. $\theta = \frac{1}{2} tan^{-1}\left(\frac{2\, A[p][q]}{A[p][p]-A[q][q]}\right)$.
4. Initialize the rotation matrix R as an identity matrix.
5. Set R[p][p] = R[q][q] = cos(θ).
6. Set R[p][q] = -sin(θ).
7. Set R[q][p] = sin(θ).
8. Update matrix A using the similarity transformation: A = $R^T\ A\ R$.
9. Update matrix P using the same transformation: P = P R
10. Continue the steps 2 to 9 until the matrix A contains non-zero values only on its diagonal elements.

Now the eigenvalues are the diagonal values of the matrix A.

The columns of the P matrix are the corresponding eigenvectors of the A matrix.

SVD

SVD (Singular Value Decomposition) for a data matrix *Mat* is-

$$Mat = U \Sigma \; V^T$$

- U , the left singular matrix, *is an* orthonormal eigenvector of *Mat* x Mat^T
- Σ , the singular matrix, is a diagonal matrix of singular values that are the square root of the eigenvalues
- V^T, the right singular matrix, is an orthonormal eigenvector of mat^T x *mat*

$$\mathrm{U} = [\; U_1 \; U_2 \; U_3 \ldots\ldots \; U_m \;] \qquad \Sigma = \begin{matrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_n \end{matrix}$$

$$V^T = [\; V_1 \; V_2 \; V_3 \ldots\ldots \; V_n \;]^T$$

**Each of the U_i and V_i are column of the matrix*

Finding SVD

- To find the left singular matrix, we need to find the normalized eigenvectors of $A = mat \times mat^T$
- *The sigma matrix is a diagonal matrix whose diagonal elements are the square root of the eigenvalues*
- *To find the right singular matrix, we use the following formula-*

$$U = V \sum^{+} A$$

Pseudo Inverse of Matrix

The pseudo inverse, or Moore-Penrose inverse, of a matrix is a generalization of the inverse, applicable to non-square or singular matrices.

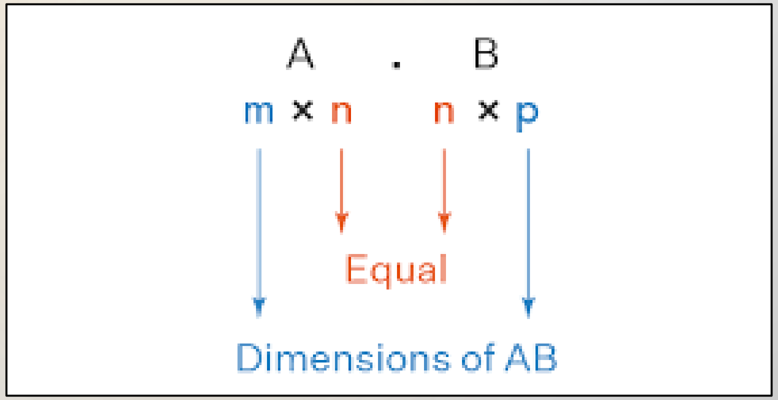
$$\boldsymbol{A}^{+} = \boldsymbol{V}\,\boldsymbol{\Sigma}^{+}\,\boldsymbol{U}^{\boldsymbol{T}}$$

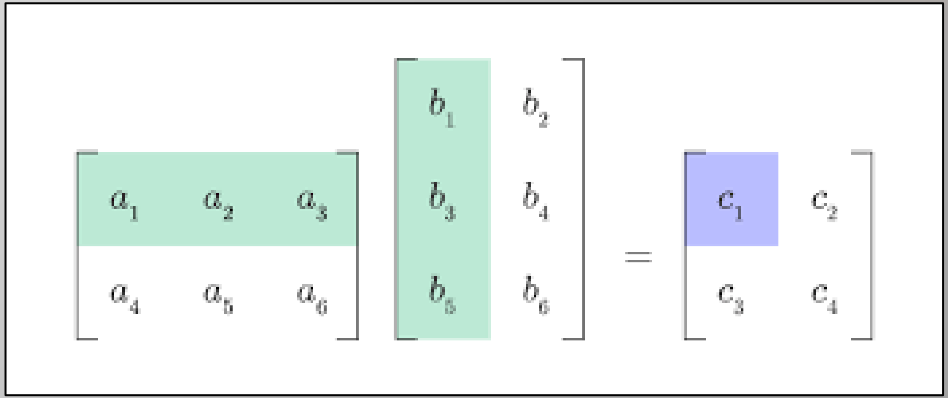
- Why pseudo inverse?

In OLS, the inverse of the matrix $(X^T X)$ is required. If that matrix is singular then we cannot invert it. Pseudo inverse allows us to invert a matrix even if it is a singular or non-square matrix.

Multiplication of two matrices

- Two matrices can be multiplied if the number of columns of the first matrix and the number of rows of the second matrix are same
- The dimension of the result matrix will be (number of rows of the first matrix) x (number of columns of the second matrix)
- Each element of the result matrix will be the dot product of each rows and each columns





Transpose of a matrix

Transpose of a matrix turns the rows of the matrix into its columns and the columns of a matrix into its rows

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}_{2 \times 3} \qquad A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}_{3 \times 2}$$

Code for transpose of a matrix:

```
void transpose(vector<vector<double>> &mat1,vector<vector<double>> &mat2)
{
    mat2.clear();
    int row=mat1.size();
    int col=mat1[0].size();
    int i,j;
    vector<double> temp;
    for(i=0;i<col;i++)
    {
        for(j=0;j<row;j++)
        {
            temp.push_back(mat1[j][i]);
        }
        mat2.push_back(temp);
        temp.clear();
    }
}
```

Matrix Norm

- One of the matrix norms is Euclidean Norm
- Euclidean Norm is the square root of the sum of square of every elements of a vector (a row or a column matrix)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \Rightarrow ||\mathbf{x}||_2 = \sqrt{x_1^2 + x_2^2 + \dots x_n^2}$$

Code for Euclidean Norm:

```
double euclidean_norm_col(vector<vector<double>> mat,int col)
{
    double norm=0;
    int i,j;
    for(i=0;i<mat.size();i++)
    {
        norm+=mat[i][col]*mat[i][col];
    }
    return sqrt(norm);
}
```

Dot product of matrix

Vectors are row or column matrices

Dot product of two vectors is the sum of the product of the elements of two vectors with same index

$$\begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = A_x B_x + A_y B_y + A_z B_z = \vec{A} \cdot \vec{B}$$

Code for dot product:

```
double inner_product(vector<vector<double>> A,int row,int k)
{
    double prod=0;
    for(int i=0;i<n;i++)
    {
        prod+=A[row][i]*x[k][i];
    }
    return prod;
}
```

User Interface

- The user has two options- Proceed for solve or Quit
- If the user proceeds for solve, he/she is prompted for the dimension of the matrix and asked to provide the augmented matrix or generate an augmented matrix
- Number of operations to be used for the Kaczmarz method is also taken from the user

```
Choices:
1. Enter solution process
2. Quit

Enter your choice: 1

Enter size of the matrix: 3

Do you want to generate an augmented matrix of 3 size?
Enter 1 for yes and anything else for no

Enter your choice: 2

Enter the augmented matrix:
1 1 1 6
4 3 1 17
7 -3 -1 16

Enter number of iterations for kaczmarz operation: 1000
```

Output

- The result shows the solutions acquired by the Kaczmarz method and OLS method.
- It gives a verdict on the differences between the solutions

```
Solution using Kaczmarz:
4.843
-2.899
2.911


Solution using OLS:
5.000
-3.000
3.000


There is 'significant differences' between the solutions of Kaczmarz method and OLS method when using 400 iterations
```

Challenges Faced

- Understanding the algorithms for the Kaczmarz and Jacobi iteration for eigenvalues
- Calculating the eigenvalues and eigenvectors
- Applying dynamic memory because of large matrices
- Debugging the code for a long time
- Handling big amount of code and multiple source files for a single purpose for the first time

*Lines of code 1090+

References

- https://www.adeveloperdiary.com/data-science/machine-learning/introduction-to-coordinate-descent-using-least-squares-regression/#:~:text=In%20Coordinate%20Descent%20we%20minimize,fixed%20and%20then%20vice%2Dversa, Introduction to Coordinate Descent using Least Squares Regression, 16 Dec 2023
- KaczmarzSGDrevised.pdf, Xuemei Chen
- kacz Lecture21Notes.pdf, Mark Schmidt, April 9, 2015
- Numerical Methods, Rao V. Dukkipati

Thank you