SPL-1 Project Report

ALGORITHMS FOR THE SOLUTION OF THE SYSTEM OF LINEAR EQUATIONS

Submitted by

Mohammed Yasin

BSSE Roll No : 1406

BSSE Session: 2021-2022

Submitted to

Dr. Md. Shariful Islam

Professor

Institute of Information Technology

University of Dhaka

Supervisor's Approval: _____

(Signature)

Institute of Information Technology

University of Dhaka

17-12-2023

Table of Contents

Index of Figures

1. Introduction

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. There are various methods for solving them.

A linear equation system is a powerful mathematical tool for representing and solving relationships among variables. Its solution methods are diverse, ranging from direct techniques to iterative and optimization-based approaches, each with its own advantages and applications. The ability to analyze and solve linear equation systems is fundamental to understanding and addressing complex problems in diverse domains.

The solution to a linear system can be found using various methods, such as substitution, elimination, or matrix methods like Gaussian elimination or matrix inversion. The method chosen depends on the characteristics of the system and the preferences of the solver. Linear algebra provides a robust framework for understanding and solving systems of linear equations.

It has numerous applications in real life. Here are some examples of its importance:

Finance and Budgeting:

- Linear equations are often used in budgeting to represent income, expenses, and savings.
- Financial planners use systems of linear equations to model and analyze investment portfolios, loan repayments, and interest rates.

Engineering and Physics:

- Electrical circuits can be analyzed using systems of linear equations to determine current and voltage at different points.
- Structural engineering relies on linear equations to model forces, stresses, and deformations in materials.

Economics:

- Supply and demand in economic systems can be modeled using linear equations to find equilibrium points.
- Cost functions and revenue functions in business are often represented by linear equations.

Transportation and Logistics:

- Systems of linear equations are used in optimizing transportation routes and logistics planning.
- Traffic flow and congestion models can be expressed using linear equations.

Statistics and Data Analysis:

- Linear regression involves finding the best-fit line through a set of data points, which can be expressed as a linear equation.
- Systems of linear equations are used in statistical modeling and data analysis to make predictions and draw conclusions.

Telecommunications:

- Signal processing in telecommunications involves solving systems of linear equations to filter and analyze signals.
- Network optimization, including the routing of data, can be formulated using linear equations.

Computer Graphics:

- Linear equations are fundamental in computer graphics for tasks such as image transformation, rotation, and scaling.
- They are used in algorithms for rendering 3D graphics and simulating physical phenomena.

2. Background of the Project

Before we deeply discuss the project, some important terms should be clarified for proper comprehension of the implementation.

➢ **Eigenvalue:** Eigenvalues are essential mathematical concepts that arise in the context of linear transformations. For a given matrix A, an eigenvalue λ is a scalar factor such that when multiplied by a vector v, the result is a scaled version of the original vector: Av=λv. Eigenvalues play a crucial role in various mathematical applications, offering insights into the intrinsic properties and behavior of linear systems. The determination of eigenvalues involves solving the characteristic equation associated with the matrix, providing a valuable tool for understanding the dynamics of linear transformations in diverse mathematical and scientific fields.

$$|\boldsymbol{A} - \boldsymbol{\lambda I}| = \mathbf{0};\ \text{where}\ \boldsymbol{\lambda}\ \textit{is the eigenvalue of}\ A$$

➢ **Eigenvector:** An eigenvector is a vector that, when multiplied by a given matrix, results in a scaled version of itself. In mathematical terms, for a matrix A and an eigenvector v, the product Av equals a scalar multiple of v, denoted as Av=λv. Eigenvectors are fundamental in various applications, including data analysis and machine learning, where they help reveal the principal components or patterns within

datasets. Understanding eigenvectors provides key insights into the behavior of linear transformations, making them a central concept in linear algebra.

$$|\boldsymbol{A} - \boldsymbol{\lambda I}|\boldsymbol{x} = \mathbf{0};$$ where $\mathbf{x}$ is the eigenvectors for $\boldsymbol{\lambda}$.

- **Euclidean Norm:** The Euclidean norm, often referred to as the L2 norm, is a measure of the magnitude or length of a vector in Euclidean space. It captures the distance of a vector from the origin in a geometric sense. The calculation involves summing the squares of each element in the vector and taking the square root of the result. This norm provides a straightforward way to quantify the size of a vector.

$$\textbf{L2 Norm} = \sqrt{\sum_{i=1}^{n} x^2}$$

- **SVD:** The Singular Value Decomposition (SVD) is a powerful matrix factorization technique in linear algebra. For a given matrix, the SVD decomposes it into three matrices: U, Σ, and V^T, where U and V are orthogonal matrices, and Σ is a diagonal matrix with singular values. It enables the representation of a matrix in terms of its essential features, facilitating dimensionality reduction and providing insights into the underlying structure of the data.

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

- **Pseudo Inverse of Matrix:** The pseudo-inverse of a matrix (Moore-Penrose Inverse) is a generalization of the matrix inverse that can be applied to non-square and singular matrices. Denoted as A^+ , it possesses properties similar to those of a true inverse, facilitating solutions to systems of linear equations even when the original matrix is not invertible. The pseudo-inverse is computed using methods such as the Singular Value Decomposition (SVD), allowing for the approximation of solutions and enabling applications in various fields, including statistics, optimization, and machine learning, where robust solutions to ill-conditioned or underdetermined systems are often required.

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

- **Kaczmarz Method:** The Kaczmarz method is an iterative algorithm employed for solving systems of linear equations, particularly when the matrix involved is large and sparse. This method focuses on solving one equation at a time by iteratively adjusting the solution to satisfy each equation. In each iteration, the method selects an equation and updates the solution to minimize the discrepancy between the left- and right-hand sides of that specific equation. The Kaczmarz method is especially useful for large-scale problems.

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

- **OLS Method:** The Ordinary Least Squares (OLS) estimator is a widely used method for estimating the coefficients (β) in a linear regression model. In the context of linear regression, OLS minimizes the sum of the squared differences between the observed

and predicted values of the dependent variable. The objective is to find the values of β that minimize the sum of the squared residuals, providing the "best-fitting" line through the given data. Here this method is used to solve the system of linear equations.

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

Why Pseudo Inverse of a matrix for OLS?

The Ordinary Least Squares method needs the inversion of a matrix. That matrix's determinant can be zero (0). In that case the matrix cannot be inverted. But pseudo inverse is a process which gives the inverse of any kind of matrix. Even a matrix with zero (0) determinant or a non-square matrix can be inverted by using pseudo inverse of a matrix.

3. Description of the Project

I have implemented the Kaczmarz Method, the OLS Method and the Coordinate Descent Method. Using the Kaczmarz Method and the OLS Method, I solved system of linear equations.

Reading Input: An augmented matrix is read from the user. The matrix is Ax = b. Here a is the coefficient matrix, x is the variable column and b is the column of constant values.

Kaczmarz Method: It is an iterative method for solving system of linear equations.

- Initialization: Choose an initial guess for the solution vector x^0. Every value is assigned with zero (0).
- Iteration: For each equation i in the system: $x^{k+1} = x^k + \frac{b_i - < a_i, x^k >}{|| a_i ||^2} \underline{a_i}$
  where a_i is the i-th row of matrix A, b_i is the i-th element of vector b, and $|| a_i ||$ is the Euclidean norm of a_i .
- Repeat: Iterate through the equations for a specified number of iterations given by the user.

Then the solutions acquired by using Kaczmarz method is printed.

OLS Method: For OLS method various matrix operations are used such as transpose of matrix, matrix multiplication etc. The following formula is used to calculate the solutions using OLS:

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

If the matrix $(X^T X)^{-1}$ is a singular matrix, then pseudo inverse is implemented. Because if the determinant of a matrix is zero (0) then the inverse of that matrix cannot be calculated using Gauss-Jordan elimination method. Pseudo Inverse can get the inverse of that type of matrices. If the matrix is not singular then the inverse is calculated using Gauss-Jordan elimination method. Then the solution of the system using OLS is printed. After that a verdict is given if there are significant differences between the solutions of the two methods.

Pseudo Inverse: We need the SVD of the matrix to calculate the pseudo inverse. SVD breaks down the matrix into three parts $U, \boldsymbol{\Sigma}\ and\ V^T$. Pseudo inverse of the matrix is-

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

SVD: We get the SVD of a matrix using Jacobi iteration method for eigenvalues and eigenvectors. This method gives a vector of eigenvalues and a 2d vector whose each column is the eigenvectors of the corresponding eigenvalue. Before proceeding to jacobi iteration, we multiply the matrix with its transpose in order to make a symmetric matrix. For SVD we need the eigenvalues in sorted order. Along with sorting the eigenvalues, the columns 2d vector of eigenvectors need to be swapped. After that we normalize the matrix or 2d vector of eigenvectors and it becomes the right singular matrix. Then the singular matrix is built which is a diagonal matrix whose diagonal elements are the square root of the sorted eigenvalues. We get the left singular matrix using the formula-

$$U = V\,\boldsymbol{\Sigma}^{-1}\,A$$

Jacobi Iteration: This method is used to calculate the eigenvalues and eigenvectors of a symmetric matrix. The method uses the following algorithm:

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[q][p] = sin(Θ).

7. Set R[p][q] = -sin(Θ). Update matrix A using the similarity transformation:

8. $A = R^T\,A\,R$.

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.

4. Implementation and Testing

My project is designed to calculate the solution of system of linear equations by using Kaczmarz method and OLS method. It includes the following attributes:

- Matrix Dot Product
- Matrix Multiplication
- Matrix Norm
- SVD
    - Eigenvalue
    - Eigenvector
    - Orthonormal Matrix
- Pseudo Inverse of Matrix
- Kaczmarz Method
- OLS Method

Matrix Dot Product

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.

```
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;
}
```

Figure 4.1: Matrix Dot Product

Matrix Multiplication

Matrix multiplication is a fundamental operation in linear algebra, involving the combination of two matrices to produce a third. Given two matrices A and B, the product AB is obtained by multiplying corresponding elements and summing them according to a specific pattern. Each element of the resulting matrix is determined by taking the dot product of the corresponding row of matrix A and column of matrix B. Matrix multiplication is a non-commutative operation, and the dimensions of the matrices involved dictate the dimensions of the resulting product.

```
void multiplication(vector<vector<double>> &mat1,vector<vector<double>> &mat2,vector<vector<double>> &mat3)
{
    int i,j,k;
    if(mat1[0].size()!=mat2.size())
    {
        cout<<"Dimension doesn't match for multiplication!"<<endl;
        return;
    }
    double value;
    vector<double> temp;
    for(i=0;i<mat1.size();i++)
    {
        for(j=0;j<mat2[0].size();j++)
        {
            value=0;
            for(k=0;k<mat2.size();k++)
            {
                value+=mat1[i][k]*mat2[k][j];
            }
            temp.push_back(value);
        }
        mat3.push_back(temp);
        temp.clear();
    }
    for(i=0;i<mat3.size();i++)
    {
        for(j=0;j<mat3[i].size();j++)
        {
            if(fabs(mat3[i][j])<EPS)
            {
                mat3[i][j]=0;
            }
        }
    }
}
```

Figure 4.2: Matrix Multiplication

Matrix Transpose

Transpose of a matrix means converting the rows into the columns and the columns into the rows.

```
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();
    }
}
```

Figure 4.3: Matrix Transpose

Swapping Rows

```
void swap_rows(vector<vector<double>> &mat,int I,int J)
{
    vector<double> temp;
    int i,j;
    for(i=0;i<mat[I].size();i++)
    {
        temp.push_back(mat[I][i]);
    }
    for(i=0;i<mat[I].size();i++)
    {
        mat[I][i]=mat[J][i];
    }
    for(i=0;i<mat[I].size();i++)
    {
        mat[J][i]=temp[i];
    }
}
```

Figure 4.4: Swapping Rows

Matrix Norm

The Euclidean norm, often referred to as the L2 norm, is a measure of the magnitude or length of a vector in Euclidean space.

```
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);
}
```

Figure 4.5: Matrix Norm

Eigenvalue and Eigenvectors

Eigenvalues and eigenvectors are calculated using Jacobi iteration method.

```
void eigen_values_and_eigen_vectors(vector<vector<double>> A,vector<double> &eigen_values,vector<vector<double>> &eigen_vectors)
{
    int inI,inJ,i,j;
    double theta;
    double max=max_off_diagonal_value(A,inI,inJ);
    vector<vector<double>> S,t_S,A_new,temp,temp_S;
    make_identity(S,A.size());
    if(A[inI][inI]==A[inJ][inJ])
    {
        theta=0.5*asin(1);
        if(A[inI][inI]<0)
        {
            theta=-theta;
        }
    }
    else
    {
        theta=0.5*atan((2*A[inI][inJ])/fabs(A[inI][inI]-A[inJ][inJ]));
    }
    S[inI][inJ]=-sin(theta);
    S[inJ][inI]=sin(theta);
    S[inI][inI]=cos(theta);
    S[inJ][inJ]=cos(theta);
    transpose(S,t_S);
    multiplication(t_S,A,temp);
    multiplication(temp,S,A_new);
    multiplication(eigen_vectors,S,temp_S);
    matrix_copy(temp_S,eigen_vectors);
    if(!is_diagonal(A_new))
    {
        eigen_values_and_eigen_vectors(A_new,eigen_values,eigen_vectors);
    }
    else
    {
        for(i=0;i<A_new.size();i++)
        {
            if(fabs(A_new[i][i])<10e-3)
            {
                A_new[i][i]=0;
            }
            eigen_values.push_back(A_new[i][i]);
        }
        for(j=0;j<eigen_vectors.size();j++)
        {
            for(i=0;i<eigen_vectors.size();i++)
            {
                eigen_vectors[i][j]/=eigen_vectors[eigen_vectors.size()-1][j];
            }
        }
    }
}
```

Figure 4.6: Eigenvalue and Eigenvectors

Orthogonal Matrix

An orthogonal matrix is a square matrix that, when multiplied by its transpose, results in the identity matrix. The left singular matrix (U) and the right singular matrix (V) in SVD are orthogonal matrices.

```
transpose(t_eigen_vectors,eigen_vectors);
normalized_eigen_vector(eigen_vectors);
sigma_matrix(eigen_values,sigma_mat,n,n);

vector<vector<double>> i_sigma_mat,temp,temp1;
inverse_sigma(sigma_mat,i_sigma_mat);
multiplication(mat,eigen_vectors,temp);
multiplication(temp,i_sigma_mat,U_mat);
transpose(eigen_vectors,V_mat);
```

Figure 4.7: Orthogonal Matrix

Kaczmarz Method

The following operations are used in each operations of the Kaczmarz method.

```
void kaczmarz(vector<vector<double>> A,vector<double> b,int k)
{
    int index=(k-1)%n;
    double coefficient;
    double dot_prod=inner_product(A,index,k-1);
    double norm=norm_sq(A,index);
    double value;
    coefficient=(b[index]-dot_prod)/norm;
    vector<double> temp;
    int i;
    for(i=0;i<n;i++)
    {
        value=x[k-1][i]+coefficient*A[index][i];
        if(fabs(value)<eps)
        {
            value=0;
        }
        temp.push_back(value);
    }

    x.push_back(temp);
}
```

Figure 4.8: Kaczmarz Method

OLS Method

The following snippet is for the OLS method:

```
void ols_operation(vector<vector<double>> mat,vector<double> cons,vector<double> &sol)
{
    sol.clear();
    int i,j;
    X=mat;
    for(auto c:cons)
    {
        Y.push_back({c});
    }
    transpose(X,t_X);
    multiplication(t_X,X,mul_X);
    double det;
    det=determinant(mul_X);
    if(fabs(det)<EPS)
    {
        calculate_pseudo_inverse(mul_X,inverse_mul_X);
    }
    else{
        inverse_matrix(mul_X,inverse_mul_X);
    }
    multiplication(inverse_mul_X,t_X,mul);
    multiplication(mul,Y,ans);
    for(auto a:ans){
        for(auto v:a){
            sol.push_back(v);
        }
    }
    X.clear();
    t_X.clear();
    mul_X.clear();
    inverse_mul_X.clear();
    mul.clear();
    ans.clear();
    Y.clear();
}
```

Figure 4.9: OLS Method

5. User Interface

The program provides the user with two options-

1. To enter the solving process
2. To quit the program

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

Enter your choice: 1
```

Figure 5.1.1: Choices

If 1 is input then it will proceed to the solution process and entering 2 will close the program. If any other input is given then it will again prompt to give choice.

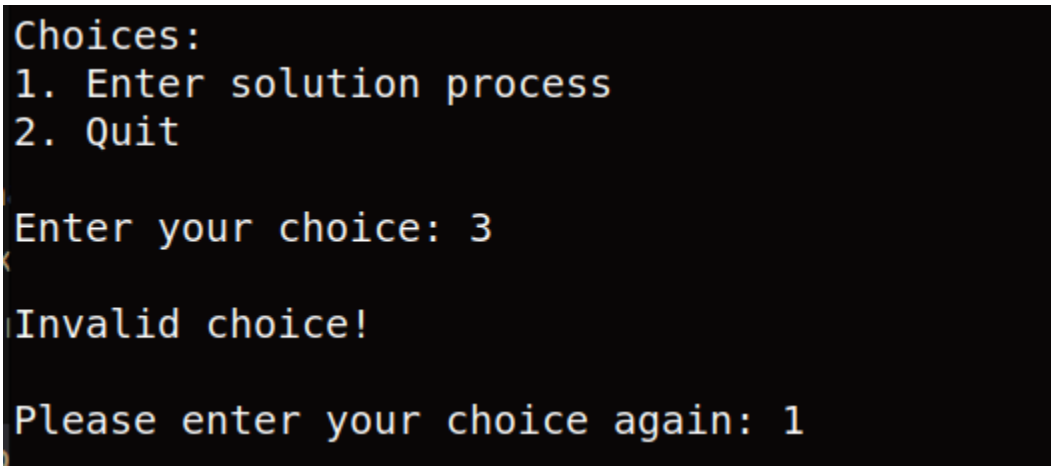

Figure 5.2: Wrong Choice

After choosing 1, the user will be asked to give the size of the matrix. Then the program will ask if the user wants to provide an augmented matrix or generate an augmented matrix. provide the augmented matrix. If the user selects to generate the augmented matrix, then an augmented matrix will be generated and printed. Before entering to the Kaczmarz method, it will prompt the user to give the number of iterations for the Kaczmarz method.

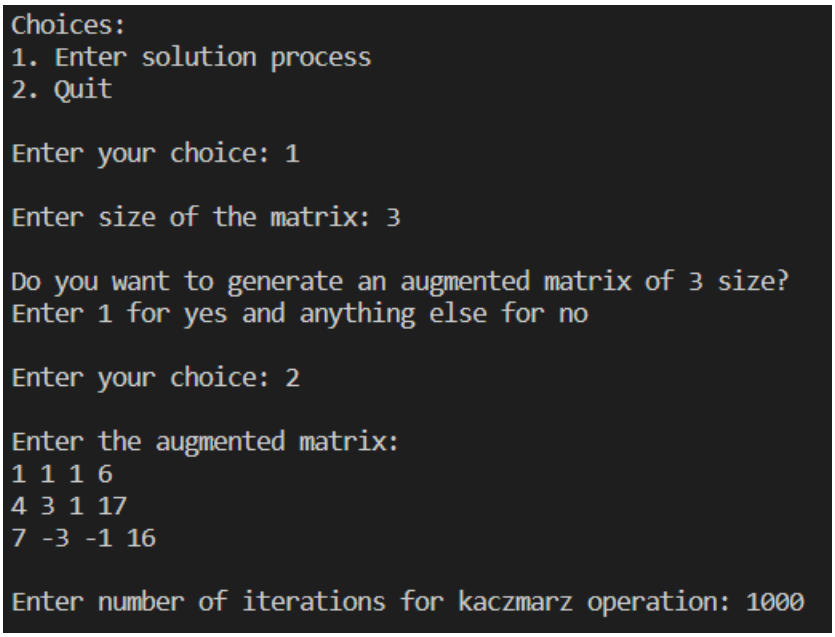

Figure 5.3: Matrix Input

Then the solving process begins. At first the Kaczmarz method is used and the solution using this method is printed. Then the same is done using the OLS method. After that a message is given whether the solutions got from the two methods have significant differences.

```
Solution using Kaczmarz:
4.936
-2.963
2.954


Solution using OLS:
5.000
-3.000
3.000


There is 'significant differences' between the solutions of Kaczmarz method and OLS method when
 using 500 iterations
```

Figure 5.4: Output

Then the user will be asked if he/she wants to use the same equations for different number of iterations in the Kaczmarz method. If choice is given 1, then the same equations will be used for different number of iterations.

```
Do you want to solve the same system for different iteration?
Enter 1 for yes. For no enter anything else.

Enter your choice: 1

Enter number of iterations for kaczmarz operation: 1000


Solution using Kaczmarz:
4.999
-2.999
3.000


Solution using OLS:
5.000
-3.000
3.000


There is 'significant differences' between the solutions of Kaczmarz method and OLS method when
 using 1000 iterations


Do you want to solve the same system for different iteration?
Enter 1 for yes. For no enter anything else.

Enter your choice:
```

Figure 5.5: Choices (Again)

6. Challenges Faced

In this project, I have faced several challenges. Some of these are-

- **Finding and Understanding Appropriate Algorithm:** At the very beginning of this project, it was very difficult to find and understand the algorithms that are needed or related to compute SVD (Singular Value Decomposition).
- **Implementing the Algorithms**: Implementing the Jacobi iteration algorithm and Kaczmarz was a bit difficult.
- **Managing Large Codes:** Handling codes from multiple files was tough for me.
- **Debugging codes:** I got Segmentation fault errors very frequently.

7. Conclusion

The importance of systems of linear equations extends across various fields and disciplines due to their fundamental role in modeling and solving real-world problems. I got to know about the system of linear equations in depth by working on the project. In this project, I got the insight of various numerical methods and algorithms. Matrix manipulation was the heart of the project. The project helped me to enhance my skill on matrix manipulation.

I had to deal with multiple files with big blocks of codes. It helped me to become familiar with practical coding. The methods I implemented require critical thinking. I also got an insight of searching through different research papers and journals.

This project can be extended further. I converged the Kaczmarz method by using a number of iterations given by the users. The process may need more or less iterations to converge. A better way can be implemented to converge the Kaczmarz method where the number of iterations don't need to be taken from the user.

Reference

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

Lecture7.pdf, Jim Lambers

https://en.wikipedia.org/wiki/Ordinary_least_squares, Ordinary least squares, 16 Dec 2023

Appendix

Numerical Analysis, Ninth edition

Numerical Methods, Rao V. Dukkipati

Introduction to Probability and Statistics for Engineers and Scientists, Sheldon M. Ross