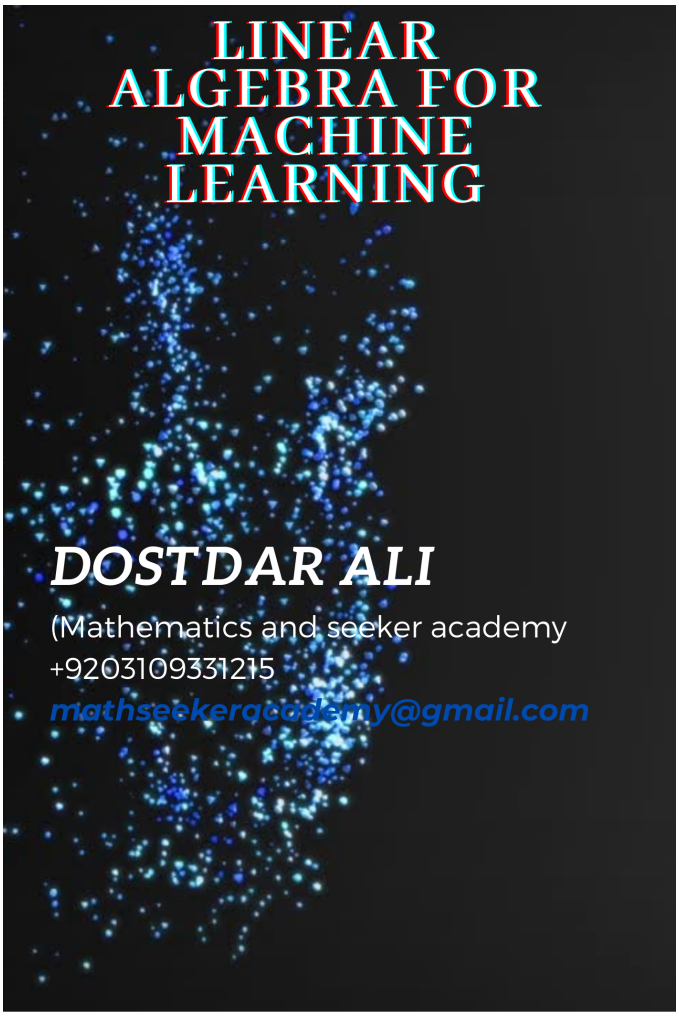

"Welcome to the world of linear algebra and machine learning! This book is designed to provide a comprehensive introduction to the key concepts and techniques of linear algebra and their applications in machine learning. The book is intended for readers with little or no prior experience in linear algebra, but with an interest in machine learning.

Linear algebra is a fundamental tool for understanding and solving problems in machine learning, and its concepts and methods are widely used in the design and analysis of algorithms. Understanding linear algebra is crucial for anyone who wants to work in machine learning or data science.

In this book, we will cover the basic concepts of vectors, matrices, and linear transformations, as well as important operations such as matrix addition, scalar multiplication, and matrix-vector multiplication. We will also explore the use of linear algebra in supervised learning algorithms such as linear and logistic regression, and neural networks. We will cover unsupervised learning algorithms such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), and we will also discuss the Singular Value Decomposition (SVD) and its application in machine learning.

Throughout the book, we will provide examples and exercises to help readers understand the concepts and apply them in practice. By the end of this book, readers will have a solid understanding of the key concepts and techniques of linear algebra and their applications in machine learning, and will be well-prepared to further explore this exciting field.": Introduction to Linear Algebra: This section could cover the basic concepts of vectors, matrices, and linear transformations, as well as important operations such as matrix addition, scalar multiplication, and matrix-vector multiplication.

Linear Regression: This section could cover the use of linear algebra in linear regression, including how to represent the problem as a system of linear equations, how to use matrix notation to represent the data and the parameters, and how to use linear algebra techniques such as matrix inversion to find the solution.

Logistic Regression: This section could cover the use of linear algebra in logistic regression, including how to represent the problem as a system of linear equations, how to use matrix notation to represent the data and the parameters, and how to use linear algebra techniques such as gradient descent to find the solution.

Neural Networks: This section could cover the use of linear algebra in neural networks, including how to represent the problem as a system of linear equations, how to use matrix notation to represent the data and the parameters, and how to use linear algebra techniques such as matrix multiplication and matrix inversion to find the solution.

Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA): This section could cover the use of linear algebra in PCA and LDA, including how to use eigenvalues and eigenvectors to find the directions of maximum variance and maximum separation in the data, respectively.

Singular Value Decomposition (SVD): This section could cover the use of SVD, a linear algebra technique used to factorize a matrix into three matrices: U, S and V, and its applications in machine learning, such as Latent Semantic Analysis, and collaborative filtering.

Conclusion: This section could summarize the key concepts covered in the book and the importance of linear algebra in machine learning.

You could also include examples and exercises throughout the book to help readers to understand the concepts and apply them in practice

Table of Contents

Introduction to Linear Algebra

This chapter provides an introduction to Linear Algebra. It covers the basics of linear equations, matrices, and vectors. The chapter also introduces some key concepts such as determinants, eigenvectors, and eigenvalues.

Linear Algebra and Machine Learning

This chapter explores the applications of Linear Algebra in Machine Learning. It covers topics such as linear regression, principal component analysis, and singular value decomposition. The chapter also discusses how Linear Algebra is used in deep learning and neural networks.

Examples of Linear Algebra in Machine Learning

This chapter provides examples of how Linear Algebra is used in Machine Learning. It covers topics such as image recognition, natural language processing, and recommendation systems. The chapter also discusses the benefits of using Linear Algebra in these applications.

Introduction to Numpy Basic Library in Python

This chapter provides an introduction to the Numpy library in Python. It covers the basics of creating arrays, indexing, and slicing. The chapter also introduces some key functions such as dot product, transpose, and inverse.

Page {{page}}

Linear algebra For Machine Learning

There is no dooubt that linear algebra is most important in machine learning. Linear algebra is the mathematics of data. it's all vectors and matrices of the

numbers. The modern statistics is described using the notaion of linear algebra and modren satatistal methods. Some of the classical methods used in this field, such as linear regression via linear least square and singular-value decomposition, are linear algebra methods and other methods such as principal components analysis, were born from the marriage of linear algebra and statistics. To read and understand machine learning, you must know be able to read and understand linear algebra. Here I am present three basic ideas from linear algebra to Machine learning piture.

- **Data Represation**

> Use basic ideas of linear algebra to represent data in a way that computer can understand Vectors

- **Vecter Embedding**

> Learn Ways to Choose these represenation wisely via matrix Factorizations.

- **Dimensionality Reduction**

> Deal with larger-dimensional data using linear maps and their eigen vectors and eigenvalues.

..... Chapter 1 .....

.....Introduction to Linear Algebra.....

---

1. Introduction to Linear Algebra

Linear algebra is a fundamental mathematical tool that is used extensively in machine learning. To write a book on the topic, you should first have a strong understanding of the concepts and techniques of linear algebra, as well as experience using them in the context of machine learning. Linear algebra is

larger field, In this series of toutorial I just focus on LG in ML perspective. In the first section consist of four parts, they are:

> 1.1 Linear Algebra

> 1.2 Numerical Linear Algebra

> 1.3 Applications of Linear Algebra

1.1 Linear Algebra

Linear algebra is the study of certain kinds of spaces called vector space and a special kinds of transformation of vector space called linear transformation. Linear algebra (LG) is a field of mathematics that is universally agreed to be a prerequisite to a deeper understanding of machine learning (ML). n the orther words, Linear algebra is the mathematics of data. Matrices and vectors are the language of data. Linear algebra is about linear combinations. That is using arithmetic on columns of numbers called vectors and arrays of numbers called matrices. Linear algebra is study of lines and planes, vector spaces and mappings that are required for linear transformation. A linear equation. $y = a.x$, where y is dependent variable and x is independent variable while a is cofficient matrix. If I represent system of equations in this form: Here are some key topics that you may want to consider covering in your section:

1.1.1 Vectors and matrices:

The basic building blocks of linear algebra, including operations such as vector addition, scalar multiplication, matrix multiplication, and inversion. Vectors and matrices are basic building blocks of linear algebra, and they are used to represent and manipulate data in many different ways. Vectors are mathematical objects that can represent quantities such as position, velocity, and force. They are usually represented as arrays of numbers, and can be added and multiplied by scalars. Matrices, on the other hand, are arrays of numbers arranged in rows and columns, and they can be used to represent a wide variety of mathematical objects such as linear transformations, linear systems of equations, and images. In linear algebra, vectors and matrices are used to represent and manipulate data, and basic operations such as vector

addition, scalar multiplication, matrix multiplication, and inversion are used to transform and analyze this data.

1.1.2 Mathematical explanations

Vector addition:

Given two vectors u = $[u_1, u_2, \ldots, u_n]$ and $v = [v_1, v_2, \ldots, v_n]$, the vector addition is defined as $w = u + v = [u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n]$.

Scalar multiplication:

Given a vector $u = [u_1, u_2, \ldots, u_n]$ and a scalar c , the scalar multiplication is defined as $v = c * u = [c * u_1, c * u_2, \ldots, c * u_n]$.

Matrix multiplication:

Given two matrices A and B with dimensions mxn and nxp respectively, the matrix multiplication is defined as C = AB , where C is an mxp matrix and the element in the i-th row and j-th column of C is calculated as $c_{ij} = \Sigma_n k = 1$, $a_{ik} * b_{kj}$.

Matrix inverse:

Given a square matrix A with dimensions n x n , the inverse of matrix A is denoted by A^{-1} and satisfies the equation $AA^{-1} = I$ where I is the identity matrix of the same dimension.

----------Examples & Exceries----------

How to define matrix and vector in Python with NumPy

Vector and matrix

```
# How to define matrix in Python with NumPy
# For example create 2 row, 3 column matrix
from numpy import array
A=array([[1,2,3],[4,5,6]])
print(A)

# vector addition
import numpy as np
```



```
# Create two vectors
v1 = np.array([1, 2, 3])
v2 = np.array([4, 5, 6])

# Add the vectors
v_sum = v1 + v2
print(v_sum)
# Output: [5 7 9]
# Scalar Multiplication
# Create a vector
v = np.array([1, 2, 3])

# Multiply the vector by a scalar
scaled_v = 3 * v
print(scaled_v) # Output: [3 6 9]

# matrix multication
# Create two matrices
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Multiply the matrices
C = np.dot(A, B)
print(C)
# Output:
# [[19 22]
#  [43 50]]

# matrix inverse
# Create a matrix
A = np.array([[1, 2], [3, 4]])

# Calculate the inverse of the matrix
A_inv = np.linalg.inv(A)
print(A_inv)
# Output:
# [[-2.   1. ]
#  [ 1.5 -0.5]]
```

```
[[1 2 3]
 [4 5 6]]
[5 7 9]
[3 6 9]
[[19 22]
 [43 50]]
[[-2.   1. ]
 [ 1.5 -0.5]]
```

1.2. Linear systems:

A system of linear equations is a set of equations that can be written in the form Ax = b, where A is a matrix, x is a vector of variables, and b is a vector of constants. Linear systems can be solved using a variety of techniques such as Gaussian elimination, LU decomposition, and matrix inversion. These techniques allow us to find the values of the variables that satisfy the equations, and they are used in many different areas of mathematics, science, and engineering.

1.2.1 Matrix Algebra

A matrix is a two dimensional array of scalars withone or more rows and one or more cloumns. It also called as a matrix a 2D array (a table) of numbers. The notation for a matrix is often an uppercase letter. such as $A = (a_{i,j})\epsilon R^{N*M}$ where i and j rows and columns whlie dimenstion are denoted as N and M are numbers of rows and column respectily. Matrices are a foundational element of linear algebra. Matrices are used throughout the field of machine learning in the **descriptions of algorithms** and processcs such as **input data** variable (X) when training an algorithm. May we can encounter a matrix in machine learning is in model training data comprised of many rows and columns

1.2.2 Vector Algebra

specifically the basic algebraic operations of vector addition and scalar multiplication.

1.2.3 Mathematical explanations

Given a system of linear equations in the form of Ax = b , where A is a matrix, x is a vector of variables, and b is a vector of constants. The goal is to find the values of x that satisfy the equations. Linear systems can be solved using a variety of techniques such as Gaussian elimination, LU decomposition, and matrix inversion. Furter solution in next chapter.

----------Examples & Exceries----------

How to define matrix and vector in Python with NumPy

Linear system

```
from scipy.linalg import lu

# Create a matrix and a vector
A = np.array([[1, 2], [3, 4]])
b = np.array([5, 6])

# Solve the system of equations Ax = b
x = np.linalg.solve(A, b)
print(x) # Output: [ -4.   4.5]
```

```
[-4.   4.5]
```

3. Eigenvalues and eigenvectors:

Eigenvalues and eigenvectors are important concepts in linear algebra that are used to understand the properties of matrices that remain unchanged under linear transformations. An eigenvalue of a matrix A is a scalar value that satisfies the equation Av = λv, where v is a non-zero vector and λ is a scalar. An eigenvector of a matrix is a non-zero vector that satisfies this equation. Eigenvalues and eigenvectors have many useful properties, such as the ability to diagonalize a matrix and to find the dominant direction of a linear transformation. They are used in many different areas of mathematics, science, and engineering, and they play a key role in many machine learning algorithms.

3.1 Diagonalization of a matrix

Diagonalization of a matrix is the process of finding a similarity transformation (a change of basis) that will transform a given matrix into a diagonal matrix. A

matrix is said to be diagonalizable if it is similar to a diagonal matrix, which means that it can be transformed into a diagonal matrix using a similarity transformation.

Diagonalization is useful in linear algebra for machine learning because it can simplify many calculations and make it easier to understand the properties of a matrix. For example, if a matrix is diagonalizable, its eigenvalues are located on the diagonal of the diagonal matrix, which makes it easy to find the eigenvalues of the original matrix. Additionally, if a matrix is diagonalizable, its eigenvectors form a basis for the space, which means that any vector in the space can be written as a linear combination of the eigenvectors. This can be useful in dimensionality reduction techniques such as PCA, as the eigenvectors with the highest eigenvalues can be chosen to form a new basis that captures the most variation in the data. \par Another example of diagonalization in the context of linear algebra for machine learning is in the training of a linear support vector machine (SVM). The SVM algorithm finds the best hyperplane that separates the data into different classes by maximizing the margin, which is the distance between the hyperplane and the closest data points of each class. The optimization problem that needs to be solved can be formulated as a quadratic programming problem, which involves the calculation of the inverse of the Gram matrix (the matrix of inner products between the data points). If the Gram matrix is not invertible, it can be regularized by adding a small multiple of the identity matrix to make it invertible. This process is called the "Kernel Regularization" and it can be considered as a form of diagonalization. \par In both examples, adding a small multiple of the identity matrix to make the matrix invertible can be seen as a form of regularization, which helps to prevent overfitting by adding some noise to the model. By adding a small multiple of the identity matrix, the eigenvalues of the matrix are slightly perturbed, which makes the matrix invertible and improves the performance of the model.

Mathematical

Given a matrix A, an eigenvalue λ is a scalar value that satisfies the equation Av = λv where v is a non-zero vector. An eigenvector of a matrix A is a non-zero vector v that satisfies this equation. Eigenvalues and eigenvectors have many

useful properties, such as the ability to diagonalize a matrix and to find the dominant direction of a linear transformation.

3.1.1 Diagonalization

Let's consider a matrix $A = [a_{11}, a_{12}; a_{21}, a_{22}]$, and its eigenvalues λ1 and λ2, and the corresponding eigenvectors $v_1 = [v_{11}; v_{21}]$ and $v_2 = [v_{12}; v_{22}]$. Then we can find a matrix $P = [v_{11}, v_{12}; v_{21}, v_{22}]$ and $D = [\lambda_1, 0; 0, \lambda_2]$ such that $A = PDP^{-1}$.

3.1.2 Diagonalization in training of a neural network:

Let's consider the Hessian matrix H, which is the matrix of second derivatives of the error function with respect to the weights. Then, we can add a small multiple of the identity matrix λI to regularize the matrix and make it invertible. The new Hessian matrix H_reg = H + λI.

3.1.3 Diagonalization in training of a linear support vector machine (SVM):

Let's consider the Gram matrix G, which is the matrix of inner products between the data points, and the regularization parameter λ. Then, we can add a small multiple of the identity matrix λI to regularize the matrix and make it invertible. The new Gram matrix $G_{reg} = G + \lambda I$.

In both examples, by adding a small multiple of the identity matrix, we are able to make the matrix invertible and improve the performance of the model. The regularization parameter λ controls the amount of regularization, and it should be chosen carefully to prevent overfitting.

How to apply Eigenvalues and eigenvectors in Python with NumPy

Eigenvalues and eigenvectors

```
#########################>> 3,1.1 << ########################
import numpy as np

# Create a matrix A
A = np.array([[3, 2], [1, 4]])
```

```
# Calculate eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(A)

# Create the diagonal matrix D
D = np.diag(eigenvalues)

# Create the matrix P
P = eigenvectors

# Confirm that A = PDP^-1
A_ = np.dot(np.dot(P, D), np.linalg.inv(P))
print(A_)
# Output:
# [[3. 2.]
#  [1. 4.]]

##########################>> 3,1.2 << ########################

import numpy as np

# Create the Hessian matrix H
H = np.array([[1, 2], [3, 4]])

# Regularization parameter
lambda_ = 0.1

# Create the identity matrix
I = np.eye(2)

# Regularize the Hessian matrix
H_reg = H + lambda_ * I

# Check that H_reg is invertible
print(np.linalg.det(H_reg))
# Output: -0.2

##########################>> 3,1.3 << ########################
import numpy as np

# Create the Gram matrix G
G = np.array([[1, 2], [3, 4]])

# Regularization parameter
lambda_ = 0.1

# Create the identity matrix
I = np.eye(2)

# Regularize the Gram matrix
G_reg = G + lambda_ * I
```



```
# Check that G_reg is invertible
print(np.linalg.det(G_reg))
# Output: -0.2
```

```
[[3. 2.]
 [1. 4.]]
-1.49
-1.49
```

> 4. Singular value decomposition: a powerful technique for factorizing a matrix into simpler components
> 5. Principal component analysis: a technique for identifying the underlying structure in high-dimensional data
> 6. Linear regression: a technique for fitting a linear model to data
> 7. Neural network: PCA, SVD, SVDD, and more

1.2. Introduction to Numerical Linear Algebra

Numerical linear algebra is a branch of mathematics that deals with the development and analysis of algorithms for solving linear algebra problems on a computer. It is concerned with the practical issues that arise when solving linear algebra problems using numerical methods, such as round-off errors, conditioning, and stability.

Numerical linear algebra algorithms are widely used in many fields such as computer science, engineering, and physics, to solve problems such as solving systems of linear equations, eigenvalue problems, and optimization problems.

Some important topics in numerical linear algebra include:

1. Gaussian elimination and LU factorization: These are methods for solving systems of linear equations, which involve transforming a matrix into an upper triangular or lower triangular form using a series of row operations.

2. Iterative methods: These are methods for solving systems of linear equations that involve iteratively approximating the solution, such as the Jacobi method, the Gauss-Seidel method, and the Conjugate Gradient method.
3. Matrix factorizations: These are methods for factoring a matrix into simpler forms, such as the LU factorization, the Cholesky factorization, and the QR factorization.
4. Eigenvalue algorithms: These are methods for finding the eigenvalues and eigenvectors of a matrix, such as the power method, the QR algorithm, and the Jacobi method.
5. Singular Value Decomposition (SVD): SVD is a factorization of a matrix into three matrices: U, S, and V. It is used in many applications such as image compression, and solving linear least squares problem.
6. Conditioning and stability: These are important concepts in numerical linear algebra that measure how sensitive a problem is to perturbations in the input data and how well-posed a problem is.

Numerical linear algebra is a challenging and active area of research, and new algorithms and techniques are constantly being developed to improve the efficiency and accuracy of solving linear algebra problems on a computer. According to Wikipedia the numerical linear algebra is define as:"Numerical linear algebra, sometimes called applied linear algebra, is the study of how matrix operations can be used to create computer algorithms which efficiently and accurately provide approximate answers to questions in continuous mathematics. It is a subfield of numerical analysis, and a type of linear algebra. Computers use floating-point arithmetic and cannot exactly represent irrational data, so when a computer algorithm is applied to a matrix of data, it can sometimes increase the difference between a number stored in the computer and the true number that it is an approximation of. Numerical linear algebra uses properties of vectors and matrices to develop computer algorithms that minimize the error introduced by the computer, and is also concerned with ensuring that the algorithm is as efficient as possible"

> In many numerical methods for solving a variety of practical computational problems is the efficient and accurate solution of linear systems

1.2.1 Gaussian elimination, LU and QR Decomposition

----------Examples & Exceries----------

Example of Gaussian elimination using Python

Gaussian elimination is a method for solving systems of linear equations by transforming a matrix into an upper triangular form using a series of row operations. Here's an example of how to perform Gaussian elimination in Python:

```
import numpy as np

# Define the matrix A and the right-hand side b
A = np.array([[3, 1, -1], [1, 4, 1], [-1, 1, 5]])
b = np.array([10, 14, 16])

# Perform Gaussian elimination
n = len(b)
for i in range(n-1):
    for j in range(i+1, n):
        factor = A[j,i] / A[i,i]
        for k in range(i, n):
            A[j,k] = A[j,k] - factor * A[i,k]
        b[j] = b[j] - factor * b[i]

# Perform back substitution to solve for the unknowns
x = np.zeros(n)
x[n-1] = b[n-1] / A[n-1, n-1]
for i in range(n-2, -1, -1):
    x[i] = (b[i] - np.dot(A[i,i+1:], x[i+1:]))/A[i,i]
print(x)
```

```
[4.44444444 1.66666667 5.        ]
```

In this example, the matrix A is a 3x3 matrix and the right-hand side b is a 3x1 vector. The code performs Gaussian elimination by iterating through the rows of the matrix and using row operations to transform the matrix into an upper triangular form. The result of the elimination is a matrix in upper triangular form, this matrix is then used to perform back substitution to solve for the unknowns.

You can also use libraries such as scipy and numpy which provide built-in functions for solving systems of linear equations, such as scipy.linalg.solve() and numpy.linalg.solve().

Please keep in mind that, Gaussian elimination is sensitive to the choice of pivot and the accuracy of the computation, and other methods such as LU decomposition or QR decomposition might be more robust and accurate.

Example for LU decomposition using Python code

LU decomposition and QR decomposition are methods for factoring a matrix into simpler forms, which can be used to solve systems of linear equations. Here's an example of how to perform LU decomposition in python using the scipy library:

```
import numpy as np
from scipy.linalg import lu

# Define the matrix A and the right-hand side b
A = np.array([[3, 1, -1], [1, 4, 1], [-1, 1, 5]])
b = np.array([10, 14, 16])

# Perform LU decomposition
P, L, U = lu(A)

# Solve for the unknowns using LU decomposition
y = np.linalg.solve(L, P @ b)
x = np.linalg.solve(U, y)
print(x)
```

```
[4.04347826 1.56521739 3.69565217]
```


Example QR decomposition using Python Code

You can use the numpy library to perform QR decomposition on a matrix. Here's an example of how to perform QR decomposition in Python using the numpy library:

```
import numpy as np

# Define the matrix A and the right-hand side b
A = np.array([[3, 1, -1], [1, 4, 1], [-1, 1, 5]])
b = np.array([10, 14, 16])

# Perform QR decomposition
Q, R = np.linalg.qr(A)

# Solve for the unknowns using QR decomposition
y = np.linalg.solve(R, np.matmul(Q.T, b))
x = np.linalg.solve(R, y)
print(x)
```

```
[ 0.1016259  -1.22843879  1.02256463]
```

In this example, the numpy function numpy.linalg.qr(A) is used to perform the QR decomposition of the matrix A into an orthonormal matrix Q and an upper triangular matrix R.

QR decomposition is a versatile and numerically stable method that can be used to solve systems of linear equations, find the eigenvalues of a matrix, and minimize the least squares of a matrix, among other things.

1.2.2 Iterative Methods

Iterative methods are a type of algorithm that are used to solve systems of linear equations by iteratively approximating the solution. These methods start with an initial guess for the solution and then use a set of rules to improve the approximation in each iteration until it reaches a desired level of accuracy. Some examples of iterative methods include:

1. Jacobi method: This method updates the solution by using the values from the previous iteration to calculate the new values for each unknown.
2. Gauss-Seidel method: This method is similar to Jacobi method, but it uses the updated values from the current iteration to calculate the new values for each unknown.
3. Conjugate Gradient method: This method is used to solve systems of equations where the matrix is symmetric and positive definite. It uses a combination of gradient descent and conjugacy to find the solution.

Example of Jacobi Method using Python

Here is an example of how to use the Jacobi method to solve a system of linear equations in python:

```
import numpy as np

# Define the matrix A and the right-hand side b
A = np.array([[10, 2, 1], [1, 5, 1], [2, 3, 10]])
b = np.array([7, -8, 6])

# Initial guess for the solution
x = np.array([1, 1, 1])

# Define the tolerance and the maximum number of iterations
tolerance = 1e-5
max_iterations = 100

# Iterate until the solution is found or the maximum number of iterations is reached
for i in range(max_iterations):
    x_new = np.zeros_like(x)
    for j in range(len(x)):
        s = np.dot(A[j, :j], x_new[:j]) + np.dot(A[j, j+1:], x[j+1:])
        x_new[j] = (b[j] - s) / A[j, j]
    if np.linalg.norm(x - x_new) < tolerance:
        break
    x = x_new

print(x)
```

```
[ 0 -1  0]
```


Example of Gauss-Seidel method using Python

here is an example of how to use the Gauss-Seidel method to solve a system of linear equations in python:

```
import numpy as np

# Define the matrix A and the right-hand side b
A = np.array([[10, 2, 1], [1, 5, 1], [2, 3, 10]])
b = np.array([7, -8, 6])

# Initial guess for the solution
x = np.array([1, 1, 1])

# Define the tolerance and the maximum number of iterations
tolerance = 1e-5
max_iterations = 100

# Iterate until the solution is found or the maximum number of iterations is reached
for i in range(max_iterations):
    x_new = np.zeros_like(x)
    for j in range(len(x)):
        s = np.dot(A[j, :j], x_new[:j]) + np.dot(A[j, j+1:], x[j+1:])
        x_new[j] = (b[j] - s) / A[j, j]
    if np.linalg.norm(x - x_new) < tolerance:
        break
    x = x_new

print(x)
```

```
[ 0 -1  0]
```

The difference between the Gauss-Seidel method and the Jacobi method is that in the Gauss-Seidel method, we use the updated values from the current iteration to calculate the new values for each unknown while in Jacobi method we use the values from the previous iteration.

It's worth noting that Gauss-Seidel method is sometimes faster than Jacobi method, but it is not always guaranteed to converge, so it is important to choose the right method depending on the problem you are trying to solve.

It's also worth noting that the Gauss-Seidel method is a special case of the Successive Over-Relaxation (SOR) method. In SOR, a relaxation factor is introduced to adjust the balance between the old and new solutions, to improve the rate of convergence

Example of Conjugate Gradient method using Python code

The Conjugate Gradient method is an optimization algorithm that can be used to solve systems of linear equations where the matrix is symmetric and positive definite. It starts with an initial guess for the solution and uses the gradient of the residual as the initial search direction. Then, in each iteration, it calculates a new search direction that is conjugate to the previous search direction and uses it to update the solution.

It's worth noting that the Conjugate Gradient method is a very efficient method that requires a small number of iterations to converge, but it is sensitive to the initial guess and the accuracy of the computation. It is also used in many other optimization problems such as minimizing least squares and eigenvalue problems.

```
import numpy as np

# Define the matrix A and the right-hand side b
A = np.array([[4, 1], [1, 3]])
b = np.array([1, 2])

# Initial guess for the solution
x = np.array([0, 0])
x = x.astype(float)

# Define the tolerance and the maximum number of iterations
tolerance = 1e-5
max_iterations = 100

# Initialize the search direction
r = b - np.dot(A, x)
p = r.copy()

# Iterate until the solution is found or the maximum number of iterations is reached
for i in range(max_iterations):
```



```
    Ap = np.dot(A, p)
    alpha = np.dot(r, r) / np.dot(p, Ap)
    x += alpha * p
    r_new = r - alpha * Ap
    if np.linalg.norm(r_new) < tolerance:
        break
    beta = np.dot(r_new, r_new) / np.dot(r, r)
    p = r_new + beta * p
    r = r_new

print(x)
```

```
[0.09090909 0.63636364]
```

In this example, we are solving the system of equations represented by the matrix A and the right-hand side b. The initial guess for the solution is [0, 0], the tolerance is set to 1e-5, and the maximum number of iterations is set to 100.

As you can see, the Conjugate Gradient method is a very efficient algorithm that can be used to solve systems of linear equations where the matrix is symmetric and positive definite, this method can be applied to different problems such as least squares, eigenvalue problems and optimization problems.

1.2.3 Matrix factorization.

Matrix factorization is a technique for expressing a given matrix as a product of two or more matrices. This can be used to simplify the matrix and make it easier to work with, as well as to reveal important properties of the matrix that may not be immediately obvious.

There are several types of matrix factorizations, some of the most common ones include:

1. LU decomposition: This factorization expresses a matrix as the product of a lower triangular matrix and an upper triangular matrix. It can be used to solve systems of linear equations and to calculate the determinant of a matrix.

2. QR decomposition: This factorization expresses a matrix as the product of an orthonormal matrix and an upper triangular matrix. It can be used to solve systems of linear equations, find the eigenvalues of a matrix, and minimize the least squares of a matrix, among other things.
3. Cholesky decomposition: This factorization expresses a matrix as the product of a lower triangular matrix and its transpose. It can be used to solve systems of linear equations and to calculate the determinant of a matrix, when the matrix is symmetric and positive definite.
4. Eigenvalue Decomposition: This factorization expresses a matrix as the product of a diagonal matrix and a matrix whose columns are the eigenvectors of the original matrix. It can be used to find the eigenvalues and eigenvectors of a matrix, and also to diagonalize a matrix.
5. Singular Value Decomposition (SVD): This factorization expresses a matrix as the product of three matrices: a unitary matrix, a diagonal matrix, and another unitary matrix. It can be used to solve systems of linear equations, find the rank of a matrix, and to perform principal component analysis (PCA) and dimensionality reduction, among other things.
6. LU-SVD decomposition : It is a combination of LU and SVD decomposition, it can be used in various applications such as image compression, image restoration, and solving linear least squares problems.

Each of these factorizations has its own advantages and disadvantages, depending on the specific problem and the properties of the matrix.

1.2.4 Applications of Linear Algebra

The applications of linear algebra in many domains but here we are presented the classical book of Gilbert Strang applications

- Matrices of Engineering
- Graphs and Networks

- Markov matrices, Population and Economics
- Linear Programming
- Fourier Series
- Least square problems
- Statistics and Probability
- Computer Graph

Example of Conjugate Gradient using python code

Here is an example of how to use the Conjugate Gradient method to solve a system of linear equations in python:

1. Matrices in Engineering:

Structural analysis: Matrices are used to model the behavior of structures, such as bridges or buildings, under various loads and conditions. Electrical circuit analysis: Matrices can be used to analyze electrical circuits, by representing the circuit equations in matrix form. Robotics: Matrices are used to model the motion and control of robots, as well as to represent the geometric transformations involved in robot movement.

2. Graphs and Networks:

Social networks: Matrices can be used to represent social networks, where each entry in the matrix represents a connection between two individuals. Transportation networks: Matrices can be used to model transportation networks, such as roads or railways, where each entry represents the distance or travel time between two locations. Internet search algorithms: Matrices are used in algorithms such as PageRank, which determines the importance of web pages based on their connections to other pages.

3. Markov matrices, Population, and Economics:

Population dynamics: Matrices can be used to model population growth and migration, where each entry represents the probability of a person moving from

one location to another. Economic models: Matrices can be used to model economic systems, such as input-output models or Leontief models, which represent the flow of goods and services between different sectors of the economy. Markov chains: Matrices are used to represent Markov chains, which are used to model stochastic processes such as random walks or queuing systems.

4. Linear Programming:

Optimization: Linear programming is used to solve optimization problems, such as minimizing costs or maximizing profits, subject to linear constraints. Resource allocation: Linear programming is used to allocate resources, such as labor or raw materials, in an efficient and optimal way. Portfolio optimization: Linear programming is used in finance to optimize investment portfolios, by selecting the right mix of assets to achieve a desired risk-return profile.

5. Fourier Series:

Signal processing: Fourier series are used in signal processing, to represent signals as a sum of sine and cosine waves of different frequencies. Image processing: Fourier series are used in image processing, to represent images as a sum of sine and cosine waves of different frequencies. Quantum mechanics: Fourier series are used in quantum mechanics, to represent wave functions as a sum of sine and cosine waves of different frequencies.

6. Least Square Problems:

Curve fitting: Least square problems are used to fit curves to data, by finding the line or curve that minimizes the sum of the squared distances between the data points and the curve. Regression analysis: Least square problems are used in regression analysis, to model the relationship between two variables and to estimate the parameters of the model. Data analysis: Least square problems are used in data analysis, to extract trends and patterns from noisy data.

7. Statistics and Probability:

Principal component analysis: Matrices are used in principal component analysis, to reduce the dimensionality of data and to identify the most important variables. Linear regression: Matrices are used in linear regression, to model the relationship between two variables and to estimate the parameters of the model. Probability theory: Matrices are used in probability theory, to represent probability distributions and to perform calculations involving probabilities.

8. Computer Graphics:

3D graphics: Matrices are used in 3D graphics, to represent and manipulate 3D objects and their transformations, such as translation, rotation, and scaling. Computer vision: Matrices are used in computer vision, to represent and process images and to perform tasks such as object recognition and tracking. Animation: Matrices are used in animation, to simulate the motion of objects and characters in a virtual environment

```
import networkx as nx
import numpy as np

# Social network example
# Generate random social network
G = nx.erdos_renyi_graph(100, 0.1)
A = nx.adjacency_matrix(G).toarray()

# Transportation network example
# Create distance matrix for a set of locations
locations = np.random.rand(10, 2)  # 10 locations with 2 coordinates each
distances = np.zeros((10, 10))
for i in range(10):
    for j in range(i+1, 10):
        distances[i, j] = np.linalg.norm(locations[i] - locations[j])
        distances[j, i] = distances[i, j]

# Internet search algorithm example
# Calculate PageRank scores for a set of web pages
adj_matrix = np.random.randint(0, 2, (100, 100))
D = np.diag(1/np.sum(adj_matrix, axis=1))
P = D.dot(adj_matrix)
alpha = 0.85  # damping factor
N = len(adj_matrix)
v = np.ones(N)/N
for i in range(10):
```



```
    v = alpha*P.dot(v) + (1-alpha)/N*np.ones(N)
```

..... Chapter 2 .....

.....Linear Algebra and Machine learning.....

Linear Algebra is a branch of mathematics that deals with linear equations, matrices, and vector spaces. In Machine Learning, it is used extensively for various tasks such as data preprocessing, feature extraction, model training, and model evaluation. Linear Algebra is used in applications such as linear regression, principal component analysis, singular value decomposition, and deep learning. It is a fundamental tool for Machine Learning and is essential for anyone interested in this field to have a solid understanding of Linear Algebra.

Sure, here are a few advanced examples of how linear algebra is used in machine learning:

2.1 Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique that is widely used in machine learning to reduce the number of features in a dataset. The basic idea behind PCA is to find a lower-dimensional representation of the data that captures as much of the variation in the original data as possible.

PCA works by computing the eigenvectors and eigenvalues of the covariance matrix of the data. The eigenvectors correspond to the principal components of the data, and the eigenvalues correspond to the amount of variance captured by each principal component. By selecting the top k eigenvectors with the highest eigenvalues, we can obtain a lower-dimensional representation of the data with k features.

2.2 Linear Regression

Linear regression is a supervised learning algorithm that is used to predict a continuous output variable based on one or more input variables. In linear regression, we assume that there is a linear relationship between the input variables and the output variable.

We can represent the linear relationship between the input variables and the output variable using a linear model, which takes the form y = w0 + w1x1 + w2x2 + ... + wnxn, where y is the output variable, xi is the i-th input variable, and wi is the weight or coefficient associated with xi.

We can represent the linear model as a matrix multiplication, where the weights are represented as a vector w and the input variables are represented as a matrix X. The linear model can then be written as y = Xw.

2.2 Neural Networks

Neural networks are a type of machine learning algorithm that are loosely inspired by the structure and function of the brain. Neural networks are composed of layers of artificial neurons, which are connected by weighted edges. Each neuron in a neural network computes a weighted sum of its inputs and applies an activation function to the result.

Neural networks are trained using an optimization algorithm called backpropagation, which uses the chain rule of calculus to compute the gradient of the loss function with respect to the weights. The gradient is then used to update the weights using an optimization algorithm such as stochastic gradient descent.

The computation in a neural network can be represented as a series of matrix multiplications and activation functions, where the weights and biases are represented as matrices and vectors, and the inputs and outputs are represented as vectors.

2.3 Linear Algebra and Statistics

The machine learning concerned, there are some clear fingerprints of linear algebra on statistic and statistical methods include:

- Multivariate statistics, where use vector and matrix notaion

- Linear regression, the methods which are use to fine the solution such as least squares and weighted least squares
- Estimates of mean and variance, from data matrices
- Multinomial Gaussian distributions, where the covariance matrix paly an importance role.

Linear algebra plays an important role in statistics, especially in multivariate statistics and linear regression. Here are some more details:

2.3.1 Multivariate statistics:

In multivariate statistics, we often work with vectors and matrices to represent data. For example, we may have a dataset of observations where each observation is represented by a vector of measurements. We can use linear algebra operations such as matrix multiplication, eigenvalue decomposition, and singular value decomposition to analyze this data and extract useful information.

2.3.2 Linear regression:

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. In its simplest form, linear regression assumes a linear relationship between the variables, and seeks to find the line that best fits the data. This can be done using methods such as least squares and weighted least squares, which involve solving systems of linear equations.

2.3.3 Estimates of mean and variance:

In statistics, we often use matrices to represent data and calculate estimates of mean and variance. For example, if we have a dataset of observations, we can represent it as a matrix where each row corresponds to an observation and each column corresponds to a variable. We can then use linear algebra operations to calculate the mean and variance of each variable.

2.3.4 Multinomial Gaussian distributions:

In statistics, the Multinomial Gaussian distribution is a probability distribution used to model multivariate data. The covariance matrix plays an important role

in this distribution, as it determines the shape of the distribution and the correlations between the variables. Linear algebra operations such as matrix inversion and eigenvalue decomposition can be used to analyze this distribution and calculate probabilities.

..... Chapter 3 .....

.....**Examlpes of Linear Algebra in Meachine Learning**.....

Regression Ananlysis

There are Saven types of Regression models

- Linear regression
- Logistic Regression
- ploynomaial Regression
- Stepwise Regression
- Ridge Regression
- Lasso Regression
- ElasticNet Regression

Linear Regression (LR):

LR is used to predict the relationship between two variables by applying a linear equation to observed data. There are two types of variable, one variable is called an independent variable, and the other is a dependent variable. Linear regression is commonly used for predictive analysis.

In the linear regression model the "goal" is to estimate the function $f(x_i, \beta)$ that is best fits the given data. most of researchers preferred statistical model, to estimate the parameters β

$h(x) = w * x + b$

here, b is the bias.

x represents the feature vector

w represents the weight vector.

..... Chapter 4

.....

.....**Introduction Numpy Basic Library in Python**.....

Hand on python code Using NumPy

This first Section we have to covered:

- NumPy N-dimensional Array
- Function to create to Array
- Combining Array

NumPy N-dimensional Array

```
# create an array
from numpy import array
A=[1.0, 2.0, 3.0]
a=array(A);
# display array
print(a)
# Display array shape
```



```
print(a.shape)
# Display array data type
print(a.dtype)
```

```
[1. 2. 3.]
(3,)
float64
```

Function to Create arrays

1. Empty
2. Zeros
3. Ones

```
# create empty array
from numpy import empty
a= empty([3,3])
# display
print(a)
```

```
[[4.64722810e-310 6.79038653e-313 2.33419537e-312]
 [2.14321575e-312 8.48798317e-313 1.08221785e-312]
 [1.12465777e-312 8.70018275e-313 0.00000000e+000]]
```

```
# Create zeros array
from numpy import zeros
a = zeros([3,5])
# Display
print(a)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

```
# Create onces array
from numpy import ones
a = ones([3,5])
# Dispaly
print(a)
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

Combining Arrays

> 1. vertical stack
> 2. Horizontal Stack

```
# Create array with vstack
from numpy import array
from numpy import vstack
# Create first array
a = array ([1,2,3])
print(a)
# create second array
b= array([4,5,6])
print(b)
# create third array
c= vstack([a,b])
print(c)
# size of the array
print(c.shape)
```

```
[1 2 3]
[4 5 6]
[[1 2 3]
 [4 5 6]]
(2, 3)
```

```
# Create array with hstack
from numpy import array
from numpy import hstack
# create first array
a= array([1,2,3])
# display array
print(a)
# create second array
b= array([4,5,6])
# display array
print(b)
# create third array
c = hstack([a,b])
# display
print(c)
# size of array
print(c.shape)
```

```
[1 2 3]
[4 5 6]
```