Welcome to the CoGrammar Linear Algebra

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

AAAAAHHHH! Math! 🥺

This is the most "complicated" lecture of all CIW lectures, since it contains the most math. So why even bother learning this?

- In the competitive world of tech interviews, a deep understanding of linear algebra **sets you apart from the crowd.** Interviewers seek candidates who can **solve complex problems, optimize algorithms, and demonstrate strong mathematical intuition.** By understanding linear algebra, you'll:

AAAAAHHHH! Math! 🥺

❖ **Impress Interviewers**
  - ➢ Showcase your ability to apply theoretical concepts to real-world problems
  - ➢ Demonstrate your problem-solving skills and attention to detail
  - ➢ Stand out as a **well-rounded, knowledgeable candidate**

AAAAAHHHH! Math! 🥺

- **Tackle Interview Questions with Confidence**
  - **Solve coding challenges** that involve matrices, vectors, and linear transformations
  - **Optimize algorithms and data structures** using eigenvalues and eigenvectors
  - Discuss the mathematical foundations of machine learning and data analysis (for DS students, although most need to understand AI in modern times)

AAAAAHHHH! Math! 🥺

- **Communicate Effectively**
  - **Explain complex ideas** clearly and concisely
  - Collaborate with interviewers to break down problems and explore solutions
  - Demonstrate your ability to **think critically and analytically**

AAAAAHHHH! Math! 🥺

- There is no exam on the other end...
- So it's OK to enjoy this lecture and have a bit of fun 🙂

Coding Interview Workshop Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions]

CoGrammar

Coding Interview Workshop Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: [Feedback on Lectures]()

Skills Bootcamp
8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

CoGrammar

Lecture Objectives

- Define **vector spaces** and understand their properties including **basis and dimension**
- Perform **basic matrix operations** and understand their applications in data transformation
- Explain the **significance of eigenvalues and eigenvectors** in dimensionality reduction techniques

CoGrammar

Lecture Objectives

- Describe **linear transformations** and their matrix representations
- Apply methods to **solve systems of linear equations** relevant to real-world data processing and software engineering problems
- Implement key linear algebra concepts in **Python and JavaScript** code examples

CoGrammar

Which of the following is NOT a property of a vector space?

A. Closure under vector addition

B. Existence of an inverse element for every vector

C. Commutativity of vector multiplication

D. Distributivity of scalar multiplication over vector addition

CoGrammar

Answer: C

- Vector spaces have several important properties:
  - Closure under vector addition and scalar multiplication
  - Associativity of vector addition and scalar multiplication
  - Existence of a zero vector and an inverse for each vector
  - Commutativity of **vector addition**
  - Distributivity of scalar multiplication over vector addition

Answer: C

- However, vector multiplication is not always commutative.
  - For example, in 3D space, the cross product of vectors is not commutative: $u \times v \neq v \times u$.

CoGrammar

Answer: C

- The other options are all valid properties of vector spaces:
  - Closure: If u and v are in the space, then u + v and cu (for any scalar c) are also in the space
  - Inverse: For every vector v, there exists a vector -v such that v + (-v) = 0
  - Distributivity: c(u + v) = cu + cv and (c + d)u = cu + du for all vectors u, v and scalars c, d

What is the time complexity of multiplying two n × n matrices using the standard algorithm?

A. O(n)

B. $O(n^2)$

C. $O(n^3)$

D. O(n log n)

```
def matrix_multiply(A, B):
    n = len(A)
    C = [[0] * n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]

    return C
```

CoGrammar

Answer: C

- We loop over n rows, n columns, and n elements.
- The total number of operations is therefore $n * n * n = n^3$.
- This gives a complexity of $O(n^3)$.
- More efficient algorithms like Strassen's algorithm or Coppersmith-Winograd can reduce the exponent below 3, but are more complex to implement and only faster for very large matrices

What does the following Python code compute for the matrix A?

```
import numpy as np

A = np.array([[1,2,3], [4,5,6], [7,8,9]])
_, eigenvectors = np.linalg.eig(A)
print(eigenvectors)
```

A. Eigenvalues of A

B. Transpose of A

C. Inverse of A

D. Eigenvectors of A

Answer: D

- The np.linalg.eig function in NumPy computes both the eigenvalues and eigenvectors of a square matrix.
- It returns two values:
  - The first is a 1D array of eigenvalues
  - The second is a 2D array where each column is an eigenvector corresponding to an eigenvalue
- In the given code, the eigenvalues are discarded (assigned to _) and only the eigenvectors are printed.

Recap and Review of Portfolio Assignments

CoGrammar

The Importance of Linear Algebra

- Linear algebra is a fundamental mathematical tool used in various fields, including data science, web development, and software engineering.
- It provides a **framework for working with vectors, matrices, and linear transformations,** which are essential for solving complex problems and building efficient algorithms.

CoGrammar

Linear Algebra

1. Vector Spaces
2. Matrix Operations
3. Eigenvalues & Eigenvectors
4. Linear Transformations
5. Systems of Linear Equations

Vector Spaces

A collection of objects called vectors, which can be added together and multiplied by scalars (real or complex numbers)

- Vector spaces satisfy the following properties:
  - Closure under addition and scalar multiplication: If u and v are vectors in the space, then u + v and cu (for any scalar c) are also in the space.
  - Associativity and commutativity of addition: (u + v) + w = u + (v + w) and u + v = v + u for all vectors u, v, w.

CoGrammar

Vector Spaces

- ➢ Existence of identity element (zero vector): There exists a unique vector 0 such that v + 0 = v for all vectors v.
- ➢ Existence of inverse elements: For every vector v, there exists a unique vector -v such that v + (-v) = 0.
- ➢ Distributivity of scalar multiplication over addition: c(u + v) = cu + cv and (c + d)u = cu + du for all vectors u, v and scalars c, d.

CoGrammar

```python
import numpy as np

# 2D vector space
v1 = np.array([1, 2])
v2 = np.array([3, 4])

# Vector addition
v3 = v1 + v2
print(v3)  # Output: [4, 6]

# Scalar multiplication
v4 = 2 * v1
print(v4)  # Output: [2, 4]
```

```javascript
// 3D vector space
const v1 = [1, 2, 3];
const v2 = [4, 5, 6];

// Vector addition
const v3 = v1.map((val, i) => val + v2[i]);
console.log(v3); // Output: [5, 7, 9]

// Scalar multiplication
const v4 = v1.map((val) => 2 * val);
console.log(v4); // Output: [2, 4, 6]
```

Linear Independence

- A set of vectors $\{v_1, v_2, \ldots, v_n\}$ is linearly independent if the equation $c_1v_1 + c_2v_2 + \ldots + c_nv_n = 0$ has only the trivial solution $c_1 = c_2 = \ldots = c_n = 0$.
- In other words, no vector in the set can be expressed as a linear combination of the others.
- Example: The standard basis vectors in R^n (e.g., (1, 0, ..., 0), (0, 1, ..., 0), ..., (0, 0, ..., 1)) are linearly independent.

CoGrammar

Linear Independence

$$\left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \checkmark$$

$$c_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + c_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0$$

$$c_1 + 0 = 0 \Rightarrow c_1 = 0$$

$$0 + c_2 = 0 \Rightarrow c_2 = 0$$

$$\left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right\} \times$$

$$c_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + c_2 \begin{bmatrix} 2 \\ 0 \end{bmatrix} = 0$$

$$c_1 + 2c_2 = 0$$

$$\Rightarrow c_1 = -2c_2$$

CoGrammar

Span

- The span of a set of vectors $\{v_1, v_2, \ldots, v_n\}$ is the set of all linear combinations of these vectors.
- In other words, it is the set of all vectors that can be obtained by multiplying the vectors by scalars and adding them together.
- Example: The span of $\{(1, 0), (0, 1)\}$ is the entire 2D space R^2, as any vector (x, y) can be expressed as $x(1, 0) + y(0, 1)$.

CoGrammar

Span

$$\left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$

∴ Entire spans entire 2D space

$$2\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \text{ and } 2\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

... take this for any number n

$$n\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} n \\ 0 \end{bmatrix} \text{ and } n\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ n \end{bmatrix}$$

⇒ add them in any way

$$\begin{bmatrix} n \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ k \end{bmatrix} = \begin{bmatrix} n \\ k \end{bmatrix}$$ where n & k are any number

Basis and Dimension

- A basis of a vector space V is a linearly independent set of vectors that spans V.
- The dimension of a vector space is the number of vectors in its basis.
- Example: The standard basis {(1, 0, ..., 0), (0, 1, ..., 0), ..., (0, 0, ..., 1)} is a basis for R^n, and the dimension of R^n is n.

CoGrammar

Basis and Dimension

$\left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\} \rightarrow$ vector space V

1. We've seen this set is linearly independent.
2. We've seen it spans all 2D real numbers.

$\therefore$ Basis of V include $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$\therefore$ Dimension = 2

Basis and Dimension

```
is_linearly_independent(V)
    count rank of matrix
    if rank = nr of vectors in V
        linearly independent
    else
        not
```

```python
def is_linearly_independent(vectors):
    matrix = np.column_stack(vectors)
    rank = np.linalg.matrix_rank(matrix)
    return rank == len(vectors)

# Example usage
v1 = np.array([1, 0])
v2 = np.array([0, 1])
v3 = np.array([2, 2])

vectors = [v1, v2, v3]
print(is_linearly_independent(vectors))  # Output: False
```

```javascript
function isLinearlyIndependent(vectors) {
  const matrix = vectors[0].map((_, colIndex) =>
    vectors.map((row) => row[colIndex])
  );
  const rank = matrix.length;

  for (let i = 0; i < rank; i++) {
    if (matrix[i][i] !== 0) {
      for (let j = 0; j < vectors.length; j++) {
        if (j !== i) {
          const ratio = matrix[j][i] / matrix[i][i];
          for (let k = 0; k < rank; k++) {
            matrix[j][k] -= ratio * matrix[i][k];
          }
        }
      }
    } else {
      return false;
    }
  }
  return true;
}
```

BTW when asked the advantage of Python over other languages it’s readability and abstracting complexity (Exhibit A 🙂👆)

CoGrammar

Matrix Operations

❖ A matrix is a rectangular array of numbers arranged in rows and columns.

Matrix Operations

- Matrices are used to represent linear transformations, solve systems of linear equations, and analyze data in various fields:
  - Data Science: Feature extraction, dimensionality reduction, recommendation systems
  - Web Development: CSS transformations, 3D graphics, image processing
  - Software Engineering: Optimization problems, graph algorithms, computer vision

CoGrammar

Matrix Addition and Subtraction

- Matrix addition and subtraction are performed element-wise, i.e., by adding or subtracting corresponding elements of two matrices with the same dimensions.

Matrix Addition and Subtraction

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \pm \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a \pm e & c \pm f \\ b \pm g & d \pm h \end{bmatrix}$$

```python
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Matrix addition
C = A + B
print(C)
# Output:
# [[6 8]
#  [10 12]]

# Matrix subtraction
D = A - B
print(D)
# Output:
# [[-4 -4]
#  [-4 -4]]
```

```javascript
const A = [[1, 2], [3, 4]];
const B = [[5, 6], [7, 8]];

// Matrix addition
const C = A.map((row, i) => row.map((val, j) => val + B[i][j]));
console.log(C);
// Output: [[6, 8], [10, 12]]

// Matrix subtraction
const D = A.map((row, i) => row.map((val, j) => val - B[i][j]));
console.log(D);
// Output: [[-4, -4], [-4, -4]]
```

Scalar Multiplication and Matrix Multiplication

- Scalar multiplication: Multiplying each element of a matrix by a scalar value.
- Matrix multiplication: Multiplying two matrices A ($m \times n$) and B ($n \times p$) to obtain a matrix C ($m \times p$).
  - To multiply matrices A and B, the number of columns in A must equal the number of rows in B.
  - Each element C[i][j] is the dot product of the i-th row of A and the j-th column of B.

CoGrammar

Scalar Multiplication and Matrix Multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times e = \begin{bmatrix} a \times e & b \times e \\ c \times e & d \times e \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a \times e + b \times g & a \times f + b \times h \\ c \times e + d \times g & c \times f + d \times h \end{bmatrix}$$

CoGrammar

```python
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Scalar multiplication
C = 2 * A
print(C)
# Output:
# [[2 4]
#  [6 8]]

# Matrix multiplication
D = np.dot(A, B)
print(D)
# Output:
# [[19 22]
#  [43 50]]
```

```javascript
const A = [[1, 2], [3, 4]];
const B = [[5, 6], [7, 8]];

// Scalar multiplication
const C = A.map(row => row.map(val => 2 * val));
console.log(C);
// Output: [[2, 4], [6, 8]]

// Matrix multiplication
const D = A.map((row, i) =>
    B[0].map((_, j) =>
        row.reduce((sum, val, k) => sum + val * B[k][j], 0)
    )
);
console.log(D);
// Output: [[19, 22], [43, 50]]
```

Interview-Style Question - Matrix Operations

❖ Given two matrices A and B, write a function to compute the matrix product AB. If the dimensions are not compatible for multiplication, return an error message.

Interview-Style Question - Matrix Operations

```
matrix_multiply(A, B):

  if nr of cols of A ≠ nr of rows of B:
      invalid operation

  else
      dot product of A & B
```

CoGrammar

```python
def matrix_multiply(A, B):
    if A.shape[1] != B.shape[0]:
        return "Error: Incompatible dimensions for matrix multiplication"

    return np.dot(A, B)
```

```javascript
function matrixMultiply(A, B) {
    if (A[0].length !== B.length) {
        return "Error: Incompatible dimensions for matrix multiplication";
    }

    return A.map((row, i) =>
        B[0].map((_, j) =>
            row.reduce((sum, val, k) => sum + val * B[k][j], 0)
        )
    );
}
```

CoGrammar

Let’s Breathe!

Let’s take a small break before moving on to the next topic.

CoGrammar

Eigenvalues and Eigenvectors

- For a square matrix A, a non-zero vector v is an eigenvector of A if $Av = \lambda v$ for some scalar λ, which is called the eigenvalue corresponding to v.
- Eigenvalues and eigenvectors help understand the behavior of a matrix when it is transformed, such as scaling, rotation, or reflection.

CoGrammar

$$A = \begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix}$$

$Ax = \lambda x \Rightarrow Ax - \lambda x = 0 \Rightarrow (A - \lambda)x = 0$

with some proofs we get $\det(A - \lambda I) = 0$

$$\det(A - I\lambda) = \det\left(\begin{bmatrix} -5-\lambda & 2 \\ -7 & 4-\lambda \end{bmatrix}\right) = 0$$

$$(-5-\lambda)(4-\lambda) - 2(-7) = 0$$

$$-20 + \lambda + \lambda^2 + 14 = 0$$

$$\lambda^2 + \lambda - 6 = 0$$

$$(\lambda - 2)(\lambda + 3) = 0$$

$$\lambda = -3 \text{ or } 2$$

$$\begin{array}{cc|c} 1 & 2 & -2 \\ 1 & 3 & +3 \\ \hline & & 1 \end{array}$$

CoGrammar

Case $\lambda = -3$

$$(A + 3I)x = 0$$

$$\begin{bmatrix} -5+3 & 2 \\ -7 & 4+3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} -2 & 2 \\ -7 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\left[\begin{array}{cc|c} -2 & 2 & 0 \\ -7 & 7 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{cc|c} -1 & 1 & 0 \\ -1 & 1 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{cc|c} -1 & 1 & 0 \\ 0 & 0 & 0 \end{array}\right]$$

this gives $-x_1 + x_2 = 0 \Rightarrow x_1 = x_2$ so $v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

CoGrammar

Case $\lambda = 2$

$$(A - 2I)x = 0$$

$$\left[\begin{array}{cc|c} -5-2 & 2 & 0 \\ -7 & 4-2 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{cc|c} -7 & 2 & 0 \\ -7 & 2 & 0 \end{array}\right]$$

$$\Rightarrow \left[\begin{array}{cc|c} -7 & 2 & 0 \\ 0 & 0 & 0 \end{array}\right]$$

$$-7x_1 = -2x_2$$

$$x_1 = \frac{2}{7}x_2$$

$$\therefore \begin{bmatrix} 1 \\ 2/7 \end{bmatrix} = V_2$$

Thus for $\begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix}$ we have

eigenvalues -3 and 2 and

eigenvectors $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 2/7 \end{bmatrix}$

Eigenvalues and Eigenvectors

❖ They have applications in various fields, such as:

- ➢ Data Science: Principal Component Analysis (PCA) for dimensionality reduction
- ➢ Computer Graphics: Transformations and animations
- ➢ Physics: Vibration analysis and quantum mechanics

```python
A = np.array([[2, 1], [1, 2]])

eigenvalues, eigenvectors = np.linalg.eig(A)

print("Eigenvalues:")
print(eigenvalues)
# Output: [3. 1.]

print("Eigenvectors:")
print(eigenvectors)
# Output:
# [[ 0.70710678 -0.70710678]
#  [ 0.70710678  0.70710678]]
```

```javascript
const math = require('mathjs');

const A = math.matrix([[2, 1], [1, 2]]);

const {values, vectors} = math.eig(A);

console.log("Eigenvalues:");
console.log(values);
// Output: [3, 1]

console.log("Eigenvectors:");
console.log(vectors);
// Output:
// [
//    [0.7071067811865475, -0.7071067811865475],
//    [0.7071067811865475, 0.7071067811865475]
// ]
```

CoGrammar

Application - Principal Component Analysis (PCA)

❖ PCA is a dimensionality reduction technique that uses eigenvalues and eigenvectors to identify the principal components of a dataset.

Application - Principal Component Analysis (PCA)

- Steps:
  - Standardize the data (zero mean and unit variance)
  - Compute the covariance matrix
  - Find the eigenvalues and eigenvectors of the covariance matrix
  - Sort the eigenvectors by their corresponding eigenvalues in descending order
  - Choose the top k eigenvectors to form a projection matrix
  - Transform the original data using the projection matrix

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Generate sample data
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])

# Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=1)
X_pca = pca.fit_transform(X_std)
```

```python
print("Original Data:")
print(X)
# Output:
# [[1 2]
#  [3 4]
#  [5 6]
#  [7 8]]

print("PCA-transformed Data:")
print(X_pca)
# Output:
# [[-1.8973666 ]
# [-0.63245553]
# [ 0.63245553]
# [ 1.8973666 ]]
```

```
const PCA = require('ml-pca');

// Generate sample data
const X = [[1, 2], [3, 4], [5, 6], [7, 8]];

// Apply PCA
const pca = new PCA(X);
const X_pca = pca.predict(X);
```

Interview-Style Question - Eigenvalues and Eigenvectors

- Given a square matrix A, write a function to check if a given vector v is an eigenvector of A. If v is an eigenvector, return the corresponding eigenvalue; otherwise, return None.

CoGrammar

Interview-Style Question - Eigenvalues and Eigenvectors

```
is_eigenvector ( Matrix, Vector ):
    Av = dot product of A and v

    for eigenvalue in Av
        if corresponding eigenvector is valid:
            return eigenvalue

    if no valid return nothing
```

CoGrammar

```python
def is_eigenvector(A, v):
    Av = np.dot(A, v)

    for eigenvalue in np.linalg.eigvals(A):
        if np.allclose(Av, eigenvalue * v):
            return eigenvalue

    return None
```

```javascript
function isEigenvector(A, v) {
    const Av = math.multiply(A, v);

    const eigenvalues = math.eig(A).values;

    for (let i = 0; i < eigenvalues.length; i++) {
        const eigenvalue = eigenvalues[i];
        if (math.deepEqual(Av, math.multiply(eigenvalue, v))) {
            return eigenvalue;
        }
    }

    return null;
}
```

CoGrammar

Linear Transformations

- A linear transformation is a function $T: V \rightarrow W$ between two vector spaces V and W that satisfies the following properties:
  - Additivity: $T(u + v) = T(u) + T(v)$ for all u, v in V
  - Homogeneity: $T(cv) = cT(v)$ for all v in V and scalar c
- Linear transformations preserve the structure of vector spaces and can be represented by matrices.

CoGrammar

Linear Transformations

$T: X \rightarrow Y$

domain

X

co-domain or range

Y

so that

1. $T(x+y) = T(x) + T(y)$
2. $T(cx) = c\,T(x)$ for any number c

CoGrammar

Common Linear Transformations

- Rotation: Rotates a vector by a given angle θ counterclockwise.
  - Rotation matrix in 2D: [cos(θ) -sin(θ)] [sin(θ) cos(θ)]
- Scaling: Scales a vector by a factor of s along each axis.
  - Scaling matrix in 2D: [s_x 0] [0 s_y]
- Shearing: Shears a vector by a factor of k along one axis.
  - Shearing matrix in 2D (shearing along x-axis): [1 k] [0 1]

```python
def rotation_matrix(theta):
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    return np.array([[cos_theta, -sin_theta], [sin_theta, cos_theta]])

def scaling_matrix(sx, sy):
    return np.array([[sx, 0], [0, sy]])

def shearing_matrix(kx, ky):
    return np.array([[1, kx], [ky, 1]])
```

CoGrammar

```python
# Example usage
v = np.array([1, 1])

# Rotation by 45 degrees
rotation = rotation_matrix(np.pi/4)
v_rotated = np.dot(rotation, v)
print("Rotated vector:", v_rotated)
# Rotated vector: [1.11022302e-16 1.41421356e+00]

# Scaling by 2 along x-axis and 3 along y-axis
scaling = scaling_matrix(2, 3)
v_scaled = np.dot(scaling, v)
print("Scaled vector:", v_scaled)
# Scaled vector: [2 3]

# Shearing by 1 along x-axis and 2 along y-axis
shearing = shearing_matrix(1, 2)
v_sheared = np.dot(shearing, v)
print("Sheared vector:", v_sheared)
# Sheared vector: [2 3]
```

CoGrammar

```
const math = require('mathjs');

function rotationMatrix(theta) {
    const cosTheta = math.cos(theta);
    const sinTheta = math.sin(theta);
    return math.matrix([[cosTheta, -sinTheta], [sinTheta, cosTheta]]);
}

function scalingMatrix(sx, sy) {
    return math.matrix([[sx, 0], [0, sy]]);
}

function shearingMatrix(kx, ky) {
    return math.matrix([[1, kx], [ky, 1]]);
}
```

CoGrammar

Interview-Style Question - Linear Transformations

❖ Given a 2D vector v and a 2x2 matrix A representing a linear transformation, write a function to apply the transformation to v and return the transformed vector.

CoGrammar

```python
def apply_transformation(A, v):
    return np.dot(A, v)
```

```javascript
function applyTransformation(A, v) {
    return math.multiply(A, v);
}
```

CoGrammar

Systems of Linear Equations

❖ A system of linear equations is a set of equations where each equation is a linear combination of variables.

Systems of Linear Equations

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$$ can be represented by

$$\left.\begin{aligned} ax^2 + bx + c \\ dx^2 + ex + f \end{aligned}\right\}$$ systems of linear equations

CoGrammar

Systems of Linear Equations

- Solving systems of linear equations is essential in various fields:
  - Data Science: Regression analysis, data fitting, optimization problems
  - Web Development: Responsive design, layout calculations, CSS transformations
  - Software Engineering: Network analysis, resource allocation, cryptography

CoGrammar

Gaussian Elimination

❖ Gaussian elimination is a method for solving systems of linear equations by transforming the augmented matrix [A|b] into row echelon form.

Gaussian Elimination

$$\begin{bmatrix} a & d & e \\ 0 & b & f \\ 0 & 0 & c \end{bmatrix}$$ is the form we want

Each row has a non-zero element (pivot) behind the prior rows elements.

CoGrammar

Gaussian Elimination

- Steps:
  - Write the system of equations in augmented matrix form [A|b]
  - Perform elementary row operations to obtain an upper triangular matrix
  - Apply back-substitution to find the solution

System of equations	Row operations	Augmented matrix
$\begin{aligned} 2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$		$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right]$
$\begin{aligned} 2x + y - z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ 2y + z &= 5 \end{aligned}$	$\begin{aligned} L_2 + \tfrac{3}{2}L_1 &\to L_2 \\ L_3 + L_1 &\to L_3 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 2 & 1 & 5 \end{array}\right]$
$\begin{aligned} 2x + y - z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ -z &= 1 \end{aligned}$	$L_3 + -4L_2 \to L_3$	$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array}\right]$
The matrix is now in echelon form (also called triangular form)		

CoGrammar

$\begin{aligned} 2x + y &= 7 \\ \tfrac{1}{2}y &= \tfrac{3}{2} \\ -z &= 1 \end{aligned}$	$\begin{aligned} L_1 - L_3 &\to L_1 \\ L_2 + \tfrac{1}{2}L_3 &\to L_2 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & 0 & 7 \\ 0 & \frac{1}{2} & 0 & \frac{3}{2} \\ 0 & 0 & -1 & 1 \end{array}\right]$
$\begin{aligned} 2x + y &= 7 \\ y &= 3 \\ z &= -1 \end{aligned}$	$\begin{aligned} 2L_2 &\to L_2 \\ -L_3 &\to L_3 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & 0 & 7 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right]$
$\begin{aligned} x &= 2 \\ y &= 3 \\ z &= -1 \end{aligned}$	$\begin{aligned} L_1 - L_2 &\to L_1 \\ \tfrac{1}{2}L_1 &\to L_1 \end{aligned}$	$\left[\begin{array}{ccc\|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right]$

Interview-Style Question - Systems of Linear Equations

- Given a system of linear equations represented by a matrix A and a vector b, write a function to determine if the system has a unique solution, infinitely many solutions, or no solution.

CoGrammar

$$\left[\begin{array}{ccc|c} a & b & c & d \\ 0 & e & f & g \\ 0 & 0 & 0 & h \end{array}\right]$$

No solution

$$\left[\begin{array}{ccc|c} a & b & c & d \\ 0 & e & f & g \\ 0 & 0 & h & i \end{array}\right]$$

Unique solution

$$\left[\begin{array}{ccc|c} a & b & c & d \\ 0 & 0 & e & f \\ 0 & 0 & 0 & 0 \end{array}\right]$$

Infinitely many solutions

```
analyze_systems(A, b):
    n = length of b

    Ab = augmented matrix

    rank_A = rank of A
    rank_Ab = rank of Ab

    if rank_A equal to rank_Ab
        if rank_A equal n:
            return unique
        else
            return infinite
    else
        return no solution
```

$$\left[\begin{array}{cc|c} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \end{array}\right]$$

$$\begin{array}{ccc} & \text{rank} = 1 & \text{rank} = 2 \\ \text{no} & \left[\begin{array}{cc} a_1 & a_2 \\ 0 & 0 \end{array}\right. & \left|\begin{array}{c} b_1 \\ b_2 \end{array}\right] \end{array}$$

```python
def analyze_system(A, b):
    n = len(b)

    # Create the augmented matrix
    Ab = np.concatenate((A, b.reshape(n, 1)), axis=1)

    # Compute the rank of A and the augmented matrix
    rank_A = np.linalg.matrix_rank(A)
    rank_Ab = np.linalg.matrix_rank(Ab)

    if rank_A == rank_Ab:
        if rank_A == n:
            return "Unique solution"
        else:
            return "Infinitely many solutions"
    else:
        return "No solution"
```

```javascript
function analyzeSystem(A, b) {
    const n = b.length;

    // Create the augmented matrix
    const Ab = A.map((row, i) => [...row, b[i]]);

    // Compute the rank of A and the augmented matrix
    const rankA = math.rank(A);
    const rankAb = math.rank(Ab);

    if (rankA === rankAb) {
        if (rankA === n) {
            return "Unique solution";
        } else {
            return "Infinitely many solutions";
        }
    } else {
        return "No solution";
    }
}
```

CoGrammar

Which of the following is an example of a linear transformation?

A. Squaring each element of a vector

B. Applying a logarithmic function to each element of a vector

C. Rotating a vector by 45 degrees counterclockwise

D. Taking the absolute value of each element of a vector

CoGrammar

Answer: C

- A linear transformation T: V → W between two vector spaces V and W satisfies the following properties:
  - Additivity: T(u + v) = T(u) + T(v) for all u, v in V
  - Homogeneity: T(cv) = cT(v) for all v in V and scalar c
- Rotating a vector by 45 degrees is an example of a linear transformation because it preserves addition and scalar multiplication.

Answer: C

- Squaring, applying logarithms, or taking the absolute value of vector elements are not linear transformations because they do not satisfy the additivity and homogeneity properties.

What is the time complexity of solving a system of n linear equations using Gaussian elimination?

A. $O(n)$

B. $O(n^2)$

C. $O(n^3)$

D. $O(2^n)$

CoGrammar

Answer: C

- Gaussian elimination is an algorithm for solving systems of linear equations by transforming the augmented matrix [A|b] into row echelon form.

Answer: C

- The time complexity of Gaussian elimination is $O(n^3)$ for a system of n linear equations.
  - The algorithm performs elementary row operations, which require $O(n)$ operations for each of the $O(n^2)$ elements in the matrix.
  - Back-substitution, used to find the solution after obtaining the row echelon form, takes $O(n^2)$ time.
- Therefore, the overall time complexity of Gaussian elimination is $O(n^3)$.

Which of the following is true about eigenvectors?

A. Eigenvectors corresponding to distinct eigenvalues are linearly independent

B. Eigenvectors are always orthogonal to each other

C. Eigenvectors are always unit vectors

D. Eigenvectors can only be found for symmetric matrices

Answer: A

- For a square matrix A, a non-zero vector v is an eigenvector of A if $Av = \lambda v$ for some scalar λ, which is called the eigenvalue corresponding to v.
- Eigenvectors corresponding to distinct eigenvalues are linearly independent.
  - This property is useful in applications like diagonalization and finding a basis for a vector space.

Answer: A

- Eigenvectors are not always orthogonal to each other or unit vectors. However, eigenvectors can be orthonormal if the matrix is symmetric.
    - Eigenvectors and eigenvalues can be found for any square matrix, not just symmetric matrices. However, symmetric matrices have some special properties, such as real eigenvalues and orthogonal eigenvectors.

Portfolio Assignment

- Feel free to adjust according to your interests and specific track (data science, web development, or software engineering)

Portfolio Assignment

- Choose one or more of the following linear algebra topics to focus on:
  - Matrix operations (addition, subtraction, multiplication)
  - Linear transformations (rotation, scaling, shearing)
  - Solving systems of linear equations
  - Eigenvalues and eigenvectors

CoGrammar

Portfolio Assignment

- Design and implement a web app that allows users to interact with the chosen topics. Some ideas include:
  - A matrix calculator that performs basic operations on user-inputted matrices
  - A 2D or 3D graphics visualiser that applies linear transformations to shapes based on user input
  - A linear equation solver that demonstrates the steps involved in Gaussian elimination

Portfolio Assignment

- An eigenvalue/eigenvector calculator that visualises the effect of eigenvalues on vector transformations

Portfolio Assignment

- Use HTML, CSS, and JavaScript to build the frontend of the app, focusing on creating a clean and intuitive user interface
- Implement the necessary mathematical calculations and algorithms using JavaScript or a backend language of your choice (e.g., Python, Java)
- Provide clear instructions on how to use the app and explain the linear algebra concepts being demonstrated

CoGrammar

Portfolio Assignment

- ❖ Host the project on GitHub Pages or a similar platform and include a brief description of the app in the README file

Portfolio Assignment

- Evaluation Criteria:
  - Correctness and accuracy of the linear algebra calculations and visualisations
  - User interface design and ease of use
  - Code quality, organisation, and documentation
  - Clarity and effectiveness of the project description and instructions
  - Creativity and originality in applying linear algebra concepts to practical problems

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar