

# Principles and Practices of Data Science

## Lecture 9

Melvin Ayala

# Lecture 9: Linear Algebra

## Sections

### 1. What is Linear Algebra?

### 2. Introduction to Vectors

2.1. Vectors and Linear Combinations

2.2. Lengths and Dot Products

2.3. Matrices

### 3. Solving Linear Equations

3.1. Vectors and Linear Equations

3.2. The Idea of Elimination

3.3. Elimination Using Matrices

3.4. Rules for Matrix Operations

3.5. Inverse Matrices

3.6. Elimination = Factorization:  $A = LU$

3.7. Transposes and Permutations

### 4. Determinants

4.1. Determinants and Their Properties

4.2. Cramer's Rule

# 1. What is Linear Algebra?

## Linear Algebra

- one of the most widely used mathematical theories.
- is the study of lines and planes, vector spaces and mappings that are required for linear transforms.
- applications in virtually every area of mathematics:
  - geometry
  - multivariate calculus
  - differential equations
  - probability theory
- applied in:
  - physics
  - chemistry
  - economics
  - psychology
  - engineering
- branch of mathematics concerning linear equations such as:  $a_1x_1 + a_2x_2 + \dots + a_nx_n = b$

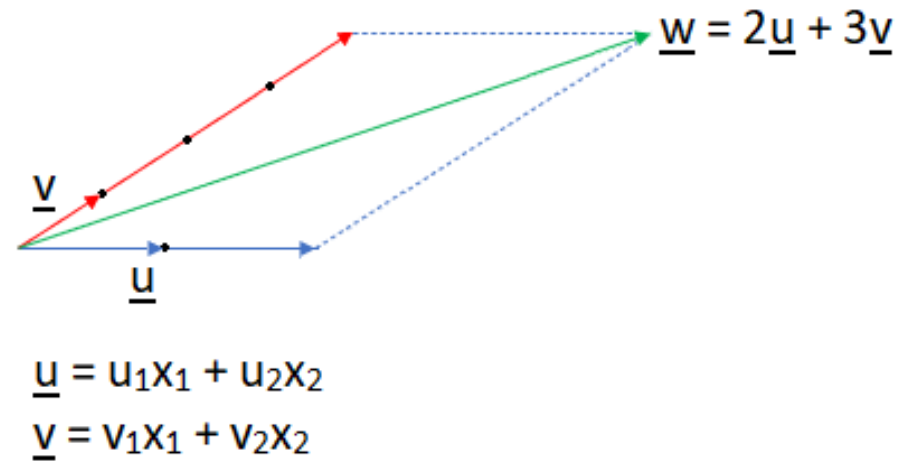
## Linear Combinations

- linear algebra is about linear combinations.
- using arithmetic on columns of numbers called vectors and arrays of numbers called matrices, to create new columns and arrays of numbers.

## 2. Introduction to Vectors

### 2.1. Vectors and Linear Combinations

A linear combination of two or more vectors is the vector obtained by adding two or more vectors (with different directions) which are multiplied by scalar values.



## 2.2. Lengths and Dot products

- the dot product or scalar product is an algebraic operation that takes two equal-length sequences of numbers and returns a single number.
- the dot product of two vectors  $\underline{u}$  and  $\underline{v}$  (of 2 dimensions) is defined as:

$$\begin{aligned}\underline{u} \cdot \underline{v} &= u_1 v_1 + u_2 v_2 \\ &= \|\underline{u}\| \|\underline{v}\| \cos(\theta)\end{aligned}$$

where  $\theta$  is the angle between the two vectors.

- the dot product of a vector  $\underline{w}$  (of 2 dimensions) with itself is:

$$\begin{aligned}\underline{w} \cdot \underline{w} &= w_1 w_1 + w_2 w_2 \\ &= \|\underline{w}\| \|\underline{w}\| \cos(\theta) \\ &= \|\underline{w}\| \|\underline{w}\|\end{aligned}$$

- this is the length of the vector. Generalizing:

$$\text{Length of a vector } \underline{w} = \|\underline{w}\|_2 = \left( \sum_{i=1}^n \|w_i\|^2 \right)^{1/2} = \sqrt{\sum_{i=1}^n \|w_i\|^2} = \sqrt{(w_1)^2 + (w_2)^2 + \dots + (w_n)^2}$$

In Python, we can calculate the norm of a vector using the Numpy or Scipy libraries.

## Python Examples

Calculate the norm of a vector in Python using the Numpy or Scipy libraries

### # Using Numpy

```
from numpy import array
from numpy.linalg import norm
arr = array([1, 2, 3, 4, 5])
print(arr)
norm_l1 = norm(arr, 1)
print(norm_l1)
```

Output:

```
[1 2 3 4 5]
15.0
```

### # Using Scipy

```
from numpy import array
from scipy.linalg import norm
arr = array([-1, -2, 3, 4, 5])
print(arr)
norm_l1 = norm(arr, 1)
print(norm_l1)
```

Output:

```
[-1 -2 3 4 5]
15.0
```

## 2.3. Matrices

### What is a matrix?

- a rectangular arrangement of numbers into rows and columns.
- arrangement of numbers into rows and columns.
- consists of dimensions and elements.

### Matrix dimensions:

The **dimensions** of a matrix tells its size: the number of rows and columns of the matrix, in that order.

### Types of Matrices:

- rectangular matrix (with  $n \neq m$ )
- square matrix: triangular (upper, lower), diagonal matrix (unity, scalar, non-scalar)
- row matrix (matrix with one row)
- column matrix (matrix with one column)
- null matrix
- scalar matrix
- symmetric matrix
- antisymmetric matrix

A diagram showing a matrix  $A$  with 2 rows and 3 columns. The matrix is represented as  $A = \begin{bmatrix} 4 & 1 & -3 \\ 2 & 0 & 9 \end{bmatrix}$ . Three blue arrows point down to the columns, labeled "3 columns". Two blue arrows point right to the rows, labeled "2 rows". Below the matrix, the text "A is a two by three matrix" is written.

### Matrix Operations:

- addition
- subtraction
- multiplication

### Matrix Transformations:

- inverse
- transpose
- adjoint
- complex conjugate

### Matrix Features:

- cofactor
- for square matrices: determinant, trace

# Matrix Operations in Python

## Example code:

```
import numpy as np

# Two matrices
mx1 = np.array([[5, 10], [15, 20]])
mx2 = np.array([[25, 30], [35, 40]])

print("Matrix1 =\n",mx1)
print("\nMatrix2 =\n",mx2)

# The addition() is used to add matrices
print ("\nAddition of two matrices: ")
print (np.add(mx1,mx2))

# The subtract() is used to subtract matrices
print ("\nSubtraction of two matrices: ")
print (np.subtract(mx1,mx2))

# The divide() is used to divide matrices
print ("\nMatrix Division: ")
print (np.divide(mx1,mx2))

# The multiply() is used to multiply matrices
print ("\nMultiplication of two matrices: ")
print (np.multiply(mx1,mx2))
```

Output:

Matrix1 =  
[[ 5 10]  
[15 20]]

Matrix2 =  
[[25 30]  
[35 40]]

Addition of two matrices:  
[[30 40]  
[50 60]]

Subtraction of two matrices:  
[[-20 -20]  
[-20 -20]]

Matrix Division:  
[[0.2 0.33333333]  
[0.42857143 0.5 ]]

Multiplication of two matrices:  
[[125 300]  
[525 800]]



## 3. Solving Linear Equations

### 3.1. Vectors and Linear Equations

#### What is a vector equation?

- an equation involving a linear combination of vectors with possibly unknown coefficients.
- an equation involving vectors with n coordinates is the same as n equations involving only numbers.
- for example, the equation:

$$x \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix} + y \begin{bmatrix} -1 \\ -2 \\ -1 \end{bmatrix} = \begin{bmatrix} 8 \\ 16 \\ 3 \end{bmatrix}$$

simplifies to:

$$\begin{bmatrix} x \\ 2x \\ 6x \end{bmatrix} + \begin{bmatrix} -y \\ -2y \\ -y \end{bmatrix} = \begin{bmatrix} 8 \\ 16 \\ 3 \end{bmatrix}$$

or

$$x - y = 8 \quad (1)$$

$$2x - 2y = 16 \quad (2)$$

$$6x - y = 3 \quad (3)$$

## 3.2. The Idea of Elimination

### Elimination Method for Solving Systems of Linear Equations

- uses the addition and multiplication property of equality.
- you can add/subtract the same value to each side of an equation.
- you can multiply/divide each side of an equation with the same value.

Example:

if you have the following system:

$$-x + y = 6 \quad (1)$$

$$x + y = 8 \quad (2)$$

you can multiply (1) with -1 and add to (2) to get:

$$x - y = -6 \quad (1) * (-1)$$

$$x + y = 8 \quad (2)$$

---

$$2x + 0 = 2$$

Therefore:

$$x = 2/2 = 1$$

then, substituting in either (1) or (2):

$$-1 + y = 6$$

$$y = 7$$

### 3.3. Elimination using Matrices

**A system of linear equations can be solved by using matrix elimination.**

For example, consider the following system:

$$a_1x + a_2y + a_3z = d_1$$

$$b_1x + b_2y + b_3z = d_2$$

$$c_1x + c_2y + c_3z = d_3$$

Remember:

- all the variables in the equations should be written in the appropriate order.
- the variables, their coefficients and constants are to be written on the respective sides.

Let:      Matrix A: represent the coefficients  $a_i, b_i, c_i$

Matrix B: represent the constants  $d_i$

A system of equations can be solved using matrix multiplication.

Rewrite the above equations in the matrix form as follows:

$$\begin{bmatrix} a_1x + a_2y + a_3z \\ b_1x + b_2y + b_3z \\ c_1x + c_2y + c_3z \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

### 3.3. Elimination using Matrices (Cont.)

Equations in matrix form:

$$\begin{bmatrix} a_1x + a_2y + a_3z \\ b_1x + b_2y + b_3z \\ c_1x + c_2y + c_3z \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

If we define:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}, \quad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad B = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

then:

$$A X = B$$

$$A^{-1} A X = A^{-1} B \quad (\text{pre-multiplying each side by the inverse of } A)$$

$$I X = A^{-1} B \quad (A^{-1} A \text{ is the identity matrix})$$

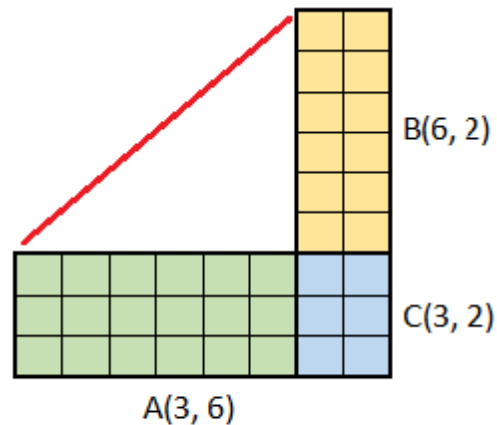
$$X = A^{-1} B \quad (\text{the solution to the system})$$

### 3.4. Rules for Matrix Operations

#### To perform matrix multiplication:

- the first matrix must have as many columns as rows are in the second matrix.
- the number of rows of the resulting matrix equals the number of rows of the first matrix
- the number of columns of the resulting matrix equals the number of columns of the second matrix.

$$A(n, m) \times B(m, k) = C(n, k)$$



$$A(n, m) \times B(m, k) = C(n, k)$$

$$A(3, 6) \times B(6, 2) = C(3, 2)$$

Matrix multiplication has some of the same properties as "normal" multiplication, such as

$$A ( B C ) = ( A B ) C$$

$$A ( B + C ) = A B + A C$$

$$( A + B ) C = A C + B C$$

But,

$A B \neq B A$  (not commutative, since dimensions rarely match up)

However,

$$( A B )^T = B^T A^T$$

# Matrix Operations in Python

- we can code explicit for-loops to perform matrix operations in Python
- we can use the **numpy** library to achieve this with one single line

## Adding two matrices:

```
import numpy as np

# input two matrices
mat1 = ([1, 6, 5], [3, 4, 8], [2, 12, 3])
mat2 = ([3, 4, 6], [5, 6, 7], [6, 56, 7])

print("Addition of two matrices:")
print("Matrix A =", mat1)
print("Matrix B =", mat2)

res = np.add(mat1, mat2)
print("A + B = ", res)
```

## Subtracting two matrices:

```
import numpy as np

# input two matrices
mat1 = ([1, 6, 5], [3, 4, 8], [2, 12, 3])
mat2 = ([3, 4, 6], [5, 6, 7], [6, 56, 7])

print("Subtraction of two matrices:")
print("Matrix A =", mat1)
print("Matrix B =", mat2)

res = np.subtract(mat1, mat2)
print("A - B = ", res)
```

## Multiplying two matrices:

```
import numpy as np

# input two matrices
mat1 = ([1, 6, 5],[3 ,4, 8],[2, 12, 3])
mat2 = ([3, 4, 6],[5, 6, 7],[6,56, 7])

print("Multiplication of two matrices:")
print("Matrix A =", mat1)
print("Matrix B =", mat2)

res = np.dot(mat1, mat2)
print("A * B = ", res)
```

### 3.5. Inverse Matrices

- a matrix can have an inverse
- we write  $A^{-1}$  instead of  $1/A$
- when we multiply a matrix by its inverse, we get the identity matrix  $I$ :

$$A A^{-1} = A^{-1} A = I$$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{example of a } 3 \times 3 \text{ identity matrix})$$

- an identity matrix is a square matrix where all elements in the main diagonal are 1 and the remaining are zeroes.
- sometimes there is no inverse at all.

# Computing the Inverse Matrix in Python

- we can write explicit code to obtain the inverse of a matrix
- we can use the numpy library to achieve this with one single line

## Obtaining the inverse matrix:

```
import numpy as np

# input one matrix
mat1 = ([1, 6, 5], [3, 4, 8], [2, 12, 3])

print("Inverse of a matrix:")
print("Matrix A =", mat1)

Ainv = np.linalg.inv(mat1)
print("Ainv = ", Ainv)
```



### 3.6. Elimination = Factorization: $A = LU$

The LU (lower-upper) decomposition of a matrix  $A$  is the pair of matrices  $L$  and  $U$  such that:

$$A = L U$$

$L$  is the lower-triangular matrix where:

- all diagonal entries equal to 1
- $U$  is an upper-triangular matrix
- LU decomposition may not exist for a matrix

LU decomposition provides an efficient means of solving linear equations.  
decomposition also called factorization.

Example:

$$A = \begin{bmatrix} 1 & 2 & 2 \\ 4 & 4 & 2 \\ 4 & 6 & 4 \end{bmatrix}$$

The LU factorization is:

$$A = L U = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 4 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & 0 & -1 \end{bmatrix}$$

# Elimination = Factorization: $A = LU$ (Cont.)

There is an algorithm for it which uses:

- forward substitution (with lower triangular matrices) and
- backward substitution (with upper-triangular matrices)

There is also a recursive LU algorithm

Code is provided below:

```
import numpy as np
def lup_decomp(A):
    """(L, U, P) = lup_decomp(A) is the LUP decomposition  $PA = LU$ 
    A is any matrix
    L will be a lower-triangular matrix with 1 on the diagonal, the same
    shape as A
    U will be an upper-triangular matrix, the same shape as A
    U will be a permutation matrix, the same shape as A
    """
    n = A.shape[0]
    if n == 1:
        L = np.array([[1]])
        U = A.copy()
        P = np.array([[1]])
        return (L, U, P)

    i = np.argmax(A[:,0])
    A_bar = np.vstack([A[i,:], A[:,i:], A[(i+1):,:]])
    A_bar11 = A_bar[0,0]
    A_bar12 = A_bar[0,1:]
```

```
A_bar21 = A_bar[1:,0]
A_bar22 = A_bar[1:,1:]
S22 = A_bar22 - np.dot(A_bar21, A_bar12) / A_bar11
L11 = 1
U11 = A_bar11

L12 = np.zeros(n-1)
U12 = A_bar12.copy()

L21 = np.dot(P22, A_bar21) / A_bar11
U21 = np.zeros(n-1)

L = np.block([[L11, L12], [L21, L22]])
U = np.block([[U11, U12], [U21, U22]])
P = np.block([
    [np.zeros((1, i-1)), 1, np.zeros((1, n-i))],
    [P22[:,:(i-1)], np.zeros((n-1, 1)), P22[:,i:]]
])
return (L, U, P)
```

## 3.7. Transposes and Permutations

### Transpose of a matrix:

- found by interchanging its rows into columns or columns into rows.
- example:

$$M = \begin{bmatrix} 6 & 3 & 1 \\ 5 & 0 & 9 \end{bmatrix}$$

$$M^T = \begin{bmatrix} 6 & 5 \\ 3 & 0 \\ 1 & 9 \end{bmatrix}$$

### Permutation matrix:

- a square binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere
- examples: all permutations of a 3 x 3 matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

# Computing the Transpose Matrix in Python

- we can write explicit code to obtain the inverse of a matrix
- we can use the numpy library to achieve this with one single line

## Obtaining the inverse matrix:

```
import numpy as np

# input one matrix
mat1 = ([1, 6, 5], [3, 4, 8], [2, 12, 3])

print("Transpose of a matrix:")
print("Matrix A =", mat1)

Atransp = np.transpose(mat1)
# also: Atransp = mat1.T
print("Atranspose = ", Atransp)
```

## 4. Determinants

### 4.1. The Determinants and their Properties

#### Determinant of a matrix:

- scalar value computed for a given **square** matrix.
- considered as the scaling factor for the transformation of a matrix.
- useful in solving a system of linear equation, calculating the inverse of a matrix and calculus operations.

For a 2 x 2 matrix A, the determinant  $\det(A)$  is given by:

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

$$\det(A) = ad - cb$$

#### Example:

$$A = \begin{bmatrix} 7 & -1 \\ 5 & 2 \end{bmatrix}$$

$$\det(A) = 7(2) - 5(-1) = 14 + 5 = 19$$

## For a 3 x 3 matrix (and higher order matrices)

- determinant can be obtained by the expansion of the matrix along any row (or column) using submatrices.
- for a matrix A such that:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\det(A) = a_{11} \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} - a_{12} \begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} + a_{13} \begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

where the sign is computed as  $(-1)^{i+j}$


- the determinant of a 3 x 3 matrix can also be computed using the Rule of Sarrus

# Rule of Sarrus for a 3 x 3 matrix

- the determinant of a 3 x 3 matrix can also be computed using the Rule of Sarrus (mnemonic)
- named after the French mathematician Pierre Frédéric Sarrus.
- for a matrix A such that:

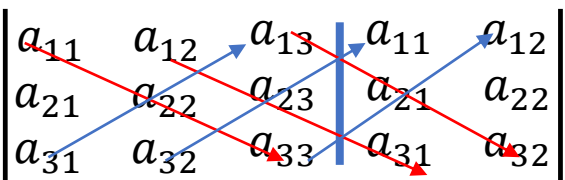
$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

**Step 1:** Augment A with the first two columns:


$$\det(A) = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \end{vmatrix}$$

**Step 2:** From upper left to lower right: Multiply the entries down the first diagonal. Add the result to the product of entries down the second diagonal. Add this result to the product of the entries down the third diagonal.

**Step 3:** From lower left to upper right: Subtract the product of entries up the first diagonal. From this result subtract the product of entries up the second diagonal. From this result, subtract the product of entries up the third diagonal.


$$\det(A) = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \end{vmatrix}$$

## Rule of Sarrus for a 3 x 3 matrix (cont.)

- the algebra is as follows:

$$\det(A) = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}$$

$$\det(A) = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \end{vmatrix}$$



# Properties of Determinants

## Reflection Property:

The determinant remains unaltered if its rows are changed into columns and the columns into rows. This is known as the property of reflection, i.e.:  $\det(A) = \det(A^T)$

## All-zero Property:

If all the elements of a row (or column) are zero, then the determinant is zero.

## Proportionality (Repetition) Property:

If all elements of a row (or column) are proportional (identical) to the elements of some other row (or column), then the determinant is zero.

## Switching Property:

The interchange of any two rows (or columns) of the determinant changes its sign.

## Scalar Multiple Property:

If all the elements of a row (or column) of a determinant are multiplied by a non-zero constant, then the determinant gets multiplied by the same constant.

## The determinant of the identity matrix is 1.

Note: There are more properties.

# Computing the Determinant of a Matrix in Python

- we can write explicit code to obtain the determinant of a matrix
- we can use the numpy library to achieve this with one single line

## Obtaining the determinant:

```
import numpy as np

# creating a 2X2 Numpy matrix
n_array = np.array([[50, 29], [30, 44]])

# Displaying the Matrix
print("Numpy Matrix is:")
print(n_array)

# calculating the determinant of matrix
det = np.linalg.det(n_array)

print("\nDeterminant of given 2X2 matrix:")
print(int(det))
```

## 4.2. Cramer's Rule

- one of the important methods applied to solve a system of equations.
- invented by Italian mathematician Gabriel Cramer in 1750s.
- used to find the solution of a system of equations with any number of variables and the same number of equations.
- the values of the variables in the system are to be calculated using the determinants of matrices.
- Cramer's rule is also known as the determinant method.

Here is the Cramer's rule formula to solve the system  $AX = B$  (or) to find the values of the variables  $x_1, x_2, x_3, \dots, x_n$ .

- Find  $\det |A|$  and represent it by  $D$ .
- Find the determinants  $D_{x_1}, D_{x_2}, D_{x_3}, \dots, D_{x_n}$ , where  $D_{x_i}$  is the determinant of matrix  $A$  where the  $i$ th column is replaced by the column matrix  $B$ .
- We divide each of these determinants by  $D$  to find the value of the corresponding variables. i.e.,
- $x_1 = D_{x_1}/D, x_2 = D_{x_2}/D, \dots, x_n = D_{x_n}/D$ .

# Cramer's Rule (Cont.)

**In a few words:**

- Cramer's rule gives the unique solution to a system of equations, if it exists.
- if the system has no solution or an infinite number of solutions → determinant = zero.

Consider a system of two equations in two variables.

$$\begin{aligned} a_1x + b_1y &= c_1 & (1) \\ a_2x + b_2y &= c_2 & (2) \end{aligned}$$

This represents the following matrix multiplication operation:

$$\underline{C} = \underline{A} * \underline{X}$$

where

$$\underline{A} = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}, \quad \underline{X} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \underline{C} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

		x
		y
a <sub>1</sub>	b <sub>1</sub>	c <sub>1</sub>
a <sub>2</sub>	b <sub>2</sub>	c <sub>2</sub>

We eliminate one variable using row operations and solve for the other.

# Cramer's Rule (derivation for the interested reader)

## To solve for x:

$$\begin{array}{ll} b_2a_1x + b_2b_1y = b_2c_1 & \text{multiply (1) by } b_2 \\ -b_1a_2x - b_1b_2y = -b_1c_2 & \text{multiply (2) by } -b_1 \end{array}$$

---

$$\begin{aligned} (b_2a_1 - b_1a_2)x + (b_2b_1 - b_1b_2)y &= b_2c_1 - b_1c_2 \\ (b_2a_1 - b_1a_2)x + (0)y &= b_2c_1 - b_1c_2 \\ (b_2a_1 - b_1a_2)x &= b_2c_1 - b_1c_2 \end{aligned}$$

Now, we solve for x:

$$x = (b_2c_1 - b_1c_2) / (b_2a_1 - b_1a_2)$$

or

$$x = \frac{b_2c_1 - b_1c_2}{b_2a_1 - b_1a_2} = \frac{\det \begin{bmatrix} c_1 & b_1 \\ c_2 & b_2 \end{bmatrix}}{\det \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}} = \frac{\begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} = \frac{D_x}{D}$$

## To solve for y:

$$\begin{array}{ll} a_2a_1x + a_2b_1y = a_2c_1 & \text{multiply (1) by } a_2 \\ -a_1a_2x - a_1b_2y = -a_1c_2 & \text{multiply (2) by } -a_1 \end{array}$$

---

$$\begin{aligned} (a_2a_1 - a_1a_2)x + (a_2b_1 - a_1b_2)y &= a_2c_1 - a_1c_2 \\ (0)x + (a_2b_1 - a_1b_2)y &= a_2c_1 - a_1c_2 \\ (a_2b_1 - a_1b_2)y &= a_2c_1 - a_1c_2 \end{aligned}$$

Now, we solve for y:

$$y = (a_2c_1 - a_1c_2) / (a_2b_1 - a_1b_2)$$

or

$$y = \frac{a_2c_1 - a_1c_2}{a_2b_1 - a_1b_2} = \frac{\det \begin{bmatrix} a_1 & c_1 \\ a_2 & c_2 \end{bmatrix}}{\det \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}} = \frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} = \frac{D_y}{D}$$

# Cramer's Rule (Cont.)

- we can use these formulas to solve for x and y, but Cramer's Rule also introduces new notation:

D: determinant of the coefficient matrix

D<sub>x</sub>: determinant of the numerator in the solution of x:

$$x = \frac{D_x}{D}$$

D<sub>y</sub>: determinant of the numerator in the solution of y:

$$y = \frac{D_y}{D}$$

with:

$$D_x = \det \begin{bmatrix} c_1 & b_1 \\ c_2 & b_2 \end{bmatrix} = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}$$

$$D_y = \det \begin{bmatrix} a_1 & c_1 \\ a_2 & c_2 \end{bmatrix} = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$$

$$D = \det \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

## Cramer's Rule Example 1

Solve the following  $2 \times 2$  system using Cramer's Rule:

$$12x + 3y = 15$$

$$2x - 3y = 13$$

Solution:

$$x = \frac{D_x}{D} = \frac{\begin{vmatrix} 15 & 3 \\ 13 & -3 \end{vmatrix}}{\begin{vmatrix} 12 & 3 \\ 2 & -3 \end{vmatrix}} = \frac{15(-3) - 3(13)}{12(-3) - 3(2)} = \frac{-84}{-42} = 2$$

$$y = \frac{D_y}{D} = \frac{\begin{vmatrix} 12 & 15 \\ 2 & 13 \end{vmatrix}}{\begin{vmatrix} 12 & 3 \\ 2 & -3 \end{vmatrix}} = \frac{12(13) - 15(2)}{12(-3) - 3(2)} = \frac{126}{-42} = -3$$

The solution is:

$$x = 2, y = -3$$

Proof:

$$12(2) + 3(-3) = 15$$

$$2(2) - 3(-3) = 13$$

## Cramer's Rule Example 2

Solve the following  $2 \times 2$  system using Cramer's Rule:

$$x + 2y = -11$$

$$-2x + y = -13$$

Solution:

$$x = \frac{D_x}{D} = \frac{\begin{vmatrix} -11 & 2 \\ -13 & 1 \end{vmatrix}}{\begin{vmatrix} 1 & 2 \\ -2 & 1 \end{vmatrix}} = \frac{-11(1) - 2(-13)}{1(1) - 2(-2)} = \frac{15}{5} = 3$$

$$y = \frac{D_y}{D} = \frac{\begin{vmatrix} 1 & -11 \\ -2 & -13 \end{vmatrix}}{\begin{vmatrix} 1 & 2 \\ -2 & 1 \end{vmatrix}} = \frac{1(-13) - (-11)(-2)}{1(1) - 2(-2)} = \frac{-35}{5} = -7$$

The solution is:

$$x = 3, y = -7$$

Proof:

$$3 + 2(-7) = -11$$

$$-2(3) + (-7) = -13$$



# Solving a System of Linear Equations in Python

- we can write explicit code to obtain the determinant of a matrix
- we can use the numpy library to achieve this with one single line

**Solve the following system of equations:**

$$\begin{aligned}x_1 + 2x_2 &= 1 \\ 3x_1 + 5x_2 &= 2\end{aligned}$$

```
# Using the solve method
import numpy as np
a = np.array([[1, 2], [3, 5]])
b = np.array([1, 2])
x = np.linalg.solve(a, b)
print(x)
```

Output:

```
array([-1., 1.])
```

```
# Solving by using  $A X = B \rightarrow X = A^{-1} B$ 
import numpy as np
a = np.array([[1, 2], [3, 5]])
b = np.array([1, 2])
x = np.linalg.inv(a).dot(b)
print(x)
```

Output:

```
array([-1., 1.])
```

End of Lecture