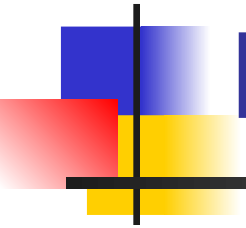
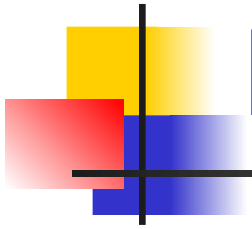


# Linear Algebra

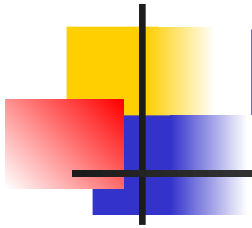
Dr. Bernard Chen

Troy University



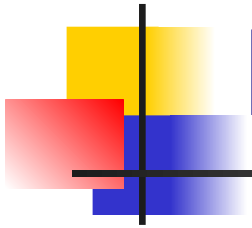
# Basics

- Sets
  - Mathematics view of the set data structure: a collection of objects
  - Denoted by a pair of braces: { }
  - Example: S = {orange, apple, banana} meaning that the set S contains orange, apple and banana.



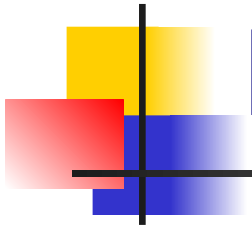
# Basics

- Union: $A \cup B$
- Intersection: $A \cap B$
- Empty set: $\emptyset$
- Contain: $a \in A$
- Set minus: $A - a$ → A minus the element $a$



# Basics

| Set Name | Symbol | Description |
|---|---|---|
| Naturals | $\mathbb{N}$ | $\mathbb{N} = \{1, 2, 3, 4, \dots\}$ |
| Wholes | $\mathbb{W}$ | $\mathbb{W} = \mathbb{N} \cup \{0\}$ |
| Integers | $\mathbb{Z}$ | $\mathbb{Z} = \mathbb{W} \cup \{-1, -2, -3, \dots\}$ |
| Rationals | $\mathbb{Q}$ | $\mathbb{Q} = \left\{\frac{p}{q} : p \in \mathbb{Z}, q \in \mathbb{Z} \backslash \{0\}\right\}$ |
| Irrationals | $\mathbb{I}$ | $\mathbb{I}$ is the set of real numbers not expressible as a fraction of integers |
| Reals | $\mathbb{R}$ | $\mathbb{R} = \mathbb{Q} \cup \mathbb{I}$ |
| Complex numbers | $\mathbb{C}$ | $\mathbb{C} = \left\{a + bi : a, b \in \mathbb{R}, i = \sqrt{-1}\right\}$ |



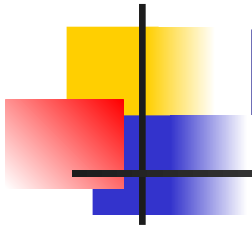
# Basics

- A **complex number** is a number that has **two parts**:
  - a **real part**
  - an **imaginary part**

- It is usually written in the form:
  - a+bi
- where:
- a = real number
- b = real number
- i = the imaginary unit

# Basics

- Vectors
- Set $R^n = \{(x_1, x_2, \ldots x_n): x_1, x_2, \ldots x_n \in R\}$
  → set of all n-tuples of real number
  → $R^3$ represents the set of real triples (x, y, z) coordinates in 3D space.

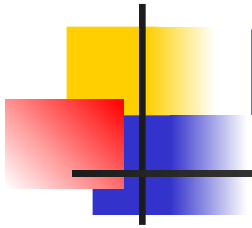


# Basics

- Row vector: vectors written horizontally
- Column vector: vectors written vertically
- Zero vector: vector in $R^n$ containing all 0s

# Basics

- Norm: a measure of the vector's length
- L1 norm (Manhattan distance): $\|v\|_1 = \sum_i |v_i|$
- L2 norm (Euclidean distance): $\|v\|_2 = \sqrt{\sum_i v_i^2}$
- p-norm ($L_p$): $\|v\|_p = \sqrt{\sum_i v_i^p}$
- $L_\infty$ norm: p-norm where p = $\infty$ (maximum absolute value in v).



# Basics

Example vector:

$$\mathbf{v} = [3, -4, 2]$$

The L1 norm is:

$$\|\mathbf{v}\|_1 = |3| + |-4| + |2|$$

$$\|\mathbf{v}\|_1 = |3| + |-4| + |2| = 9$$

So the **L1 norm value is 9**.

# Basics

For the same vector:

$$\mathbf{v} = [3, -4, 2]$$

The **L2 norm** (also called the Euclidean norm) is:

$$\|\mathbf{v}\|_2 = \sqrt{3^2 + (-4)^2 + 2^2}$$

$$\|\mathbf{v}\|_2 = \sqrt{3^2 + (-4)^2 + 2^2} = \sqrt{29} \approx 5.39$$

So the **L2 norm value is approximately 5.39**.

# Python code for vector

`numpy.array` is used in Python to create an array using the NumPy library.

Example:

Python

Run

```
import numpy as np

v = np.array([3, -4, 2])

print(v)
```

```
>>> print (v)
[ 3 -4  2]
```

# Why np.array?

## 1. Faster calculations

`np.array` is optimized in C internally, so operations on large datasets are much faster.

Python

Run

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)
```

Output:

Python

Run

```
[5 7 9]
```

# Why np.array?

- This is called Vector Addition

**Vector addition** is defined as the pairwise addition of each of the elements of the added vectors. For example, if $v$ and $w$ are vectors in $\mathbb{R}^n$, then $u = v + w$ is defined as $u_i = v_i + w_i$.

```
>>> a=np.array([1,2,3])
>>> b=np.array([4,5,6])
>>>
>>> a+b
    array([5, 7, 9])
>>> a*b
    array([ 4, 10, 18])
>>> a-b
    array([-3, -3, -3])
>>> a/b
    array([0.25, 0.4 , 0.5 ])
```

# Why np.array?

With normal Python lists:

Python

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

means concatenation:

Python

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

# Why np.array?

## 2. Vectorized operations

You can do math on the whole array at once.

Python

Run

```
a = np.array([1, 2, 3])

print(a * 2)
```

Output:

Python

Run

```
[2 4 6]
```

No loop needed.

# Why np.array?

### 3. Useful for machine learning and data science

Libraries like:

- Pandas
- TensorFlow
- PyTorch
- SciPy

all use NumPy arrays internally.

# Python code for vector/matrix

**TRY IT!** Create a row vector and a column vector, and show their shape.

```
In [1]: import numpy as np
        vector_row = np.array([[1, -5, 3, 2, 4]])
        vector_column = np.array([[1],[2],[3],[4]])
        print(vector_row.shape)
        print(vector_column.shape)

(1, 5)
(4, 1)
```

# Norm1 in Python

Python

```python
import numpy as np

a = np.array([3, -4, 2])

l1 = np.linalg.norm(a, 1)

print(l1)
```

# Norm2 in Python

Python

```python
import numpy as np

a = np.array([3, 4])

l2 = np.linalg.norm(a)

print(l2)
```

# Norm1 and Norm2 in Python

```
>>> print (v)
[ 3 -4  2]
>>> np.linalg.norm(v)
np.float64(5.385164807134504)
>>> np.linalg.norm(v,1)
np.float64(9.0)
```

# Matrix

- m x n matrix: a rectangular table of numbers consisting of m rows and n columns.

- p-norm

$$\|M\|_p = \sqrt[p]{\sum_{i}^{m}\sum_{j}^{n}|a_{ij}|^p}.$$

# Matrix

```python
import numpy as np

A = np.array([
    [1, -2],
    [3, 4]
])
```

$$\|M\|_p = \sqrt[p]{\sum_i^m \sum_j^n |a_{ij}|^p}.$$

Matrix:

$$A = \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix}$$

Matrix:

# Matrix

$$A = \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix}$$

$$\|M\|_p = \sqrt[p]{\sum_i^m \sum_j^n |a_{ij}|^p}.$$

P=2 (Frobenius Norm)
This is similar to the vector L2 norm but for all matrix entries.

Calculation:

$$\sqrt{1^2 + (-2)^2 + 3^2 + 4^2}$$

$$= \sqrt{1 + 4 + 9 + 16} = \sqrt{30} \approx 5.477$$

```
np.linalg.norm(A, 'fro')
```

# Matrix

- Matrix addition and scalar multiplication for matrices work the same way as for vectors.

```
>>> aa
array([[ 1, -2],
       [ 3,  4]])
>>> bb
array([[ 2,  6],
       [-1, -2]])
>>> aa+bb
array([[3, 4],
       [2, 2]])
>>> aa*bb
array([[  2, -12],
       [ -3,  -8]])
```

Element-wise Multiplication

# Matrix

- **Matrix multiplication** means multiplying rows of the first matrix by columns of the second matrix.

Suppose we have:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

and

$$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

# Matrix

Suppose we have:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

and

$$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$C = AB$$

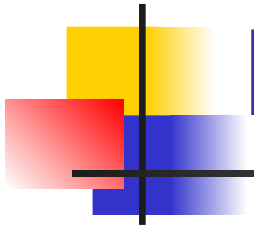
Result matrix $C$:

$$C = \begin{bmatrix} (1)(5) + (2)(7) & (1)(6) + (2)(8) \\ (3)(5) + (4)(7) & (3)(6) + (4)(8) \end{bmatrix}$$

Calculate each entry:

$$C = \begin{bmatrix} 5 + 14 & 6 + 16 \\ 15 + 28 & 18 + 32 \end{bmatrix}$$

$$C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$



# Matrix

However, **matrix multiplication** between two matrices, P and Q, is defined when P is an m×p matrix and Q is a p×n matrix.

The result of M=PQ is a matrix M that is m×n.
The dimension with size p is called the **inner matrix dimension**, and the inner matrix dimensions must match

The dimensions m and n are called the **outer matrix dimensions**.
Formally, if P is m×p and Q is p×n, then M=PQ is defined as

$$M_{ij} = \sum_{k=1}^{p} P_{ik} Q_{kj}$$

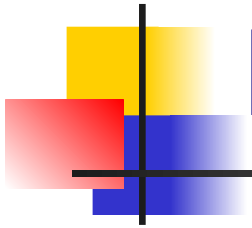
# Matrix in Python

```
P = np.array([[1, 7], [2, 3], [5, 0]])
Q = np.array([[2, 6, 3, 1], [1, 2, 3, 4]])
print(P)
print(Q)
print(np.dot(P, Q))
np.dot(Q, P)
```

```
[[1 7]
 [2 3]
 [5 0]]
[[2 6 3 1]
 [1 2 3 4]]
[[ 9 20 24 29]
 [ 7 18 15 14]
 [10 30 15  5]]
```

# Matrix in Python

```
>>> p=np.array([[1,7],[2,3],[5,0]])
>>> p
array([[1, 7],
       [2, 3],
       [5, 0]])
>>> p.shape
(3, 2)
>>> q=np.array([[2,6,3,1],[1,2,3,4]])
>>> q.shape
(2, 4)
>>>
>>>
>>> np.dot(p,q)
array([[ 9, 20, 24, 29],
       [ 7, 18, 15, 14],
       [10, 30, 15,  5]])
>>> np.dot(q,p)
Traceback (most recent call last):
  File "<pyshell#66>", line 1, in <module>
    np.dot(q,p)
ValueError: shapes (2,4) and (3,2) not aligned: 4 (dim 1) != 3 (dim 0)
>>>
```



# Matrix

For an $m \times n$ matrix $A$:

$$\text{rank}(A) = \text{number of linearly independent rows or columns of } A$$

The rank tells you:

- how much independent information the matrix contains
- how many rows or columns are not redundant

The number of independent rows always equals the number of independent columns.

# Matrix

Using NumPy:

‹/› Python

```python
import numpy as np

A = np.array([
    [1, 2],
    [3, 4]
])

print(np.linalg.matrix_rank(A))
```

# Matrix

Code:

```python
B = np.array([
    [1, 2],
    [2, 4]
])

print(np.linalg.matrix_rank(B))
```

Output:

```
1
```

# Matrix

## Important Facts

For an $m \times n$ matrix:

$$\text{rank}(A) \leq \min(m, n)$$

- Full rank means the rank equals the smaller dimension.
- Rank 0 means all entries are zero.

# Matrix

Here is a $4 \times 2$ matrix example.

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \\ 4 & 8 \end{bmatrix}$$

This matrix has:

- 4 rows
- 2 columns

Notice the second column is always 2 times the first column:

$$\begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

So the columns are linearly dependent.

Therefore:

$$\text{rank}(A) = 1$$

# Matrix

Now compare with a full-rank $4 \times 2$ matrix:

$$B = \begin{bmatrix} 1 & 2 \\ 2 & 5 \\ 3 & 1 \\ 4 & 7 \end{bmatrix}$$

The second column is NOT a multiple of the first column, so both columns are independent.

Thus:

$$\text{rank}(B) = 2$$

# Square Matrix

- Square matrix: n x n matrix

- Determinant: a special number that can be calculated directly from a square matrix. (for inverse calculation or the solution for systems of linear equations)

# Square Matrix

In the case of a $2 \times 2$ matrix, the determinant is

$$\det|M| = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc.$$

Example:

$$A = \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix}$$

Compute:

$$\det(A) = (3)(4) - (2)(1)$$

$$= 12 - 2$$

$$= 10$$

$$\det\left(\begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix}\right) = 3 \cdot 4 - 2 \cdot 1 = 10$$

# Python code

Python

```python
import numpy as np

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

# calculate determinant
det_A = np.linalg.det(A)

print("Matrix A:")
print(A)
```

Output:

Python

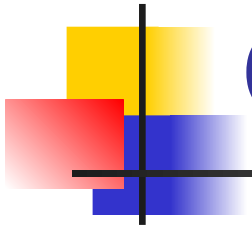
```
Matrix A:
[[3 2]
 [1 4]]

Determinant of A:
10.000000000000002
```

# Square Matrix

Similarly, in the case of a $3 \times 3$ matrix, the determinant is:

$$
\begin{aligned}
|M| = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} &= a \begin{bmatrix} \square & \square & \square \\ \square & e & f \\ \square & h & i \end{bmatrix} - b \begin{bmatrix} \square & \square & \square \\ d & \square & f \\ g & \square & i \end{bmatrix} + c \begin{bmatrix} \square & \square & \square \\ d & e & \square \\ g & h & \square \end{bmatrix} \\
&= a \begin{bmatrix} e & f \\ h & i \end{bmatrix} - b \begin{bmatrix} d & f \\ g & i \end{bmatrix} + c \begin{bmatrix} d & e \\ g & h \end{bmatrix} \\
&= aei + bfg + cdh - ceg - bdi - afh.
\end{aligned}
$$



# Overall Outline

- Basics
- Linear Transformations

# Linear Transformations

- F(ax + by) = aF(x) + bF(y) where x and y are vectors and a and b are scalars.

- Multiplication of m x n matrix A and n x 1 vector v is linear transformation of v → a matrix will be synonymous with a linear transformation function.

# Linear Transformations

## Matrix View of Linear Transformations

Most linear transformations can be written as matrix multiplication:

$$T(x) = Ax$$

where:

- $A$ is a matrix
- $x$ is a vector

# Linear Transformations

## Example

Let

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

and vector

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Transformation:

$$T(x) = Ax$$

Multiply:

$$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$$

# Linear Transformations

A linear equation must have variables only to the first power and not multiplied together.

General linear form:

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

Example of a linear equation:

$$2x_1 + 3x_2 - x_3 = 5$$

# Linear Transformations

**TRY IT!** Determine which of the following equations is linear and which is not. For the ones that are not linear, can you manipulate them so that they are?

1. $3x_1 + 4x_2 - 3 = -5x_3$
2. $\frac{-x_1+x_2}{x_3} = 2$
3. $x_1 x_2 + x_3 = 5$

# Linear Transformations

A **system of linear equations** is a set of linear equations that share the same variables. Consider the following system of linear equations:

$$
\begin{array}{ccccccccc}
a_{1,1}x_1 & + & a_{1,2}x_2 & + & \dots & + & a_{1,n-1}x_{n-1} & + & a_{1,n}x_n \\
a_{2,1}x_1 & + & a_{2,2}x_2 & + & \dots & + & a_{2,n-1}x_{n-1} & + & a_{2,n}x_n \\
 & & & & \dots & & \dots & & \\
a_{m-1,1}x_1 & + & a_{m-1,2}x_2 & + & \dots & + & a_{m-1,n-1}x_{n-1} & + & a_{m-1,n}x_n \\
a_{m,1}x_1 & + & a_{m,2}x_2 & + & \dots & + & a_{m,n-1}x_{n-1} & + & a_{m,n}x_n
\end{array}
$$

# Linear Transformations

where $a_{i,j}$ and $y_i$ are real numbers. The **matrix form** of a system of linear equations is \(\textbf{\}Ax = y} where $A$ is a $m \times n$ matrix, $A(i,j) = a_{i,j}$, $y$ is a vector in $\mathbb{R}^m$, and $x$ is an unknown vector in $\mathbb{R}^n$. The matrix form is showing as below:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{bmatrix}$$

# Linear Transformations

**TRY IT!** Put the following system of equations into matrix form.

$$\begin{aligned} 4x + 3y - 5z &= 2 \\ -2x - 4y + 5z &= 5 \\ 7x + 8y &= -3 \\ x + 2z &= 1 \\ 9 + y - 6z &= 6 \end{aligned}$$

$$\begin{bmatrix} 4 & 3 & -5 \\ -2 & -4 & 5 \\ 7 & 8 & 0 \\ 1 & 0 & 2 \\ 9 & 1 & -6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ -3 \\ 1 \\ 6 \end{bmatrix}$$

# Gauss Elimination Method

- A procedure that turns the matrix A into an **upper-triangular** form to solve the system of equations.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & a'_{2,2} & a'_{2,3} & a'_{2,4} \\ 0 & 0 & a'_{3,3} & a'_{3,4} \\ 0 & 0 & 0 & a'_{4,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y'_2 \\ y'_3 \\ y'_4 \end{bmatrix}$$

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 &= y_1, \\ a'_{2,2}x_2 + a'_{2,3}x_3 + a'_{2,4}x_4 &= y'_2, \\ a'_{3,3}x_3 + a'_{3,4}x_4 &= y'_3, \\ a'_{4,4}x_4 &= y'_4. \end{aligned}$$

# Gauss Elimination Method

- Use Gauss elimination, solve the following equations:

$$\begin{aligned} 4x_1 + 3x_2 - 5x_3 &= 2, \\ -2x_1 - 4x_2 + 5x_3 &= 5, \\ 8x_1 + 8x_2 &= -3. \end{aligned}$$

# Gauss Elimination Method

Step 1: Turn these equations to matrix form $Ax = y$.

$$\begin{bmatrix} 4 & 3 & -5 \\ -2 & -4 & 5 \\ 8 & 8 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ -3 \end{bmatrix}$$

# Gauss Elimination Method

Step 2: Get the augmented matrix [A, y]

$$[A, y] = \begin{bmatrix} 4 & 3 & -5 & 2 \\ -2 & -4 & 5 & 5 \\ 8 & 8 & 0 & -3 \end{bmatrix}$$

# Gauss Elimination Method

Step 3: Now we start to eliminate the elements in the matrix, we do this by choose a **pivot equation**, which is used to eliminate the elements in other equations. Let's choose the first equation as the pivot equation and turn the 2nd row first element to 0. To do this, we can multiply -0.5 for the 1st row (pivot equation) and subtract it from the 2nd row. The multiplier is $m_{2,1} = -0.5$. We will get

$$[A, y] = \begin{bmatrix} 4 & 3 & -5 & 2 \\ -2 & -4 & 5 & 5 \\ 8 & 8 & 0 & -3 \end{bmatrix}$$

Use 1st Row as pivot, multiple -0.5=
-2 -1.5 2.5 -1

Then use 2nd Row to minus the equation we generated

$$\begin{bmatrix} 4 & 3 & -5 & 2 \\ 0 & -2.5 & 2.5 & 6 \\ 8 & 8 & 0 & -3 \end{bmatrix}$$

-2 -4 5 5
- (-2 -1.5 2.5 -1) =
0 -2.5 2.5 6

# Gauss Elimination Method

Step 4: Turn the 3rd row first element to 0. We can do something similar, multiply 2 to the 1st row and subtract it from the 3rd row. The multiplier is $m_{3,1} = 2$. We will get

$$\begin{bmatrix} 4 & 3 & -5 & 2 \\ 0 & -2.5 & 2.5 & 6 \\ 0 & 2 & 10 & -7 \end{bmatrix}$$

# Gauss Elimination Method

Step 5: Turn the 3rd row 2nd element to 0. We can multiple -4/5 for the 2nd row, and add subtract it from the 3rd row. The multiplier is $m_{3,2} = -0.8$. We will get

$$\begin{bmatrix} 4 & 3 & -5 & 2 \\ 0 & -2.5 & 2.5 & 6 \\ 0 & 0 & 12 & -2.2 \end{bmatrix}$$

# Gauss Elimination Method

Step 6: Therefore, we can get $x_3 = -2.2/12 = -0.183$.

Step 7: Insert $x_3$ to the 2nd equation, we get $x_2 = -2.583$

Step 8: Insert $x_2$ and $x_3$ to the first equation, we have $x_1 = 2.208$.

# Class Assignment

$$Ax = b$$

Coefficient matrix:

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

Constant vector:

$$b = \begin{bmatrix} 6 \\ 3 \\ 3 \end{bmatrix}$$

# Python

Python

```python
import numpy as np

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

b = np.array([6, 3, 3])

x = np.linalg.solve(A, b)

print(x)
```

Output:

Python

```
[2. 2. 2.]
```