1.Python Basics

In [1]:
```
from datetime import datetime
current_datetime = datetime.now()
print("Current date and time:", current_datetime.strftime("%Y-%m-%d %
```

```
Current date and time: 2024-09-18 17:02:19
```

In [2]:
```
color_list = ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']
new_list = [color_list[i] for i in range(len(color_list)) if i not in
print(new_list)
```

```
['Green', 'White', 'Black']
```

In [3]:
```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")

student1 = Student("Pang", 22)
student1.display_info()
```

```
Name: Pang
Age: 22
```

2.Linear Algebra

In [4]:
```
import numpy as np

A = np.array([[1, -2, 4], [1, -1, 1], [1, 0, 0], [1, 1, 1]])
B = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3], [1, 2, 3]])

print("A:")
print(A)
print("B:")
print(B)
```

```
A:
[[ 1 -2  4]
 [ 1 -1  1]
 [ 1  0  0]
 [ 1  1  1]]
B:
[[1 2 3]
 [1 2 3]
 [1 2 3]
 [1 2 3]]
```

In [5]:
```
print("\nSecond row of A:")
print(A[1, :])

print("\nThird column of B:")
print(B[:, 2])
```

```

Second row of A:
[ 1 -1  1]

Third column of B:
[3 3 3 3]
```

In [6]:
```
print("\nA + B:")
print(A + B)

print("\nA - B:")
print(A - B)
```

```

A + B:
[[2 0 7]
 [2 1 4]
 [2 2 3]
 [2 3 4]]

A - B:
[[ 0 -4  1]
 [ 0 -3 -2]
 [ 0 -2 -3]
 [ 0 -1 -2]]
```

In [7]:
```
new_matrix = np.hstack((A, B))
print("\nNew 4x6 matrix [A, B]:")
print(new_matrix)
```

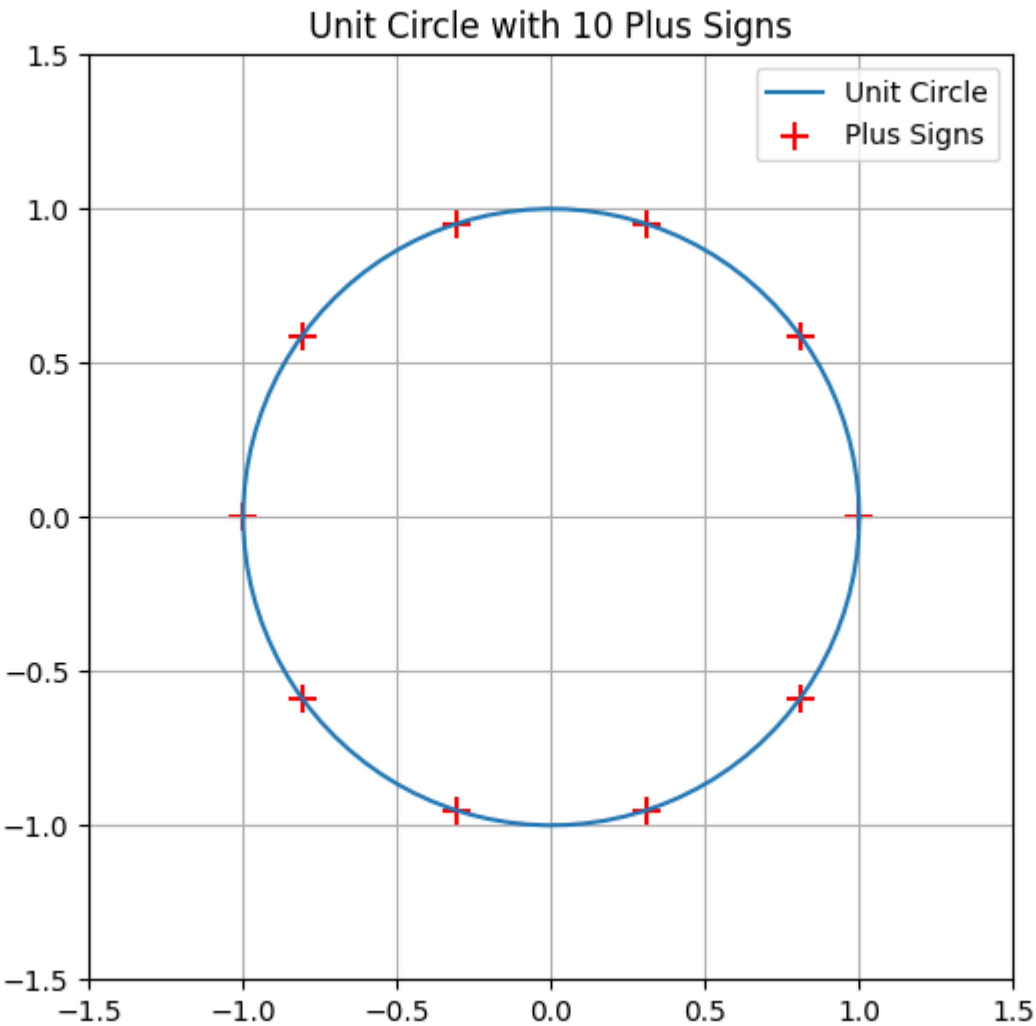
```

New 4x6 matrix [A, B]:
[[ 1 -2  4  1  2  3]
 [ 1 -1  1  1  2  3]
 [ 1  0  0  1  2  3]
 [ 1  1  1  1  2  3]]
```

In [8]:
```
AT_B = np.dot(A.T, B)
print("\nA^T * B:")
print(AT_B)
```

```

A^T * B:
[[ 4  8 12]
 [-2 -4 -6]
 [ 6 12 18]]
```

3.Matplotlib

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt

theta = np.linspace(0, 2 * np.pi, 100)
x = np.cos(theta)
y = np.sin(theta)

plt.figure(figsize=(6, 6))
plt.plot(x, y, label="Unit Circle")

num_points = 10
angles = np.linspace(0, 2 * np.pi, num_points, endpoint=False)
x_points = np.cos(angles)
y_points = np.sin(angles)

plt.scatter(x_points, y_points, color='red', marker='+', s=100, label

plt.gca().set_aspect('equal', adjustable='box')

plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)

plt.grid(True)
plt.legend()
plt.title("Unit Circle with 10 Plus Signs")

plt.show()
```



4. Column and Null Space

(1)

A can be converted to matrix:

$$\begin{bmatrix} 1 & 3 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

we can get $rank(A) = dim(col) = 2$,so the $dim(null)$= 3 - 2 = 1.

(2)

Let the column vector of the matrix be a_1, a_2, a_3,the relation of them is $a_3 = 2a_1 - a_2$,so:

$$A \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix} = 0$$

we can get the set of basis vectors for null A is

$$\begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix}$$

(3)

A set of basis vectors for $col(A)$ is:

$$\begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \\ 1 \end{bmatrix}\begin{bmatrix} -1 \\ 2 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

(4)

To judge if $col(C) = col(A)$,we should check if the column of C can be linearly represented by the column of A, we already know the set of basis vectors for $col(A)$:

$$a_1, a_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 2 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

C can be converted to:

$$\begin{bmatrix} 1 & 5 & 4 & 3 \\ 0 & 3 & 3 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$rank(C) = 2$$

$$dim(null) = 4 - 2 = 2$$

Let the column vector of the matrix be c_1, c_2, c_3, c_4.the relation of them is

$$c_1 - c_2 + c_3 = 0$$

$$c_1 - 2c_2 + 3c_4 = 0$$

we can get the set of basis vectors for $col(C)$:

$$c_1, c_2 = \begin{bmatrix} -2 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 5 \\ -1 \\ 2 \\ 5 \end{bmatrix}$$

the column of C:c_1, c_2 be linearly represented by the column of A:a_1, a_2. the relationship is:

$$[c_1, c_2] = [a_1, a_2]K$$

this relationship equal to the equation:

$$[a_1, a_2]X = [c_1, c_2]$$

has solution.then we can check the extension matrix $\begin{bmatrix} a, c \end{bmatrix}$ it can be converted to:

$$\begin{bmatrix} 1 & -1 & -2 & -1 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$rank(\begin{bmatrix} a_1, a_2, c_1, c_2 \end{bmatrix}) = rank(A) = rank(C) = 2$$

We proof that the equation has solution, and the column of C:$[c_1, c_2]$ be linearly represented by the column of A:(a_1, a_2).the column space has relationship: $col(C) = col(A)$

(5)

By inspection:

$$B = \begin{bmatrix} -1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

5.Speak the Matrix Language

(a)

In matrix language, this can be expressed as:

$$Z = TY$$

T is a matrix that represents these linear combinations. Each row of Z depends on the rows of Y from the i-th row to the n-th

(b)

In matrix language, this can be expressed as:

$$W = VP$$

P is a permutation matrix used to swap adjacent columns. The permutation matrix P represents the reordering of the columns of V.

(c)

In matrix language, this can be expressed as:

$$\text{diag}(P^T Q) > 0$$

$P^T Q$ represents the matrix of dot products between the columns of P and Q, with the diagonal values indicating the dot products of corresponding columns. The condition $\text{diag}(P^T Q) > 0$ ensures that the angle between each pair of corresponding columns is acute.

(d)

In matrix language, this can be expressed as:

$$A_1^T A_2 = 0$$

Where A_1 represents the first k columns of A, and A_2 represents the remaining columns. The equation $A_1^T A_2 = 0$ indicates that the dot product between any column of A_1 and any column of A_2 is zero, thus ensuring orthogonality.

6.Matrix Rank

(a)

$$A = aa^T = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 a_1 & a_1 a_2 & \dots & a_1 a_n \\ a_2 a_1 & a_2 a_2 & \dots & a_2 a_n \\ \dots & \dots & \dots & \dots \\ a_n a_1 & a_n a_2 & \dots & a_n a_n \end{bmatrix}$$

Assume that the rows of the matrix are $r_1, r_2 \dots r_n$, Perform transformations on matrix:

$$r_2 = r_2 - \frac{a_2}{a_1} r_1$$
$$\dots$$
$$r_n = r_n - \frac{a_n}{a_1} r_1$$
$$A = \begin{bmatrix} a_1 a_1 & a_1 a_2 & \dots & a_1 a_n \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$
$$rank(A) = 1$$

(b)

$AB = 0$ can be considered as $AX = 0$, X is a part of null space of A, n is num of the columns:

$$rank(B) \leq n - r = n - rank(A)$$
$$rank(A) + rank(B) \leq n$$

A,B are nonzero square matrices:

$$rank(A) \neq 0, rank(B) \neq 0$$

We can get :

$$rank(A) < 0, rank(B) < 0$$

(c)

If $rank(A) \neq rank(\begin{bmatrix} A & b \end{bmatrix})$, they can be translated to:

$$\begin{bmatrix} 1 & 0 & \dots & 0 & b_1 \\ 0 & 1 & \dots & 0 & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & b_{n-1} \\ 0 & 0 & \dots & 0 & b_n \end{bmatrix}$$

By the last row of matrix , the system has no solution. And if $rank(A) = rank(\begin{bmatrix} A & b \end{bmatrix})$, the matrix $\begin{bmatrix} A & b \end{bmatrix}$ can be converted to the simplest form, and get solution.

7.Ellipsoids

(a)

The ellipsoid $E_1(P, x_c)$ is defined as $E_1(P, x_c) = \{x : (x - x_c)^T P^{-1}(x - x_c) \leq 1\}$.

To express this in terms of A and x_c, start with $x = Au + x_c$, where $||u||^2 \leq 1$. Substitute x into the equation for E_1:

$$(Au + x_c - x_c)^T P^{-1}(Au + x_c - x_c) \leq 1$$

Simplifies to:

$$(Au)^T P^{-1}(Au) \leq 1$$
$$u^T A^T P^{-1} Au \leq 1$$

For this to be equivalent to $||u||^2 \leq 1$, we need:

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

Therefore, A should be such that:

$$A = \sqrt{P}$$

(b)

To solve the eigenvalues of P ,need solve the determinant of $P - \lambda I$:

$$\det(P - \lambda I) = \begin{vmatrix} 4-\lambda & 1 \\ 1 & 4-\lambda \end{vmatrix} = 0$$

This gives the solutions:

$$\lambda_1 = 3, \quad \lambda_2 = 5$$

For each eigenvalue, we compute the corresponding eigenvector:

For $\lambda_1 = 3$, the eigenvector is $v_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

For $\lambda_2 = 5$, the eigenvector is $v_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

The matrix A is constructed using the eigenvectors v_1 and v_2. The column vectors of A are scaled by the square roots of the corresponding eigenvalues:

$$A = \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{5}} \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{5}} \end{pmatrix}$$

The center of the ellipsoid remains unchanged, so:

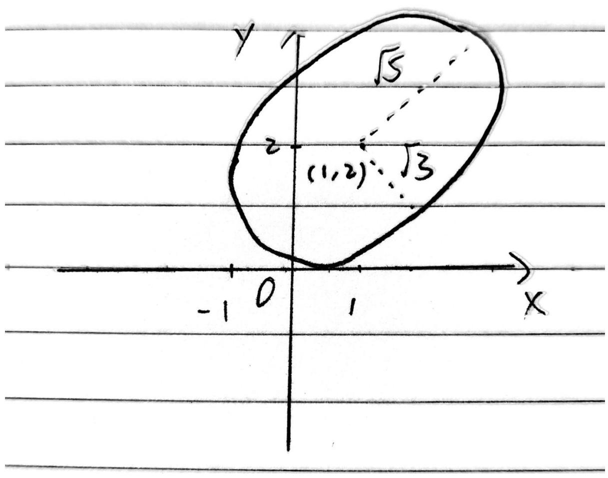
$$b = x_c = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Based on the eigenvalues and eigenvectors calculated, the lengths of the semi-axes of the ellipsoid are $\sqrt{3}$ and $\sqrt{5}$, and their directions are given by:

axis 1: Length = $\sqrt{3}$, direction = $v_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

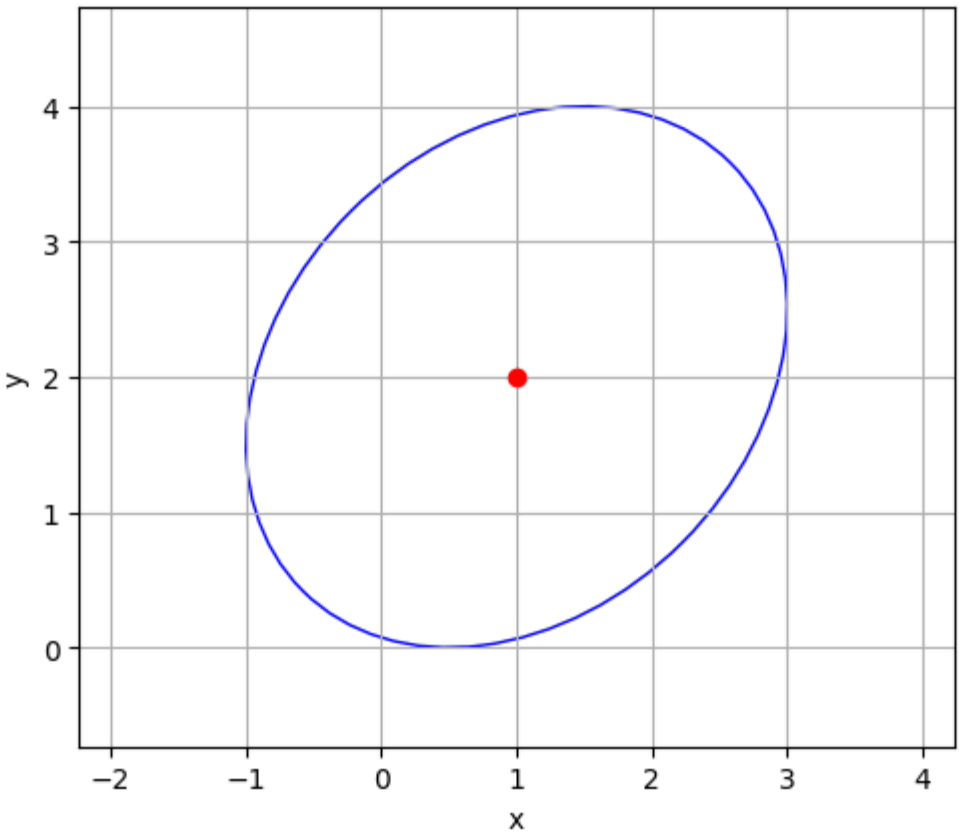
axis 2: Length = $\sqrt{5}$, direction = $v_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

The center of the ellipsoid is located at $(1, 2)$.



(c)

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse

center = np.array([1, 2])
eigenvalues = np.array([3, 5])
eigenvectors = np.array([[1, 1], [1, -1]])

semi_axes = np.sqrt(eigenvalues)

angle = np.arctan2(eigenvectors[1, 0], eigenvectors[0, 0]) * 180 / np

fig, ax = plt.subplots(subplot_kw={'aspect': 'equal'})

ellipse = Ellipse(xy=center, width=2*semi_axes[1], height=2*semi_axes

ax.add_patch(ellipse)
ax.plot(center[0], center[1], 'ro')
ax.set_xlim(center[0] - semi_axes[1] - 1, center[0] + semi_axes[1] +
ax.set_ylim(center[1] - semi_axes[0] - 1, center[1] + semi_axes[0] +
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.grid(True)

plt.show()
```



7.Polyhedron

(a)

The intersection of the two polyhedra $P_1 \cap P_2$, can be expressed as a new polyhedron with combined inequalities. That is:

$$P_1 \cap P_2 = \{x \in \mathbb{R}^n : A_1 x \leq b_1 \text{ and } A_2 x \leq b_2\}$$

In matrix form, this can be written as:

$$P_1 \cap P_2 = \{x \in \mathbb{R}^n : \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} x \leq \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\}$$

(b)

In [5]:
```
import numpy as np
from scipy.optimize import linprog

A1 = np.array([[0, 1], [5, -2], [-1, -2], [-4, -2]])
b1 = np.array([7, 36, -14, -26])

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

A_combined = np.vstack([A1, a])
b_combined = np.hstack([b1, b_halfspace])

c = np.zeros(2)

res = linprog(c, A_ub=A_combined, b_ub=b_combined, method='highs')

if res.success:
    print("P1 intersects with the half-space.")
else:
    print("P1 does not intersect with the half-space.")
```

```
P1 does not intersect with the half-space.
```