Linear Algebra Functions ใน NumPy

NumPy มีโมดูลย่อยชื่อ numpy.linalg ซึ่งรวบรวมฟังก์ชันสำหรับการคำนวณทาง Linear Algebra (พีชคณิตเชิงเส้น) เช่น การหาดีเทอร์มิแนนต์ (determinant), การหาอินเวอร์ส (inverse), การแก้ระบบสมการเชิงเส้น (solving linear equations), ค่าลักษณะ (Eigenvalues/Eigenvectors)และอื่นๆ

1. การคำนวณดีเทอร์มิแนนต์ (Determinant)

- ฟังก์ชัน: numpy.linalg.det()
- คำอธิบาย: ใช้หาดีเทอร์มิแนนต์ของเมทริกซ์(สามารถใช้กับ square matrix เท่านั้น)
- ตัวอย่าง:

In [1]:
```
import numpy as np

# สร้างเมทริกซ์ 3x3
B = np.array([[1, 2, 3],
              [0, 4, 5],
              [1, 0, 6]])

# คำนวณ Determinant ของ B
det_B = np.linalg.det(B)
print("เมทริกซ์ B:")
print(B)
print("\nDeterminant ของ B:")
print(det_B)
```

```
เมทริกซ์ B:
[[1 2 3]
 [0 4 5]
 [1 0 6]]

Determinant ของ B:
22.000000000000004
```

2. การหาอินเวอร์ส (Inverse)

- ฟังก์ชัน: numpy.linalg.inv()
- คำอธิบาย: ใช้อินเวอร์สของเมทริกซ์ (เมทริกซ์ที่คูณกับเมทริกซ์เดิมแล้วได้เมทริกซ์เอกลักษณ์)
- เงื่อนไข: เมทริกซ์ต้องเป็น square matrix และ เมทริกซ์ต้องเป็น non-singular (มี determinant ไม่เป็น 0)
- ตัวอย่าง:

In [2]:
```python
import numpy as np

# สร้างเมทริกซ์ 2x2
A = np.array([[4, 7],
              [2, 6]])

# คำนวณเมทริกซ์ผกผันของ A
A_inv = np.linalg.inv(A)
print("เมทริกซ์ A:")
print(A)
print("\nเมทริกซ์ผกผันของ A:")
print(A_inv)

# ตรวจสอบว่า A * A_inv = Identity Matrix
identity = np.dot(A, A_inv)
print("\nA * A_inv (ควรได้ Identity):")
print(identity)
```

```
เมทริกซ์ A:
[[4 7]
 [2 6]]

เมทริกซ์ผกผันของ A:
[[ 0.6 -0.7]
 [-0.2  0.4]]

A * A_inv (ควรได้ Identity):
[[ 1.00000000e+00 -1.11022302e-16]
 [ 1.11022302e-16  1.00000000e+00]]
```

3. การแก้ระบบสมการเชิงเส้น (Solving Linear Equations)

- ฟังก์ชัน: numpy.linalg.solve()
- คำอธิบาย: ใช้แก้ระบบสมการเชิงเส้นในรูปแบบ Ax=b โดยที่ A เป็นเมทริกซ์สัมประสิทธิ์ และ b เป็นเวกเตอร์ผลลัพธ์
- ตัวอย่าง:

In [3]:
```python
import numpy as np

# กำหนดระบบสมการ: 2x2
#  3x +  y = 9
#  x  + 2y = 8
A = np.array([[3, 1],
              [1, 2]])
b = np.array([9, 8])

# แก้สมการ Ax = b
x = np.linalg.solve(A, b)
print("คำตอบของระบบสมการ Ax = b:")
print(x)
```

```
คำตอบของระบบสมการ Ax = b:
[2. 3.]
```

4. การหาค่าไอเกน (Eigenvalues และ Eigenvectors)

- ฟังก์ชัน: numpy.linalg.eig()
- คำอธิบาย: ใช้หาค่าไอเกน (eigenvalues) และเวกเตอร์ไอเกน (eigenvectors) ของเมทริกซ์
- ตัวอย่าง:

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

# สร้างเมทริกซ์ 2x2
C = np.array([[3, 1],
              [0, 2]])

# คำนวณ Eigenvalues และ Eigenvectors ของ C
eigenvalues, eigenvectors = np.linalg.eig(C)
print("เมทริกซ์ C:")
print(C)
print("\nEigenvalues ของ C:")
print(eigenvalues)
print("\nEigenvectors ของ C (ในแต่ละคอลัมน์):")
print(eigenvectors)
```

```
เมทริกซ์ C:
[[3 1]
 [0 2]]

Eigenvalues ของ C:
[3. 2.]

Eigenvectors ของ C (ในแต่ละคอลัมน์):
[[ 1.         -0.70710678]
 [ 0.          0.70710678]]
```

5. การคำนวณ Rank ของเมทริกซ์

- ฟังก์ชัน: numpy.linalg.matrix_rank()
- คำอธิบาย: ใช้หาค่า Rank ของเมทริกซ์ (จำนวนแถวหรือคอลัมน์ที่อิสระเชิงเส้น)
- ตัวอย่าง:

In [5]:
```
A = np.array([[1, 2], [2, 4]])
rank_A = np.linalg.matrix_rank(A)
print("Rank of A:", rank_A)
```

```
Rank of A: 1
```

6. การคำนวณ Trace ของเมทริกซ์

- ฟังก์ชัน: numpy.trace()
- คำอธิบาย: ใช้หาผลรวมของสมาชิกบนเส้นทแยงมุมหลักของเมทริกซ์
- ตัวอย่าง:

In [6]:
```
A = np.array([[1, 2], [3, 4]])
trace_A = np.trace(A)
print("Trace of A:", trace_A)
```

```
Trace of A: 5
```

7. การคำนวณ Dot Product

- ฟังก์ชัน: numpy.dot()
- คำอธิบาย: ใช้คำนวณผลคูณจุด (dot product) ของเวกเตอร์หรือเมทริกซ์
- ตัวอย่าง:

In [7]:
```
import numpy as np

# ตัวอย่าง 1: dot product ระหว่างเวกเตอร์ 1 มิติ
v1 = np.array([1, 2, 3])
v2 = np.array([4, 5, 6])
dot_product = np.dot(v1, v2)
print("Dot product ของเวกเตอร์ v1 และ v2 =", dot_product)
# คำนวณ: 1*4 + 2*5 + 3*6 = 32

# ตัวอย่าง 2: ผลคูณของเมทริกซ์ 2 มิติ
A = np.array([[1, 2],
              [3, 4]])
B = np.array([[5, 6],
              [7, 8]])
C = np.dot(A, B)
print("A:")
print(A)
print("\nB:")
print(B)
print("\nผลคูณของเมทริกซ์ A และ B โดยใช้ np.dot:")
print(C)
# คำนวณ:
# C[0,0] = 1*5 + 2*7 = 19, C[0,1] = 1*6 + 2*8 = 22
# C[1,0] = 3*5 + 4*7 = 43, C[1,1] = 3*6 + 4*8 = 50
```

```
Dot product ของเวกเตอร์ v1 และ v2 = 32
A:
[[1 2]
 [3 4]]

B:
[[5 6]
 [7 8]]

ผลคูณของเมทริกซ์ A และ B โดยใช้ np.dot:
[[19 22]
 [43 50]]
```

In [8]:
```python
import numpy as np

# กำหนดเมทริกซ์ A ขนาด 3x3
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# กำหนดเมทริกซ์ B ขนาด 3x3
B = np.array([[9, 8, 7],
              [6, 5, 4],
              [3, 2, 1]])

# ผลคูณของเมทริกซ์โดยใช้ np.dot
C_dot = np.dot(A, B)

# ผลคูณโดยใช้ operator @
C_at = A @ B

# ผลคูณโดยใช้ np.matmul
C_matmul = np.matmul(A, B)

print("A:")
print(A)
print("\nB:")
print(B)
print("\nผลคูณของ A และ B โดย np.dot:")
print(C_dot)
# print("\nผลคูณของ A และ B โดย operator '@':")
# print(C_at)
# print("\nผลคูณของ A และ B โดย np.matmul:")
# print(C_matmul)
```

```
A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

B:
[[9 8 7]
 [6 5 4]
 [3 2 1]]

ผลคูณของ A และ B โดย np.dot:
[[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]
```

In [9]:

```python
import numpy as np

# กำหนดเมทริกซ์ A ขนาด 3x4
A = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])

# กำหนดเมทริกซ์ B ขนาด 4x2
B = np.array([[1, 2],
              [3, 4],
              [5, 6],
              [7, 8]])

# ผลคูณของเมทริกซ์โดยใช้ np.dot
C_dot = np.dot(A, B)

# ผลคูณโดยใช้ operator @
C_at = A @ B

# ผลคูณโดยใช้ np.matmul
C_matmul = np.matmul(A, B)

print("A (3x4):")
print(A)
print("\nB (4x2):")
print(B)
print("\nผลคูณของ A และ B (3x2) โดย np.dot:")
print(C_dot)
# print("\nผลคูณของ A และ B โดย operator '@':")
# print(C_at)
# print("\nผลคูณของ A และ B โดย np.matmul:")
# print(C_matmul)
```

```
A (3x4):
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

B (4x2):
[[1 2]
 [3 4]
 [5 6]
 [7 8]]

ผลคูณของ A และ B (3x2) โดย np.dot:
[[ 50  60]
 [114 140]
 [178 220]]
```

In [ ]: