Lab3_Sol_A

January 31, 2025

[28]:
```python
import sympy as sp

# 3.1 Basic Matrix Operations
# Matrix Creation
A = sp.Matrix([[1, 2], [3, 4]])
B = sp.Matrix([[2, 0], [1, 3]])

# Print matrices
print("Matrix A:")
sp.pprint(A)

print("Matrix B:")
sp.pprint(B)

# Addition and Subtraction
print("A + B:")
sp.pprint(A + B)

print("\nA - B:")
sp.pprint(A - B)

# Matrix Multiplication
print("A * B:")
sp.pprint(A * B)

# Scalar Multiplication
print("3 * A:")
sp.pprint(3 * A)

# Transpose
print("Transpose of A:")
sp.pprint(A.T)

# Determinant
print("Determinant of A:")
sp.pprint(A.det())
```

```
# Inverse
print("Inverse of A:")
sp.pprint(A.inv())
```

```
Matrix A:
1  2

3  4
Matrix B:
2  0

1  3
A + B:
3  2

4  7

A - B:
-1  2

2   1
A * B:
4   6

10  12
3 * A:
3  6

9  12
Transpose of A:
1  3

2  4
Determinant of A:
-2
Inverse of A:
-2    1

3/2  -1/2
```

[29]:
```
# 3.2 Advanced Matrix Operations
#3.2.1 Symbolic Matrix Operations
x, y = sp.symbols('x y')
M = sp.Matrix([[x, 1], [2, y]])

print("Determinant of M:")
sp.pprint(M.det())
```

```
Determinant of M:
x y - 2
```

[36]:
```
#3.2.2 Solving Matrix Equations
M1 = sp.Matrix([[2, 3], [1, 4]])
M2 = sp.Matrix([[x], [y]])
M3 = sp.Matrix([5, 6])
equation = sp.Eq(M1 * M2, M3)
solution = sp.solve(equation, (x, y))
sp.pprint(solution)
```

```
{x: 2/5, y: 7/5}
```

[31]:
```
#3.2.3 Matrix Equation
lhs_matrix = sp.Matrix([[x + y, 2], [3, x - y]])
rhs_matrix_eq = sp.Matrix([[4, 2], [3, 1]])

print("\nSolution to the matrix equation [x+y 2; 3 x-y] = [4 2; 3 1]:")
equation_solutions = sp.solve(lhs_matrix - rhs_matrix_eq, (x, y))
sp.pprint(equation_solutions)
```

```
Solution to the matrix equation [x+y 2; 3 x-y] = [4 2; 3 1]:
{x: 5/2, y: 3/2}
```

[32]:
```
#3.2.4 Matrix Exponentiation using Loops
n = 3  # Example power
result = A
for _ in range(1, n):
    result = result * A

print(f"\nA^{n} using loops:")
sp.pprint(result)
```

```
A^3 using loops:
 37  54

 81  118
```

[33]:
```
#3.2.5 Generating Fibonacci Sequence using Matrix Multiplication
def fibonacci(n):
    F = sp.Matrix([[1, 1], [1, 0]])
    result = sp.eye(2)  # Identity matrix
    for _ in range(n - 1):
        result = result * F
    return result[0, 0]

n = 10  # Example Fibonacci number to compute
```

```
print(f"\nFibonacci({n}):")
sp.pprint(fibonacci(n))
```

```
Fibonacci(10):
55
```

[34]:
```
# 3.3 Comparison with NumPy
import numpy as np

# 3.3.1Compare determinant of A
A_np = np.array([[1, 2], [3, 4]])
det_sympy = A.det()
det_numpy = np.linalg.det(A_np)

print("Determinant comparison:")
print(f"SymPy: {det_sympy}")
print(f"NumPy: {det_numpy}")
```

```
Determinant comparison:
SymPy: -2
NumPy: -2.0000000000000004
```

[26]:
```
#3.3.2 Compare inverse of A
inv_sympy = A.inv()
inv_numpy = np.linalg.inv(A_np)

print("Inverse comparison:")
print(f"SymPy:\n{inv_sympy}")
print(f"NumPy:\n{inv_numpy}")
```

```
Inverse comparison:
SymPy:
Matrix([[-2, 1], [3/2, -1/2]])
NumPy:
[[-2.   1. ]
 [ 1.5 -0.5]]
```

[35]:
```
#3.3.3 Demonstrate symbolic inversion
a, b, c, d = sp.symbols('a b c d')
M_symbolic = sp.Matrix([[a**2, b**3], [c**2, d**4]])
print("\nInverse of symbolic matrix M:")
sp.pprint(M_symbolic.inv())
```

```
Inverse of symbolic matrix M:
      4              3
     d             -b
```

```
 2  4    3  2   2  4    3  2
a  d  - b  c   a  d  - b  c

      2             2
    -c              a

 2  4    3  2   2  4    3  2
a  d  - b  c   a  d  - b  c
```

[ ]:

Lab3_Sol_B

January 30, 2025

[65]:
```python
import sympy as sp

# 3.1 Basic Matrix Operations
# Matrix Creation
A = sp.Matrix([[2, 5], [8, 10]])
B = sp.Matrix([[-2, 5], [-1, 8]])

# Print matrices
print("Matrix A:")
sp.pprint(A)

print("Matrix B:")
sp.pprint(B)

# Addition and Subtraction
print("A + B:")
sp.pprint(A + B)

print("\nA - B:")
sp.pprint(A - B)

# Matrix Multiplication
print("A * B:")
sp.pprint(A * B)

# Scalar Multiplication
print("3 * A:")
sp.pprint(3 * A)

# Transpose
print("Transpose of A:")
sp.pprint(A.T)

# Determinant
print("Determinant of A:")
sp.pprint(A.det())
```

```
# Inverse
print("Inverse of A:")
sp.pprint(A.inv())
```

```
Matrix A:
 2  5

 8  10
Matrix B:
 -2  5

 -1  8
A + B:
 0  10

 7  18

A - B:
 4  0

 9  2
A * B:
 -9   50

 -26  120
3 * A:
 6   15

 24  30
Transpose of A:
 2  8

 5  10
Determinant of A:
-20
Inverse of A:
 -1/2   1/4

 2/5   -1/10
```

[66]:
```
# 3.2 Advanced Matrix Operations
#3.2.1 Symbolic Matrix Operations
x, y = sp.symbols('x y')
A = sp.Matrix([[x, y], [1, x+y]])
B = sp.Matrix([[2*x+1, y+1], [2*x, x+2*y]])

print("Determinant of M:\n")
```

```
sp.pprint(B.T*A.T)
print("Determinant of M:\n")
sp.pprint((A*B).T)
```

```
Determinant of M:

  2 x y + x (2 x + 1)      2 x (x + y) + 2 x + 1

x (y + 1) + y (x + 2 y)  y + (x + y) (x + 2 y) + 1
Determinant of M:

  2 x y + x (2 x + 1)      2 x (x + y) + 2 x + 1

x (y + 1) + y (x + 2 y)  y + (x + y) (x + 2 y) + 1
```

[67]:
```
#3.2.2 Solving Matrix Equations
M1 = sp.Matrix([[4, 6], [1, 2]])
M2 = sp.Matrix([[x], [y]])
M3 = sp.Matrix([26, 8])
equation = sp.Eq(M1 * M2, M3)
solution = sp.solve(equation, (x, y))
sp.pprint(solution)
```

```
{x: 2, y: 3}
```

[70]:
```
#3.2.3 Matrix Equation
lhs_matrix = sp.Matrix([[2*x - y, 2], [3, x -y**2]])
rhs_matrix_eq = sp.Matrix([[4, 2], [3, 1]])

print("\n")
equation_solutions = sp.solve(lhs_matrix - rhs_matrix_eq, (x, y))
sp.pprint(equation_solutions)
```

```
 17   √17  1   √17   √17   17  1   √17
  -   ,   -   ,     +   ,   +
 8     8   4    4     8    8   4    4
```

[69]:
```
import sympy as sp

# Define the variable x
x = sp.symbols('x')

# Define the matrix M
M = sp.Matrix([
    [x**2+x-13,  1],
    [x-12, x**2]
])
```

```
# Compute the determinant
det_M = M.det()

# Solve for x when determinant is zero
solutions = sp.solve(det_M, x)

# Display results
print("Determinant of M:", det_M)
print("Values of x such that det(M) = 0:", solutions)
```

```
Determinant of M: x**4 + x**3 - 13*x**2 - x + 12
Values of x such that det(M) = 0: [-4, -1, 1, 3]
```

[32]:
```
#3.2.4 Matrix Exponentiation using Loops
n = 3  # Example power
result = A
for _ in range(1, n):
    result = result * A

print(f"\nA^{n} using loops:")
sp.pprint(result)
```

```
A^3 using loops:
 37  54

 81  118
```

[48]:
```
#3.2.5 Generating Fibonacci Sequence using Matrix Multiplication
def fibonacci(n):
    F = sp.Matrix([[1, 1], [1, 0]])
    result = sp.eye(2)  # Identity matrix
    for _ in range(n - 1):
        result = result * F
    return result[0, 0]+result[0,1]

n = 10  # Example Fibonacci number to compute
print(f"\nFibonacci({n}):")
sp.pprint(fibonacci(4))
```

```
Fibonacci(10):
5
```

[ ]:
```

```