

Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 9

In the last lecture, we discussed:

- * Insertion and deletion of elements in a one-dimensional array.
- * Insertion and deletion of rows and columns in a two-dimensional array.
- * Arithmetic operations on arrays.
- * Commands to compute the shape (size) and number of dimensions of an array.

1. Array Operations: Sum, Product, Mean, Median

We will continue with operations on arrays. Given an array `v`, you may want to find the sum of its elements.

Python

```
print(v)
```

```
# Example output: [ 1  2  3  4  5  6  7  8  9 10]
```

To find the sum, instead of writing a loop as in C/C++, Python (NumPy) makes it easier:

Python

```
np.sum(v)
```

To find the product of all elements:

Python

```
np.prod(v)
```

```
# This is 10! (10 factorial) = 3,628,800
```

To find the average (mean):

Python

```
np.mean(v) # Output: 5.5
```

To find the median:

Python

```
np.median(v) # Output: 5.5
```

2. Vector Norm and Dot Product

Another important operation is finding the norm (magnitude) of a vector. The norm is calculated as the square root of the sum of the squares of its elements:

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2}$$

You can compute this manually:

Python

```
np.sqrt(np.sum(v * v))
```

There is also a dedicated command in the NumPy linear algebra (`linalg`) sub-library:

Python

```
np.linalg.norm(v)
```

Both methods will give the same result.

To compute the dot product of two vectors `v` and `w`:

Python

```
w = np.array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])
```

```
print(w)
```

```
# Dot product: 1*10 + 2*9 + ... + 10*1 = 220
```

```
np.dot(v, w) # Output: 220
```

3. Matrix Operations

We can perform various matrix operations. Let's use a matrix `m` for examples.

Python

```
print(m)
```

```
# Example output:
```

```
# [[1 2 3]
```

```
# [4 5 6]
```

```
# [7 8 9]]
```

a) Transpose:

The transpose of a matrix is found simply:

Python

```
m.T
```

b) Determinant:

The determinant is calculated using the `linalg` module.

Python

```
np.linalg.det(m) # Output: ~0 (singular matrix)
```

c) Inverse:

The inverse is also found in the 'linalg' module. Since 'm' is singular (determinant=0), let's create an invertible matrix 'n'.

Python

Create a 3x3 identity matrix

```
I = np.eye(3)
```

```
print(I)
```

Output:

```
# [[1. 0. 0.]
```

```
# [0. 1. 0.]
```

```
# [0. 0. 1.]]
```

```
n = m + I
```

```
print(n)
```

Output:

```
# [[ 2.  2.  3.]
```

```
# [ 4.  6.  6.]
```

```
# [ 7.  8. 10.]]
```

Find the inverse of n

```
np.linalg.inv(n)
```

d) Matrix Rank:

The rank of a matrix is found with:

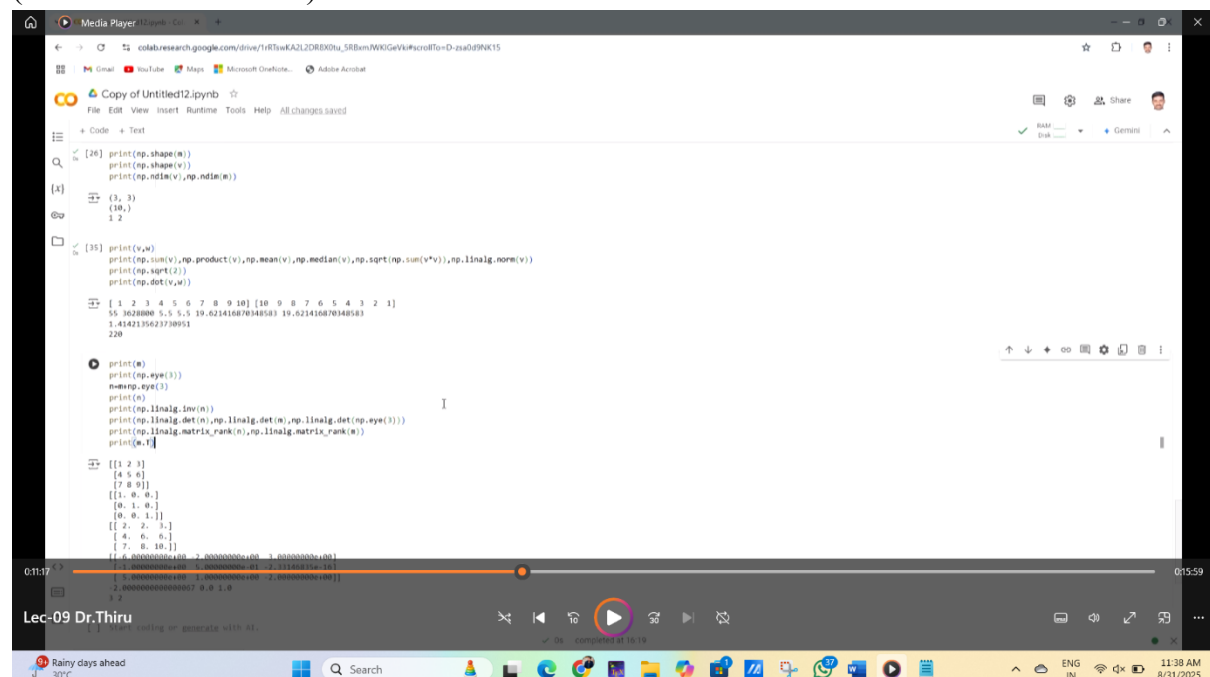
Python

```
np.linalg.matrix_rank(m) # Output: 2
```

```
np.linalg.matrix_rank(n) # Output: 3
```

```
'''
```

(Refer slide time 11:17)



4. Conditional Statements (`if`, `elif`, `else`)

Conditional statements control the flow of a program based on whether conditions are `True` or `False`. The keywords are `if`, `elif` (else-if), and `else`.

Basic Syntax:

Python

if condition:

 # Command 1

 # Command 2

 # ... (all indented commands run if the condition is True)

else:

 # Command 3

 # Command 4

 # ... (all indented commands run if the condition is False)

Code here runs after the if-else block

Example: Comparing Two Numbers

Python

a = 4

b = 5

if a > b:

 print("a is big")

else:

 print("b is big") # This will be printed

Importance of Indentation:

In Python, indentation (spaces at the beginning of a line) defines code blocks. Unlike C/C++ (which uses `{}`) or MATLAB (which uses `end`), Python uses indentation to group statements. This is mandatory.

The `elif` Statement:

To check for multiple conditions, use `elif`.

Python

a = 5

b = 5

if a > b:

 print("a is big")

elif a == b: # Use == for comparison, = is for assignment

 print("a and b are equal") # This will be printed

else:

 print("b is big")

Comparison and Logical Operators:

- * Comparison Operators:
 - * Equal to: `==`
 - * Not equal to: `!=`
 - * Greater than: `>`
 - * Less than: `<`
 - * Greater than or equal to: `>=`
 - * Less than or equal to: `<=`
- * Logical Operators:
 - * `and`: True only if both conditions are True.
 - * `or`: True if at least one condition is True.
 - * `not`: Inverts the Boolean value.

(Refer slide time 24:40)

CONDITIONAL STATEMENTS: if, elif, else.

```
if condition:  
    ↳ Command 1  
    ↳ Command 2  
    ↳  
    ↳ Command l  
else:  
    ↳ Command l+1  
    ↳  
    ↳ Command l+m  
Next statement
```

```
if condition 1:  
    ↳ Routine 1  
elif condition 2:  
    ↳ Routine 2  
    ↳  
elif condition 27:  
    ↳ Routine 27  
else:  
    ↳ Routine 28
```

⊕: Assignment, ⊖: checking
>, <, >=, <=, != → Conditional operators
or, and → logical operators

Examples with Logical Operators:

Python

a, b, c = 4, 5, 6

Using 'and'

```
if (a > b) and (a > c):
```

```
    print("a is the biggest")
```

```
else:
```

```
    print("a is not the biggest") # This will be printed
```

Using 'or'

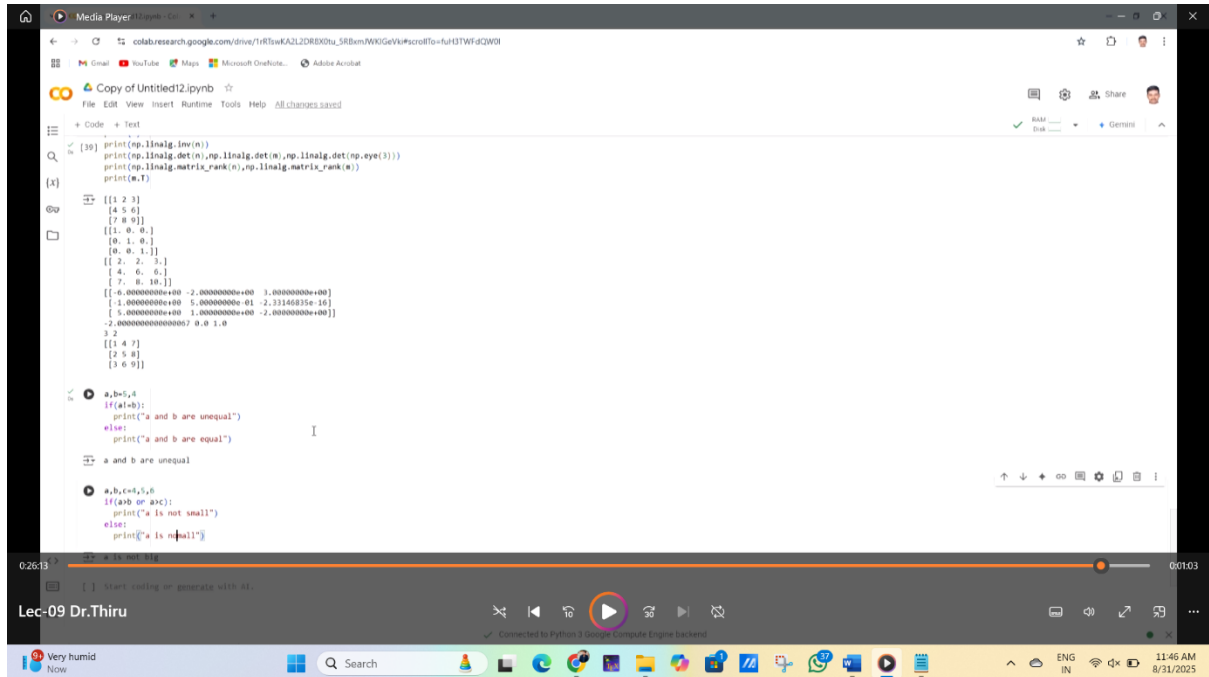
if (a > b) or (a > c):

print("a is greater than at least one number")

else:

print("a is the smallest") # This will be printed

(Refer slide time 26:13)



The screenshot shows a Jupyter Notebook titled 'Copy of Untitled12.ipynb' running in a web browser. The notebook contains three code cells. The first cell defines a 3x3 matrix 'x' and prints its transpose, determinant, inverse, and rank. The second cell defines variables 'a' and 'b' and uses an 'if-else' statement to print a comparison. The third cell defines variables 'a', 'b', 'c', and 'd' and uses an 'if-elif-else' statement to print a comparison. The output of the first cell shows the transpose, determinant, inverse, and rank of the matrix 'x'. The output of the second cell shows 'a and b are unequal'. The output of the third cell shows 'a is not small'.

```
print(np.linalg.inv(n))
print(np.linalg.det(n), np.linalg.det(m), np.linalg.det(np.eye(3)))
print(np.linalg.matrix_rank(n), np.linalg.matrix_rank(m))
print(m.T)

[[[ 2.  3.]
  [ 4.  5.  8.]
  [ 7.  8.  9.]]
 [[ 1.  0.  0.]
  [ 0.  1.  0.]
  [ 0.  0.  1.]]
 [[ 2.  2.  3.]
  [ 4.  6.  6.]
  [ 7.  8. 10.]]
 [[-0.00000000e+00 -2.00000000e+00  3.00000000e+00]
 [ 1.00000000e+00  5.00000000e-01  2.33468335e-16]
 [ 5.00000000e+00  1.00000000e+00 -2.00000000e+00]]
 3.2
 [[ 1.  4.  7.]
  [ 2.  5.  8.]
  [ 3.  6.  9.]]

a,b=5,4
if(a>b):
    print("a and b are unequal")
else:
    print("a and b are equal")

a and b are unequal

a,b,c=4,5,6
if(a>b or b>c):
    print("a is not small")
else:
    print("a is small")

a is not small
```

Summary:

In this lecture, we covered:

- * Array operations: `sum`, `prod`, `mean`, `median`, `norm`, `dot`.
- * Matrix operations: transpose (`T`), determinant (`det`), inverse (`inv`), rank(`matrix_rank`).
- * Conditional statements: `if`, `elif`, `else`, and the use of comparison and logical operators.

In the next lecture, we will look at loops and functions. Thank you
