TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A LAB REPORT ON

NUMPY

Lab No: 07
Experiment Date: 05 - 11
Submission Date: 05- 11

SUBMITTED BY:
Name: Ishant yadav
Group : B
Roll no: 081BEL036

SUBMITTED TO:
Department of
Electrical Engineering

NumPy (Numerical Python) is the fundamental package for scientific computing in Python. It provides a powerful N-dimensional array object, sophisticated broadcasting functions, tools for integrating C/C+ + and Fortran code, and useful linear algebra, Fourier transform, and random number capabilities.

This laboratory report demonstrates the implementation and documentation of NumPy's built-in functions, along with practical examples from the official NumPy documentation.

Objective

The primary objectives of this laboratory exercise are:

1. Implement and document all built-in functions of NumPy
2. Execute and analyze examples from the NumPy QuickStart guide
3. Execute and analyze examples from the NumPy Absolute Beginners guide
4. Demonstrate proficiency in NumPy array operations and manipulations

---

Part 1: NumPy Built-in Functions Implementation

1.1 Array Creation Functions

np.array()

Purpose: Creates an array from a list or tuple

Syntax: numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

```
import numpy as np
# Example implementation
def create_array_example():

    """Demonstrates np.array() functionality"""
    # From list
    arr1 = np.array([1, 2, 3, 4, 5])
    print(f"1D array: {arr1}")
    # From nested list (2D)
    arr2 = np.array([[1, 2, 3], [4, 5, 6]])
    print(f"2D array:\n{arr2}")
    # With specific dtype
    arr3 = np.array([1, 2, 3], dtype=float)
    print(f"Float array: {arr3}")
    return arr1, arr2, arr3



# Execute
result = create_array_example()
```

np.zeros()

Purpose: Returns a new array of given shape and type, filled with zeros

Syntax: numpy.zeros(shape, dtype=float, order='C')

```
def zeros_example():

    """Demonstrates np.zeros() functionality""" # 1D array of zeros zeros_1d = np.zeros(5) print(f"1D zeros: {zeros_1d}") # 2D array of zeros zeros_2d = np.zeros((3, 4)) print(f"2D zeros:\n{zeros_2d}") # Integer zeros zeros_int = np.zeros(5, dtype=int) print(f"Integer zeros: {zeros_int}") return zeros_1d, zeros_2d, zeros_int




# Execute
zeros_result = zeros_example()
```

np.ones()

Purpose: Returnsa newarrayof givenshape andtype,filledwithones

Syntax:numpy.ones(shape, dtype=None, order='C')

```
def ones_example():
    """Demonstrates np.ones() functionality"""
    # 1D array of ones
    ones_1d = np.ones(4)
    print(f"1D ones: {ones_1d}")
    # 2D array of ones
    ones_2d = np.ones((2, 3))
    print(f"2D ones:\n{ones_2d}")
    return ones_1d, ones_2d



# Execute
ones_result = ones_example()
```

np.full()

Purpose: Returns a new array of given shape and type, filled with fill_value

Syntax: numpy.full(shape, fill_value, dtype=None, order='C')

```
def full_example():

    """Demonstrates np.full() functionality"""
    # Array filled with 7
    full_arr = np.full((2, 3), 7)
    print(f"Full array with 7:\n{full_arr}")
    # Array filled with string
    full_str = np.full(4, 'hello')
```

```
    print(f"Full string array: {full_str}")
    return full_arr, full_str


# Execute
full_result = full_example()
```

1.2 Array Range Functions

np.arange()

Purpose: Returns evenly spaced values within a given interval
Syntax: numpy.arange([start,] stop[, step,], dtype=None)

```
def arange_example():
    """Demonstrates np.arange() functionality"""
    # Simple range
    range1 = np.arange(10)
    print(f"Range 0-9: {range1}")
    # Range with start and stop
    range2 = np.arange(5, 15)
    print(f"Range 5-14: {range2}")
    # Range with step
    range3 = np.arange(0, 20, 2)
    print(f"Even numbers 0-18: {range3}")
    # Float range
    range4 = np.arange(0, 5, 0.5)
    print(f"Float range: {range4}")
    return range1, range2, range3, range4




# Execute
arange_result = arange_example()
```

np.linspace()

Purpose: Returns evenly spaced numbers over a specified interval
Syntax: numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

```
def linspace_example():
    """Demonstrates np.linspace() functionality"""
    # Linear space
    lin1 = np.linspace(0, 10, 5)
    print(f"Linear space 0-10 (5 points): {lin1}")
    # Without endpoint
    lin2 = np.linspace(0, 10, 5, endpoint=False)
    print(f"Without endpoint: {lin2}")
    # With step information
    lin3, step = np.linspace(0, 10, 5, retstep=True)
    print(f"With step info: {lin3}, step: {step}")
```

```
    return lin1, lin2, lin3

# Execute
linspace_result = linspace_example()
```

1.3 Mathematical Functions

np.add()

Purpose: Element-wise addition of array elements

Syntax: numpy.add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None)

```
def mathematical_functions_example():
    """Demonstrates various mathematical functions"""
    arr1 = np.array([1, 2, 3, 4, 5])
    arr2 = np.array([6, 7, 8, 9, 10])
    # Addition
    add_result = np.add(arr1, arr2)
    print(f"Addition: {add_result}")
    # Subtraction
    sub_result = np.subtract(arr1, arr2)
    print(f"Subtraction: {sub_result}")
    # Multiplication
    mul_result = np.multiply(arr1, arr2)
    print(f"Multiplication: {mul_result}")
    # Division
    div_result = np.divide(arr2, arr1)
    print(f"Division: {div_result}")
    # Power
    pow_result = np.power(arr1, 2)
    print(f"Power (squared): {pow_result}")
    # Square root
    sqrt_result = np.sqrt(arr1)
    print(f"Square root: {sqrt_result}")
    return add_result, sub_result, mul_result, div_result, pow_result, sqrt_result

# Execute
math_result = mathematical_functions_example()
```

1.4 Statistical Functions

Statistical Operations

Purpose: Performstatisticalcalculationsonarrays

```
def statistical_functions_example():
    """Demonstrates statistical functions"""
```

```
    data = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]) print(f"Data:\n{data}") # Mean mean_all = np.mean(data) mean_axis0 = np.mean(data, axis=0) mean_axis1 = np.mean(data, axis=1) print(f"Mean (all): {mean_all}") print(f"Mean (axis=0): {mean_axis0}") print(f"Mean (axis=1): {mean_axis1}") # Standard deviation std_all = np.std(data) print(f"Standard deviation: {std_all}") # Variance var_all = np.var(data) print(f"Variance: {var_all}") # Min and Max min_val = np.min(data) max_val = np.max(data) print(f"Min: {min_val}, Max: {max_val}") # Sum sum_all = np.sum(data) sum_axis0 = np.sum(data, axis=0) print(f"Sum (all): {sum_all}") print(f"Sum (axis=0): {sum_axis0}") return mean_all, std_all, var_all, min_val, max_val, sum_all
```

```
# Execute
stats_result = statistical_functions_example()
```

1.5 Array Manipulation Functions

Reshaping and Transposing

```
def array_manipulation_example():
    """Demonstrates array manipulation functions"""
    # Original array
    original = np.arange(12)
    print(f"Original array: {original}")

    # Reshape
    reshaped = original.reshape(3, 4)
    print(f"Reshaped (3x4):\n{reshaped}")

    # Transpose
    transposed = reshaped.T
    print(f"Transposed:\n{transposed}")

    # Flatten
    flattened = reshaped.flatten()
    print(f"Flattened: {flattened}")

    # Concatenate
```

```
    arr1 = np.array([1, 2, 3]) arr2 = np.array([4, 5, 6]) concatenated = np.concatenate([arr1, arr2]) print(f"Concatenated: {concatenated}") # Stack stacked = np.stack([arr1, arr2]) print(f"Stacked:\n{stacked}") return reshaped, transposed, flattened, concatenated, stacked



# Execute
manipulation_result = array_manipulation_example()
```

Part 2: QuickStart Tutorial Examples

2.1The Basics

```
def quickstart_basics():
    """Examples from NumPy QuickStart - The Basics"""
    print("=== QuickStart Tutorial Examples ===\n") # Creating
    arrays
    a = np.arange(15).reshape(3, 5)
    print(f"Array 'a':\n{a}")
    print(f"Shape: {a.shape}")
    print(f"Dimensions: {a.ndim}")
    print(f"Data type: {a.dtype.name}")
    print(f"Item size: {a.itemsize}")
    print(f"Size: {a.size}")
    print(f"Type: {type(a)}") # Different ways to create arrays
    b = np.array([6, 7, 8])
    print(f"\nArray 'b': {b}") c = np.array([2, 3, 4])
    print(f"Array 'c': {c}")
    print(f"Data type of c: {c.dtype}") # Array with floats
    d = np.array([1.2, 3.5, 5.1])
    print(f"Array 'd' (floats): {d}")
    print(f"Data type of d: {d.dtype}") # 2D array
    e = np.array([(1.5, 2, 3), (4, 5, 6)])
    print(f"\n2D Array 'e':\n{e}") # Specify data type
    f = np.array([[1, 2], [3, 4]], dtype=complex)
    print(f"\nComplex array 'f':\n{f}") return a, b, c, d, e, f
```

```
# Execute
quickstart_basics_result = quickstart_basics()
```

2.2 Array Creation

```
def quickstart_array_creation():
    """Examples from QuickStart - Array Creation"""
    print("\n=== Array Creation Examples ===\n")

    # Zeros, ones, empty
    zeros_arr = np.zeros((3, 4))
    print(f"Zeros array (3x4):\n{zeros_arr}")

    ones_arr = np.ones((2, 3, 4), dtype=np.int16)
    print(f"\nOnes array (2x3x4, int16):\n{ones_arr}")
    empty_arr = np.empty((2, 3))
    print(f"\nEmpty array (2x3):\n{empty_arr}")

    # Sequences
    arange_arr = np.arange(10, 30, 5)
    print(f"\nArange (10 to 30, step 5): {arange_arr}")

    arange_float = np.arange(0, 2, 0.3)
    print(f"Arange (0 to 2, step 0.3): {arange_float}")
    linspace_arr = np.linspace(0, 2, 9)
    print(f"Linspace (0 to 2, 9 numbers): {linspace_arr}")

    # Reshape example
    x = np.linspace(0, 2*np.pi, 100)
    f = np.sin(x)
    print(f"\nSine function example (first 10 values): {f[:10]}")
    return zeros_arr, ones_arr, empty_arr, arange_arr, linspace_arr, f
```

```
# Execute
creation_result = quickstart_array_creation()
```

2.3 Printing Arrays

```
def quickstart_printing():
    """Examples from QuickStart - Printing Arrays"""
    print("\n=== Printing Arrays Examples ===\n")

    # 1D array
    a = np.arange(6)
    print(f"1D array: {a}")

    # 2D array
    b = np.arange(12).reshape(4, 3)
    print(f"\n2D array:\n{b}")

    # 3D array
    c = np.arange(24).reshape(2, 3, 4)
    print(f"\n3D array:\n{c}")

    # Large array (NumPy skips middle)
```

```
    large = np.arange(10000)
    print(f"\nLarge array (truncated): {large}")
    # Force print entire array (commented out for brevity)
    # np.set_printoptions(threshold=sys.maxsize)
    return a, b, c, large


# Execute
printing_result = quickstart_printing()
```

2.4 Basic Operations

```
def quickstart_basic_operations():
    """Examples from QuickStart - Basic Operations"""
    print("\n=== Basic Operations Examples ===\n") a = np.array([20, 30, 40, 50])
    b = np.arange(4)
    print(f"Array a: {a}")
    print(f"Array b: {b}") # Subtraction
    c = a - b
    print(f"a - b: {c}") # Square
    square_b = b**2
    print(f"b squared: {square_b}") # Sine
    sin_a = 10 * np.sin(a)
    print(f"10 * sin(a): {sin_a}") # Boolean operations
    bool_result = a < 35
    print(f"a < 35: {bool_result}") # Matrix operations
    A = np.array([[1, 1], [0, 1]])
    B = np.array([[2, 0], [3, 4]])
    print(f"\nMatrix A:\n{A}")
    print(f"Matrix B:\n{B}") # Element-wise product
    elementwise = A * B
    print(f"Element-wise product:\n{elementwise}") # Matrix product
    matrix_product = A @ B
    print(f"Matrix product (A @ B):\n{matrix_product}") # Alternative matrix product
    matrix_product2 = np.dot(A, B)
    print(f"Matrix product (np.dot):\n{matrix_product2}") return c, square_b, sin_a,
bool_result, elementwise, matrix_product
```

```
# Execute
operations_result = quickstart_basic_operations()
```

2.5 Universal Functions

```
def quickstart_universal_functions():
    """Examples from QuickStart - Universal Functions"""
    print("\n=== Universal Functions Examples ===\n") # Random array
    rg = np.random.default_rng(1) # Create generator with seed
    a = rg.random((2, 3))
    print(f"Random array:\n{a}") # Sum operations
    print(f"Sum of all elements: {a.sum()}")
    print(f"Min of all elements: {a.min()}")
    print(f"Max of all elements: {a.max()}") # Operations along axes
    b = np.arange(12).reshape(3, 4)
    print(f"\nArray b:\n{b}") print(f"Sum of each column: {b.sum(axis=0)}")
    print(f"Min of each row: {b.min(axis=1)}")
    print(f"Cumulative sum along rows: {b.cumsum(axis=1)}") return a, b
```

```
# Execute
ufunc_result = quickstart_universal_functions()
```

Part 3: Absolute Beginners Tutorial Examples

3.1WhatisanArray?

```
def beginners_what_is_array():
    """Examples from Absolute Beginners - What is an array?"""
    print("\n=== Absolute Beginners Tutorial ===\n")
    print("=== What is an Array? ===\n")
    # 1D array example
    a1D = np.array([1, 2, 3, 4])
    print(f"1D array: {a1D}")
    # 2D array example
    a2D = np.array([[1, 2], [3, 4]])
    print(f"\n2D array:\n{a2D}")
    # 3D array example
    a3D = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
    print(f"\n3D array:\n{a3D}")
    # Array attributes
    print(f"\n1D array attributes:")
```

```
    print(f" Number of dimensions: {a1D.ndim}")
    print(f" Shape: {a1D.shape}")
    print(f" Size: {a1D.size}")
    print(f"\n2D array attributes:")
    print(f" Number of dimensions: {a2D.ndim}")
    print(f" Shape: {a2D.shape}")
    print(f" Size: {a2D.size}")
    return a1D, a2D, a3D



# Execute
array_basics = beginners_what_is_array()
```

3.2 Array Creation from Existing Data

```
def beginners_array_creation():
    """Examples from Absolute Beginners - Array Creation"""
    print("\n=== Array Creation from Existing Data ===\n")

    # From lists
    list_array = np.array([1, 2, 3, 4, 5, 6])
    print(f"Array from list: {list_array}")

    # From tuples
    tuple_array = np.array((1, 2, 3, 4, 5, 6))
    print(f"Array from tuple: {tuple_array}")

    # From nested sequences
    nested_array = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
    print(f"\nArray from nested list:\n{nested_array}")

    return list_array, tuple_array, nested_array



# Execute
creation_basics = beginners_array_creation()
```

3.3 Array Creation from Scratch

```
def beginners_array_from_scratch():
    """Examples from Absolute Beginners - Creating arrays from scratch"""
    print("\n=== Creating Arrays from Scratch ===\n") # Zeros
    zeros_1d = np.zeros(4)
    zeros_2d = np.zeros((4, 6))
    print(f"1D zeros: {zeros_1d}")
    print(f"2D zeros:\n{zeros_2d}") # Ones
    ones_1d = np.ones(4)
    ones_2d = np.ones((3, 8))
    print(f"\n1D ones: {ones_1d}")
    print(f"2D ones:\n{ones_2d}") # Empty (uninitialized)
    empty_arr = np.empty(4)
    print(f"\nEmpty array: {empty_arr}")
```

```
    #Range of numbers
    range_arr = np.arange(4)
    print(f"Range array: {range_arr}")
    #Evenly spaced numbers
    even_spaced = np.linspace(0, 10, num=5)
    print(f"Evenly spaced: {even_spaced}")
    #Specify data types
    int_array = np.ones(5, dtype=np.int64)
    print(f"\nInteger ones: {int_array}")
    return zeros_2d, ones_2d, range_arr, even_spaced, int_array



# Execute
scratch_basics = beginners_array_from_scratch()
```

3.4 Adding, Removing, and Sorting Elements

```
def beginners_adding_removing_sorting():
    """Examples from Absolute Beginners - Adding, removing, and sorting"""
    print("\n=== Adding, Removing, and Sorting Elements ===\n") arr = np.array([2, 1, 5, 3, 7,

    4, 6, 8])
    print(f"Original array: {arr}") # Sorting
    sorted_arr = np.sort(arr)
    print(f"Sorted array: {sorted_arr}") # Concatenate arrays
    a = np.array([1, 2, 3, 4])
    b = np.array([5, 6, 7, 8])
    concatenated = np.concatenate((a, b))
    print(f"\nArray a: {a}")
    print(f"Array b: {b}")
    print(f"Concatenated: {concatenated}") # 2D concatenation
    x = np.array([[1, 2], [3, 4]])
    y = np.array([[5, 6]])
    concat_2d = np.concatenate((x, y), axis=0)
    print(f"\n2D concatenation:\n{concat_2d}") return sorted_arr, concatenated, concat_2d



# Execute
sorting_basics = beginners_adding_removing_sorting()
```

3.5 Array Shape and Size

```
def beginners_shape_and_size():
    """Examples from Absolute Beginners - Shape and size"""
    print("\n=== Array Shape and Size ===\n")

    # Create test arrays
    array_example = np.array([[[0, 1, 2, 3],
```

```
                                    [4, 5, 6, 7]],
                                   [[0, 1, 2, 3],
                                    [4, 5, 6, 7]],
                                   [[0, 1, 2, 3],
                                    [4, 5, 6, 7]]])

    print(f"Array shape: {array_example.shape}")
    print(f"Number of dimensions: {array_example.ndim}")
    print(f"Array size: {array_example.size}")
    # Number of elements in each dimension
    print(f"Number of rows: {array_example.shape[0]}")
    print(f"Number of columns: {array_example.shape[1]}")
    return array_example



# Execute
shape_basics = beginners_shape_and_size()
```

3.6 Reshaping Arrays

```
def beginners_reshaping():
    """Examples from Absolute Beginners - Reshaping arrays"""
    print("\n=== Reshaping Arrays ===\n") # Original array
    a = np.arange(6)
    print(f"Original 1D array: {a}") # Reshape to 2D
    b = a.reshape(3, 2)
    print(f"Reshaped to 3x2:\n{b}") # Reshape to 3D
    c = a.reshape(2, 1, 3)
    print(f"Reshaped to 2x1x3:\n{c}") # Using reshape with -1 (automatic
    dimension)
    d = a.reshape(2, -1)
    print(f"Reshaped with -1:\n{d}") # Add new axis
    a2 = np.array([1, 2, 3, 4, 5, 6])
    expanded = a2[np.newaxis, :]
    print(f"\nOriginal: {a2}")
    print(f"With new axis: {expanded}")
    print(f"New shape: {expanded.shape}") # Column vector
    column = a2[:, np.newaxis]
    print(f"Column vector:\n{column}") return b, c, d, expanded, column



# Execute
reshape_basics = beginners_reshaping()
```

3.7 Indexing and Slicing

```
def beginners_indexing_slicing():
    """Examples from Absolute Beginners - Indexing and slicing""" print("\n=== Indexing and Slicing ===\n") # 1D array indexing data = np.array([1, 2, 3]) print(f"Array: {data}") print(f"First element: {data[0]}") print(f"Second element: {data[1]}") print(f"Last element: {data[-1]}") # 2D array indexing a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]) print(f"\n2D Array:\n{a}") print(f"Element at row 0, column 1: {a[0, 1]}") print(f"Element at row 1, column 3: {a[1, 3]}") print(f"Element at row -1, column -1: {a[-1, -1]}") # Slicing print(f"\nFirst two rows: \n{a[0:2]}") print(f"All rows, first two columns: \n{a[:, 0:2]}") print(f"Last row, all columns: {a[-1, :]}") # Boolean indexing b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]) condition = b > 5 print(f"\nCondition (b > 5):\n{condition}") print(f"Elements greater than 5: {b[condition]}") return data, a, b[condition]
```

```
# Execute
indexing_basics = beginners_indexing_slicing()
```

3.8 Basic Array Operations

```
def beginners_basic_operations():
    """Examples from Absolute Beginners - Basic operations""" print("\n=== Basic Array Operations ===\n") data = np.array([1, 2]) ones = np.ones(2, dtype=int) print(f"Data: {data}") print(f"Ones: {ones}") # Basic arithmetic print(f"Addition: {data + ones}") print(f"Subtraction: {data - ones}") print(f"Multiplication: {data * ones}") print(f"Division: {data / ones}") # More operations a = np.array([1, 2, 3, 4]) print(f"\nArray a: {a}") print(f"Sum: {a.sum()}") print(f"Maximum: {a.max()}")
```

```
print(f"Minimum: {a.min()}") print(f"Mean: {a.mean()}") print(f"Standard deviation: {a.std()}") # 2D operations b = np.array([[1, 2], [3, 4]]) print(f"\n2D Array b:\n{b}") print(f"Sum: {b.sum()}") print(f"Sum along axis 0: {b.sum(axis=0)}") print(f"Sum along axis 1: {b.sum(axis=1)}") return data + ones, a.sum(), b.sum(axis=0)



# Execute
operations_basics = beginners_basic_operations()
```

Conclusion

This laboratory exercise has provided a comprehensive overview of NumPy's built-in functions and practical applications. Through the implementation of various functions and execution of examples from the official documentation, we have demonstrated:

Key Findings:

1. **Array Creation:** NumPy provides multiple methods for creating arrays, from basic **np.array()** to specialized functions like **np.zeros()**, **np.ones()**, and **np.linspace()**.
2. **Mathematical Operations:** Universal functions (ufuncs) enable efficient element-wise operations on arrays, providing significant performance benefits over pure