Chapter 6: Applications in Linear Algebra, Differential Calculus and Probability

6.1 Basic Linear Algebra

Linear algebra is the study of vectors and matrices. It is used to represent and solve systems of equations and to manipulate data.

1. Vectors and matrices with NumPy

- **Vector**: a list of numbers.
- **Matrix**: a two-dimensional array (rows and columns).

```
import numpy as np

# Creating a vector
v = np.array([1, 2, 3])
print("Vector v:", v)

# Creating a 2x2 matrix
A = np.array([[1, 2],
              [3, 4]])
print("Matrix A:\n", A)
```

Explanations:

- `import numpy as np`: imports the **NumPy** library, which is used to manipulate arrays and matrices.
- `np.array([1, 2, 3])`: creates a **vector** with three elements.
- `np.array([[1, 2], [3, 4]])`: creates a 2x2 **matrix** (2 rows, 2 columns).

2. Simple operations

- **Addition**: add element by element
- **Multiplication**: matrix multiplication

```
B = np.array([[5, 6], [7, 8]])

# Addition
C = A + B
print("Addition A + B:\n", C)

# Multiplication
D = A @ B  # or np.dot(A, B)
print("Multiplication A*B:\n", D)
```

Explanations:

- `A + B`: adds the matrices **element by element**.
- `A @ B`: performs **matrix multiplication**, following the rules of linear algebra (row-column product).

3. Solving simple linear systems

Example:
Solve the system

$2x + y = 5$ and $x + 3y = 6$

Ex

```
A = np.array([[2, 1], [1, 3]])
b = np.array([5, 6])

x = np.linalg.solve(A, b)
print("Solution [x, y]:", x)
```

****Tip**: `np.linalg.solve(A, b)` automatically calculates the solution to the system Ax=b.

Explanations:

- `A`: matrix of the system's coefficients.
- `b`: vector of results.
- `np.linalg.solve(A, b)`: automatically calculates the values of `x` and `y` that satisfy both equations.
- `print(x)`: displays the result `[x, y]`.

6.2 Introductory Differential Calculus

Differential calculus allows us to study the variations of functions.

1. Simple numerical derivative with NumPy

The derivative measures **the slope of a function** at a point.

```
def f(x):
    return x**2 + 3*x + 1

x0 = 2
h = 1e-5   # small step
df_dx = (f(x0 + h) - f(x0 - h)) / (2*h)
print("Approximate derivative at x0:", df_dx)
```

Explanations:

- `def f(x):` : defines the function f(x) = x^2 + 3x + 1.
- `h = 1e-5`: small step size for calculating the derivative numerically.
- `(f(x0+h) - f(x0-h)) / (2*h)`: formula for **the centred derivative**.
- `print(df_dx)`: displays the approximate slope of the function at `x0`.

2. Introduction to gradients

The **gradient** is a vector that indicates the direction in which the function increases most rapidly.

The **gradient** indicates the direction of maximum variation for a function of several variables.

```
def f(x, y):
    return x**2 + y**2

h = 1e-5
x0, y0 = 1, 2
grad_f = np.array([
    (f(x0+h, y0) - f(x0-h, y0)) / (2*h),
    (f(x0, y0+h) - f(x0, y0-h)) / (2*h)
])
print("Gradient at (x0, y0):", grad_f)
```

Explanations:

- `f(x, y)`: function of two variables.
- Gradient = `[∂f/∂x, ∂f/∂y]`
- First line: derivative with respect to `x` (horizontal variation).
- Second line: derivative with respect to `y` (vertical variation).
- `np.array([...])`: combines the two derivatives into a vector.

6.3 Probability and Basic Statistics

Probability allows us to model chance.
Statistics allow us to summarise and analyse data.

1. Random variables and simple simulations with NumPy

```
# 10 uniform random numbers between 0 and 1
rand_nums = np.random.rand(10)
print("Uniform random:", rand_nums)

# 10 numbers following a normal distribution
norm_nums = np.random.randn(10)
print("Normal random:", norm_nums)
```

Explanations:

- `np.random.rand(10)`: 10 random numbers uniformly distributed between 0 and 1.
- `np.random.randn(10)`: 10 random numbers following a **normal distribution** (mean 0, standard deviation 1).

2. Calculation of descriptive statistics

- **Mean**: central value
- **Median**: middle value
- **Variance**: dispersion around the mean

```
data = np.random.randn(100)

mean = np.mean(data)
median = np.median(data)
variance = np.var(data)

print("Mean:", mean)
print("Median:", median)
print("Variance:", variance)
```

Explanations:

- `np.mean(data)`: calculates the **mean**.
- `np.median(data)`: calculates the **median**.
- `np.var(data)`: calculates the **variance** (dispersion of the data).

6.4 Practical Projects

1. Data visualisation

```
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y)
plt.title("Example of a sin(x) graph")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.show()
```

Explanations:

- `np.linspace(0, 10, 100)`: creates 100 points between 0 and 10.
- `y = np.sin(x)`: calculates sin(x) for each point.
- `plt.plot(x, y)`: plots the curve.

- `plt.show()`: displays the graph.

2. Solving and visualising linear systems

- Create a random system
- Solve using `np.linalg.solve`
- Visualise the equations and solutions

3. Probability simulation(Histogram of Random Data)

- Generating random data
- Calculate statistics
- Plot a histogram to visualise the distribution

```
plt.hist(data, bins=10)
plt.title("Distribution of random data")
plt.show()
```

Explanations:

- `plt.hist(data, bins=10)`: creates a histogram with 10 columns (intervals).
- Shows the **distribution of the random values**.

Useful references

1. NumPy Documentation
2. Matplotlib Documentation
3. Wes McKinney, *Python for Data Analysis*
4. Al Sweigart, *Automate the Boring Stuff with Python*