Modeling and Simulation - Lecture Notes 2

June 2023

Python Basics

Overview of Python: Syntax, Data Types, Control Structures

Python is a high-level, interpreted programming language known for its simplicity and readability. Its syntax is clear and intuitive, making it a popular choice for beginners and experienced developers alike.

- **Syntax:** Python's syntax is one of its defining features. Unlike many programming languages, Python uses indentation to denote blocks of code. This leads to more readable code as it enforces the proper structuring of code blocks. Here are some examples:
  - Python uses a colon (:) and indentation to denote separate blocks of code. Consider the following example:

    ```
    if 5 > 2:
    print("Five is greater than two!")
    ```

    Here, the print statement is part of the if block and is executed if the if condition is true.
  - For loops and while loops in Python also use indentation to mark the scope of the loop. For example:

    ```
    for i in range(5):
    print(i)
    ```

    In this example, the print(i) statement is executed five times as it's within the scope of the for loop.
  - Python uses indentation to define function scope as well:

    ```
    def greet(name):
    print("Hello, " + name + ". Good morning!")
    ```

    In this example, the print statement is a part of the function greet.

Remember, in Python, the amount of whitespace (spaces or tabs) for indentation can vary, but all statements within the block must be indented the same amount.

- **Data Types:** Python supports various data types including integers, floats (floating point numbers), strings (text), and booleans (True or False). Python also has several built-in data structures like lists, tuples, dictionaries, and sets.

  Here are some of the most common data types:

  - **Integers:** These are positive or negative whole numbers with no decimal point. For example, ‘age = 21‘.
  - **Floats:** These are real numbers with a decimal point. For example, ‘height = 1.75‘.
  - **Strings:** These are sequences of character data. The string type in Python is called ‘str‘. For example, ‘name = "John"‘.
  - **Booleans:** These represent truth values and can be either ‘True‘ or ‘False‘.

  Python also has several built-in data structures. These are complex types which organize collections of basic data types:

  - **Lists:** A list in Python is an ordered group of items. They are mutable, meaning items can be added, removed, and changed. For example, ‘fruits = ["apple", "banana", "cherry"]‘.
  - **Tuples:** A tuple is similar to a list in that it is an ordered group of items. However, tuples are immutable, meaning they cannot be changed after they are created. For example, ‘dimensions = (20, 30, 40)‘.
  - **Dictionaries:** Dictionaries are unordered collections of key-value pairs. For example, ‘student = "name": "John", "age": 21‘. Here ‘"name"‘ and ‘"age"‘ are keys, and ‘"John"‘ and ‘21‘ are their corresponding values.
  - **Sets:** A set is an unordered collection of unique elements. For example, ‘unique_numbers = 1, 2, 3, 3, 3, 4‘ will result in the set ‘1, 2, 3, 4‘.

```
# Integers:
age = 21
print(age)  # Output: 21

# Floats:
height = 1.75
```

```
print(height)  # Output: 1.75

# Strings:
name = "John"
print(name)  # Output: John

# Booleans:
is_tall = True
print(is_tall)  # Output: True

# Lists:
fruits = ["apple", "banana", "cherry"]
print(fruits)  # Output: ['apple', 'banana', 'cherry']

# Add an element to the list
fruits.append("orange")
print(fruits)  # Output: ['apple', 'banana', 'cherry', 'orange']

# Tuples:
dimensions = (20, 30, 40)
print(dimensions)  # Output: (20, 30, 40)

# Dictionaries:
student = {"name": "John", "age": 21}
print(student)  # Output: {'name': 'John', 'age': 21}

# Access a value in the dictionary by its key
print(student["name"])  # Output: John

# Sets:
unique_numbers = {1, 2, 3, 3, 3, 4}
print(unique_numbers)  # Output: {1, 2, 3, 4}
```

- **Control Structures:** Python uses control structures to dictate the flow of the program. Below are some examples:
  - **Decision Making:** The 'if', 'elif', and 'else' keywords are used in Python for decision making. For example:

    ```
    age = 21
    if age >= 18:
        print("You are an adult.")
    else:
        print("You are a minor.")
    ```

In this example, if the age is greater than or equal to 18, the program will print "You are an adult.". If not, it will print "You are a minor.".

- **Loops:** The 'for' and 'while' keywords are used for looping in Python. For example:

```
for i in range(5):
    print(i)
```

This example will print numbers 0 through 4, each on a new line.

- **Controlling Loop Execution:** The 'break' and 'continue' keywords are used to control the flow within loops. For example:

```
for num in range(10):
    if num == 5:
        break
    print(num)
```

This program will print numbers 0 through 4. When 'num' equals 5, the 'break' statement is executed, stopping the loop.

- **Functions:** Functions in Python are defined using the 'def' keyword. For example:

```
def greet(name):
    print("Hello, " + name)
greet("John")
```

This program defines a function named 'greet' that takes one argument, 'name'. The function then prints a greeting with the provided name. The function is then called with the argument "John".

Introduction to Python Libraries: NumPy, SciPy, Matplotlib

Python's power comes in large part from its vast collection of libraries, which are pre-written pieces of code available for you to use. For our purposes in this course, we'll focus on a few key libraries:

- **NumPy:** Short for 'Numerical Python', NumPy is a library that provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these elements.
- **SciPy:** Built on NumPy, SciPy is a library used for scientific and technical computing and adds more functionality to NumPy arrays, including mathematical algorithms and convenience functions.
- **Matplotlib:** This is Python's basic plotting library. It provides tools for making plots and graphs, and works well with NumPy and SciPy.

NumPy

NumPy, short for Numerical Python, is a fundamental package for numerical computation in Python. It provides support for arrays, matrices, and higher-dimensional tensors, making it extremely useful for a wide variety of mathematical tasks in science and engineering. In particular, for a modeling and simulation course in electrical and electronics engineering, NumPy's support for efficient array operations makes it an invaluable tool for manipulating data and implementing mathematical models.

An array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The dimensions are defined as the size of the array along each direction, and the rank is the number of dimensions. A list in Python is a collection of items which can be of different types. Unlike arrays, lists are a part of Python's syntax, while arrays are a part of the numpy library.

A matrix is a special case of a two-dimensional array where both dimensions are of the same length. Tensors are a generalization of matrices to higher dimensions and are used in fields such as physics and computer graphics.

NumPy provides many functions to create arrays:

```
import numpy as np

# Create a 2x2 array of zeros
a = np.zeros((2,2))

# Create a 1D array of ones
b = np.ones(3)

# Create a 3x3 identity matrix
c = np.eye(3)

# Create an array with random values
d = np.random.random((2,2))
```

These arrays can be manipulated in a variety of ways:

```
# Transpose an array
a = np.array([[1,2],[3,4]])
b = np.transpose(a)

# Inverse of a matrix
c = np.linalg.inv(a)
```

For visualization, NumPy works seamlessly with Matplotlib, a plotting library. Here's an example:

```
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```

This will create a simple sine wave and display it using matplotlib's interface.

Vectors

In Python, a vector can be represented using a one-dimensional NumPy array. Here is an example of a vector:

```
# Creating a vector in NumPy
v = np.array([1, 2, 3])
print(v)
```

The output will be:

```
array([1, 2, 3])
```

Matrices

A matrix can be represented using a two-dimensional NumPy array. Here is an example of a matrix:

```
# Creating a matrix in NumPy
M = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(M)
```

The output will be:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

NumPy provides a suite of functions to perform operations on vectors and matrices. Here is an example of matrix-vector multiplication:

```
# Matrix-vector multiplication in NumPy
v = np.array([1, 2, 3])
M = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
result = np.dot(M, v)
print(result)
```

The output will be:

```
array([14, 32, 50])
```

This demonstrates that NumPy makes it easy to perform complex mathematical operations, which are fundamental to topics in modeling and simulation.

NumPy Sublibraries and Functions

NumPy includes several sublibraries that provide additional functionality beyond basic array manipulation. Some of these include:

NumPy.linalg

This sublibrary includes several functions to solve linear algebra problems. These include:

- **np.linalg.inv**: This function computes the (multiplicative) inverse of a matrix.

```
# Compute the inverse of a matrix
A = np.array([[1, 2], [3, 4]])
A_inv = np.linalg.inv(A)
print(A_inv)
```

- **np.linalg.eig**: This function computes the eigenvalues and right eigenvectors of a square array.

```
# Compute the eigenvalues and eigenvectors of a matrix
A = np.array([[1, 2], [3, 4]])
eigenvalues, eigenvectors = np.linalg.eig(A)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:", eigenvectors)
```

- **np.linalg.solve**: This function solves a linear matrix equation, or system of linear scalar equations.

```
# Solve the system of equations 3 * x0 + x1 = 9 and x0 + 2 * x1 = 8
A = np.array([[3,1], [1,2]])
b = np.array([9,8])
x = np.linalg.solve(A, b)
print(x)
```

NumPy.fft

This sublibrary is used to compute the one-dimensional n-point discrete Fourier Transform (DFT) and its inverse.

```
# Compute the one-dimensional n-point discrete Fourier Transform
x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])
y = np.fft.fft(x)
print(y)
```

NumPy.random

This sublibrary is used to generate random numbers for various distributions including normal, uniform, and binomial among others.

```
# Generate five random numbers from the normal distribution
x = np.random.normal(size=5)
print(x)

# Generate five random integers between 10 and 50
y = np.random.randint(low=10, high=50, size=5)
print(y)
```

Matplotlib

Matplotlib is a plotting library in Python which produces quality figures in a variety of formats and interactive environments. The module `pyplot` in Matplotlib is a collection of functions that provide a simple interface for plotting.

```
import matplotlib.pyplot as plt
```

Here are some examples of how to use Matplotlib in combination with NumPy.

Plotting a Sine Wave

Let's start with a simple example of plotting a sine wave:

```
import numpy as np
import matplotlib.pyplot as plt

# Create an array of x values from 0 to 2*pi
x = np.linspace(0, 2*np.pi, 100)

# Compute the corresponding y values
y = np.sin(x)
```

```
# Create a new figure
plt.figure()

# Plot x against y
plt.plot(x, y)

# Display the plot
plt.show()
```

In this example, we first create an array of x-values using NumPy's `linspace` function. We then compute the corresponding y-values by taking the sine of the x-values. We plot the result using Matplotlib's `plot` function and display it using `show`.

Plotting a Histogram

Next, let's plot a histogram of data drawn from a normal distribution:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate 1000 random values from a normal distribution
data = np.random.normal(size=1000)

# Create a new figure
plt.figure()

# Plot a histogram of the data
plt.hist(data, bins=30)

# Display the plot
plt.show()
```

In this example, we generate 1000 random values from a normal distribution using NumPy's `random.normal` function. We then plot a histogram of this data using Matplotlib's `hist` function.

Plotting a Scatter Plot

Finally, let's create a scatter plot:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate 100 random x values and 100 random y values
x = np.random.random(size=100)
y = np.random.random(size=100)
```

```
# Create a new figure
plt.figure()

# Create a scatter plot of x against y
plt.scatter(x, y)

# Display the plot
plt.show()
```

In this example, we generate 100 random x-values and 100 random y-values using NumPy's `random.random` function. We then create a scatter plot of the x-values against the y-values using Matplotlib's `scatter` function.

SciPy

SciPy is a library in Python that is used for scientific and technical computing. It builds on NumPy and provides many additional functionalities, such as modules for optimization, linear algebra, integration, interpolation, special functions, and more.

```
import scipy
```

Solving Differential Equations with SciPy

A common task in modeling and simulation, especially in electrical engineering, is to solve differential equations. SciPy's `scipy.integrate.solve_ivp` function can be used to solve initial value problems for a system of ODEs.

Here is an example of solving a simple first order RC circuit (resistor-capacitor circuit). The differential equation is:

$$\frac{dV}{dt} = \frac{1}{RC}(V_{in} - V) \tag{1}$$

where V_{in} is the input voltage, V is the output voltage across the capacitor, R is the resistance, and C is the capacitance.

Let's assume $R = 1$ Ohm, $C = 1$ Farad, and $V_{in} = 1$ Volt. The Python code to solve this ODE is:

```
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt

# Define the differential equation
def RC_circuit(t, V, Vin, R, C):
    return 1/(R*C) * (Vin - V)
```

```
# Parameters
Vin = 1
R = 1
C = 1

# Time grid
t = np.linspace(0, 5, 100)

# Initial condition
V0 = [0]

# Solve the differential equation
sol = solve_ivp(RC_circuit, [0, 5], V0, args=(Vin, R, C), dense_output=True)

# Get the solution on the defined grid
V = sol.sol(t)[0]

# Plot the solution
plt.figure()
plt.plot(t, V)
plt.xlabel('Time [s]')
plt.ylabel('Voltage [V]')
plt.show()
```

In this example, we first define the differential equation as a Python function. We then solve the ODE using the `scipy.integrate.solve_ivp` function, which requires the function defining the ODE, the time span, the initial conditions, and the parameters of the function as arguments. Finally, we plot the output voltage over time.

Problem Set

1. **Python Basics:**
   (a) Write a Python function to calculate the factorial of a positive integer.
   (b) Write a Python function to check if a number is prime or not.
   (c) Write a Python function to sort a list of integers in ascending order. Do not use any in-built Python sorting function.

2. **NumPy:**
   (a) Create a NumPy array with values ranging from 10 to 50.
   (b) Write a Python program to create a 3x3 matrix with values ranging from 0 to 8.
   (c) Compute the mean, standard deviation, and variance of a given NumPy array.

3. **Matplotlib:**
   (a) Plot a sine wave from 0 to 4π and label the axes.
   (b) Plot a scatter plot using random distributions to generate balls of different sizes.
   (c) Create a histogram of a set of observations - use NumPy's random number generator to create the observations.

4. **SciPy:**
   (a) Use SciPy's numerical integration function to integrate a function from 0 to 10.
   (b) Use SciPy's interpolation function to interpolate a sine wave.
   (c) Use SciPy's optimization functions to find the minimum of a simple function.