ME 5401: Scientific Computing using Python
Tutorial 1

Duration: 3 hours

Topics covered

Introduction to Python for scientific computing using Google Colab/Jupyter notebooks, Python objects and basic data structures, lists, NumPy arrays, vector and matrix operations, floating-point representation and numerical precision, vectorization and basic scientific visualization using Matplotlib.

Setup

Run the following imports before starting the tutorial:

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

# Reproducible random numbers for the whole tutorial
rng = np.random.default_rng(5401)

# Make NumPy output easier to read
np.set_printoptions(precision=4, suppress=True)
```

Questions

1. **Basic Python commands, lists**

   (a) Use `print` to display the text `Scientific Computing using Python`.

   (b) Create a Python `list` containing the numbers $0, 2, 4, 6, 8$ using `range`.

   (c) Print all the items in the `list`, the datatype of the `list`, and the length of the `list`.

   (d) Add an extra item 10 to the first `list` and print the `list` after item addition.

   (e) Create a second Python `list` containing the square of each number in first `list`.

2. **NumPy array creation, shapes, and reshaping**.

   (a) Create a 2×3 NumPy array `A` with values

   $$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

   (b) Use the following commands to create numpy arrays: `np.zeros, np.ones, np.eye, np.arange, np.linspace`. Print each array.

   (c) For `A`, print its shape (`A.shape`), size (`A.size`), and number of dimensions (`A.ndim`). Interpret each of the outputs.

(d) Print the transpose of `A` and a reshaped version of `A` with shape `(3, 2)`. Comment on the differences between both outputs.

3. **Scalars, vectors, matrices, and Python bracket types**

   (a) Create two scalar variables: 'n = 4', and 'alpha = 0.8'. Print their values and datatypes.

   (b) Create the following NumPy arrays and print each array, its datatype, its shape, and its number of dimensions. Comment on the shape and number of dimensions of each case.

      i. 1-D NumPy array: `v1d = np.array([1, 2, 3])`
      ii. row vector: `vrow = np.array([[1, 2, 3]])`
      iii. column vector: `vcol = np.array([[1], [2], [3]])`
      iv. matrix: `M = np.array([[1, 2, 3], [4, 5, 6]])`

   (c) Demonstrate parentheses in a function call by evaluating $sin(\pi/2)$

   Note: Python uses different brackets in different contexts:

   (a) `[]` : lists, indexing, and slicing. Example: `samplelist = [1,2,3], a[0], a[1:4]`

   (b) `()` : function calls and grouping expressions. Example: `np.exp(x), (a + b) * c`

   (c) {} : sets and dictionaries (see Additional Question 4)

4. **Vector operations**

   (a) Create a row vector containing values from 0 to 30 in steps of 2 and a column vector containing values from 0 to 30 in steps of 2. Print their values and shapes.

   (b) Create two random length-5 vectors `u` and `v`. Compute their dot product.

5. **Matrix computations**

   Generate an $n \times n$ random square matrix A and a random vector x of length $n = 4$. Then:

   (a) Compute the determinant of A and the transpose of A

   (b) Compute vector y such that $y = Ax$.

   (c) Print the shapes of A, x, and y.

   (d) Create another $n \times n$ random square matrix B and compute the matrix $C1 = AB$ and $C2 = BA$. Check if $C1$ and $C2$ are equal.

6. **Basic plotting with `matplotlib.pyplot`**

   (a) Create a figure with two subplots using `plt.subplots`

   (b) In the top subplot, create a simple line plot of $y = sin(x)$ using 200 uniformly spaced points between $x = 0$ and $x = 2\pi$.

   (c) In the bottom subplot, first create random vectors x and y of equal length and then generate a scatter plot with the values from vector x on the x-axis and values from vector y on the y-axis.

   (d) Use good scientific plotting habits: axis labels, title, grid lines, legend and readable figure size.

7. **NumPy indexing, slicing, and Boolean masks**

   Create a NumPy array containing the numbers 0 to 20. Then:

   (a) Select the first five values.

(b) Select every second value.

(c) Select values greater than 10.

(d) Replace values divisible by 3 with `-1` in a copy of the array.

(e) Normalize the original array to the range 0 to 1 using

$$\text{normalized} = \frac{\text{data} - \min(\text{data})}{\max(\text{data}) - \min(\text{data})}.$$

8. **Functions**

Write a Python function that computes the root-mean-square (RMS) value of a sequence using a `for` loop, as described below.

$$\text{RMS}(x) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}.$$

(a) Create a NumPy array `samplearray` containing values $[1, 2, 3, ..., n]$ for predefined n.

(b) Write a function named `rmsloop(values)` that uses a `for` loop to compute the RMS value of the elements in `values`.

Additional Questions (Optional)

Run the following imports for the additional questions

```
from time import perf_counter
```

1. **Vectorization**

Expanding on Question 8 from the main tutorial:

(a) Write a function named `rmsvectorized(values)` that uses a vectorized NumPy expression to compute the RMS value of the elements in `values`.

(b) Apply both functions `rmsloop(values)` and `rmsvectorized(values)` to the `samplearray` and compare the time required to evaluate the result for (i) $n = 100$, (ii) $n = 10,000$ and (iii) $n = 1,000,000$. Comment on the run time difference between both approaches for large n.

2. **Random numbers, basic statistics**

(a) Create and print a 3×4 numpy array `R` of uniformly distributed random numbers between 0 and 1.

(b) Create and print a length-5 vector `z` of normally distributed random numbers with mean 0 and standard deviation 1.

(c) For `R`, compute and print the minimum, maximum, sum, mean, and standard deviation.

(d) Save `R` to the file `examplearray.npy`. Load it back and verify that the loaded array is equal to the original.

3. **Floating-point representation and numerical precision**

Explore floating-point arithmetic in Python:

(a) Evaluate `0.1 + 0.2 == 0.3`.

(b) Use `np.isclose` to compare floating-point numbers safely.

(c) Print the machine epsilon for double-precision floating point numbers.

4. **Dictionary, Set and Tuple**

   Expanding on Question 3 from the main tutorial, create and print an example of a tuple, a list, a dictionary, and a set to illustrate the usage of different types of brackets in Python: `[]`, `()`, {}.