MATM063: Principles of Data Science, Python LAB

Worksheet 1 (Week 1)

MATM063: Python basics

Werner Bauer
Department of Mathematics

Key Learning: Scripting, Arrays, import, Plotting, Types, Loops, If-Else, Numpy, Slicing and Broadcasting

1 LAUNCHING SPYDER ON LINUX LAP MACHINES

Please run the following commands in a terminal to load Anaconda on your machine using env module:

```
source /etc/profile.d/zz_00-lmod.sh
ml Anaconda3/2021.05 # for an older version of Spyder
ml Anaconda3/2022.05 # for a newer version of Spyder
```

Listing 1. load anaconda in virtual environment

This will allow you to use Anaconda, which comes with Spyder installed in it. Once you are done using Anaconda via env module, it can be unloaded using the following command:

```
ml unload Anaconda3/2021.5
```

Listing 2. unload module

2 CREATING AND RUNNING PYTHON SCRIPTS

We will mostly use the Integrated Development environment (IDE) Spyder, where writing a Python script and executing it can be easily done with the controls. It is useful however, to also understand how python scripts can be created and executed without an IDE. Let's assume we have created and saved a python script named **test.py** with the text editor of your choice. Then, we can execute it in the following ways:

```
 python3 test.py
```

Listing 3. run scripts in bash shell

```
 # start ipython with:
 ipython3
 # run python script within ipython shell with:
 In [1]: run test.py
```

Listing 4. run scripts in ipython

If Anaconda is installed on your machine (as for the Labs), you can install Spyder and additional python packages as follows:

```
 conda install spyder # execute this in a shell
 conda install numpy scipy matplotlib
 # start spyder with:
 spyder
```

Listing 5. install python packages and start spyder

3 IMPORT PACKAGES

Python packages that contain additional functions can be easily imported by the following commands:

```
 import math # imports all functions from package math
 from math import sin # imports only sin function
 import numpy as np # imports numpy package and gives it the name np
```

Listing 6. importing packages and functions

4 VARIABLES

Can consist of: uppercase and lowercase letters (A–Z , a–z), digits (0–9), and the underscore character (_); also, the first character of a variable name cannot be a digit

You can clear your workspace of all variables using the command `%reset` in the console. You can delete a specific variable using the `del()` function. For example, define the variable `x = "hello"`. Look at the variable in the Variable Explorer. Now delete `x` using `del(x)` and look in the variable explorer again.

5 PLOTTING

Matplotlib is a data visualization library for Python. It is an excellent 2D and 3D graphics library for generating scientific figures. You can find out more on the official Matplotlib web page: `http://matplotlib.org/`.

In the following you find an example how to plot functions in the domain $x = [-2, 6]$ with stepsize $\Delta x = 0.5$:

```python
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-2,6.5,0.5)
y1 = x**2
y2 = x**2 -2
plt.plot(x,y1,'g',label='x^2')
plt.plot(x,y2,'r',label='x^2 - 2')
plt.xlabel('x')
plt.ylabel('y1,y2')
plt.title('Plotting functions')
plt.legend()
plt.show()
```

Listing 7. Plotting simple functions

6 WORKING WITH TUPLES, LISTS, SETS, DICTS

Here is a short list of how to create and work with these data array types. Note that Python uses 0-based indexing (unlike Matlab).

```python
# Tuples are ordered and cannot be modified:
t = (2, 'a', 11.2, (3, 4), True)
print(t[2]) # 11.2
```

```python
# Lists are ordered and can be modified:
l = [5, 4, 3]
l.append(2)     # add 2 to the list
print(l)        # [5, 4, 3, 2]
print(len(l))   # 4
l = l[::-1]     # revert list
print(l)        # [2, 3, 4, 5]
l.insert(1, 7)  # insert 7 at the first index
print(l)        # [2, 7, 3, 4, 5]
l.pop()         # remove last item
print(l)        # [2, 7, 3, 4]
print(l[-1]) # 4 # Access the last elements of a list using negative indices
```

```python
# Sets are unordered collections of variables:
s = {'a', 'b', 'c'}
print(s == {'b', 'a', 'c', 'a'}) # True
s1 = s.copy() # copy set
s.add('d') # add an element to the set
print(s)   # {'c', 'b', 'a', 'd'}
print('a' in s) # True
print('z' in s) # False
print(s1.issubset(s))  # True
```

```python
# Dictionaries are unordered collections of key-value pairs:
d = {'red':  [1, 0, 0], 'green':[0, 1, 0], 'blue': [0, 0, 1]}
print(d['red'])   # [1, 0, 0]
print(d.keys())   # dict_keys(['red', 'green', 'blue'])
print(d.values()) # dict_values([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
print(d.items()) # dict_items([('red', [1, 0, 0]), ('green', [0, 1, 0]), ('blue', [0, 0, 1])])
print(d['yellow']) # KeyError: 'yellow'
d.pop('red')  # drop key-value pair: 'red':  [1, 0, 0],
print(d) # {'green': [0, 1, 0], 'blue': [0, 0, 1]}
```

7 LOOPS

There are two types of loop, a `for` loop that is performed a fixed number of times, and a `while` loop that is repeated until a given condition is satisfied.

A `for` loop provides a simple way of doing repeated calculations. For example, for a given vector of x values, we may want to calculate the corresponding vector of values `y = f(x)` for some function `f`.

```python
import numpy
import math
import matplotlib

x = numpy.linspace(0,10,11) # generates a 1D array of integers 0 to 10
y = numpy.zeros(11) # initialises y to be a 1D array of zeros

for i in range(11): #index i cycles through 0 to 10
   y[i] = math.exp(-x[i]/2) # applies the function for each value of x

matplotlib.pyplot.plot(x,y) # plots the vectors
```

Listing 8. FOR loop

As an example for a `while` loop, let us find the smallest value of n such that $2^n \geq 1000$. For this we need to evaluate 2^n for a range of values of n, but we do not know in advance how far we need to go. Thus, a while loop that terminates when a given condition (for continuing with the loop) is no longer satisfied.

```python
n = 1 # initialise n to 1 (=2^1)
p = 2**n # calculate p = 2^n
while p<1000: # keep iterating as long as p < 1000
  n += 1 # increase n by 1
  p = 2**n # recalculate p
# the while loop ends here, due to indentation of next line
print(f"The smallest n is {n-1}")
The smallest n is 9 # output
```

Listing 9. WHILE loop

8 IF-ELSE FUNCTION AND LOGICAL STATEMENTS

You can use following logical statement in if-else functions:

```
Equals: a == b
Not Equals: a != b
Less than: a < b
Less than or equal to: a <= b
Greater than: a > b
Greater than or equal to: a >= b
```

Then the statements are written as:

```python
if expression:
  statements
elif expression:
  statements
else:
  statements
```

if-elif-else statements

9 NUMERICS WITH NUMPY RATHER THAN LIST

Here we show briefly the difference between List (build-in data structure in Python) and the external Numpy package (which has to be imported). The following short example should illustrate why Numpy arrays are more suitable for mathematical and statistical tasks (handling only numbers) than the List structure.

```python
list = [3, 6, 9, 12]
print(type(list)) #output: <class 'list'>
==================================================
division = list/3
#output:
TypeError                                 Traceback (most recent call last)
<ipython-input-7-63612f103a6a> in <module>
```

```
 8  ----> 1 division = list/3
 9
10  TypeError: unsupported operand type(s) for /: 'list' and 'int'
11  ==================================================
12  for i in range(len(list)):
13  list[i]=list[i]/3
14  print(list) #output: [1.0, 2.0, 3.0, 4.0]
```

Listing 10. list structure can apply mathematical operation only one element at a time

Using Numpy arrays, however, we can divide each element in the array by 3 as shown in Line 3, which is far simpler than List structure. All other arithmetic can be done in the same way.

```
1  import numpy as np
2  ar = np.array([3, 6, 9, 12])
3  division = ar/3
4  print(division)
5  print (type(division))
6  #output:
7  [1. 2. 3. 4.]
8  <class 'numpy.ndarray'>
```

Listing 11. array wide division

10 INDEXING, SLICING AND BROADCASTING

Indexing and slicing for Numpy arrays have some similarities to List operations (particularly in 1D). For example, the first index is 0 and negative indexing is supported, i.e. the last element is -1. See the following examples for more details.

10.1 Indexing and slicing 1-D array

```
 1  import numpy as np
 2  a=np.array([10,20,30,40,50,60,70])
 3  print(a[2]) #  take out index 2 # output: 30
 4  print(a[-1]) # take out the end # output: 70
 5  print(a[0:2]) #slice 2 elements starting from index 0 # output: [10 20]
 6  print(a[2:]) #slice elements with index >= 2 # output: [30 40 50 60 70]
 7  print(a[:2]) #slice elements with index <2 # output: [10 20]
 8  print(a[0:6:2]) #slice every other elements between 0 (inclusive) and 6 (not included, i.e. up to 5) # output:
       [10 30 50]
 9  print(a[0::3]) #slice every other three elements from index 0 # output: [10 40 70]
10  print(a[3:-1]) #slice elements from the index 3 to the end (not inclusive) # output: [40 50 60]
11  print(a[-3:-1]) #slice from index -3 to the end (not inclusive) # output: [50 60]
```

Listing 12. slicing in 1D

10.2 Indexing and slicing 2-D array

Indexing 2-dimensional array follows the same logic as 1-D by applying the 1-D indexing onto the row and column separately. In Line 1, notice how the 2-D array is constructed by combining 3 arrays of the same size together.

```
 1  import numpy as np
 2  arr = np.array([[1, 2, 3, 4, 5],
 3          [6, 7, 8, 9, 10],
 4          [11, 12, 13, 14, 15] ])
 5  print(arr[0]) #slice row-index 0 # output: [1 2 3 4 5]
 6  print(arr[:,1]) #slice column-index 1 # output: [ 2  7 12]
 7  print(arr[0:2, 2])  #slice 0<=row-index<2 and column-index 2 # output: [3 8]
 8  print(arr[0:2, 1:4])  #slice 0<=row-index<2 and 1<=column-index<4 # output:
 9  [[2 3 4]
10   [7 8 9]]
11  print(arr[-1, 1:4]) #slice the last row and 1<=column-index<4 # output: [12 13 14]
```

Listing 13. slicing in 2D

10.3 Mathematical Operation and Broadcasting

In Numpy, multiplication, addition, subtraction, division and other elementary functions are performed element-wise for the arrays of the same dimension. Notice that the division by zero is reported as ”inf” (i.e. infinity).

```
import numpy as np
arr1 = np.array([[1, 2, 3, 4, 5],
        [6, 7, 8, 9, 10],
        [11, 12, 13, 14, 15] ])
arr2 = np.array([1, 0, 1, 0, 1],
        [1, 0, 1, 0, 1],
        [1, 0, 1, 0, 1])
print(arr1*arr2) #output:
[[ 1  0  3  0  5]
[ 6  0  8  0 10]
[11  0 13  0 15]]
print(arr1+arr2) #output:
[[ 2  2  4  4  6]
[ 7  7  9  9 11]
[12 12 14 14 16]]
print(arr1/arr2) #output:
[[ 1. inf  3. inf  5.]
[ 6. inf  8. inf 10.]
[11. inf 13. inf 15.]]
```

Listing 14. working with numpy arrays

Numpy also allows these mathematical operations to be applied on the arrays with different dimensions. However, the dimensions of the arrays must be “compatible”. For example, for the 2D array, the number of columns must be identical; hence the operation will be “broadcasted” column-wise for each row, see Line 7-9 below. In Line 10 below, the rule is violated and an error is reported.

```
1  import numpy as np
2  arr1 = np.array([[1, 2, 3, 4, 5],
3  [6, 7, 8, 9, 10],
4  [11, 12, 13, 14, 15] ])
5  arr2 = np.array([1, -1, 1, -1, 1])
6  arr3 = np.array([[1],[-1],[1],[-1],[1]])  #array with 5-rows-1-col
7  print(arr3)
8  print(arr1*arr2)
9  print(arr1+arr2)
10 print(arr1/arr2)
11 print(arr1+arr3)
12 #output:
13 [[ 1]
14 [-1]
15 [ 1]
16 [-1]
17 [ 1]]
18
19 [[  1  -2   3  -4   5]
20 [  6  -7   8  -9  10]
21 [ 11 -12  13 -14  15]]
22
23 [[ 2  1  4  3  6]
24 [ 7  6  9  8 11]
25
26 [12 11 14 13 16]]
27
28 [[  1.  -2.   3.  -4.   5.]
29 [  6.  -7.   8.  -9.  10.]
30 [ 11. -12.  13. -14.  15.]]
31
32 -----------------
33 ValueError  Traceback (most recent call last)
34 <ipython-input-94-46cc542badcb> in <module>
35 9 print(arr1+arr2)
36 10 print(arr1/arr2)
37 ---> 11 print(arr1+arr3)
38
39 ValueError: operands could not be broadcast together with shapes (3,5) (5,1)
```

Listing 15. broadcasting example

There are a large number of Numpy function to manipulate the multi-dimensional array. You can consult the Numpy “manual” at `https://numpy.org/devdocs/user/basics.html`.

10.4 Shape, reshape, size

Here some useful command to determine the size of an array and to modify its shape. For arr above, we can do the following commands:

```
print(arr.shape) # output: (3, 5)
print(arr.size)  # number of all elements: 15
print(arr.reshape(1,15)) # output:
[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]
```

11 EXERCISES

Question 1: Creating, saving, executing py-scripts

1. Write the obligatory “Hello, World!” program in a Python script. The programme should simply output this phrase in the bash.
2. Store the script as a Python-file, e.g., hello-world.py in a folder with suitable naming (e.g. Lab-M063) on your OneDrive or home-directory.
3. Execute the programme in the bash, in Spyder, and in the iPython console.
4. Log out of the system, log in again, find and open hello-world.py with Spyder and execute it again.

Question 2: Numerical calculations

1. Calculate with Python the following expressions: $6 + 12/2 + 4$, $6 + 12/(2 + 4)$, $(6 + 12)/(2 + 4)$, $2 + 20/4 * 5$
2. Determine output of Python commands: $5/2$, $5//2$, $5\%2$
3. Determine the square and square-root of 9, the natural logarithm of 4 and exponential of 2
4. How would you represent π, $2 * 10 * *(-8)$ and $3 * 10 * *6$ in Python?
5. Please determine $sin(\pi)$. What do you realize?

Question 3: Creating functions and plotting them

1. Define $f(x) = x^2 + 6, g(x) = 2x^2 - 1$ as (i) anonymous/lambda functions and (ii) as normal functions. By trying different values, find an integer value of x such that $f(g(x)) = g(f(x))$.
2. Plot both function $f(x), g(x)$ in the domain $x = -10, 10$, give the plot a title, x,y-labels, and add a legend.

Question 4: Create and handle: tuples, lists, sets, dicts

1. Create a tuple that contains (’a’, 3, 3.1, (9,9), False). Verify that the created tuple t is indeed of type ’tuple’. How do you access the entry ’(9,9)’? Try to set the 0th value to 0; what is happening?
2. Create a similar list by using the list() command! Try again to set the 0th value to 0. Append the numbers 2,3 to the list and insert ’True’ at the 3rd index position. Are there similar functions available for tuples?
3. Define the sets $A = \{$’b’, ’a’, ’c’, ’a’$\}$ and $B = \{$’a’, ’c’$\}$. Show that $B \subset A$. Verify this by (i) using a build-in function and (ii) element-wise (hint: check for each element in B if it is in A). Add ’b’ to B (or remove ’b’ from A) and verify that A = B.
4. Create the dictionary $d = \{$ ’key1’:’value1’,’key2’:’value2’ $\}$. How do you access ’value2’? Substitute ’value2’ by ’apples’, add a new item pair with ’key3’:’bananas’ and remove item1 with ’key1’! Print out: (i) only the values of d, and (ii) the key-item pairs. Try to create also $d2 = \{$10:’item1’,(’key2’):’item2’$\}$ and $d3 = \{$10:’item1’,[’key2’]:’item2’$\}$. Please explain what’s happening!

Question 5: Loops, if/else conditions

1. Consider the following list of exam marks: (39, 60, 70, 65, 55, 20, 89, 95, 10, 75, 80). Write a script that iterates over this array and counts the number of ’Fail’ ($x < 40$), ’Lower Second’ ($40 <= x < 60$), ’Upper Second’ ($60 <= x < 70$), and ’First’ ($70 <= x$). Output your results.

Question 6: Solving iteration problems

1. The iteration
$$x_{n+1} = f(x_n) = \frac{1}{2}x_n(1 - x_n)$$
converges to the fixed point $x = 0$ for some initial values x_0. Write a Python function that has inputs `x_0`,`f` (a lambda function- see above),`tol` and the maximum number of iterations `maxit` and returns the number of iterations that are required to converge to zero to an accuracy of `tol`.
   Investigate how the number of iterations for convergence varies with different values of `tol`. What range of values `x_0` gives convergence of the iteration to zero?
2. For the iteration
$$x_{n+1} = 4x_n(1 - x_n), \quad x_0 = 0.1,$$
how many iterations are required for convergence with `tol` $= 10^{-2}, 10^{-3}, 10^{-4}$ and 10^{-5}? Do you think that this iteration will converge as $n \to \infty$? (To help understand this iteration, plot the first 100 iterates of the map by plotting the points (n, x_n), $n = 0, \ldots, 100$.))

Question 7: Solving equations

1. Use `scipy.optimize.fsolve()` to find two real solutions of the equation $x^4 - 3x^3 - 1 = 0$ by choosing different values of the initial guess.
2. Show that the function $f(x) = \sin^2(x) - \sin(2x) - 1$ changes sign between $x = -1$ and $x = 0$. Find a solution of the equation $f(x) = 0$ between $x = -1$ and $x = 0$.
3. Find two solutions of the equations
$$x^3 - y^3 = 1,$$
$$x^4 + xy + y^4 = 2.$$