Introduction to Scientific Computing with Python L2

May 11, 2020

Dr. Pietro Giampa

TRIUMF

2020 Winter Term

Lecture 2/3

1 Outline

```
1) Introduction to Arrays
2) Array Operations
3) Data-Visualization with Matplotlib
4) Exercises
```

2 Introduction to Arrays

An array is a special variable, which can hold more than one value at a time. Imagine if you are measuring some data with your experiment if you have three readings you can enter them in Python as the following (based on our last lecture):

[1]:
```
data_point1 = 45.666
data_point2 = 52.987
data_point3 = 28.735
```

This works great if the number of data points is small, but what about if you have 100+ or 1000+ data points? This is where arrays come into play. An array can hold many values under a single variable name, and you can access the values by referring to an index number. NB: the Array index starts at 0. For example, the above can be re-written using arrays as:

[2]:
```
data = [45.666, 52.987, 28.735]
print('First Data Point',data[0])
```

```
First Data Point 45.666
```

In Python, arrays can store any of the available variable types (Numbers, Strings, Boolean). Arrays can also be formed out of combinations of variable types. In general, Python always assumes what the datatype is based on the input, however, if necessary the variable type can be directly passed

to the program (requires NumPy). It is always possible to identify the current data-type of one array with the command dtype.

At any point you can check the length of an Array by using the function len(). Example:

[3]:
```
nentries = len(data)
print('Number of Entries in the Array:',nentries)
```

```
Number of Entries in the Array: 3
```

Arrays can also be augmented by adding new entries. This is done by simply appending a new value to an already defined array by using the function append. Note, this changes the length of your Array.

[4]:
```
data.append(56.565)
data.append('Entry')
nentries = len(data)
print('Current Data Array:',data)
print('Number of Entries in the Array:',nentries)
```

```
Current Data Array: [45.666, 52.987, 28.735, 56.565, 'Entry']
Number of Entries in the Array: 5
```

Similar to the ability to add entries to an array, it is also possible to remove parts from it. To remove entries there are two options: pop(index) or remove(value). The pop command requires an index and will remove the value at the given index, while remove gets rid of all entries equal to a given value.

[5]:
```
print('Current Data Array:',data)
data.pop(0)
print('Current Data Array:',data)
data.remove(56.565)
print('Current Data Array:',data)
```

```
Current Data Array: [45.666, 52.987, 28.735, 56.565, 'Entry']
Current Data Array: [52.987, 28.735, 56.565, 'Entry']
Current Data Array: [52.987, 28.735, 'Entry']
```

Arrays can be purged at any time via the command clear(). This action will remove all entries from a given array permanently, so use carefully.

[6]:
```
print('Current Data Array:',data)
data.clear()
print('Current Data Array:',data)
```

```
Current Data Array: [52.987, 28.735, 'Entry']
Current Data Array: []
```

Thus far we have only dealt with one-dimensional arrays, however, in Python, it is possible to use an array as a multi-dimension object. (Basically, NxM matrices). For instance here how to construct a 3x3 array:

[7]:
```
data_3b3 = [['x1','x2','x3'],['y1','y2','y3'],['z1','z2','z3']]
print('Select Element 1x3:',data_3b3[0][2])
```

```
Select Element 1x3: x3
```

2.0.1 Example 2.1.1

Use the data below to setup two arrays where the first corresponds to the data timestamp and the second is the actual data. Remove all the data with timestamp prior to 13:00:00, then add a new entry with timestamp = 15:52:00 and data = 56.7. Combine the two one-dimensional arrays into one multi-dimensional array, and print the first entry in the new array.

```
TIME - Data
12:40:00 45.3
13:52:00 53.7
14:32:00 100.4
14:53:00 134.8
15:32:00 76.3
```

[8]:
```
############################################################
## Example 2.1.1                                          ##
## Introduction to Scinetific Programming with Python ##
##                                                        ##
## Pietro Giampa, TRIUMF, 2020                            ##
############################################################

### Define Needed Libraries ###
import numpy as np

### Define Arrays ###
timestamp = ['12:40:00','13:52:00','14:32:00','14:52:00','15:32:00']
data_points = [45.3, 53.7, 100.4, 134.8, 76.3]
entry_to_remove = []

### Find Early Datapoint ###
for k in range(len(timestamp)):
    if timestamp[k]<'13:00:00':
        entry_to_remove.append(k)

### Remove Early Data-point ###
for i in range(len(entry_to_remove)):
    timestamp.pop(entry_to_remove[i])
    data_points.pop(entry_to_remove[i])

### Add new Data-point ###
timestamp.append('15:52:00')
data_points.append(56.7)
```

```
### Combine Arrays into nD Array ###
data = [timestamp,data_points]
print('New n-dim Array:',data)
```

```
New n-dim Array: [['13:52:00', '14:32:00', '14:52:00', '15:32:00', '15:52:00'],
[53.7, 100.4, 134.8, 76.3, 56.7]]
```

3 Array Operations

In the last lecture, we went through basic operations with variables, Arrays can be treated very similarly thanks to the NumPy library. Furthermore, NumPy offers standard Matrix operations for multi-dimensional arrays.

Create Pre-Set Arrays Python and NumPy will let you create and fill basic arrays with a quick and fast 'one-line' commands, which can be quite handy in long programs. The following are a list of useful NumPy commands to remember for array definitions:

- ARANGE(n) = creates an array integer (or floats) of n entries, starting from 0 and increasing to n-1.
- RANDOM.RANDOM((x,y) or n) = creates an array of size X*Y (or a 1D array of size n) filled with random numbers generated from 0 to 1.
- ZEROS((x,y) or n) = Similar to the random option, generates a one or multi-dimensions array filled with zeros.
- FULL((x,y),k) = Generates an array of size X*Y filled with values equal to the input k.
- EYE(d) = Creates a unit matrix of dimension d.

[9]:
```
### Generates an array of 10 entries from 0 to 9
array1 = np.arange(10)
print('Array1 ARANGE(n):','\n',array1,'\n')

### Generates a 2x2 array of random numbers
array2 = np.random.random((2,2))
### Generates a linear array with 4 random numbers
array3 = np.random.random(4)
print('Array2 RANDOM.RANDOM((x,y)):','\n',array2,'\n')
print('Array3 RANDOM.RANDOM(n):','\n',array3,'\n')

### Generates a 3x1 array of Zeros
array4 = np.zeros((3,1))
print('Array4 ZEROS((x,y)):','\n',array4,'\n')

### Generates a 2x3 array fillied with a given value
val = 'Finn'
array5 = np.full((2,3),val)
print('Array5 FULL((x,y),k):','\n',array5,'\n')
```

```
### Generates a 2x2 unit matrix
array6 = np.eye(2)
print('Array6 EYE(d):','\n',array6,'\n')
```

```
Array1 ARANGE(n):
 [0 1 2 3 4 5 6 7 8 9]

Array2 RANDOM.RANDOM((x,y)):
 [[0.49463498 0.88820492]
 [0.82458858 0.80664305]]

Array3 RANDOM.RANDOM(n):
 [0.68637724 0.96639085 0.03839259 0.28231334]

Array4 ZEROS((x,y)):
 [[0.]
 [0.]
 [0.]]

Array5 FULL((x,y),k):
 [['Finn' 'Finn' 'Finn']
 ['Finn' 'Finn' 'Finn']]

Array6 EYE(d):
 [[1. 0.]
 [0. 1.]]
```

Basic Math Similarly to what we saw in the last lecture, the basic math operators can be used for arrays the same way we learn to use them for individual variables. Recall that similarly to variables, you can either create a new array or over-write the current array with the operation.

[10]:
```
### Generate Some Basic Arrays
A1 = np.arange(5)
A2 = np.full(5,2)
print('Array A1:',A1)
print('Array A2:',A2,'\n')

### Addition and Subtraction
Asum = np.add(A1,A1)
Asub = np.subtract(A1,A1)
print('Addition       A1+A1:',Asum)
print('Subtraction    A1-A1',Asub)

### Multiplication and Division
Amul = np.multiply(A1,A2)
```

```
Adiv = np.divide(A1,A2)
print('Multiplication A1*A1:',Amul)
print('Division       A1/A1',Adiv)

### Square Root
Asqrt = np.sqrt(A1)
print('Square Root    A1:',Asqrt)

### Summing Over All Elements
AsumE = np.sum(A1)
print('Elements Sum   A1:',AsumE)
```

```
Array A1: [0 1 2 3 4]
Array A2: [2 2 2 2 2]

Addition       A1+A1: [0 2 4 6 8]
Subtraction    A1-A1 [0 0 0 0 0]
Multiplication A1*A1: [0 2 4 6 8]
Division       A1/A1 [0.  0.5 1.  1.5 2. ]
Square Root    A1: [0.         1.         1.41421356 1.73205081 2.        ]
Elements Sum   A1: 10
```

Matrix Algebra Arrays can be interpreted as lists of data-points, but they can also be considered as standard matrices. In NumPy, all basic matrix algebra operators are available and Python can be used for all those calculations (other packages could offer more complex options!). Here is a list of the primary functions:

- numpy.dot(x,y): Dot product between arrays (matrices) X and Y
- numpy.cross(x,y): Cross product between (matrices) arrays X and Y
- x.T: Transposition of array (matrix) X
- numpy.linalg.det(x): Determinant of array (matrix) X
- numpy.linalg.norm(x): Magnitude of array (matrix) X
- numpy.var(x): Variance of an array (matrix)

3.0.1 Example 2.2.1

If you have a particle of mass m=0.02 [kg], with a given position vector $r = (2\mathbf{i} - \mathbf{j} - 3\mathbf{k})$ [m] and a velocity matrix of $v = (3\mathbf{i} + 5\mathbf{j} - 4\mathbf{k})$. Estimate the particle's angular momentum about the origin and calculate the magnitude.

Recall that the angular momentum $\mathbf{L}$ of a particle of mass $\mathbf{m}$ is given by:

$\mathbf{L} = \mathbf{mr} \times \mathbf{v}$

[11]:
```
############################################################
## Example 2.2.1                                          ##
## Introduction to Scinetific Programming with Python ##
##                                                        ##
```

```
## Pietro Giampa, TRIUMF, 2020                          ##
############################################################

### Define Needed Libraries ###
import numpy as np

### Define Variables
m = 0.02 # [kg] partical mass

### Define Data Arrays
r_array = [2., -1., -3.] # [m] position array
v_array = [3., 5., -4.] # [m/s] velocity array

### Create the multi-D Angular Momentum Array
L_array = np.cross(r_array,v_array)
L_array = np.multiply(L_array,m)
L_det = np.linalg.norm(L_array)

### Print the Results
print('Angular Momentum L:',L_det,'[kg*m2/s]')
```

```
Angular Momentum L: 0.46086874487211654 [kg*m2/s]
```

4 Data-Visualization with Matplotlib

Learning how to properly visualize your data and results is a crucial component of data analysis. If a plot is well constructed, whoever sees it can extrapolate all the conclusions he/she/they need.

Matplotlib is the best available option for data-visualization in Python (not even close). The latest documentation versions can be found at https://matplotlib.org/3.1.1/gallery/index.html (lots and lots of tutorials and examples).

Mostly a translation from the Matlab library, Matplotlib is available for Linux, ChromeOS, macOS, and Windows.

Let's start with the basic, first like any other external library we need to call Matplotlib. I suggest the following:

[12]:
```
import matplotlib.pyplot as plt
```

Matplotlib has plenty of option when it comes to visualization and data representation, here some of the standards:

- plt.plot(x,y) - Creates a standard 1D histogram, given arrays x and y.
- plt.hist(x, nbins) - Creates a histogram based on array x, given a number of bins equal to nbins.
- plt.bar(x,y,w,align='center') - Creates a Bar-chart given x and y, with bars of width w. Bars center will be placed at value x.

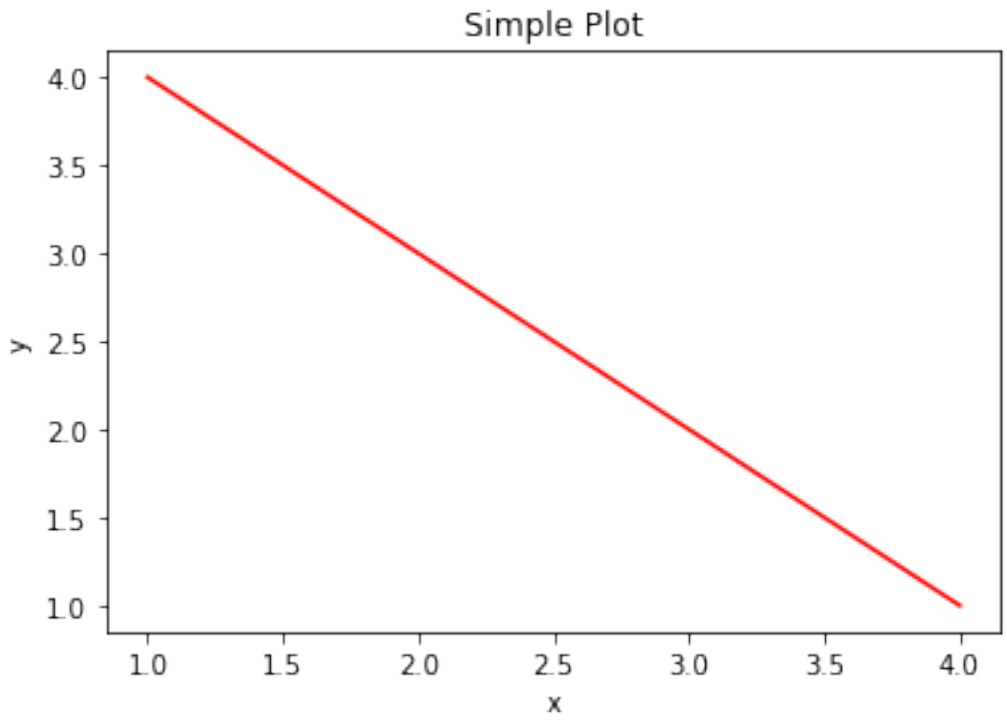
- plt.scatter(x,y) - Creates a scatter plot with Markers, given arrays x and y.
- plt.errorbar(x,y,xerr,yerr) - Creates a standard plot where each point is assigned an error given by the arrays xerr and yerr.
- plt.hist2d(x,y,nbins) - Creates a 2D histogram based on arrays x and y, and given a bin number of nbins.

A few useful Tips for visualization:

- Always include a Title
- Always label your axis, including units where suitable
- Data should occupy a minimum of 2/3 of your canvas
- Add texts or highlights only if it enhances the message of the results
- Less is more

Here a quick example for a simple plot:

[13]:
```
### Define Needed Libraries ###
import matplotlib.pyplot as plt

### Define Same Arrays
x = [1.,2.,3.,4.]
y = [4.,3.,2.,1.]

### Draw Basic Plot, Red Line
plt.plot(x,y,'r-')
plt.title('Simple Plot')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



4.0.1 Example 2.3.1

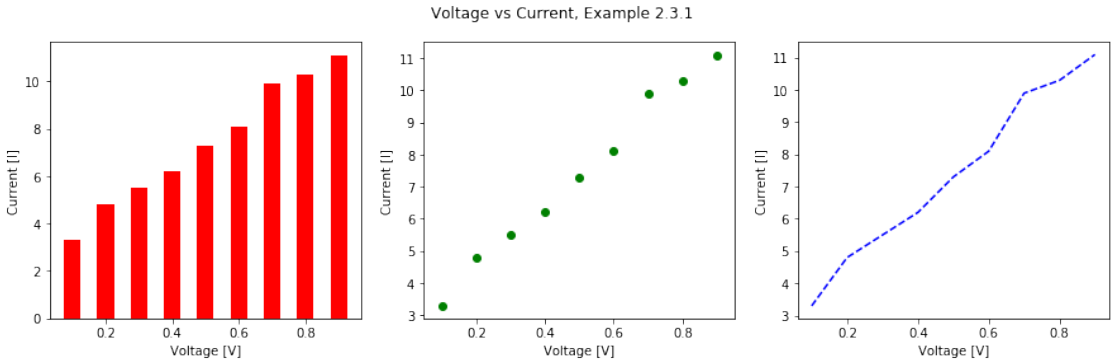
You are in the lab and you are measuring the current of a circuit, as a function of the applied voltage. Your results are given below:

Voltage = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] Current = [3.3, 4.8, 5.5, 6.2, 7.3, 8.1, 9.9, 10.3, 11.1]

Plot the results above as a standard plot, a bar chart, and a scatter plot. Include all three of them on the same Canvas.

[14]:
```
############################################################
## Example 2.3.1                                          ##
## Introduction to Scinetific Programming with Python ##
##                                                        ##
## Pietro Giampa, TRIUMF, 2020                            ##
############################################################

### Define Needed Libraries ###
import numpy as np
import matplotlib.pyplot as plt

### Define X array
Voltage = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] # [Volts]
```

```
Current = [3.3, 4.8, 5.5, 6.2, 7.3, 8.1, 9.9, 10.3, 11.1] # [mAmp]

### Split the Canvas in 3
fig, axs = plt.subplots(1, 3, figsize=(15,4)) #Note the size can be adjusted
fig.suptitle('Voltage vs Current, Example 2.3.1') #Set Title

### Plot Bar Chart, V vs I, width=0.05 with Center aligment, Red Color
axs[0].bar(Voltage,Current,width=0.05,align='center',facecolor='red')
axs[0].set_xlabel('Voltage [V]')
axs[0].set_ylabel('Current [I]')

### Scatter plot, V vs I, Markers set to green cricles
axs[1].scatter(Voltage,Current,c='g',marker='o')
axs[1].set_xlabel('Voltage [V]')
axs[1].set_ylabel('Current [I]')

### Basic Plot, V vs I, Blue dotted line
axs[2].plot(Voltage,Current,'b--')
axs[2].set_xlabel('Voltage [V]')
axs[2].set_ylabel('Current [I]')

### Plot the results
plt.show()
```



5 Exercises

Problem 1 Given the following two matrices A and B:

A = [[0.4, 6.4, 2.2],[0.1, 0.3, 2.5],[1.8, 9.9, 0.4]]

B = [[2.1, 2.4, 2.7],[2.1, 5.1, 4.3],[3.3, 8.8, 6.4]]

Write a program that first transposes each matrix, then calculates the determinant, magnitude and variance. Moreover, the program should calculate the dot and cross product between A and B. All

results have to be printed on screen with appropriate descriptions.

Problem 2 You are designing an experiment and you need to understand how far alpha particles, between 1-5 MeV, travel through foils of Au, Al, Cu. Go to the ASTAR database (https://physics.nist.gov/PhysRefData/Star/Text/ASTAR.html) and look up the range (CSDA) tables for the three elements. Extrapolate the data from 1 to 5 MeV and generates four different arrays (Energy, Range_Au, Range_Al, Range_Cu). Plot the range as a function of energy, for all three materials, on the same canvas (label everything).

NB: that the range given on ASTAR is in g/cm2, so you will have to scale your array by the material density to get units of length.

Problem 3 Start by writing a generic function that model Gaussian distribution with x, μ, and σ as input parameters. First set the mean to 2.5 and sigma to 1.2, plot the function for values of x from 0 to 5 (label everything as needed). Using the function above generates a 3x3 matrix where each entry is a different combination of x, μ, and σ. Print the newly created matrices on the screen with required comments.

Now, consider that each row corresponds to the cartesian coordinates of a unique triangle, use matrix algebra to determine the area of each of the three triangles.