COMP0043 Python-Matlab syntax conversion explanation

The primary packages utilized across all scripts were NUMPY as it mimics majority of Matlab’s functionalities (eg: trapz,exp,pi,linspace,etc) and matplotlib was used to chart the content across almost all scripts.

Packages used:

From numpy as np to replicate certain functions such as exp, inserting, searching, deriving pi,etc.

From pandas as pd to convert arrays or matrices into dataframes.

From Scipy.stats import \*, whenever using \* from a library, it will import all modules that constitute the library. This library was used to import distribution functions such as log.norm,norm,etc.

From Pytictoc for replicate Tic Toc CPU time count as done in Matlab

From Mpl\_toolkits.mplot3d import Axes3D

From Matplotlib import cm (color patterns)

From Matplotlib.ticker.import LinearLocator to design 3D modules

Import QuantLib as ql

From QuantLib import \*, whenever using \* from a library, it will import all modules that constitute the library. This library was used to derive an analytical solution in the bs script.

From scipy.fft import fft,ifft,fftshift this package was used to replicate Fourier’s transform.

Script:

Parameters, these can either be labelled one by one or in Python it is possible to define various variables in one row, for example:

0.4,0.2,120=mu,sigma,nsteps

All lists/arrays in python start with 0, hence when specifying the start value of an array use [0] and [-1] as the last value, instead of using end as shown in matlab. Notice [] are used instead of () to represent a list. When specifying a specific length in an array by using calculations, for example:

Array1[int(10/2):int(10/2)+10]

the use of int is needed to convert floats into integers so that the array can correctly isolate the new length of the array.

To be able to set a value to the power of another, in Matlab its ^, in Python its \*\* or use np.power(value, its power).

To transpose in python, you need to specify .transpose() instead of using .’ in MATLAB.

To multiply matrices use np.dot(array1,array2) while in Matlab you would use .\*.

No need to use ; to specify the ending of a variable, just write on another line to start defining new variable or function or action.

In python # is commonly used to indicate a comment row instead of % in MATLAB.

To reference a dynamic value within a text string, you can use format, for example:

Plt.title(‘Arithmetic Brownian Motion using mean={} and sigma={}’.format(mu,sigma))

So that if the variables dynamically change, so will the values indicated imported in the string.

To derive a grid:

-linspace can be used in Python

-To replicate the logic of deriving a grid from using Matlab’s: a:deltax:b, np. Arrange is used as an alternative in python.

Np.arange(start\_value,end\_value, difference in timesteps)

Functions requiring specific libraries:

- pdf(‘Normal’) in Matlab but norm.pdf (using scipy.stats) where you specify the array, the location which would be represented by the mean and the scale as the standard deviation.

- cdf(‘Normal’) in Matlab but norm.cdf (using scipy.stats) where you specify the array, the location which would be represented by the mean and the scale as the standard deviation.

- pdf(‘ncx2’) in Matlab but nxc2.pdf( using scipy.stats).

- pdf(‘Exponential’) or cdf(‘Exponential) in matlab, in python it would be expon.pdf(dataframe,location=0,scale=mu) where location refers to where the exponent curve starts and scale refers to the mean set.

-pdf(‘lognorm’) or cdf(‘lognorm’) in MATLAB where in python (using scipy.stats) lognorm.pdf(dataframe,s=sigma,loc=mu) was used where s refers to the sigma used and scale as mu.

- Np.insert to adds an element to an array, for example, if you want to add 0 as the starting value for each row, use np.insert(dataframe,value,position) and apply cumsum separately instead of using [0;cumsum(dataframe)] in one go as shown in Matlab.

-.reshape() (built-in python) where if you want to specify zeros(nsteps,npaths), in python np.zeros() can only take 1 value so provide nsteps\*npaths then reshape the array intro matrix .reshape(npaths,nsteps).

plt for plotting (using matplotlib). The logic is almost identical to MATLAB just for plt.legends(), legends can either be specified as label=”” in plt.plot or within [] in plt.legend().

-timing.tic() and timing.toc() (using TicToc), which replicates the use of tic and toc in Matlab. To then store the values of the timers, timing.tocvalue() is used.

-np.random.rand() (using NUMPY), where it replicates the same rand logic as Matlab, but in Python random needs to be specified first.

-plt.plot() with plt.xscale(log) and plt.yscale(log) (using matplotlib) to replicate loglog in MATLAB.

-plt.hist(array,bins=x,density=True) (using matplotib), by incorporating density=True the package automatically derives binedges and normalizes the frequency (ie no need to divide the histogram values by the sample\*(x(2)-x(1))

-xcorr(dataframe,’unbiased’) in MATLAB but in python to replicate an autocorrelation you need to use np.correlate(dataframe,dataframe,mode=2) where mode=2 refers to mode=full which matches the same xcorr in MATLAB of xcorr(dataframe,’normalized’). To try and match unbiased xcorr, np.correlate was multiplied by 1/number of steps+1 as indicated in MathWorks Help Center.

-blsprice(initial stock price,strike,rate,time,sigma and dividend rate) in Matlab but to derive a analytical solution in Python you need to use QuantLib to import all modules to replicate each step of the pricing mechanism: ql.Date, ql.Option.Call, ql.EuropeanExercise and many others.

-real(fftshift(fft(ifftshift(….))) in Matlab, in Python np.fft.fftshift was used hence fft modules under Numpy were used to replicate the Fourier Transform method.

-poissrnd() in MATLAB where in python (using NUMPY) its np.random.poission().

Creating Loops in Python:

- Create an empty list signified by [] and another list will be appended into it using the list.append(‘list you want to append’) function. Then specify a list or a range where if the length of a array is used, in python len() must be used. Moreover, if you want to reference many lists in one go in a single for loop, use zip as shown below:

combo=[]

for i,j in zip(len(combo),range(10):

comb=i+1

combo.append(comb)

-Alternatively, create an array similarly to matlab made up of a calculation or use np.zeros or np.ones to fill an array to then be able to reference the position of each new value indicated in the for loop, for example:

combo=np.zeros(10)

for i in range(10):

combo[i]=i+1

-Alternatively, you can use lambda to derive the mean (or variance using np.var) per timestep for example for an entire dataframe:

list(dataframe.apply(lambda row: np.mean(row),axis=1))

compared to:

mean(dataframe,2) where MATLAB already does the mean per time step.

Object-Oriented Programming:

-Defining a function in Python vs Matlab:

def calibration\_ml(dataframe,timestep)):

return mu,sigma,alpha

vs

function [mu,sigma,alpha]=calibration\_ml(dataframe,timestep)

-To utilize variables from a defined function that was created in another Jupyter notebook, you need to import the function from a .py file (needs to be .py). Example:

from calibration\_ml import calibration\_ml

muS,sigmaS,alphaS=calibration\_ml(dataframe,timestep)

Where it’s from ‘name of the .py file’ import ‘name of the function’

While in MATLAB just define the variables:

[muS,sigmaS,alphaS]=calibration\_ml(dataframe,timestep)