MS6021 Scientific Computing

TOPICS:

Python BASICS,

INTRO to PYTHON for Scientific Computing

Preliminary Notes on Python (v MatLab + other languages)

- Install Anaconda (NOT urgent! ; unless you already have Python on your computer)
  - Check whether you have a 64bit (more likely) or 32bit machine
  - Install the relevant **Python 3.6 version** (Python 2.7 is slightly different, but can be used as well)
  - On completing the installation, try to open **Spyder**

- When you enter **Spyder** (available on installing **Anaconda**), you'll be able to (as in MatLab)
  - use the Command Window, or
  - create scripts (they will have an extension .py)

- Simplest commands (play with them):

```
print("hello world!")
x=3; print("x =", x)
# comments
5/3
5./3
2**3  (NOT that this is equivalent to 2^3 in MatLab!)
```

- **Basic Mathematical functions need to be imported (i.e. loaded)!!!**

  sin(1) will produce an error, i.e. Python does not recognize mathematical functions such as sin.

- To get access to this function (and many other mathematical functions), we need to **import** them from a **Python library**, also called a **module** (such as **math** or **numpy**). A few approaches are available (one may employ all of them on different occasions):

  (a)
  ```
  import math
  math.sin(1)  #  . indicates that we use function sin from module math
  ```
  (b)
  ```
  from math import sin
  sin(1)
  ```
  (c)
  ```
  from math import *
  #  here * means that ALL functions are imported from math
   sin(1), log(3)     # we can use other math functions, such as log
  ```

- In the above (a)-(c) , the module math can be replaced by a bigger module numpy (replace also sin by exp in your examples) Furthermore, a version of (a) with numpy can be replaced by

  (a')
  ```
  import numpy as np          # very typical abbreviation!!
  np.exp(1)                   # instead of numpy.exp(1)
  ```

COPYING complex data types (lists, arrays…) --- BEWARE!!!

- When you create x=5. and then y=x, then changing y does NOT change x
- For more complex data types, assignments in Python usually do not copy the actual objects (to achieve high performance):

```
A = [1, 2, 3, 5]
B = A                 # now B is referring to the same list data as A
B[0]=100              # so changing B affects A (and visa versa)
                      # print A and B to check this!
```

- If you want to really create a copy, instead of B=A, use **B = A[:]**

  Perform the above example with this replacement and see the difference!

- See also [Section 3.11, Python for Computational Science and Engineering] (the link is given below)
  --- for similar examples with arrays

- Instead of end to finish a loop, if statement, a function, **program blocks are defined by their indentation level:**
- Compare:

PYTHON:	MATLAB

if x>0:
  print ("x is positive")
  print("x =", x)
  # end of **indentation**=end of loop
	if x>0
	  'x is positive'
	  x
	end

if x>0:
  print ("x is positive")
  # end of **indentation**=end of loop
print("x =", x)
	if x>0
	  'x is positive'
	end
	x

def myfun(x,y):
  return x+y**3 # end of function
 	
print(myfun(1,4))
	function z = myfun(x,y)
	  z=x+y.^3
	end
	myfun(1,4)

Indexing Range in a list (array)

- **Indexing starts at 0!** (while in MatLab from 1)
  Negative indices are possbile

```
A=[8,4,3,5,9,12]        # here we have created a LIST A
print(A[0])             # prints the FIRST entry 8
print(A[2])             # prints the third entry 3
print(A[-1])            # prints the FINAL entry 12
                        # check A[-2], A[-3], A[-6]
```

Note that A[7] and A[-7] will result in an error!

- List can be created and used in **loops** using range(start, stop) or range(start, stop, step)

Try range(3, 12) and range(3, 15, 4) inside the following for loop.

Note that in both cases, the final element is 11 (NOT 12 or 15!).

```
for x in range(-3,3):
   print(x)
```

- We can extract a list part using **[start : stop]** or **[:]** or **[: stop]** or **[start :]**

```
A[0:3]      # does NOT include A[3]
A[1:-1]     # try to predict the output before running this command!
A[1:6:2]    # try to predict/explain the output of A[1:6:2]
```

lambda functions
(similar to anonymous functions in MatLab)

- In Python we can define functions, using the lambda keyword:

```
f1 = lambda x: x**2
# is equivalent to
def  f2(x):
    return x**2
```

To check this, evaluate:

```
f1(2),  f2(2)
```

- Independently of how the function is defined, it can be used as an input for another function…

MatLab-ish capabilities

- **numpy** – multidimensional data arrays
- **matplotlib** – 2d and 3d plotting
- **scipy** library of scientific algorithms
  - builds on top of the low-level NumPy framework for multidimensional arrays
  - provides a large number of higher-level scientific algorithms
  - sparse matrices among other features
  - will be skipped in this course

ARRAYs with **numpy**

- To use module **numpy**, you need to import the module functions, using for example:

  ```
  from numpy import *
  ```

- Now, to create new vector and matrix arrays, we can use the numpy.array function:

  ```
  M = array( [ [1, 2], [3, 4], [5, 6] ] )        # each interior [..] defines a row
  print (M)
  ```
  Try M[0,0] , M[1,2] , M[:,0] , M[1,:] , M[-1,:] etc, with various ranges
  Operations with arrays: try M*2+3 and M**2 and sin(M)
  ---all array operations are applied **elementwise**!

- Assignments B = M do not copy the actual array!

  ```
  B = M                    # now B is referring to the same list data as M
  B[0,0]=100 # so changing B affects M (and visa versa)
                           # print M and B to check this!
  ```

Replace B = M by **B=M[:]** or **B=copy(M)** to **create an actual copy** of M (check!)

- For linear systems, use numpy.linalg.solve(A,b)
  (similar to A\b in MatLab)

2d plotting with matplotlib

- You may start with

```
from matplotlib.pyplot import *
from numpy import *    # unless you already imported numpi
```

- Then we can plot almost like in MatLab

```
x = linspace(0,1,20)                                  # linspace is from module numpy
plot(x, x**2)                                         # graph of one function
xlabel("text for x-axis") # plot, xlabel, ylabel are from matplotlib.pyplot
ylabel("text for y-axis")
plot(x, x**2, '-o',   x, cos(x), '--*')     # graph of two functions
```

- Alternative (and more preferable version of the above)

```
import matplotlib.pyplot as plt
from numpy import *                    # unless you already imported numpi
x = linspace(0,1,20)                   # linspace is from module numpy
plt.plot(x, x**2, '-o',   x, cos(x), '--*')
plt.xlabel("x-axis")
plt.ylabel("x^2 and cos(x)")
```

- To add curves to the existing figure (from a script!)

```
plt.hold(True)                         # ---similar to MatLab hold on
plt.show()                             # ---to finalize + show you a figure
plt.plot(x, 1-x**3, '-o')
```

Further Reading → Exercises

- As a crash course, you may look through the slides:
  http://www.seas.upenn.edu/~cis391/Lectures/python-tutorial.pdf
  (Pp. 17- 32, ignore tuples; pp. 50-61, 72-74, 76-79, 86-88)

- Main REF (may be of use in future as well):
  Python for Computational Science and Engineering (A beginner's guide), by Hans Fangohr
  http://www.southampton.ac.uk/~fangohr/training/python/pdfs/Python-for-Computational-Science-and-Engineering.pdf
  (Sections 2.1, 2.3-2.5, 2.6.2, 2.7-2.9, 2.11 and beyond)

- NumPy for MatLab users notes:
  https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

- Also you may look into an open-source Springer book
  Programming for Computations – Python by Svein Linge, Hans Petter Langtangen,
  http://link.springer.com/book/10.1007%2F978-3-319-32428-9
  ---this is a Python version of the MatLab book you already know…
  (Sections 1+2 and beyond)

Python Basics Worksheet

- *Programming for Computations – Python: Ex. 2.2, 2.4, 2.6*

- Rewrite your code(s) of Ass.#3, part II, into Python and reproduce the results obtained (excluding the use of the sophisticated MatLab build-in function quad).

- Rewrite your codes of Ass.#4 into Python and reproduce the results obtained (optional)