UECM1703 Introduction to Scientific Computing Topic 1: Introduction

Dr Liew How Hui

Oct 2021

About This Subject — MS TeamCode: 8n0q0ct

Week 1: No practical in Week 1 (Friday morning no practical). Friday Topic 1 Lecture (Liew HH)

Week 2: Topic 2 (Array) Lecture + Practical by Liew HH

Week 3: Friday Test (20%) during lecture (Liew HH) + Practical by Liew HH + Tuesday Lecture by Dr Goh YK

Week 4: Programming Lecture + Practical + Assignment (20%) by Dr Goh YK

Week 5: Graphics Lecture + Practical by Dr Goh YK

Week 6: Modelling Lecture + Practical + Quiz (10%) by Dr Yong CK

Week 7: Model Fitting Lecture + Practical + Quiz (10%) by Dr Yong CK

Course Learning Outcomes

1. Perform vector and matrix operation using computer software.
2. Plot graphs, curves, surfaces and contours using computer software.
3. Write program scripts for mathematical software.
4. Apply computer software to solve system of linear equations, eigenvalue problems or matrix factorisation problems.
5. Apply computer software to perform curve fitting on a set of data.

Course Assessment

The Teaching Plan is not clear:

- Test (20%): Topic 1 (CO3) & Topic 2 (CO1, CO3) + Part of Topic 3
- Assignment (20%): Remainder of Topic 3 & Topic 4 (CO2)
- Quiz (20%): Topic 5 (CO4) & Topic 6 (CO5)

Final Assessment / Exam (3 Hours)

After Week 7:

- Question 1: 10% (CO1, Topic 1 & 2)
- Question 2: 10% (CO2, Topic 4)
- Question 3: 10% (CO3, Topics 1, 2 & 3)
- Question 4: 5% (CO4, Topic 5)
- Question 5: 5% (CO5, Topic 6)
- 2.5 Hours Working
- 0.5 Hour scan and submission
- Write in Word, Convert to PDF
- Submit PDF file

Software

Why Python?

Answer:

- Popular & “easy” to learn
- Many online resources: just search “Python (language)” on the Internet.

How to work with Python?

Answer: Python Software Environments:

- Text Editor + Python Shell (`https://www.youtube.com/watch?v=QKBcHuA3VJE`)
- Jupyter Notebook (used by many instructors)
- Spyder: all-in-one IDE from Anaconda Python

Reference

Pratap, R., 2006. Getting started with matlab 7: A quick introduction for scientists and engineers. New York: Oxford University Press.

This course follows MATLAB's syllabus using equivalent Anaconda Python commands.

https://www.mathworks.com/pricing-licensing.html
MATLAB education annual licence: USD 275 $\approx$ RM1,151

Outline

Online Help

- https://www.python.org/about/help/
- General help: help()
- Specific help: help([object]) $\rightarrow$ If the argument is a string, then the string is looked up as the name of a module, function, class, method, keyword, or documentation topic, and a help page is printed on the console. If the argument is any other kind of object, a help page on the object is generated.
- Numpy specific help (similar to MATLAB): np.info(), np.lookfor()

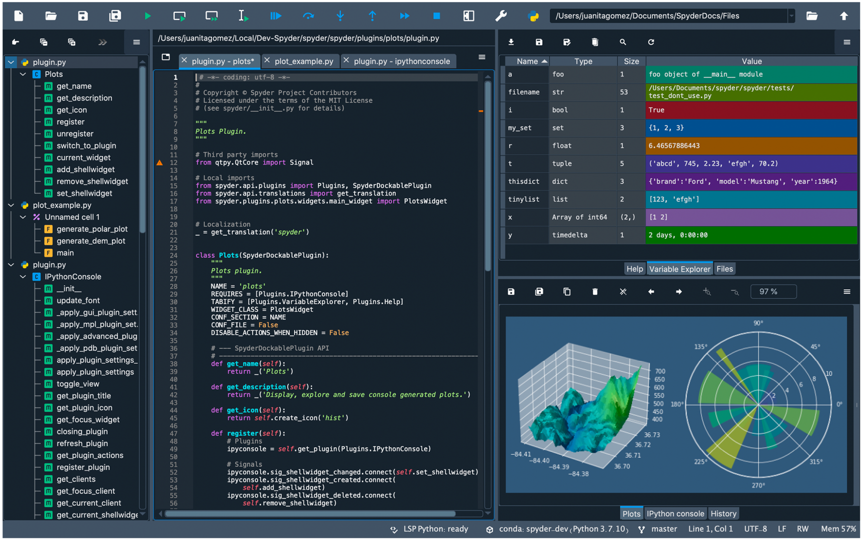
Input-Output

Input:

- Spyder: Directory Listing, Editor Window, Command Window, Workspace window
- Jupyter: Interactive with In & Out
- Python Shell: No spacing in front of usual command

Input-Output

Spyder: Directory, Editor, Command, Workspace



/Users/juanitagomez/Documents/SpyderDocs/Files

- plugin.py
  - Plots
    - get_name
    - get_description
    - get_icon
    - register
    - unregister
    - switch_to_plugin
    - current_widget
    - add_shellwidget
    - remove_shellwidget
    - set_shellwidget
- plot_example.py
  - Unnamed cell 1
    - generate_polar_plot
    - generate_dem_plot
    - main
- plugin.py
  - IPythonConsole
    - __init__
    - update_font
    - _apply_gui_plugin_sett
    - _apply_mpl_plugin_set
    - _apply_advanced_plug
    - _apply_pdb_plugin_set
    - apply_plugin_settings_
    - apply_plugin_settings
    - toggle_view
    - get_plugin_title
    - get_plugin_icon
    - get_focus_widget
    - closing_plugin
    - refresh_plugin
    - get_plugin_actions
    - register_plugin
    - get_clients
    - get_focus_client
    - get_current_client
    - get_current_shellwidge

/Users/juanitagomez/Local/Dev-Spyder/spyder/spyder/plugins/plots/plugin.py

```python
# -*- coding: utf-8 -*-
#
# Copyright © Spyder Project Contributors
# Licensed under the terms of the MIT License
# (see spyder/__init__.py for details)

"""
Plots Plugin.
"""


# Third party imports
from qtpy.QtCore import Signal

# Local imports
from spyder.api.plugins import Plugins, SpyderDockablePlugin
from spyder.api.translations import get_translation
from spyder.plugins.plots.widgets.main_widget import PlotsWidget


# Localization
_ = get_translation('spyder')


class Plots(SpyderDockablePlugin):
    """
    Plots plugin.
    """
    NAME = 'plots'
    REQUIRES = [Plugins.IPythonConsole]
    TABIFY = [Plugins.VariableExplorer, Plugins.Help]
    WIDGET_CLASS = PlotsWidget
    CONF_SECTION = NAME
    CONF_FILE = False
    DISABLE_ACTIONS_WHEN_HIDDEN = False

    # --- SpyderDockablePlugin API
    # ------------------------------------------------------------------------
    def get_name(self):
        return _('Plots')

    def get_description(self):
        return _('Display, explore and save console generated plots.')

    def get_icon(self):
        return self.create_icon('hist')

    def register(self):
        # Plugins
        ipyconsole = self.get_plugin(Plugins.IPythonConsole)

        # Signals
        ipyconsole.sig_shellwidget_changed.connect(self.set_shellwidget)
        ipyconsole.sig_shellwidget_created.connect(
            self.add_shellwidget)
        ipyconsole.sig_shellwidget_deleted.connect(
            self.remove_shellwidget)
```

Name	Type	Size	Value
a	foo	1	foo object of __main__ module
filename	str	53	/Users/Documents/spyder/spyder/tests/test_dont_use.py
i	bool	1	True
my_set	set	3	{1, 2, 3}
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 70.2)
thisdict	dict	3	{'brand':'Ford', 'model':'Mustang', 'year':1964}
tinylist	list	2	[123, 'efgh']
x	Array of int64	(2,)	[1 2]
y	timedelta	1	2 days, 0:00:00

LSP Python: ready conda: spyder-dev (Python 3.7.10) master Line 1, Col 1 UTF-8 LF RW Mem 57%

Input-Output

Output:

- `np.set_printoptions`
  - precision: Number of digits of precision for floating point output (default 8). E.g. np.array([2.1,3.2]) / np.array([3.7,5.5])
  - Beware of Numpy rounding: E.g. `np.set_printoptions(precision=0); np.array([1.5,2.5])`
  - threshold: Total number of array elements which trigger summarization rather than full repr (default 1000). E.g. `np.r_[1:1000]` vs `[1:2000]`.
  - linewidth: The number of characters per line for the purpose of inserting line breaks (default 75).

File Types

There are only two types of files that we need to know for Python:

- py-files: Python program text file
- pyc-files: Automatically generated by the Python interpreter

In Windows, one needs to turn off ‘hide file extension’ in Explorer to see the file extension ‘.py’ and ‘.pyc’.

The file types related to Input-Output can be roughly categorised to ‘text’ and ‘binary’.

- Read a text file: open(”file.txt”, ”r”)
- Read a binary file: open(”file.bin”, ”rb”)

General Commands

General commands that we should remember:

- Command history: Using arrows to navigate (they are automatically stored in .python_history in GNU/Linux)
- Getting help: help, np.info, np.lookfor
- Workspace information:
  - dir(): list variables in the workspace
  - type(): check the type of the variable
  - del: delete the variables from the workspace
- Directory information:
  - os.getcwd(): show the current workding directory
  - os.lisdir(): list contents of the current directory
  - os.chdir("D:/mydir/") for windows

General Commands (cont)

Getting the information about Python and the computer:

- Get Python version: sys.version
- Computer type & os: import platform; platform.machine(); platform.platform(); platform.system(); platform.release(); platform.version()

Existing:

- Python shell: quit() or Ctrl-Z in Windows; Ctrl-D in GNU/Linux
- Spyder / Jupyter: File Menu > Quit

General Commands (cont)

Time related commands:

- Time in seconds since the epoch (1/1/1970): time.time()
  - import time
  - tstart = time.time()
  - do something in Python
  - tdiff = time.time() - tstart
- Human readable time: time.gmtime()
- Today's date: datetime.date.today()

Outline

Numbers: Double & Float

- Scientific notation: $\pm$ X.DDDDD $\times 10^{y}$
- Into computer: $\pm$ 1.MMMMM $\times 2^{E}$
- Syntax: -.1, +3.14, 1_0000.0, 1e3, 2.999e8
- +, -, *, /, **, abs()
- $>$, $>=$, $==$, $!=$, $<=$, $<$
- Special constants: math.e, math.pi, math.tau, math.inf, math.nan

Python only supports double (64-bit floating point numbers)

Numbers: Double & Float (cont)

Numpy supports both double (64-bit floating point numbers) and floats (32-bit floating point numbers)

- Numpy website: https://numpy.org/
- import numpy as np
- Double example:
  `np.array(3, dtype='double').dtype`
- (Single) Float example:
  `np.array(3, dtype='single').dtype`
- Load ‘real (floating) number’ library: import math
- Special ‘numbers’: Nan (np.nan), ∞ (np.inf).
- Information about the floating point number capability: np.finfo('double'), np.finfo('single')

Numbers: Integers

- Syntax: +1023, -1_0000 (underscore allow)
- Range: ..., -3, -2, -1, 0, 1, 2, 3, ... (can be arbitralily large)
- 0b111, 0o123, 0xABC, int(”123”, 7)
- +, -, *, /, //, **, %, abs()
- Bit arithmetic: &, |, ˆ, <<, >>
- $>, >=, ==, != , <=, <$

Python’s integer is ‘big’ integers in other programming environment.

Numbers: Integers (cont)

Numpy supports 8, 16, 32 and 64-bit integers usually used in other programming environment:

- `np.array(1, dtype='int8').dtype`
- `np.array(2, dtype='int16').dtype`
- `np.array(3, dtype='int32').dtype`
- `np.array(4, dtype='int64').dtype`
- 1 byte = 8 bit
- `b.nbytes` gives the numbers of bytes used in memory

Numbers: Complex

- Complex (Floating) Numbers: 1+2j, 1.1e2 - 3.503j, complex(3,5)
- Complex Double: np.array(1+2j).dtype
- Complex Single: np.array(1+2j,dtype='complex64').dtype
- +, -, *, /, **, abs(), .conjugate(), .imag, .real
- Load 'complex (floating) number' library: import cmath

Logic: Booleans

- Syntax: True, False
- and, or, not, &, |
- Applications: relations (E.g. $a < b$), if statement, while loop

Outline

Arithmetic operations

- The following arithmetic operations are defined for all numbers (floats, integers, complex) a and b (of the same type):

  $$a + b, \quad a - b, \quad a * b, \quad a * *b$$

  Note that $0^0 = 1$ by definition.
- a/b is 'well-defined' defined for $b! = 0$.
- // and % are intended for integers but also work for floats

Arithmetic operations (cont)

After we have learn the numbers and arithmetic operators, we can express rational expressions in Python.

Example: Express the expression from SPM in Python:

$$\frac{27^{-\frac{5}{3}}}{9^{-4} \times (4^{\frac{1}{3}})^6}$$

Answer: (27**(-5/3)) / (9**-4 * (4**(1/3))**6)
Using fractions.Fraction(...), we get $\frac{27}{16}$, same as the one found in SPM but fractions.Fraction(30.1) does not give $\frac{301}{10}$ due to number rounding.

Outline

Functions

In maths, a function is a mapping (or association), which takes value(s) and returns a single value. For example, sine is a function

$$\sin(\frac{\pi}{2}) = 1$$

What kind of mapping is **not** a function? The ‘square root’ is not a function when we do not restrict it to nonnegative return value:

$$\sqrt{9} = \pm 3.$$

Functions (cont)

A **function** in Python (usually) takes in **parameters** and returns a **value**.

- Defining our own function:

```
def f(x,y,z):
    return a value derive from x,y,z
```

Example:

```
def myaverage(x,y,z):
    return (x+y+z)/3.0
```

- Standard functions provided by Python are grouped into “modules”/“libraries”: E.g.

```
from math import *       # everything
from cmath import sqrt # specific
```

Functions (cont)

Elementary functions in `math` module:

- Radian $\leftrightarrow$ Degree: `radians(degree)`, `degrees(radian)`
- Square root function $\sqrt{x}$ (only returns a non-negative value): `sqrt(x)`
- Power function x^n: `pow(x,n)`
- Exponential function e^x: `exp(x)`
- Logarithmic functions: `log(x)`, `log10(x)`
- Trigonometric & Hyperbolic functions: `sin(x)`, `cos(x)`, `tan(x)`, `sinh(x)`, `cosh(x)`, `tanh(x)`
- Inverse Trigonometric functions: `asin(x)`, `acos(x)`, `atan(x)`, `atan2(y,x)`

Examples of elementary functions

Example: Write down the Python command to calculate $\frac{\sin 10^\circ \sin 50^\circ}{\sin 30^\circ}$.

Answer 1: `sin(radians(10))*sin(radians(50))/sin(radians(30))`

Answer 2: `sin(10*pi/180)*sin(50*pi/180)/sin(30*pi/180)`

Exercise: Write down the Python command to calculate $\tan 6^\circ \tan 42^\circ \tan 66^\circ \tan 78^\circ$.

Exercise: Write down Python command to calculate $\cos 42^\circ \cos 18^\circ - \sin 42^\circ \sin 18^\circ$. Are you able to find the exact value using trigonometric equality $\cos A \cos B - \sin A \sin B = \cos(A + B)$?

Functions (cont)

Note: Elementary functions refer those functions which can be ‘differentiated’ in Calculus.

Other (non-elementary) functions in the math module:

- Distance: `dist(x,y)`, `hypot(...)`
- Truncating functions: `ceil(x)`, `floor(x)`, `trunc(x)`
- “Check” functions: `isclose(a,b)`, `isfinite(x)`, `isinf(x)`, `isnan(x)`
- Combinatorics: `comb(n,k)`, `perm(n,k)`, `factorial(n)`, `gcd(x,y)`

Final Exam Sept 2012, Q2(b)

Write a Python function that converts a temperature in Fahrenheit to a temperature in Celcius according to the following equation

$$[^\circ C] = \frac{5([^\circ F] - 32)}{9}.$$

Show an example of using the function `f2c` in the Python command window. (8 marks)

Answer:

```
def f2c(fahrenheit):
    return 5.0*(fahrenheit-32.0)/9.0
```

Outline

Formatting numbers

We usually do not come across a very large or very small numbers but scientists and engineers do come across such numbers an it is essential have special 'number' formats.

Constant	Decimal	Scientific Notation
Speed of Light in Vaccum	299792458.0	2.997925e+08
Plank constant	0.0000000000000000000000000000000006 62607015	6.626070e-34

Formatting numbers in MATLAB

- format short: fixed with 5 digits
- format long: fixed with 15 digits for double, 7 digits for single.
- format short e: Engineering format with 5 digits, e.g. 3.14159e+001
- format long e: Similar to format long but in engineering format
- format short g: Best of fixed or engineering format with 5 digits
- format long g: Best of fixed or engineering format with 15 or 7 digits
- format +: Turn on the positive sign
- format bank: Fixed dollars and cents. E.g. 31.42
- format hex: Hexadecimal representation of a binary double-precision number. E.g. 403f6a7a2955385e

Formatting numbers in Python

Different from MATLAB but very close to C/C++:

- Fixed digits: `"{num:.5f}".format(num=10*math.pi)`
- Engineering format: `"{num:.5e}".format(num=10*math.pi)`
- 'Shortest' format: `"{num:.5g}".format(num=10*math.pi)`
- Positive sign: `"{num:+.5g}".format(num=10*math.pi)`
- Fixed dollars and cents: `"{num:.2f}".format(num=10*math.pi)`
- Hexadecimal representation: `hex(struct.unpack("Q", struct.pack("d", 10*math.pi))[0])`

  Here Q stands for 64-bit integer, d stands for double. Ref: help(struct)

Formatting numbers in Numpy

For formatting the numbers in Python's Numpy array, we need to do something more complicated using `np.set_printoptions`. `'float': lambda x: "{num:.5e}".format(num=x) })`

Example: $\begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/4 & 1/5 & 1/6 \end{bmatrix}$

```
print(np.array([[1,1/2,1/3],[1/4,1/5,1/6]]))
def format_func(x): return "{num:.5e}".format(num=x)
np.set_printoptions(formatter={'float': format_func})
print(np.array([[1,1/2,1/3],[1/4,1/5,1/6]]))
```

Formatting numbers in Numpy (cont)

Example (cont):

Before we set the output number format, we get

```
[[1.          0.5         0.33333333]
 [0.25        0.2         0.16666667]]
```

After we set the output number format, we get

```
[[1.00000e+00 5.00000e-01 3.33333e-01]
 [2.50000e-01 2.00000e-01 1.66667e-01]]
```

Outline

The reality about numbers

Numbers can be stored into a text file or a binary file.

```
# Write 1/3 to text and binary.  Try using Notepad to
# open the two files to see the difference!
print(1/3, file=open("test.txt","w"))
import struct
open("test.bin","wb").write(struct.pack('d', 1/3))
```

Now read the two files for the number 1/3:

```
float(open("test.txt").readline())
struct.unpack('d', open("test.bin", "rb").read())[0]
```

A character refer to capital letters, small letters, digits, punctuation marks, etc.

So ‘test.txt’ is formed from characters while ‘text.bin’ is formed from computer low level ‘bytes’!

The Reality of Characters

Modern day 1-byte = 8-bit. How to represent a character in 8-bit?

EBCDIC (Extended Binary Coded Decimal Interchange) code used mainly by IBM mainframes in the 1950s $\sim$ 1960s.

US-ASCII (American Standard Code for Information Interchange, see https://en.wikipedia.org/wiki/ASCII) is used in AT&T's telegraph system in 1961 and standardised in 1969.

binary	decimal	char	binary	decimal	char
00101010_2	42	*	00101011_2	43	+
00110000_2	48	0	01000001_2	65	A
01100001_2	97	a	00101110_2	46	.

The Reality of Characters (cont)

To verify that a text file is just a special case of a binary file, we can try out guess: 0, 1, 2, 3 is 48, 49, 50, 51 in ASCII and the dot '.' is 46

We try to write them as a binary file as see if this is the case:

```
open("ascii.bin", "wb").write(
  struct.pack('BBBBB', 48,46,49,50,51))
```

Note that 'B' is unsigned byte.

Open ascii.bin using notepad and see what we have (in Windows, you may need to close Python to unlock the file ascii.bin for opening)

The Reality of Characters (cont)

Modern day characters are more complex because an extension of ASCII call https://en.wikipedia.org/wiki/UTF-8 (the modern encoding of https://en.wikipedia.org/wiki/Unicode characters) is used and Python's documentations use UTF-8 encoded characters.

Characters, strings, texts

A string is just a sequence of characters.

- Syntax: ”a string”, ’a string’ (both sequences are the same and has 8 characters)
- Strings can be used in statistics for categorical data (e.g. blood types: A, B, O, AB)
- Strings are used in report generation and widely used in various social network platforms such as reddit, Facebook, etc.
- String operations: +, len(), ord(), chr(), str()

A sentence can be encoded in to a string. E.g. ”Python is quite popular in Scientific Computing.”

A text is just a long string string with many sentences or a sequence of strings.

Outline

Variables

A variable is created the moment we **assign** a value to it.

A variable is something starts with small or capital letters and is usually an English word followed by a number or underscore and then other English words.

E.g. variables can be:
aVariable, Grade, A, some_variable_with_long_name, a1, a2, a3, ...

Variables **cannot** be written as follows:
1stname, 2nd-name, *star*, +plus+, ...

Variables (cont)

If we don't use variables, many calculations will be very **troublesome** because we need to "copy and paste" answers.
Consider the scenario: find the (population) standard deviation of 68, 82, 77

- The mean using Python: (68 + 82 + 77)/3
- Python gives us: 75.66666666666667
- Usually, we will do rounding in calculation and so we may calculate the population variance: ((68-75.6667)**2 + (82-75.6667)**2 + (77-75.6667)**2)/3
- Python gives us: 33.55555555666667
- The standard deviation is 33.5556**0.5
- Python gives us 5.792719568561903

Variables (cont)

For the same scenario: find the (population) standard deviation of 68, 82, 77, let us use variables to simplify the process.

- a1 = 68
- a2 = 82
- a3 = 77
- mean = (a1+a2+a3)/3
- stddev = (((a1-mean)**2 + (a2-mean)**2 + (a3-mean)**2)/3)**0.5
- print(stddev) or print(”sd:.6f”.format(sd=stddev))

The use of variables make the calculation much shorter avoiding copy and paste and also clearer.

Variables (cont)

One class of problems that is asked a lot in SPM is the quadratic equations. For example, find the solution to

$$x + 2 = \frac{12 - x}{2x}$$

Assuming $x \neq 0$ (otherwise, we have division by zero). This can be transform to

$$2x^2 + 5x - 12 = 0.$$

Using factorisation in Python is “impossible” because Python does not do algebra. So we will use the general solution to $ax^2 + bx + c = 0$ to find the solutions:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{or} \quad \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Variables (cont)

Based on what we have learned, we can write the expression using Python:

```
a = 2
b = 5
c = -12    # beware of the negative sign
x1 = (-b + (b**2-4*a*c)**0.5)/2/a
x2 = (-b - (b**2-4*a*c)**0.5)/2/a
print("x1={x1:.6f}, x2={x2:.6f}".format(x1=x1, x2=x2))
```

If put the wrong value for c, i.e. c=12 instead of −12, we will get complex numbers as answers! Try it out and try to replace **0.5 with sqrt from math and cmath libraries to see the difference.