EP417 – Computational Optics – Python notes.

The software for the Computational Optics module has been translated to Python (from C++ and Matlab). The main reason for this is that Python is freely available and you can install it on your home or portable PC (the Matlab package which was used in the past for graphical routines requires a licence). You will not be required to write your own python scripts for this module, but you will need to make minor modifications to the scripts that will be provided with the module notes. This document provides instructions on running the scripts and making the required modifications. It is assumed that you are using the Anaconda/Spyder distribution of python, but the scripts should run successfully with other distributions such as Enthought/Canopy. The software should also be platform independent, so the scripts should run under Windows, Linux, or Mac OS. Computer laboratory sessions will be scheduled as part of the EP417 module during which the programming environments and python scripts will be demonstrated.

(If you encounter any problems with Anaconda/Spyder, or incompatibility issues with Enthought/Canopy, please let me know).

Spyder – required settings:

1) Interactive graphics/ floating graphics windows: the default setting in Spyder is for graphs to be drawn within the Ipython console on the lower right. Such graphs cannot be scaled or moved (they are non-interactive). To obtain graphs in a floating window which can be panned/zoomed, do the following within the Spyder environment:

Tools – Preferences – Ipython Console – Graphics:
- set ‘backend’ to ‘automatic’ and click ‘apply’ (do not select ‘inline’ - this is the default setting).

2) Working directory: it is best to keep your python scripts in a separate ‘working directory’; you might choose to have different working directories for different tasks, e.g. you might create

c:\ep417pythonscripts

and use this folder on your c: disk drive to store all of the python files associated with the ep417 module (these files would also include possible input/output data files).

You could simply use the ‘folder’ icon in the upper right corner of the Spyder environment to connect to this directory, or you could use:

Tools – Preferences – current working directory

to set the name of the directory in which you are going to work (Spyder assumes that this directory already exists). Select the ‘file explorer’ tab to show the list of available python script files in this working directory (see the next item in this list). Click on a file name to load its contents into the editor window.

3) Variable explorer: the upper right window in Spyder has multiple modes; select a mode using one of the tabs (variable explorer, file explorer, help). The ‘variable explorer’ displays all the variables used in a script, and lists their properties and contents. To correctly display ‘vector’ variables (used in ep417 in optical raytracing), use the following setting:

- click on the small ‘cog’ icon above the top right corner of the upper right window, and enable ‘tick’ marks in front of ‘exclude private references’ and ‘show arrays min/max’. Do not have tick marks on any other option.

4) To activate changes: always restart Spyder after making any changes to the environment.

Running the EP417 python scripts:

Raytracing – vector programming: ensure that Vec2D.py is in the same working directory as the python scripts which import it.

Variable names:

It is generally a good idea to use naming conventions which assist you in recognizing variable types. This is particularly the case in scripts involving vector types, as it is important that you do not assign a vector variable to a floating point variable (or any other non-vector type). Where possible I have used the following naming convention:

- upper case first letter => Vector
- lower case ‘z’ as first letter => array
- lower case letter other than ‘z’ => scalar (integer or real).

Some warnings about Python – pitfalls:

Python is an ‘object’ oriented language – the variable type (real, integer, character, etc.) is determined by the nature of the object:

```
test = 1234          ……  ‘test’ is an integer variable
test = 1.234         ……. ‘test’ is now a real variable
test = ‘abcd’        ……. ‘test’ is now a character variable
```

In ‘type’ orientated languages such as C++ or Fortran, the variable type is set by a declaration statement:

```
int test1;
float test2;

test1 = 1234;
test2 = 1.234;
```

C++ will issue an error if you try:
test1 = 1.234;   (you are trying to assign a real number to an integer-type variable)

Python will NOT issue such errors (or even a warning!) as it does not associate a particular type with a variable name. You need to be aware of this as it can lead to problems. The ‘Variable Explorer’ window in Spyder is useful for monitoring variable types – see point 3 above.

Mfc 17/9/’19