# SET 15: COMPUTING FOR PHYSICAL SCIENCE

## Hand in date: 10th March 2021
## Submit electronically through Moodle; no hard copy submission required

*This coursework tests the use of functions. Note that it is not really necessary to write functions for the simple tasks required here – it is just to illustrate their use.*
In Python, we normally define our functions in our main Python script (or Jupyter Notebook). It is customary to have them at the top, after any `import` statements, but they can actually be defined at any point before they are first called. For this coursework, it will be easier (for the markers) if you define each function at the relevant part of your code, just before the function is first used. You only need to keep the final version of your program file with all modifications made and with all function definitions included in that same file. Alternatively, you could also have your functions collected in a separate file (a module), called say functionsfile.py, but then in your Jupyter Notebook (or in your main Python script if you are working in Spyder) you would need to import any function used by adding `from <functionsfile> import <function>`. In that case you would, of course, need to upload both your Jupyter Notebook (or main script) and your functions file.

1. In Physics, we often want to convert between different types of coordinates. Here, we will look at conversion from plane polar coordinates to Cartesians. In this question you are invited to write your own Python function to do this.

   (a) Define a function called `polar2cart` to convert plane-polar coordinates $\{r, \theta\}$ into Cartesian coordinates $x = r\cos\theta$ and $y = r\sin\theta$ (where $\theta$ is in Radians). Your function should work also when $r$ and $\theta$ are vectors containing more than one value (and of the same length as each other). Test your function works by defining vectors of several values (of your choice) for $r$ and $\theta$ and then using your function to convert the polar values to Cartesians. {**7**}

   (b) Include some text in your function so that if the user runs `help(polar2cart)` (in a Jupyter Notebook cell, in the IPython console, or in a script file) they will be told what the function does and how to use it. {**2**}

2. According to a model for the pattern of florets in the head of a sunflower proposed by H. Vogel in 1979, florets are found at positions $(r_f, \theta_f)$ in plane polar coordinates, given by[1]

   $$r_f = c\sqrt{n} \quad \text{and} \quad \theta_f = (2\pi/\phi^2)n,$$

   

   where $n$ is an integer, $c$ is a constant and $\phi = (1+\sqrt{5})/2$ (which is known as the Golden Mean). The positions all lie on a Fermat spiral, which is given by

   $$r = c\sqrt{\phi^2\theta/(2\pi)}.$$

   Other plants have florets at different positions.
   Download the script 'Fermatspiral.py' from Moodle, which plots both the Fermat Spiral and the first 20 floret positions when $c = 1$. If you are working in Jupyter Notebook, copy the code from this script to a new Jupyter Notebook. Different floret patterns (that don't line on a Fermat spiral) can be generated by changing the parameter `ndiff` (`ndiff` $= 0$ corresponds to floret positions that line on a Fermat spiral).

[1] See the end of `http://en.wikipedia.org/wiki/Fibonacci_number`. This is also found to be the ideal layout for mirrors in a concentrating solar power plant (where sunlight is focussed on a central tower).

(a) Modify the code from 'Fermatspiral.py' to use your function `polar2cart` from question 1 (in two places) rather than using the formulae to convert to Cartesians that are currently typed directly into the program. {**2**}

(b) Modify your program so that the part that calculates the Cartesian coordinates of the floret positions (but not the part that plots the florets) is done in a function. Input parameters for the function should be the maximum number of florets and `ndiff`, and output parameters should be vectors of $x$ and $y$ values. {**5**}

(c) Add a new section to your program that uses your function in (b) to calculate the floret positions (not the continuous line representing the Fermat spiral, which doesn't need to be plotted for this part) for values of `ndiff` from -0.25 to 0.25 in steps of 0.1. Plot the results in a grid with two rows and three columns (using, for example, Matplotlib's `add_subplot` to add an array of axes objects in your figure). Add a title to each subplot giving the value of `ndiff` and some accompanying text, like 'ndiff ='. Given that your figure will contain six subplots, you will need to specify a large size for your figure window (e.g. 12 inches $\times$ 7 inches) and you may need to finetune the layout with Matplotlib's `subplots_adjust` to add space between the subplots. For full marks, your program should use a loop and no `if` statements. {**9**}

*Upload to Moodle*

- *Your final Jupyter Notebook (or script file if you are working in Spyder) including all function definitions. If you have chosen to collect your functions in a separate file then you must upload both your Jupyter Notebook/script file and your functions file.*
- *If you have uploaded a Python script (rather than a Jupyter Notebook file) also upload a png or jpg version of your final figure from 2(c).*