EMSC8014 Assessment

October 5, 2018

EMSC8014 Assessed exercise

Due: 5pm Friday 26th October

Once your solution is complete, you should:

1. Ensure it has been saved (select `File > Save and checkpoint` from the Jupyter menu)
2. Select `File > Download as... > Notebook (.ipynb)` from the Jupyter menu
3. Save the file on your computer
4. Submit this file via Wattle.

N.B. Since it is possible to accidentally delete cells from this Jupyter notebook, you are also provided with a `.pdf` version. It is your responsibility to ensure that your solution is complete before submitting it.

Important information

- This assessed exercise is made up of a number of tasks. You should attempt all of them. The approximate weight to be given to each task during grading is shown as a percentage for each.
- You should write your solutions within this Jupyter notebook. You may add additional Markdown (text) and code cells as appropriate. Your solution should be entirely self-contained (i.e., you should not create additional notebooks or other files). Everything appearing in the submitted notebook should be intended for assessment; do not include rough working.
- Your solution should run without error on the RSES Jupyter server, when all cells are executed in the order in which they appear (e.g. by selecting `Kernel > Restart & Run All` from the Jupyter menu).
- You may choose to make use of the following modules:
  - `numpy`
  - `matplotlib`
  - `pandas`
  - `cartopy`
  - `datetime`
  - `math`
  - `re`

No other modules may be used unless you have obtained permission from the course convenor (Dr Andrew Valentine). Note that you may not need to use all of the modules in the above list.

- Where a task asks you to "write a function", you are expected to produce one function that can be called to generate all required output. However, you are free to organise this as you wish, and you may choose to write additional functions which get called within your main function.
- You should include an example of how your function is intended to be used.

Marking policy

- There is never a single 'correct' way to solve a programming problem. You will gain marks for any solution that achieves the desired goal(s). Higher marks will be awarded for solutions that are:
  - Clear: code is easy for someone else to understand (and perhaps modify)
  - Documented: code contains docstrings and comments as appropriate
  - Concise: code is not excessively long or complicated for the task at hand
  - General: code can easily be adapted to solve related problems
  - Robust: code appropriately handles any errors that might reasonably be expected to occur, especially those related to user input.
- Some tasks require you to produce figures. These should be of a standard suitable for inclusion in a research paper. Marks will be awarded for:
  - Correctness of information presented; remember to include titles, axis labels and legends where appropriate.
  - Clarity of layout: is your figure easy to interpret and visually-pleasing?
  - Attention to detail: are there any inconsistencies in the way you are presenting information?
- If you are unable to complete a task, you should:
  1. Submit your best attempt, and
  2. Include a note (comment or Markdown) explaining which piece(s) of your solution you believe to be incorrect or incomplete, and why.
  3. If you believe you understand the algorithm (sequence of operations) to perform a task, but have not been able to implement this in Python, you should explain it in full.

  Partial credit may be given for such work.
- You are allowed to make use of internet resources, class notes, and to talk to other people about how to solve these tasks. However, **everything that appears in your solution must be your own, individual work**. You should not collaborate with someone else to produce a solution, or copy their work. In particular, you **must** understand, and be able to fully explain, everything that appears in your solution. **If you cannot, you will not receive any marks for that task.**

- Drop-in sessions (optional) will be held from 12-2 pm on:

  - Wednesday 10 October
  - Monday 15 October
  - Friday 19 October
  - Wednesday 24 October

  These will all take place in the TerraView Room at RSES (Jaeger 2, top floor). One of the course tutors will be available to answer any questions you have.

- Oral exams will be held on Tuesday 30th and Wednesday 31st October, in Dr Valentine's office at RSES (Jaeger 2, Room 147a). You will be informed of the time at which you should attend. During the oral exam, you will be asked to explain your solutions to this assessed exercise, and discuss why you chose to implement them in the way that you did. No special preparation is required, but you may wish to re-read your solutions prior to the examination. **Failure to attend the oral examination (without good cause) will result in you receiving no marks for this assessment.**

For each task, a **failing mark** will be awarded where:

- The submitted solution does not accomplish the goal(s) described for the task.
- The student is unable to explain how their solution works, and why they chose to tackle the problem in a certain way.
- Output (figures/text files) contain significant errors. Figures are well below the standard expected for a good-quality journal, through omission of important information or poor visual standard.
- The examiners are not satisfied that the solution is the student's own, independent work.

A **passing mark** will be awarded where:

- The solution accomplishes the principal goal(s) described for the task, perhaps with some minor errors or omissions.
- The code is functional, but unsophisticated; it may contain significant redundancy or unnecessary computation.
- It is difficult for someone looking at the code for the first time to understand the program logic; variable names are poorly-chosen, and the program is not well-organised.
- There is little or no documentation (e.g. comments and docstrings) provided.
- Little or no attempt is made to handle common sources of error (e.g. bad user input).
- Output (figures and/or text files) are accurate but lacking polish.
- The student has demonstrated familiarity with only a limited subset of the Python capabilities covered in the course.
- The student is able to explain, in basic terms, how the code solves the assigned task.

A **good mark** will be awarded where:

- The solution accomplishes goal(s) described for the task without any errors or omissions.
- The code is well-structured and appropriate to the task in hand.
- Code is reasonably understandable, with meaningful variable names and a clear organisation.
- Basic documentation is provided through comments and docstrings.
- Common sources of error (e.g. bad user input) are handled in a sensible fashion.
- Outputs are well laid out and (for figures) visually pleasing.
- The student displays knowledge of a broad range of Python capabilities, as apropriate to the task.
- The student is able to explain their code well, perhaps showing some awareness of alternative ways in which the problem could have been tackled.

An **excellent mark** will be awarded where:

- The solution fully accomplishes all aspects of the task without error.
- Code is clear, concise and readily-understood by anyone who reads it.
- Code is fully-documented with comprehensive docstrings and comments that help the reader understand how the code works.
- The code is robust, with all common sources of error handled elegantly and effectively.
- Figures are of a high standard, with care taken to maximise clarity and information content.
- The student displays full knowledge of the Python programming language (to the extent covered in the course), and consistently chooses the most appropriate tools at their disposal.
- The student has a comprehensive understanding of their code, and is able to discuss the advantages and disadvantages of various solution strategies, justifying why they chose to adopt one and not the others.
- The solution may go beyond what is strictly required to answer the assigned task, or employ aspects of Python that were not explored during the course.

Good luck!

---

1. The Fibonacci Sequence (10%)

The Fibonacci seqence is a famous sequence of numbers. Each term in the sequence is obtained by adding together the previous two terms. The first two terms are defined to be (1, 1, ...).

Write a function that will return a list containing the first n terms of a Fibonacci sequence.

2. The ideal gas law (10%)

The ideal gas law relates pressure (P), volume (V) and temperature (T) for an 'ideal' gas. It states that

$$PV = nRT$$

where n is the number of 'moles' of gas present, and R is a constant, $R = 8.3145\,\mathrm{L\,kPa\,mol^{-1}\,K^{-1}}$. Write a function that allows the user to specify **any three** of the quantities P, V, n and T, and obtain the fourth.

3. The substitution cipher (20%)

In class, we encountered the Caesar cipher for encoding messages. Another kind of cipher is a 'substitution cipher'. To encode a message, we first choose a 'keyword' - for example, `CHEESE`. If any letter appears in the keyword more than once, we delete the subsequent occurrences (giving `CHES`).

Next, we write out a full alphabet, and a second version where the characters that appear in the keyword are shifted to the front:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z    <== Plaintext
C H E S A B D F G I J K L M N O P Q R T U V W X Y Z    <== Cipher
\_____/    | |   |                   |
Keyword    Characters in keyword moved to front
```

To encode a message, we look up each character in the 'plaintext' row and replace it with the corresponding character from the 'cipher' row. So, using our keyword `CHEESE`, the message `I AM WORKING ON MY ASSIGNMENT` becomes `G CL WNQJGMD NM LY CRRGDMLAMT`. To decode a message, we look up characters in the 'cipher' row, and replace them with the corresponding character from the 'plaintext' row. Any characters not in the alphabet (e.g. punctuation characters and spaces) should be unchanged by the encoding/decoding operation.

Write a function which can encode and decode messages using any keyword. To help you check that it is working properly, here is a message that has been encoded using the keyword `PYTHON`: `VOFF HJIO, XJS BPUO QSTTOQQNSFFX HOTJHOH P GOQQPAO!`

4. Global seismicity (20%)

This question has two parts; you should attempt both.

The file `earthquakes.dat` contains information about earthquakes that have occurred in the last 10 years, drawn from the Global CMT catalogue. The file contains columns corresponding to the earthquake location, magnitude and date/time, as follows:

Latitude	Longitude	Depth	Magnitude	Year	Month	Day	Hour	Mins	Secs
36.87	69.95	54.3	5.1	2000	01	01	05	24	35.3
-60.72	153.67	10.0	6.0	2000	01	01	05	58	19.8
...	...	...	...	...	...	...	...	...	...

1. Make a global map plotting the location of every earthquake in the file.

2. Write a function that allows the user to specify ranges for any or all of: latitude, longitude and depth. The function should then

    1. produce a map of only those events that fall within the specified range(s), with information about earthquake depth shown using colour, AND
    2. write out a new file (in the same format as `earthquakes.dat`) that contains only the selected events.

5. Infrared Spectroscopy (20%)

The file `11octrll32bs1_2016-10-11T12-48-25.asp` is produced by an infrared spectrometer. Look at the file contents. It is formatted as follows:

- Line 1: Number of data points in file
- Line 2: Wavenumber (cm^{-1}) associated with first measurement in file
- Line 3: Wavenumber associated with last measurement in file
- Line 4-6: Miscellaneous information (not required for this exercise)
- Line 7-end: Reflectance measurements (%).

You can assume that the measurements are evenly-spaced in wavenumber.
The filename encodes some additional information:

```
11octrll32bs1_2016-10-11T12-48-25.asp
\___/\____/\| \________/ \______/
  A     B   C      D        E

A - Date of experiment (11 October)
B - Location of experiment ('rll32b')
C - Sample number (Sample 1) -- starting with 's'
D - Date of experiment (again; 11 October 2016)
E - Time of experiment (12:48:25)
```

Write a function that takes the filename as argument, and produces a high-quality figure, showing reflectance (y-axis) plotted against wavenumber (x-axis). The following information should also be displayed on the figure: - The date and time at which the experiment was run - The location, panel and sample numbers - The filename of the datafile.

Your function should work for *any* file of this format. To help you test this, you have been provided with a second file from the same instrument - `12octrll61s1_2016-10-12T10-36-43.asp`.

6. Tic-Tac-Toe (20%)

Tic-tac-toe (or 'noughts and crosses') is a game for two players. Players take it in turns to place their mark ('X' for player 1, 'O' for player 2) in an empty space on a 3 x 3 grid. A player getting 3 marks in a line (row, column or diagonal) wins.

Write code to allow you to play Tic-Tac-Toe against the computer. The computer should play randomly, placing its mark in any empty cell.