PHYS 210 - Projects - Marking rubric

Item	Lowest grade	Average grade	Highest grade
Code execution	Code does not execute, for whatever reason(s) 0	Runs without execution bugs or errors, without crashing but **`pep8`** returns some warnings/errors 15	Runs without execution bugs or errors, without crashing and **`pep8`** does not return warnings/errors 20
Code returns correct results	Code runs but all results are bogus 0	Only some results are correct, or some are incorrectly presented or project objectives not completely met 1-14	Code runs and all results are correct, all questions from projects are answered 15
Comments in the code	None to very few comments 0-4	Some comments 5-9	Key parts of the code contain comment(s) explaining what the part does. The code should start with a short description of what it does and instructions how to use it (if called with arguments, specific syntax, etc…) 10
Definition (incl. comments) of variables, functions and parameters.	No variable, function or parameters is defined 0	Some variables, functions, parameters are defined 1-9	All variables, functions and parameters are clearly defined. Explicit variable names 10

Plots: legend box, labels and title	None 0	Some items 1-3	All items present 4
Plots: units	None 0	Some 1-3	All present 4
Plots: legibility	Very poor 0-2	Average 3-6	Fonts are readable. Lines are thick enough to be readable. 7-9
Plots: colours and line styles. (plots should be legible if printed in black and white)	All lines have the same style. Colours have poor constrast relative to background 0	1-2	Good choice of colours and line styles that help the legibility of the graphs. 3
Code compactness (does not apply to comment, to some extend).	The code is too long, e.g. the code includes useless functions, use too many redundant variables, does not use compact expressions to define variable/arrays, comments are WAY too heavy. 0-3	4-9	The code achieves its goal in less lines than average. Python language remains clear an compact 10-15
Code optimization/speed	The code wastes time doing useless or time consuming calculations. 0 - 1	Some effort in optimization (e.g. using arrays instead of loops when possible) 2 -3	The result is obtained using less computing time than average. 4 - 5

Use of functionalities not covered in class	None 0	Some effort 2-3	The code uses several new routines or functionalities, or uses one or two challenging functionalities 4-5
Bonus points (max: 5 marks): • the project is developed beyond expectations.			

Code execution: A code should execute without crashing, it is not acceptable to deliver a code that does not even run! you get a 0 or full mark 15 for checking if your code runs or not. It does not mean the code should return exact results, it means it should execute when used "normally" (i.e. not pushing it to generate an error on purpose), it should have no runtime or syntax error. In addition, your code must be written in proper Python style, that means running **`pep8`** on it should return no warning/error message, it is again a pass/fail test, you get either 0 or the full 5. (Note: **`pep8`** is installed on the PHAS server, see https://pypi.python.org/pypi/pep8 for usage).

Code returns correct results: This rubric asses whether or not you have met the objectives of the project in terms of actual results.

Comments in the code: A good code contains on average one line of comments per ~4 coding lines (it is not a rule, it is just a basic observation based on experience). Here are some tips on commenting:

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-189-a-gentle-introduction-to-programming-using-python-january-iap-2011/lectures/MIT6_189IAP11_comment.pdf

http://www.cs.utah.edu/~germain/PPS/Topics/commenting.html

Also every good code starts with a description of what it does and how to use it, pretty much like the package routines (e.g. numpy or scipy) described online. Your code should start with this too.

Definition of variables, functions and parameters: use clear names, e.g. **`aaa=6.67e-11`** is NOT an acceptable definition of the constant of gravity, instead something like this is good (clear name and comment):

```
G_grav=6.67e-11   #  Constant of gravity in SI units
```

Define/definition includes you have to write a short comment what that variable/function/parameter is (e.g. above). You don't have to define every single variable in your code, e.g. the loops and iterators, usually given by a single letter, **`i`** or **`n`**, don't have to be defined/commented.

Plots: should contain as much information as it is deemed useful, clear, not too crowded (make several plots if there is too much information to show).

Compactness: efficient and clear programming also comes with the use of short and compact programing statements, when possible and when it does not sacrifice clarity. e.g. think of using tuples, list and/or dictionaries as containers for values/variables; this helps compactify the writing. One example of something to avoid (from assignment 10):

```
a=midpoint(f,10,-1,2*math.pi)
b=midpoint(f,20,-1,2*math.pi)
c=midpoint(f,100,-1,2*math.pi)
d=midpoint(f,200,-1,2*math.pi)
e=midpoint(f,400,-1,2*math.pi)
g=midpoint(f,800,-1,2*math.pi)
h=midpoint(f,1600,-1,2*math.pi)
```

The reasons it is not good programing style: it is long, it is hardcoded (changes are difficult), it is needlessly repetitive. You can instead define a list and add a useful comment:

```
n_values=[10,20,100,200,400,800,1600]  # number of sampling points for midpoint()
```

and e.g. loop over **`n_values`** elements in **`midpoint()`**, so **`midpoint()`** could returns an array.

Code optimization/speed: although you will not write codes that are time consuming, you have to demonstrate that you have a sense of optimization in your choice of variables and functions, e.g. prefer the use of arrays over iterative loops where appropriate, do not make unnecessary intermediate calculations if you can reach the same result faster with less commands while still preserving clarity.

Use of functionalities not covered in class: In your comments section at the beginning of the program, you have to include a -short- comment indicting/listing which new functionality(ies) not covered in class you used (this does not apply to new functionality(ies) specifically mentioned in the project sheet).

Bonus points: if you extend the project significantly beyond its initial scope, you can gain up to 5 bonus points. Write also at the beginning of the code (as a comment) what it is you have done extra. The instructors are the only judges how significant your "extension" is.