Relativity Python activity- June 2022

Long-baseline neutrino experiments study the nature of neutrinos, in particular neutrino oscillations, by generating neutrino beams in accelerator facilities. The success of experiments like T2K, NOVA and MiniBooNE relies on the correct modelling of the neutrino beam. Accurate simulations which make use of random generation techniques, also known as Monte Carlo methods, are an essential element of these studies. You can find an example in the preprint
https://arxiv.org/pdf/0806.1449.pdf

Neutrino beams are typically produced by accelerating protons to high energies and colliding them into a fixed target (e.g.a graphite block). The highly energetic interactions between the protons and the carbon nuclei produce secondary particles, principally mesons, such as pions and kaons. These are short-lived particles which produce neutrinos in their decays; the most probable decays are:

$\pi^+ \rightarrow \mu^+ + \nu_\mu$,

$K^+ \rightarrow \mu^+ + \nu_\mu$

In this activity you'll simulate these two decays. You don't need any prior knowledge of particle physics for this task; it's about applying the physics of special relativity. Most of the physics you will need is the physics from the second half of seminar 2. Once you understand what's going on with the seminar, you should be in a good position to start the task. You might need to check back for some physics from the lectures in a few places.

Much of the code is correct, but you will probably find some errors. Your job is to find the bugs and correct them, keeping a record of the bugs you have found in a separate debugging report (details shortly).

The bugs are in the code, not the questions being asked in the script that the code was written from – the code should answer the questions being asked in the script. Remember, many bugs will not cause the code to crash, so you will need to carefully check the code, line by line, as well as the output it generates.
Ask yourself: does it make sense? Is it correct? Does the output make sense? Does it follow the principles that you know should apply?

We are not necessarily expecting you to find every single error. You can get excellent marks on the activity if you find most of the problems in the code, correct them and explain the physics. In fact, you will never be able to be completely sure whether you have found all the problems – unfortunately, coding in real life is also like this! The best you can do is to check it carefully and check carefully that its output seems to be sensible.

Please do not leave it to the last minute to start – you may run out of time! The activity is worth 7.5% of the module credit, compared to 5% for an APS, and the time it will take you should be roughly in the same proportion: **it will probably take longer to complete than an APS**. We expect it might take most people very roughly **in the region of 10 hours to complete**; it may take you personally less time or more time than this though. If you find yourself spending a lot longer, consider stopping and just handing in what you've done. Similarly, if you do run out of time, just submit what you have done.

You may think that you would have written the code differently if you had started from scratch yourself, but in this task your job is to debug the code, so you should not be completely rewriting

code, just correcting the errors. However, if you would like to use different names for any of the variables used in the notebook, feel free to rename them to some other sensible name according to your personal preference. For example, you can do this by opening the .ipynb file in a text editor (clear all the image outputs first) and then applying a search-and-replace e.g. to replace "_lf" with "_lab-frame" if you are someone who prefers long variable names.

If you are feeling apprehensive about tackling something Python-related, the key thing is to get started, rather than putting it off! We suggest tackling the activity in small chunks rather than all in one go – it is difficult to focus on this sort of activity for more than an hour in one go. The notebook is structured into three sections, so you could set an initial goal of completing the first section.

Debugging report

At the end of the task you need to submit **both your corrected code and a debugging report**, both before the deadline. Note that submissions will be checked for plagiarism using plagiarism checkers including Turnitin. This is an individual activity. Do the task on your own!

The focus of your debugging report is *the physics*, not the Python. You should explain why your modified version is correct from a physics point of view (and what was wrong with the original version). Include the code snippet for your corrected code and an explanation. We want to see that you understand the physics of what is happening; this is why your explanation is necessary: **if you include just the code snippet for the corrected code you will only get about half of the marks.** However, each explanation can be concise. Check the sample report available on Blackboard to see the kind of thing the report should be.

If you find an error but can't work out the problem, just explain in your report why you think it's an error. If you know what is wrong with the physics but can't work out how to write this in Python, explain what is wrong with the physics (and what the correct physics should be) in your report. You will still get some credit for this.

Make the debugging report *as you go along*. Don't leave it to the end or you may forget which parts you changed. It might be worth keeping one working copy of the debugged code and one unmodified copy of the code so that you can easily cut and paste the originally 'buggy' section of the code into your debugging report.

You need to write your debugging report within the provided Jupyter-notebook template. The debugging report does not need to be like a formal scientific report would be. It should instead be functional: for each bug you find, describe it within the numbered section provided in the template notebook. Do not modify any of the section headings in the notebook. Have a look at the sample report we have provided to see the kind of thing we are expecting. Your report should be submitted as a Jupyter notebook, i.e. as a .ipynb format text file.

If you need to install Jupyter notebooks on your computer, you can download Anaconda from softwarehub.imperial.ac.uk and then open Jupyter notebooks and navigate to your file location. (Just doubleclicking the Jupyter notebook file you were sent in File Explorer may not work for you since your computer does not know the right application to open it.)

If you are not very familiar with Jupyter notebooks, you can get help about shortcuts by clicking in the borders -or pressing Escape- and then pressing H, or from the help menu. Some of the most useful shortcuts are A, B, M and Y.

-If you want to, you can include the code snippets you need to include by doing a screenshot for each one. However, if you are considering doing this, please check the following look ok in a PDF of

your debugging report (check when you first start, not for the first time right at the end!): the screenshots must be trimmed appropriately (just the code, not your whole screen!) and must show the code legibly: the resolution must be sufficient and the text should be a normal size - not very small! Note that the *explanation* for each bug must be written in the debugging report though, rather than being shown in the screenshot. Have a look at the sample notebook for an example of this.

As well as recording the errors you find and how you fix them, there are a few places in the notebook where the script for the notebook tells you to include something in particular in your debugging report, such as a graph, or explaining a result – remember to include these things too!

Your submission will be marked based on your debugging report; if you correct places in the code which you do not include in the debugging report, you will not get credit for those. We might check your code if we need to though (that's why you need to submit the code as well)

Debugging tips

Remember, many bugs will not cause the code to crash, so you will need to carefully check the code, line by line.
Ask yourself: does it make sense? Is it correct? Does the output make sense? Does it follow the principles that you know should apply?

The notebook is written to be run from the top sequentially downwards, so be cautious about executing cells out of sequence. This could cause additional errors.

As with any other debugging, you may find it helpful to add extra print statements to look at the values of particular variables, or extra comments on how bits of the code work. There are no marks for doing/not doing this, but it may make it easier for you to understand the code and debug it.

If you add additional variables or define different functions, we recommend using different variable names from the ones already in the code to avoid problems if that variable is used again further down the code.

If you come across some Python syntax you don't understand, make use of online help to find out what it means.