

MANUAL FOR USING GEANT4 WITH THE DREXEL BUBBLE CHAMBER

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 2 | Setup | 2 |
| 2.1 | Accounts | 2 |
| 2.2 | Obtaining the GEANT Code | 3 |
| 3 | Running GEANT | 3 |
| 3.1 | Adding New Components to the Chamber | 4 |
| 3.1.1 | PICO250DetectorConstruction.cc | 4 |
| 3.1.2 | Other Files | 6 |
| 3.2 | Bash Scripts For Automation | 6 |
| 3.3 | Setting Up Macro Files | 8 |
| 4 | Analysis Code | 10 |

1 Introduction

As part of my senior thesis for graduating in June 2019, I adapted the GEANT4 software to the Drexel Bubble Chamber, wrote some analysis code, and wrote some bash scripts for automating many functions. I spent many months painfully learning and figuring out how to work with GEANT, so hopefully this manual will save you many of the troubles that I faced.

Within the manual, I will show how to get started with GEANT and the PICO code, how to run it, go through my scripts and code, and explain some of the common issues that I ran in to. It's not necessary to follow everything step by step, so skip around to what you actually need. In the spirit of helping future people who might work with this code, you should update this manual with any changes, challenges you faced, etc. to help facilitate the transition of knowledge.

If you do run in to any issues that you can't seem to solve, feel free to contact me at: salvatorezerbo37@gmail.com. I'll try my best to respond in a timely matter with useful advice but don't get your hopes up.

2 Setup

Hopefully you have access to a linux machine, since operating GEANT on windows is a nightmare, and my scripts/code were all written on a linux machine. If you don't have one, it's never too late to set up a linux dual boot on your laptop. It's fairly quick, easy, and is extremely useful to have when you need it. I have no idea how anything works on mac, so you're on your own there for installation and adapting the scripts as necessary. For reference, below is a list of versions of everything that I used. These versions might not be necessary, but if you run in to issues, try to obtain something similar.

| Software | Version |
|-------------|--------------------|
| Linux | Mint 18.3 (Sylvia) |
| Bash | 4.3.48(1)-release |
| GEANT4 | 10.3 (From SVN) |
| Python | 3.5.2 |
| –Pandas | 0.24.1 |
| –Numpy | 1.16.2 |
| –Matplotlib | 3.0.3 |
| –astroML | 0.4 |
| –astropy | 3.1.2 |

2.1 Accounts

In order to access the SVN that contains the PICO GEANT code, you will need to have a Snolab account. I had already obtained my Snolab account from my brief visit to Snolab,

so I cannot say too much on this. Professor Neilson should be able to guide you to getting an account.

I don't believe you need to obtain Fermilab accounts for the purpose of the GEANT simulations, but it might be useful to also get it out of the way during this stage if you think you'll need it for other purposes. Docdb 903 has a guide for obtaining a Fermilab account.

2.2 Obtaining the GEANT Code

The SVN repository already contains a version of GEANT4 along with the code for the DBC and other chambers. To obtain the code, follow the steps on slide 3 of Docdb 3482. After cloning the svn and following the steps from the slides, your file structure should like the one seen below in Figure 1. For simulations with the DBC, you will primarily interact with code within the pico-svn/DBC-from-40/ file structure.

3 Running GEANT

In this section, I'll go over how to run the simulations, make changes to the DBC construction, set up different sources with macro files, and explain the bash scripts I wrote to

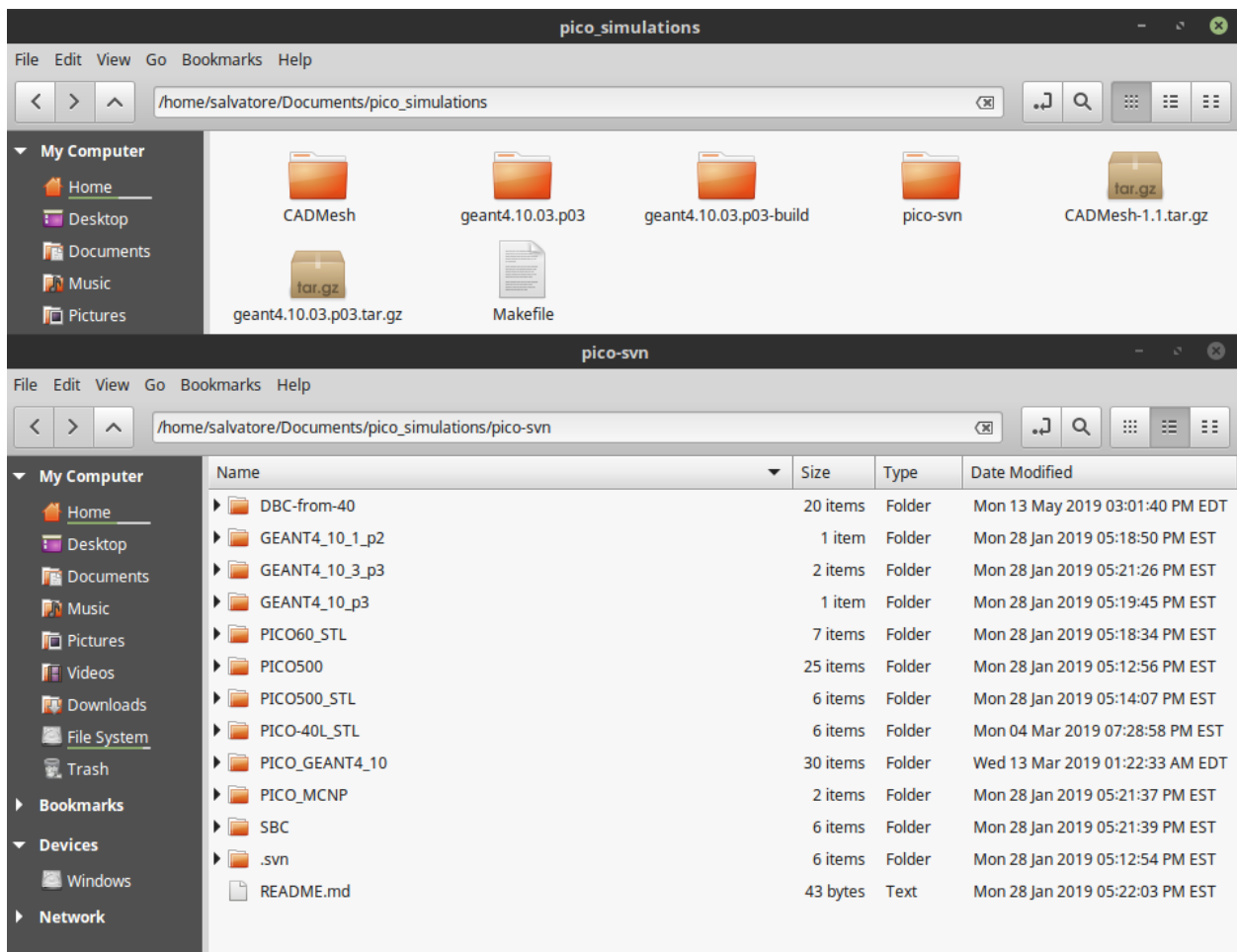


Figure 1

```

183 // G4Colour black (0.0, 0.0, 0.0) ;
184
185 //*****
186 // ***** DBC *****
187 //*****
188 if (DBCFlag){
189 // rotation if needed
190 G4RotationMatrix* yRot = new G4RotationMatrix();
191 G4RotationMatrix* jarRot = new G4RotationMatrix();
192 yRot -> rotateY(90.*deg);
193 jarRot -> rotateX(180.*deg);
194
195 // Scale detector to larger chambers - set value to 1 to match DBC
196 G4double SCALE = 25.0;
197
198 // Universe - Room made of air
199 G4double wallThick = (10.*cm) * SCALE;
200 G4double worldWidth = (40.0*cm + 2.*wallThick) * SCALE;
201 G4double worldLength = (40.0*cm + 2.*wallThick) * SCALE;
202 G4double worldHeight = (40.0*cm + 2.*wallThick) * SCALE;
203
204 //World
205 G4Box* world_box = new G4Box("world_box",
206 | 0.5*worldWidth,
207 | 0.5*worldLength,
208 | 0.5*worldHeight );
209 world_log = new G4LogicalVolume(world_box,
210 | world_mat,
211 | "world_log");
212 world_phys = new G4PVPlacement(0,
213 | G4ThreeVector(0.,0.,0.),
214 | "world_phys",
215 | world_log,
216 | 0,
217 | false,
218 | 0);
219
220 G4VisAttributes* world_vat= new G4VisAttributes(yellow);
221 world_log->SetVisAttributes(world_vat);
222

```

Figure 2

automate much of the process. Before jumping in to the DBC code, it may be helpful to become familiar with GEANT4 through the examples given. These are found under the file structure `geant4.10.03.p03/share/Geant4-10.3.3/examples`.

3.1 Adding New Components to the Chamber

The last version of the DBC that I constructed within the simulations is fairly simple, so I'm sure you'll want to know how to add more to it for more accurate simulations. Again for everything related to the DBC, all file paths I list will have an implied `pico-svn/DBC-from-40` in front of them. The primary files that need to be edited to add additional components are as follows:

```

include/PICO250DetectorConstruction.hh
include/PICO250DetectorMaterial.i.hh
src/PICO250DetectorConstruction.cc
src/PICO250DetectorMaterial.icc

```

3.1.1 PICO250DetectorConstruction.cc

The first step is to create the component within the `PICO250DetectorConstruction.cc` file. You should open the file and scroll down until you see a large commented out header saying

DBC, as seen in Figure 2. All code written should be contained within the if (DBCFlag) statement. Hopefully you have some idea as to what generic shape you would like to use to model the component. I primarily used the G4Tubs, which is a cylinder, and the G4Polycone, which is a radially defined object used for the C₃F₈ and its container.

First, you should define the dimensions of the object you are trying to create. As an example, I'll go through the code of constructing a generic cylinder, abbreviated GC in the code. The first group of code might look something like:

```
G4double GCRadius = 5. * cm; \\GEANT4 has its own units defined conveniently
G4double GCHeight = 5. * cm;
G4double GCPosZ = 10. * cm; \\You can define other position coordinates, but I pretty
                               \\much only cared about the z-axis.
```

Next, you'll want to actually define the object, set its dimensions, give it a logical volume, and give it a physical volume.

```
G4Tubs* GC =                               \\Set the cylinder's dimensions
    new G4Tubs("GC",                       \\Name for referencing
        0.*cm,                             \\Inner radius
        GCRadius,                          \\Outer radius
        0.5*GCHeight,                     \\Height extends in -z and +z
        0.*deg,                            \\Starting angle
        360.*deg);                         \\Ending angle
GC_log =                                   \\Define the logical volume
    new G4LogicalVolume(GC,                 \\Attach to the object
        GC_mat,                            \\Material as defined in DetectorMaterial files
        "GC_log");                         \\Name for referencing
GC_phys =                                 \\Define the physical volume
    new G4PVPlacement(0,                    \\Rotation matrix to rotate object
        G4ThreeVector(0.,0.,GCPosZ),      \\Position
        "GC_phys",                         \\Name for referencing
        GC_log,                            \\Attach to logical volume
        world_phys,                        \\Mother physical volume to position relative to
        false,                             \\Currently unused, always false
        0);                                \\Overlap check
```

The final step in this file is to add the visibility attributes. Here, the color is set and the visibility is set to true

```
G4VisAttributes* GC_vat = new G4VisAttributes(magenta);  \\Set color
GC_vat->SetVisibility(true);                             \\Set visibility true
GC_log->SetVisAttributes(GC_vat);                         \\Attach visibility attributes
```

3.1.2 Other Files

These next steps are all extremely simple compared to the previous file, so I will combine them all together here. In these files, you can place the new lines of code pretty much anywhere, though make sure you can find them again. Next navigate to the include/PICO250DetectorConstruction.hh file. Here, you'll want to add 2 new lines of code to define the logical and physical volumes set in the previous step. This will look like:

```
G4LogicalVolume* GC_log;
G4PhysicalVolume* GC_phys;
```

And that's all for that file! Next navigate to include/PICO250DetectorMaterial.ihh. Here, add 1 line:

```
G4Material* GC_mat;
```

Finally, go to src/PICO250DetectorMaterial.icc. If the material you want to use is already defined, then this step is as simple as the others. Just add the following line to the code:

```
GC_mat = metalAl; \\I set this to aluminum, but it can be anything
```

If the material you want to use is not already defined, then you should follow the example below and add it a bit further up in the same file. Here I show how to create a material for Beryllium Oxide:

```
G4Material* BeO = new G4Material(name = "BeO", density = 3.02*g/cm3,
                                ncomponents = 2);
BeO->AddElement(Be, 1);
BeO->AddElement(O, 1);
```

Fairly simple. The ncomponents defines how many different elements there are in your composition. Each element is then added, and you can specify how many atoms of that element there are. With this, you're all done constructing, and you are now ready to compile the code to run.

3.2 Bash Scripts For Automation

There are three bash scripts that I wrote for aiding in compiling the code and a few other useful features. These files are all contained within the build/ directory, and you should make sure each file has proper permissions. If it does not, you can change them with the

command "chmod 700 file.sh".

The first file we'll look at is the compile.sh file. This is the primary file that I used for compiling, running macros, storing data, etc. In fact, the remaining two bash scripts rely heavily on this script. At the top of the file, you should see two commented out lines. These lines set the source directories for GEANT and CADMesh. You should either uncomment these lines, or set the paths in your .bashrc file. If you do not set them, you will run into errors compiling. The two lines beginning with "cmake" should be updated to reflect the full path to GEANT and the full path to the build directory. Next, I'll describe the parameters for the script:

| | |
|----------------|--|
| guiType | - "pico", "gun", "nomake" Compile for certain macro commands or don't make, see next section. |
| runScript | - "piconovis", "gpsnovis", "norun" Choose to run macro from console, no visualization. If empty, opens gui. |
| particleScript | - "neutron", "gamma", "cm" (gps only) Choose macro file to run, should update with you're own if necessary |
| moveOutput | - "move" Moves the output of the simulations to archive folder with date if set. |
| fileName | - Anything If moveOutput set, adds descriptor to filename such as: gamma_ \$fileName.root |

The next file to look at is run_multiple.sh. This file is used to automate the running of n simulations. You should already have compiled whatever changes that you may have made to the source code, and changes to the macro files you would like to use should also be made. This script draws two random numbers to set the seeds in the macro file and runs as many times as you would like. The seeds used are output to a file to be used again if desired. There are a few changes you can make here:

1. line='grep "/random/setSeeds" neutron.mac'
2. sed -i "s|\$line|/random/setSeeds \$seedOne \$seedTwo|g" neutron.mac
3. ./compile.sh nomake piconovis neutron move "25x_ \$i"

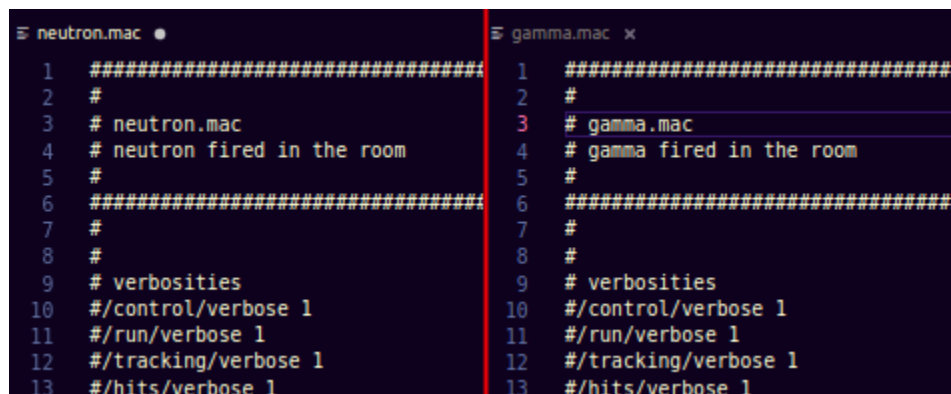
In the first and second lines, you should replace neutron.mac with the macro file that you would like to run. In the third line, you should make sure to compile with either pico or gun as appropriate and change the name of the files to reflect what you would like them to be called.

The final bash script I wrote is run_multiple_from_file.sh, which is an extension of the previous file to run the simulations using seeds already used previously. This will allow you to rerun simulations to reproduce results, restart simulations if they have to be interrupted for some reason, and set your own custom seeds. The same three lines as the previous file can still be altered. In addition, at the end of the while loop, you should replace "done < neutron_seeds_18mm.txt" with the file containing your seeds that you would like to run.

3.3 Setting Up Macro Files

Macro files, which are files ending in the ".mac" extension, are used to define the type of source and its properties such as location, isotope, strength, position, size, etc. Several other simulation parameters such as the number of events, seeds, output verbosity, etc are also set here. There are two types of sets of commands that I used: "gun" and "pico". These commands can only be used when compiled with the corresponding variable set, as described in the previous section. The pico commands should be fine for most of the applications you may want. The gps command set is useful for defining a custom emission spectrum of the source and setting a custom angular distribution for the direction that the particles are emitted.

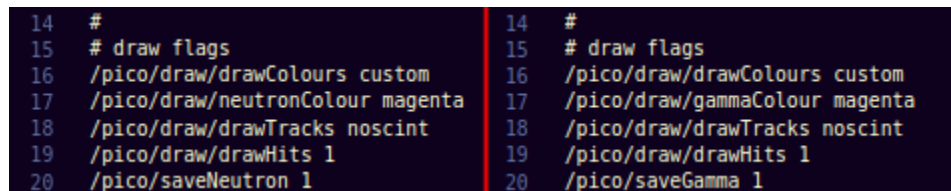
Here, I'll run through the basics of the neutron and gamma.mac files that I used for simulating a ^{207}Bi gamma source with a ^9Be target. I'll break down the macro files in groupings of parameters with the neutron file on the left and the gamma file on the right. The first group is the verboisities. These parameters control how much output there is to the console or to the GUI; however, they do not control the actual data that is output to the files. These commands are useful for checking event by event results and can be piped to an output file for storing the output.



```
1 #####
2 #
3 # neutron.mac
4 # neutron fired in the room
5 #
6 #####
7 #
8 #
9 # verboisities
10 #/control/verbose 1
11 #/run/verbose 1
12 #/tracking/verbose 1
13 #/hits/verbose 1
```

Figure 3

The next grouping is the draw parameters. These control the colors, which lines are drawn, and which events are saved. The saveNeutron flag here will only save nuclear recoils in the target volume. The saveGamma flag will save all recoils with the target volume, including nuclear recoils. It is important to only set one or the other, otherwise you will double count hits.



```
14 #
15 # draw flags
16 /pico/draw/drawColours custom
17 /pico/draw/neutronColour magenta
18 /pico/draw/drawTracks noscint
19 /pico/draw/drawHits 1
20 /pico/saveNeutron 1
```

Figure 4

The next two flags determine the output of the results. The primary flag here is the savePMT flag, which will output to a file specified further down. This file is what I used with my Python code to perform the analysis of the results. I'm not sure if the saveHits flag affects the results, so I've left it set to true. I also am not sure why the results of the simulation are output to a PMT file instead of the hits file, considering there is no PMT for the DBC, but that seems to be how it is.

```
21 #
22 # file output
23 /pico/savePmt 1
24 /pico/saveHits 1
```

Figure 5

The flags here set which detector is currently being used. This is useful if you end up doing simulations on other detectors for some reason. If you recall back to the PICO250DetectorConstruction.cc file with the DBCFlag variable, there are certain properties and physics attached to each detector.

```
25 #
26 #Detector
27 /pico/PICO250 0
28 /pico/COUPP60 0
29 /pico/PICO2L 0
30 /pico/COUPP4 0
31 /pico/DBC 1
```

Figure 6

This line is fairly self-explanatory. I'm not sure how relevant it is for the DBC, so I've left it set to false.

```
32 #
33 # kill gammas in lab wall
34 /pico/KillGammasInConcrete 0
35 #
```

Figure 7

The grdm commands allow you to use the full properties of the isotope. For the neutron source, I only cared about the neutron emission, which was modeled using calculations I had previously done. For gamma emission from the ^{207}Bi source, I used the full isotope model. The key is the nucleusLimits variable which determines how far down the decay chain GEANT will track.

```
36 # radioactive decay module
37 #/grdm/analogueMC 1
38 #/grdm/verbose 0
39 #/grdm/allVolumes
40 #
40 #/grdm/nucleusLimits 207 207 83 83
```

Figure 8

This block contains the meat of the macro file. It outlines all of the properties of the source, including its energy type (monoenergetic vs spectrum), energy(ies), angular distribution, position, size and shape, etc. Note, for grdm, I set the particle to be an ion. Here the energy refers to the movement of the ion itself, not the emitted particles. As a result, 0 energy means that the ion is sitting still in one spot.

```

41 # gun: particle
42 /pico/gun/verbose 1
43 /pico/gun/particle neutron
44 /pico/gun/energytype Mono
45 /pico/gun/energy 94.14 keV
46 #
47 # gun: isotropic
48 /pico/gun/angtype iso
49 #
50 # OR gun: shoot at detector
51 #/pico/gun/angtype direction
52 #/pico/gun/direction -1 0 0 m
53 #
54 # gun: source
55 /pico/gun/confine NULL
56 /pico/gun/type Volume
57 /pico/gun/shape Sphere
58 #/pico/gun/centre 18 0 30 mm
59 #/pico/gun/centre 216 0 360 mm
60 /pico/gun/centre 411 0 750 mm
61 /pico/gun/radius 0.1 cm

```

Figure 9

Finally, this block details the names of the files that will be output. In particular, I used the neutron_C3F8.out/gamma_C3F8.out files for the analysis code in the next section. Also output is the neutron.root and gamma.root files, which are useful for using the ROOT program, although I did not use that much. Also in this block is the line for setting seeds, which was explained more in the section on my bash scripts. The final line is how many total emissions of the source GEANT will track, not the total hits on the target volume. So, 1000000 neutrons will be emitted from the source; however, you might only see a fraction of them recorded in the results. This number is very important in determining the real-time equivalent for a source, so keep track of it, as it is used in the analysis code in the next section.

```

62 #
63 /pico/hitsfile neutron.out
64 /pico/pmtfile neutron_C3F8.out
65 /pico/histogramfile neutron
66 #
67 /random/setSeeds 35 54
68 #
69 /run/beamOn 1000000

```

Figure 10

4 Analysis Code

In this section, I'll discuss all of the code I wrote briefly in hopes that it is useful to you. I attempted to maintain good programming practices; however, as my thesis came to a close,

I began to get a little sloppy, so hopefully nothing is too confusing to understand. All of these files are stored in the directory build/Analysis.

The first file that I'll explain is `create_histograms.py`. This file is the primary file to run for creating histograms and contains most of the setup. This file creates the plot as seen below in Figure 11 by making a histogram of the simulation results in the `C3F8.out` file, calculating the bubble rate per hour, and displaying the bubble nucleation threshold. The parameters are as follows:

- `file_list` - Full directory to either an exact `C3F8.out` file, or a folder containing many `.out` files. All data will be plotted together as in Figure 11.
- `column_id` - Column that you want to histogram. 6 is the deposited energy.
- `tot_events` - The total number of events the simulation ran for, as described at the end of the previous section.
- `thresh` - Bubble nucleation threshold that you would like to see results for.
- `rate` - The rate of emission of the source. For ^{207}Bi , I had estimated a rate of 0.3 neutrons/s.

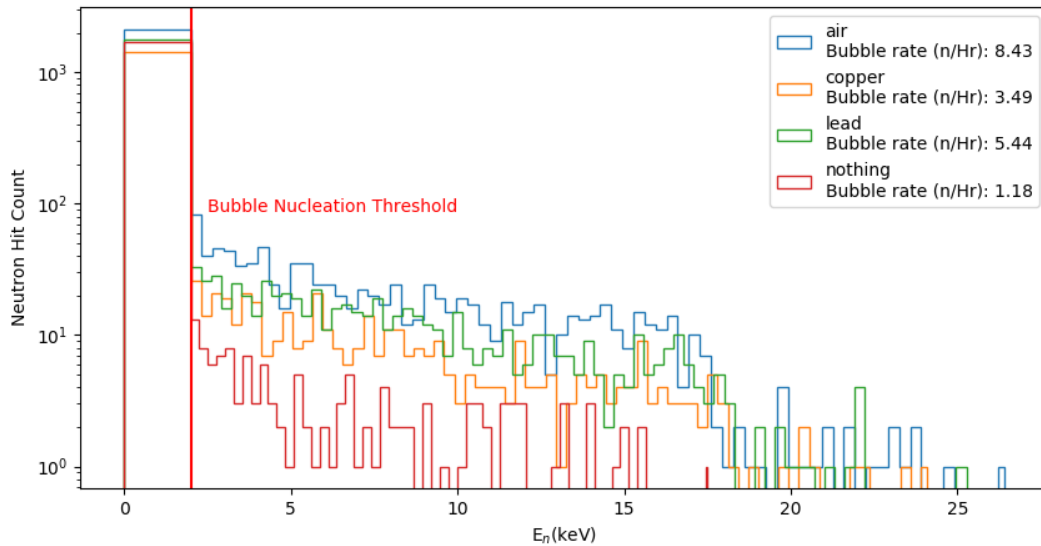


Figure 11

The next two files I'll discuss are secondary files split from `create_histograms.py` to improve readability. These two are `data_operations.py` and `file_operations.py`. Since these are secondary files, you do not ever need to run them directly, and you should not have to make changes to them unless you discover a bug or need additional functionality. The `file_operations.py` file performs the initial functions to read and load data from a file and column. The `data_operations.py` performs all of the calculations such as bubble multiplicities, bubble rates, some statistics, and adds a histogram to the figure once calculations are finished.

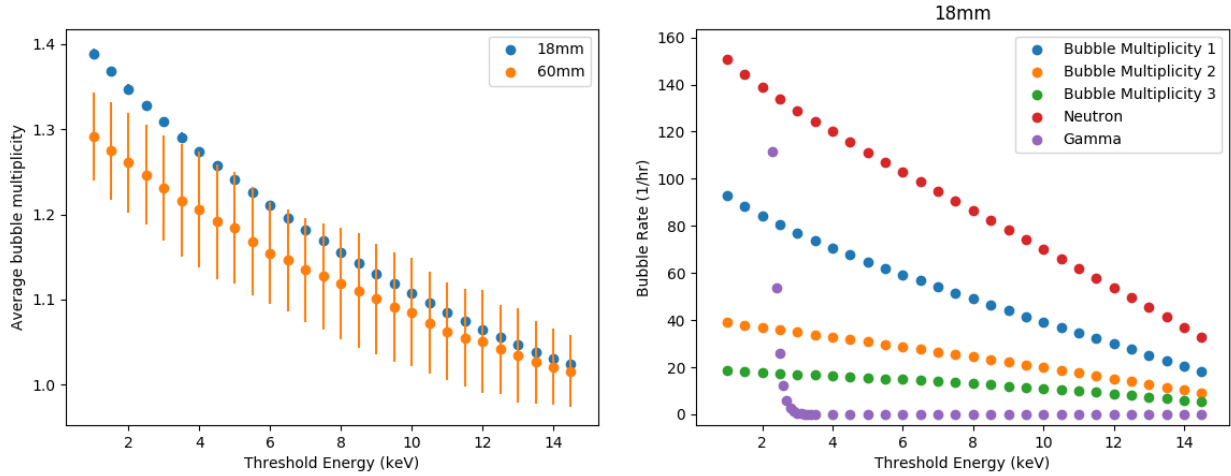


Figure 12

The `bubble_multiplicity.py` file explores a wide range of bubble nucleation thresholds and calculates the average bubble multiplicity with error bars at each threshold. The results from one of my plots is shown above in Figure 12A. There are two different input combinations. The first is a single parameter with the full directory to a single C3F8.out file or a folder containing many C3F8.out files. The second combination is two parameters, each being the full directory to a file. This is how I created Figure 12A. There may be some bugs with the first case, because as I made changes, I sloppily did not check that previous functionality was preserved.

The `create_threshold_plot.py` is used to similarly explore a range of bubble nucleation threshold energies, but this time, it calculates the expected bubble rate per hour for each threshold. I would guess that this file contains the sloppiest pieces of code, as I did not find a simple method to break down the rates by multiplicity, as seen in Figure 12B, without adding a method that reruns the entire loop again. As such, this code is fairly slow. In addition, this code was run on 100 simulations of 100000 events each. I do not know how it will function if you were to do a single simulation of 10000000 events. Something important to note is that this script uses data obtained from the PICO electron recoil paper to determine the gamma efficiency function to calculate bubble nucleations from gammas. The parameters are as follows:

- `neutron_path` - Full directory to the folder containing the `neutron_C3F8.out` files.
- `gamma_path` - Full directory to the folder containing the `gamma_C3F8.out` files.
- `column_id` - Column that you want to histogram. 6 is the deposited energy.
- `tot_events` - The total number of events the simulation ran for, as described at the end of the previous section.
- `neutron_ratee` - The rate of emission of the source. For ^{207}Bi , I had estimated a rate of 0.3 neutrons/s.
- `gamma_ratee` - The rate of emission of the source. For ^{207}Bi , I the source had a quoted strength of 0.01 mCi or 370000 s^{-1} .

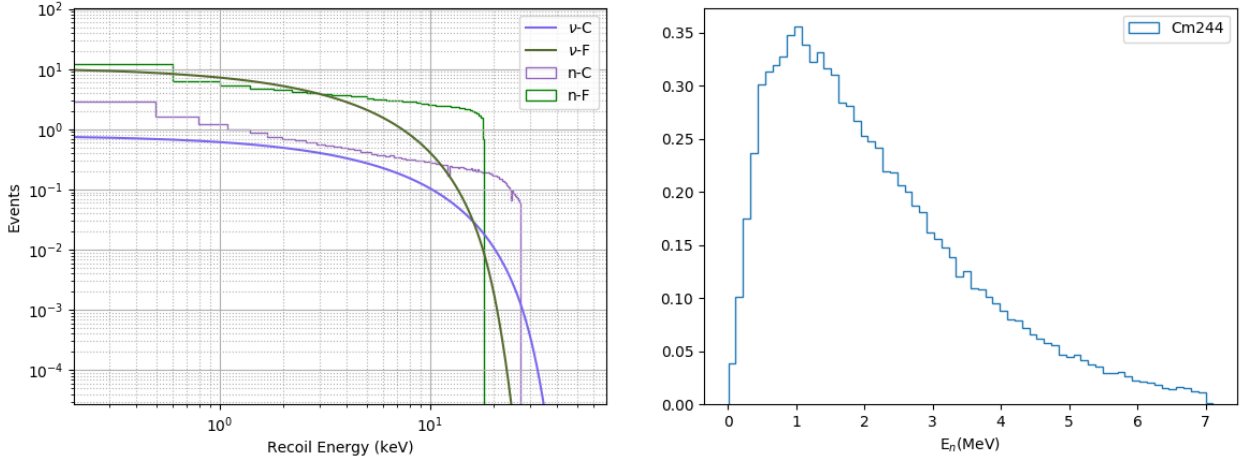


Figure 13

The `create_split_histograms.py` file is an extension of the original histogram python script. This particular script was used to explore the possibility of the neutron spectrum resembling the ν spectrum. This script normalizes the neutron spectrum to match the ν spectrum at around 3 keV; however, the spectra did not match close enough as we had hoped. This script does not have much use outside of this scenario, so unless you are exploring a similar comparison of spectra, you will probably not need to use this script. Figure 13A shows what the output looked for the scenario I described above.

The final two files work together to plot the spectrum of a non-monoenergetic source. This requires you to first pipe the output of the `compile.sh` script to some file. Then, the `read_output.sh` script will scan that file for the energies and event numbers. Finally, the `energy_distribution.py` file takes the output of the previous script and creates a histogram as seen in Figure 13B. This was useful to ensure that the spectrum of emitted particles matched what the actual source's spectrum would look like. These files won't be very useful for monoenergetic sources, and their only real use is if you suspect that there are issues with the emission spectra.