Calculating the Differential Cross Section Using IceCube Simulation Data

Ryan Hill

Queen Mary, University Of London Undergraduate

`r.l.hill@se12.qmul.ac.uk`

August 27, 2015

Abstract

This review will summarise the work I have been doing with Dr Teppei Katori over the summer of 2015 with regards to calculating the Differential Cross Section (DCS) using simulated data from the IceCube experiment at the South Pole. We will start by reviewing the equations used to calculate each part needed for the DCS, then look at the methods implemented in the macros themselves, before finally considering what the next steps would be for the development of the code.

Contents

1 Introduction

1.1 Terminology

There are a few terms that I will be using throughout this summary which you may or may not be familiar with depending on your exposure to using ROOT or neutrino physics so I will just explain them quickly here for those who need them.

- Macro - A Macro is a snippet of code that ROOT can run. The benefit of them is that running long pieces of code in the command line can be tedious and annoying, so macros allows for repeated use of and ease of editing commands. I will use the term macro interchangeably with program and code.

- Zenith angle - This is technically the angle measured from directly overhead down to the horizon with a range of $[0,\pi]$. However, 99% of the time if this summary mentions the zenith angle it is talking about the cosine of the zenith angle with a range of [-1,1], as this is much more useful for physical purposes.

- Azimuth angle- This is the rotational angle on the plane of the horizon with a range of $[0,2\pi]$, commonly denoted as ϕ.

- Bin - Bins are how values are represented in histograms; each bin has a height that is the count of how many data points had a value that fell between the two edge values of the bin.

1.2 Differential Cross Section

The DCS is a very important quantity in the world of particle physics and is more useful than just the regular cross section as it quantifies the intrinsic rate of an event that can be detected at a given angle, something the scalar cross section cannot tell you. With this in mind we can see why it might be important to want to calculate this quantity. The main equation we will be using in this summary is similar to the usual DCS equation with some additional quantities due to way in which the data is simulated. That equation can be written as

$$\frac{d\sigma}{dE_{\mu_i}} = \frac{\sum\limits_{j} U_{ij}(d_j - b_j)}{\Delta E_{\mu_i} N t \Phi \epsilon_i S r} \tag{1.1}$$

where U_{ij} is the unfolding matrix, d_j, b_j are the data and background respectively, ΔE is the energy bin width, N is the number of target nuclei, t is the simulated time length, Φ is the integrated neutrino flux, ϵ_i is the bin efficiency and Sr is the solid angle for that bin. The outputs of the main macro, included two of the DCS for simulated IceCube data are shown in figure 1.1.
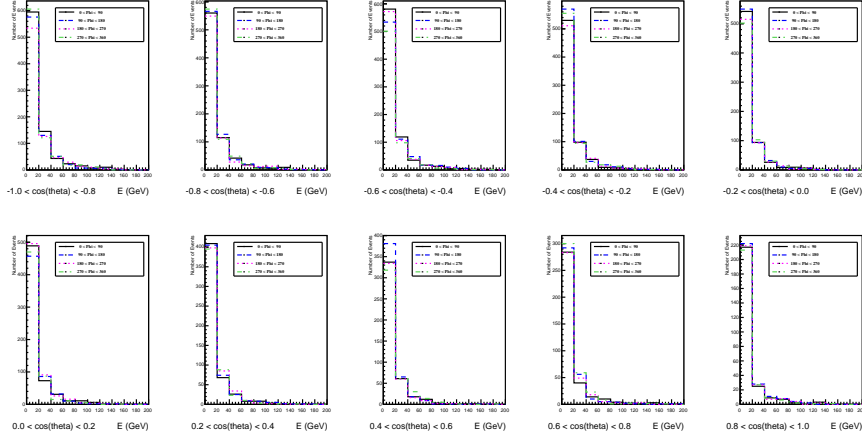
(a) True data distribution just for comparison and sanity checks
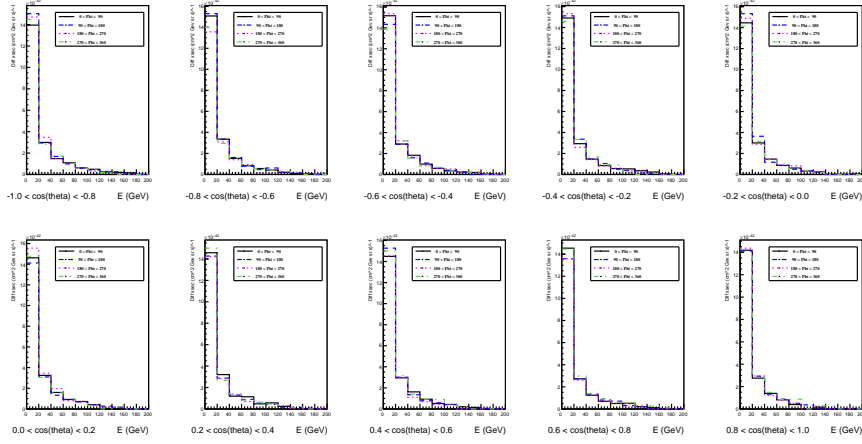
(b) The DCS of the fake data from the IceCube file without any smearing to the data
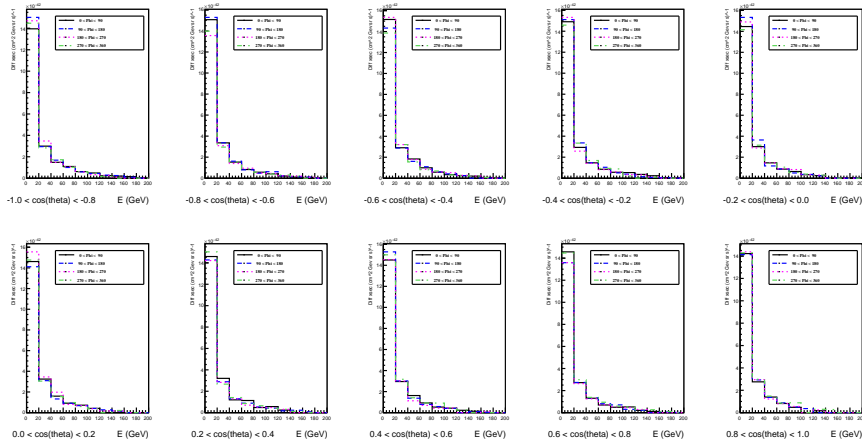
(c) The DCS of the fake data from the IceCube file with a Gaussian smear to the data

Figure 1.1: Example outputs from the main macro in the project. Each image is split into 10 plots representing 10 $\cos(\theta)$ ranges and each plot has 4 historgrams representing each ϕ range.

1.3 Unfolding Matrix

The unfolding matrix is the most complicated part of the calculation as it involves creating it from a data set that contains both the *true* data and the *fake* data; by that I mean that the true data is the correct values for all quantities relating to all particles involved, and the fake data is the values that are calculated/measured after detection has taken place. Normally this would be impossible, how could we know the true values from before detection? We can get around this problem by using a simulation (well, 1000 simulations to be more specific) so we can know the true and fake data and work out the unfolding matrix from this.

Before we can create our unfolding matrix we must create a migration matrix, M_{ij}. In generality it would be possible to have a different number of bins for the fake and true data, and whilst it probably would be possible to change the code to reflect this decision, in this summary we will only consider the case where M and U are square matrices i.e both true and fake data have the same binning. The reason you might want a non-equal number of bins for true and fake data is simple; if you know you only simulated your neutrinos to have a maximum energy of 195GeV then you know no true muons of a greater energy can exist, but there may be some in the fake data. As such you might want a higher range for your fake data compared to your true data (as making it square increases computational time and memory usage).

The migration matrix defines the correlation between the true and fake data such that

$$d_j^{true} = \sum_i M_{ij} \tag{1.2}$$

and

$$d_i^{fake} = \sum_j M_{ij}. \tag{1.3}$$

Some examples of migration matrices for different bin selections are shown in figure 1.2, to avoid having to have a 6 dimensional histogram we decided to wrap the variables inside each other, so that each energy bin will contain all the $\cos(\theta)$ bins and so on. Now that we have this we can create the unfolding matrix by normalising the columns of the migration matrix, it is worth noting that some other sources will include the efficiency in the calculation of U, but we keep that for only the final calculation. To normalise the columns we simply use

$$U_{ij} = \frac{M_{ij}}{\sum_k M_{kj}}. \tag{1.4}$$

Now that we have the unfolding matrix we can start looking at the other components of equation (1.1).

1.4 Number of Nuclei Targets

The number of targets is hardcoded into the calculations. It was calculated assumed a cylinder of $R = 600m$, $H = 1200m$ and an average Antarctic ice density of $0.922 Mg\,m^{-3}$.
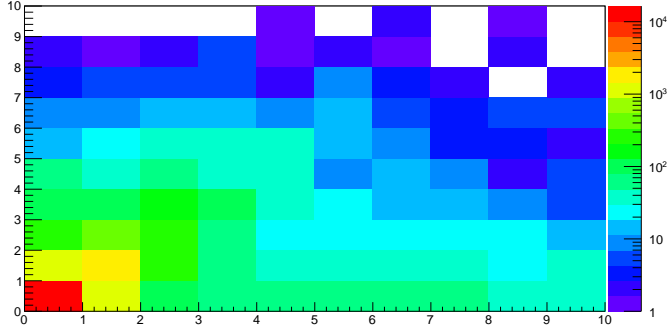
(a) 10 Energy Bins, 1 Cos(θ) bin, 1 ϕ bin
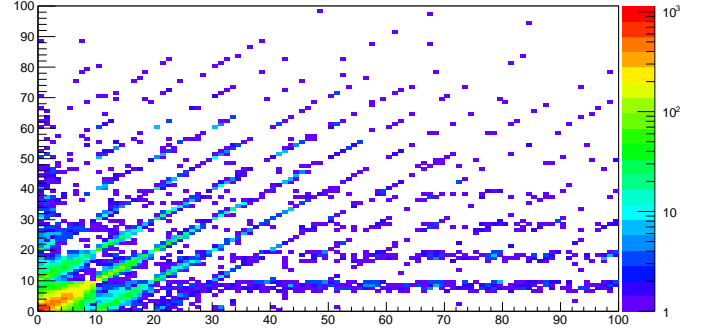
(b) 10 Energy Bins, 10 Cos(θ) bins, 1 ϕ bin
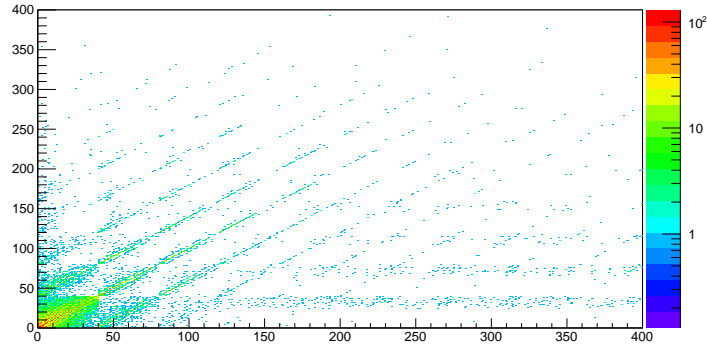
(c) 10 Energy Bins, 1 Cos(θ) bin, 4 ϕ bins

Figure 1.2: Different migration matricies, axis are simply bin numbers, in (1.2b) every 10 bins are the cos(θ) bins for a given energy, in (1.2c) every 4 bins are ϕ bins for a given cos(θ) and a given energy.

This combined with the knowledge that the molar mass of H_2O is $12.01528 g \, mol^{-1}$ we can calculate the number of H_2O molecules using

$$N = \frac{Vol \times \rho \times N_A}{n_{H_2O}} \approx 8.366 \times 10^{37} \quad (1.5)$$

where ρ is the density of the ice, N_A is Avagadro's number n_{H_2O} is the number of moles of water.

1.5 Other Variables

Finally we can just list off the other variables as they are all quite simple.

- d_j - this would be the post-detection data that we would be provided. As it is this is our fake data. Obviously if we just use the fake data we would just arrive back at our true data so we smear the fake data in the macro before we use it.

- b_j - background data, this is any neutrinos that are non-atmospheric. In the simulation data there is not background and as such it has not been included in the code.

- ΔE - this is the energy bin width in GeV. The limits on energy for the histograms are hard coded to be [0,200] but the number of bins can be easily changed by the user.

- t - time the simulation was run for. This was 3 years and has been hard coded into the calculation.

- Φ - This is the integrated flux density of neutrinos. One of the macros is written to explicitly calculate this and we will discuss this more in section (3.2).

- ϵ_i - the detection efficiency is the probability of the detector detecting an event that occurred. It varies dependent on energy and angle, there is a mecro written to do this and we will discuss this more in sec(3.3).

- Sr - the amount of solid angle included in this bin. Solid angle is measured with the integral of $\cos(\theta)$ and ϕ with integral limits of $[-1, 1]$ and $[0, 2\pi]$ respectfully, giving a maximum of 4π if all angular directions were considered at once. These are calculated by the initial bin number selection and do not need to be calculated by the user.

2 Software Requirements

There are a few pieces of software that you will need to have installed to be able to use the code this summary is about. The main one can only be installed on a linux machine so if you don't have one you will need to install a VirtualBox system. There is a lot of webpages mentioned in this section but they will all be included in the Useful Links section at the end of this summary.

2.1 ROOT

ROOT is a data analysis package created by CERN that allows for running of *macros*, pieces of code that allow you to run long or complicated commands in ROOT without having to type them out each time. You can find instillation instructions on the ROOT main website as well as tutorials and help for how to use the package. However I would advise that you don't install ROOT just yet as you are going to have to do a special type of instillation for it when you go to install the next program.

2.2 GENIE

GENIE is a monte carlo generator for neutrino interactions. It can be quite difficult to install from scratch but fortunately the creators of the program have written a script that *should* install it all for you. If you go to the *GENIEMC* github page there is a repo called *lamp*; this script will install all the pre-requisites for GENIE and also GENIE itself, this includes ROOT. Helpful. You will need to make sure you are installing version 2.9.0 as this is the version the new flux driver you will need was written for. There is a chance that the install will fail if you don't have pre-requisites for the pre-requisites, I suggest reading the error message and finding out what is missing, or just google the error messages and hope to fix it. Good luck, you will need it.

2.3 Honda Flux Driver

Whilst not currently implemented into the DCS code it was intended that you would be able to compare the weighted interactions with the Honda flux interactions generated by GENIE, that and if you are using this guide it's very likely that you would want to use Honda flux files to generate interactions with GENIE anyway. All that being said, GENIE can't read Honda flux files as it currently stands so you will need to download an edited version of the GENIE code to be able to run simulations with these flux files. Why did I have you install it via *lamp* if I was just going to ask you to do it by hand anyway? Because the pre-requisite programs and paths are the hard parts to set up. So with this in mind, once again go to github but find the user *rlh1994* and go to the *GENIE_2_9_0* repo and the branch *honda_flux_rv5318*. Once there you will have to download the entire folder as the file structure was changed from the publicly available 2.9.0 and this version, replace your current GENIE folder, make sure you're environment variables have been set correctly by typing *source environment_setup.sh* in your *lamp* folder, then inside the GENIE folder type *make*. It will take a while but it should completely rebuild GENIE with the new flux driver included.

The other option to doing this is to set up the lamp installation such that it installs from this github repo and branch which may be easier depending on your experience with installing GENIE.

Assuming there are no errors you have now correctly installed the new flux driver to allow GENIE to read Honda flux files. To run GENIE with these flux files, all the options for the *gevgen_atmo* command are the same except when specifying the flux file use *ATMNC*

instead of *FLUKA/BGLRS* and if you wish to use simulate multiple neutrino types you must type the file name each time.

3   File Package

The macros that you will actually be running to calculate and plot the DCS is split into 4 programmes both for each of understanding and also to minimise run time. Each piece does small parts of the calculation and then the final piece combines all of the variables to give you the final DCS. Every effort has been made to require the least amount of editing from the user however, due to me not understanding how to give command line arguments easily in ROOT, there will be some editing that the user must do in each case to specify input file names and the number of each bin you would like. I have tried to make it so all the changes the user must make are close to the top of code and clearly commented. Whilst you don't need to understand how the code works the following will also explain some of the main methods the code uses and I made an attempt to comment the code so that users with some ROOT or C++ familiarity should be able to understand what is going on.

3.1   Calculating the Migration Matrix

The first macro you will need to run is the one that will calculate the migration matrix as this will be used to get the true and fake data in the future using equations (1.2) and (1.3). The code itself is called *make_complete_histo.cpp* and will output a root file of the form *ice-cube_energy_#_cos_#_phi_#.root* where each # is the number of bins for energy, $\cos(\theta)$ and ϕ respectively.

This macro is the slowest running of all of them it runs with $\mathcal{O}(i^2 j^2 k^2)$ where i, j and k are the number of each set of bins which is obviously horrible if you want to add even just one more bin of any type. Fortunately if you don't want to change the number of bins or input file you will only need to run this macro once and the migration matrix is then saved to that output file, meaning you can quickly reconstruct the true or false data from the file without having to run this macro again.
The two things the user has control over in this macro is the number of each type of bin and the input data file for the true and fake data. The first are easily identifiable variables declared immediately after the root variable initialising; they are *nbins_en, nbins_cos_theta* and *nbins_phi* which are the number of energy, $\cos(\theta)$ and ϕ bins respectively. A few lines below this is when the input file is added and the user will need to change it so that it read their IceCube input file, the user will have to replace the current *Level7_genie_ic.1460.0000XX_LowE.root* with the name of their file.

The code itself creates 2D sub-histograms from the true and fake data in the IceCube file using a series of cuts in the energy and the value of $\cos(\theta)$ then places these into the main 2D histogram that represents the migration matrix where the content of each bin is the value of that matrix element. This histogram is then saved into the output file and will be used in most future calculations so please do not move this or any other output file to another

7

folder. The matrix itself is currently formatted such that it cycles over the ϕ bins for each $\cos(\theta)$ bin, and it cycles over the $\cos(\theta)$ bins for each energy bin; so on the largest scale it looks like just the energy bins.

3.2 Calculating Integrated Flux

The next thing that needs to be calculated is the integrated flux; this also only needs to be done once and is done with the file named *honda_integrate.cpp*. After this has been run it will create a file called *integrated_flux.dat* and this will be the same for all future calculations assuming you don't change the Honda flux file you use in GENIE. With that in mind the user will have to specify the flux file name that you wish to calculate the integrated flux for, you will need to replace the default *kam-mu-nu-20-12-n3650.dat* with the name of your Honda flux file.

The code itself does a very rough approach to working out the integrated flux; obviously it is impossible to get an accurate integral as you only have discrete data and as such we just calculate the combined area of the bins in the histogram. Bin edges for Honda files and the energy range of 3-195 are hardcoded into this program due to it only needed to be run once and for my purpose I didn't have to consider a different range as this is what IceCube simulations use. This macro is the one that will produce the most error just because of the data from the Honda flux not lining up with the IceCube simulation range.

3.3 Calculating Detector Efficiency

The efficiency of a detector is calculated simply by

$$\epsilon_i = \frac{N_i^{after}}{N_i^{before}} \tag{3.1}$$

where $N_i^{after/before}$ is the number of events after/before cuts respectively. By cuts this includes anything such as the event not being detected, the computer deciding to not record the event etc. It is important to note here that whilst we calculated the total *neutrino* flux, we are trying to calculate the DCS for the *muons* and so the efficiency will be of the muon detection (as this is what IceCube actually detects anyway). The detector will have different efficiencies for different energies, zenith angles, and despite the fact it should be uniform over the azimuth angle we discovered there was some differences in efficiencies for these and as such we left the efficiency to be per energy, zenith, and azimuth (so technically it should be ϵ_{ijk} but we have wrapped the dimensions within each other).

The macro itself, called *ice_genie_loop.cpp*, relies on being passed 3 files: the migration matrix file, a file of neutrino events, and a file of anti-neutrino events. The latter two files must be of a 1:1 ratio, be generated with an E^{-2} flux, with a target of water and with energies in the range 3-195 GeV. Two such files have been provided with the package as well as an extra macro that can generate files in the style of a FLUKA flux file if you wish to generate your own (although this isn't advised without a working knowledge of GENIEs flux driver code

8

as it will involve editing that code, instructions for how to do this are provided in Appendix A, probably, it's probably A).

As with most of these macros you will need to edit the three bin number values at the beginning of the code as well as changing the two root file names should you be providing your own. If you are providing your own and they do not total to 100k events you will need to change one more piece of the code; around line 106 at time of writing there is a line that says

<div align="center">*simulation_histo->Scale(300);*</div>

and you will need to change the 300 such that when multiplied by the total number of events in the files you have provided it equals to 30 million. If you used the provided files you will not need to change this line. The reason we have to scale this is because the IceCube simulation is actually 30 million simulated events to start with before all the cuts.

The macro uses a very similar method to the one that was used to create the migration matrix to fill a histogram with the interaction file data and uses the migration matrix itself to get the true data (remember that the fake data is what we have *reconstructed* the values for, the true data contains the values for the simulated events that were detected). Once it has these it simply uses equation (3.1) to find the efficiency for that bin then writes these all out to a file of the form *efficiency_energy_#_cos_#_phi_#.dat* where the #s represent what they did previously. Some examples of the efficiency as a function of energy, $\cos(\theta)$ and ϕ are shown in figure 3.1.

3.4 Calculating the DCS

The final step to the calculation is just to combine all the parts into equation (1.1) and perform the sum, this is all done with the file named *diff_cross_split.cpp*. Once run this macro will output two canvases, each with m plots with n histograms per plot, where m is the number of $\cos(\theta)$ bins, n is the number of ϕ bins, all with the x-axis being energy in GeV. The first canvas titled *MC Unfolded* is the one the user is likely to be interested in, it contains the histograms for the DCSs as functions of energy, $\cos(\theta)$ and ϕ, the second canvas titled *MC Original* is in there to examine the phi dependence of the detector and was put in just for comparison, the user can feel free to ignore this canvas if it isn't required.

Once again the user will be required to change the variables relating to the bin numbers at the top of the macro and they also have the option to change how the canvas is split for the plots as obviously this will vary depending on how many plots the user would want to have. To change the way the canvas is split the user would need to edit the lines that say

<div align="center">*c101->Divide(5,2);*</div>
<div align="center">*c102->Divide(5,2);*</div>

to contain whatever vertical and horizontal splitting numbers they want respectively (note that whilst there is no problem dividing the canvas into more plots than needed, the user will receive errors if there aren't enough divisions of the canvas). Currently there is no way for the user to have the plots printed to more than one canvas other than to change the code that plots the histograms which shouldn't be too difficult if the user has some familiarity

<div align="center">9</div>
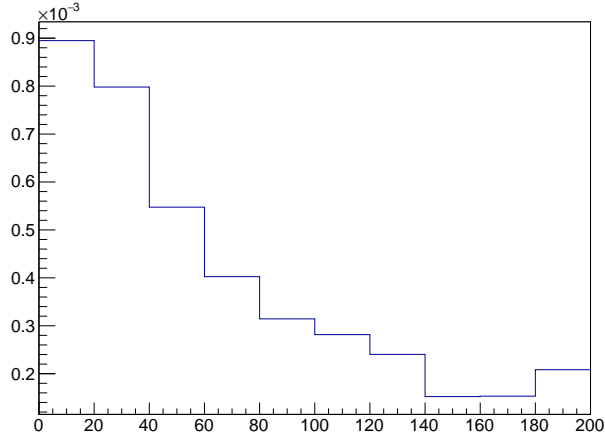
(a) Efficiency over 10 Energy Bins
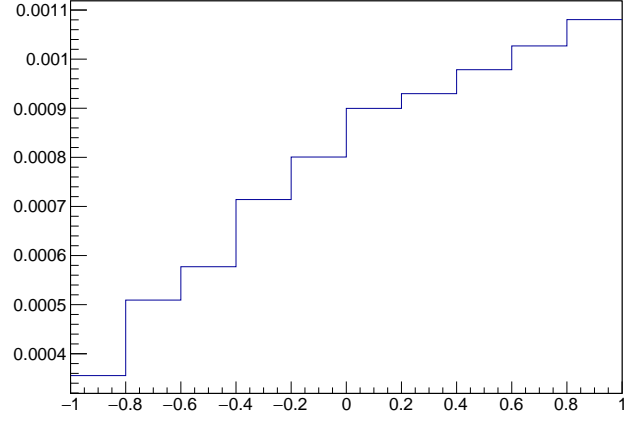
(b) Efficiency over 10 Cos(θ) bins
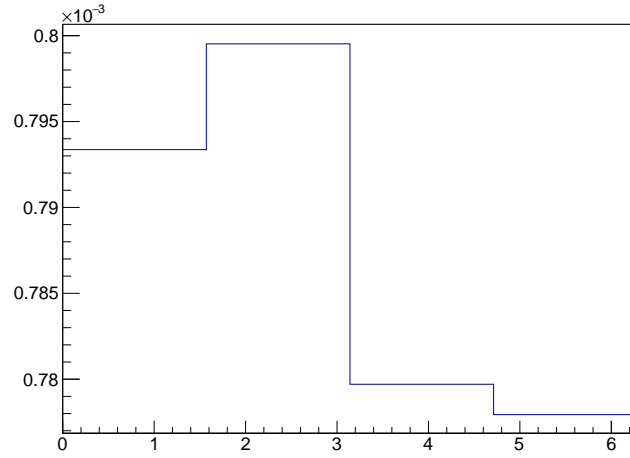
(c) Efficiency over 4 ϕ bins

Figure 3.1: Efficiencies as functions of different variables. The later few energy efficiency bins are not reliable due to low statistics.

with writing ROOT macros.

3.5 Other Included Files

As mentioned previously there are a few other included files/macros to make it easier for the user to begin calculating DCSs. There are two ROOT files that contain GENIE simulated data to be able to run the efficiency calculation which are named *genie_energy_cos_3_195_numu_50k.root* and the same but with numu replaced with numubar. These files each contain 50000 events created using the set-up described in section (3.3) for ν_μ and $\overline{\nu}_\mu$ respectively.

The next file included is named *flux_create_cos* because it creates a flux file in the FLUKA format using $\Phi(E, \cos(\theta), \phi) = E^{-2}$ i.e. only an energy dependence. This is the file that, when combined with the modifications to GENIE that can be made using Appendix A, will allow the user to calculate the efficiency for IceCube files with user defined energy limits. The user will need to change only one number in this file, the upper limit of the first for loop, to be equal to the base 10 logarithm of the highest energy they want (although I suggest going a few steps higher just to be safe). An example output file named *e-2_including_cos.dat* has been included that goes up to 1000 GeV and can be used as an input for GENIE should you need to generate events of a higher energy. The IceCube simulations I had access to were generated with energies between 3 and 195 GeV, hence why only GENIE output files for that energy range were created and included.

4 Next Steps

Whilst I believe that the work I have done is a good foundation for future work there are many gaps in the functionality of the macros as well as other problems I would have solved or options added were I to have more time. That being said what follows are just a few ideas of what could be done to expand/improve the work I started and there are certainly more options to consider than just those I will talk about here.

4.1 Using More Files With TChain

One task that was always pushed to the bottom of my to-do list was learning to properly use TChains; whilst they are used in the macros they are not used to the fullness of their ability. The first step would be to check all the places that the existing code uses TChains and make sure that is able to handle more than one file in that chain in all places. Once this is done more IceCube files could be included at once and this would increase the amount of data available at any one time which is always a good thing (this may change the scaling of the GENIE files in the efficiency calculation and that should be taken into account were you to include more IceCube files). The only other place work with TChains would be required is in the efficiency calculator where currently the neutrino and anti-neutrino simulations are in two separate TChains because of how I set up the combination of them later so this could

be made more efficient by combining them into one TChain and modifying how the macro gets the values it needs from them.

4.2 Weighted Events

One thing that would be of great interest to look at would be the distribution considering a true neutrino flux. As it stands the IceCube data and the GENIE simulation are created by using an E^{-2} flux and no other considerations are taken into account, but that needn't be the case. There is a variable saved in IceCube file that gives weighting to the events in agreement with Honda flux; to add this to the code is simple enough you simply need to multiply the entire cut string by the variable, namely *AtmoWeightHonda.value* and this will give you the weighted distribution. The tricky part comes when trying to work out the efficiency because you need to correctly normalise the number of events before the cuts, i.e the GENIE simulation data.

It was my understanding that the normalisation is linked to a constant that can be calculated by taking the events, dividing by the cross section and then finding the ratio between this distribution and that of the input Honda flux (or at least the average of the ratios for each bin) as

$$N \propto \Phi \otimes \sigma. \tag{4.1}$$

You then either divide or multiply the GENIE output by this and should in theory have normalised the number of genie events correctly. Unfortunately this gives a number that is horribly off what it should be. The problem is that I was not certain of what information the weighted IceCube data contained, I tried with just the cross section convoluted with the flux and multiplied by the time of the simulation but this gave efficiencies of the order of 10^{-17} which was clearly incorrect. Anyone wanting to work on this would need to find the correct way to normalise the GENIE data by knowing what was included in the IceCube weighting, and then using this to calculate the new efficiency.

4.3 Different Binning for True and Fake Data

Something which I hadn't really considered until writing up this summary was that a user might want to have different binning for true and fake data and as such the code is written throughout to work with a square migration matrix. A next step in the work could be to make it so that the migration matrix could be a non-square matrix and this would involve modifying pretty much every for loop in all the macros that use the number of bins as a limit in one way or another. This would be useful for the reasons I explained in section 1.3.

4.4 Optimisation of the Code

As it stands some of the code is horribly inefficient; a lot of it was just made to work before carrying onto the next step and as such there are probably many places where the code could be optimised to run faster/more efficiently.

The bottleneck of any run is in generating the migration matrix; this takes by far the longest

of any of the macros and runs at $\mathcal{O}(i^2 j^2 k^2)$ where i, j and k are the number of each set of bins. This is obviously not going to be useful if you want to take a large number of bins, as it stands even small numbers of bins can take a long time to run. This should be the main starting point for any code optimisation as even if other parts of the code were improved this would still be the part a user was waiting on.

The method by which the migration matrix is made involves, as mentioned previously, creating small histograms, getting the bin content, then setting this content into the larger histogram; this method is applied throughout all the macros and if it were improved upon in one could almost certainly be improved upon in the others as this is the part of the macros that takes the longest to run in each case.

5   Conclusion

The work I had done over the summer is definitely a good starting point for someone to develop on more into the future, most of the main infrastructure is there to be built upon and excluding one bug is in working condition to the point where it could be used right now. It does not accomplish tasks is the best way, sometimes just brute-forcing them, but it does get them done.

6   Useful Links

ROOT: `https://root.cern.ch`
GENIE main site: `http://genie-mc.org`
Lamp github page: `https://github.com/GENIEMC/lamp`
Honda Flux Driver github page: `https://github.com/rlh1994/GENIE_2_9_0/tree/honda_flux_rv5318`
VirtualBox download: `https://www.virtualbox.org`
IceCube website: `http://icecube.wisc.edu`
Honda flux files: `http://www.icrr.u-tokyo.ac.jp/~mhonda/`

A Modifying GENIE to use larger FLUKA files

As talked about in section 3.5 we can generate a FLUKA flux file with a custom flux function and an arbitrary energy range. The problem in trying to use these is that GENIE expects to receive a FLUKA file with 61 energy values per $\cos(\theta)$ range and as such will not work if we just try to give it a file with a higher energy range. Fortunately the code is quite easy to change and only requires one modification as long as you leave the minimum energy and the energy steps the same, only changing the maximum energy value in the file. To pick the maximum energy you must take the base 10 logarithm of the energy, add one more step size to it, then set this value as your upper limit in the first for loop in the file.

If you are currently in the top level of your GENIE directory (not the lamp folder but the actual GENIE_2_9_0 folder) then the file you are going to edit is
src/FluxDrivers/GFlukaAtmo3DFlux.h
so open that up in your editor of choice and then find line 50 which should be
const unsigned int kGFlk3DLogNumEvBins = 61;
which just means that there is a total of 61 bin energies given that are logarithmically spaced. Replace the number 61 with the number of steps your first for loop in *flux_create_cos* remembering that a for loop uses [min, max), i.e. it doesn't use the maximum value. As it currently stands the for loop uses steps of 0.05 so to calculate the number of steps you take you can just use

$$n_{Steps} = (i_{max} - i_{min}) \times 20. \tag{A.1}$$

This equation will work if you have changed the minimum energy value which we will talk about in a second, but will not work if you have changed the step size.

The following steps have not been tested and can't be guaranteed to work so proceed at your own risk.

If you wish to change the minimum energy value to be lower than 100MeV, you will have to follow the same steps above to find the minimum value required in the for loop, except you will just use the base 10 log value as the minimum in included in the loop, compared to the maximum which is not. In the same file as above you will also have to change line 52,
const unsigned int kGFlk3DEvMin = 0.100;
and replace 0.100 with your minimum energy value in GeV then recalculate the number of bins you have and make sure that number has been correctly changed.

Finally if you wish to change the step size for more or less detailed flux data then you will need to change the step size in the for loop, remembering that these are log energy steps. Once you have changed this you will need to change the number of bins because equation (A.1) becomes

$$n_{Steps} = \frac{i_{max} - i_{min}}{step\_size}. \tag{A.2}$$

You will also need to change line 51 which is usually
const unsigned int kGFlk3DLogNumEvBinsPerDecade = 20;
and replace the 20 with 1 divided by the step size.

Once you have made any changes that you require you will need to make GENIE again as before, although this time it will be much quicker as it only has to rebuild the parts you have changed. GENIE should now accept your modified FLUKA file, I recommend you run a short test to make sure that everything is working correctly and you get a correct looking histogram for the output data. Finally, make sure if you want to use a normal FLUKA file after this you change the numbers back and make GENIE once more.