Neutron Detection Efficiency Analysis Software for CLAS12

Keegan Sherman

July 19, 2017

Abstract

The purpose of this document is to show how to get and use the entire Neutron Detection Efficiency (NDE) analysis chain for the CLAS12 detector from Github. Other required software to run the entire chain is the *gemc* CLAS12 simulation software and the *coatjava* reconstruction package. Installation instructions for *gemc* can be found at `https://www.jlab.org/12gev_phys/packages/sources/ceInstall/devel_install.html` and the newest version of *gemc* can be downloaded from `https://github.com/gemc/clas12Tags`. Installation instructions for *coatjava* can be found at `https://github.com/JeffersonLab/clas12-offline-software` and the newest version can be downloaded under the releases tab.

Contents

1 Getting the Software

The event generation software, gemc gcard, and analysis scripts are all stored on Github at the below URL. You can either download a zip file from Github or run the command below to download it.

URL: `https://github.com/Siax135/Neutron_Efficiency`
Command: `git clone https://github.com/Siax135/Neutron_Efficiency.git`

This should put a directory called `Neutron_Efficiency` in which ever directory you unpacked the zip file or ran the command. The table below shows the directory/file structure and gives a small description of each file. The following sections will go into more detail for each file.

Main Directory	Sub-Directories	Files	Description
Neutron_Efficiecny	Documentation	NDE_Analysis.pdf	This document
	Event_Generation	charges.f	Contains particle charge information for event generation.
		inelastic1.f	Front end code that initializes Pythia and sets things up to generate $ep \rightarrow e'\pi^+n$ events.
		pythia-6.4.28.f	Event generator back end distributed and maintained by Fermilab.
		Makefile	Used to make building the event generator easier.
	Histograms	Graph.groovy	Used to track plots using the *GraphErrors* class
		Hist1D.groovy	Used to track 1D histograms using the $H1F$ class
		Hist2D.groovy	Used to track 2D histograms using the $H2F$ class
		HistHandler.groovy	Histogram helper class that is used to initialize and maintain all histograms/plots
	Neutron_Efficiency_Analysis	epiplusSkimmer.groovy	Goes through reconstructed HIPO files and skims out any $ep \rightarrow e'\pi^+n$ events into another file
		ndeHistAdder.groovy	Used to add histograms together for large cluster runs
		ndeHistViewer.groovy	Used to view histograms and plots created by the analysis scripts below
		neutronEfficiencySkimmed.groovy	Performs the NDE analysis on HIPO event files created by `epiplusSkimmer.groovy`
		neutronEfficiencyTBT.groovy	Performs NDE analysis on HIPO event files
	Simulation	NDE.gcard	Settings file for simulating events in *gemc*

2 Event Generation

To measure the NDE of CLAS12 for the neutron magnetic form factor (G_M^n) experiment we will be using a Hydrogen target to produce tagged neutrons from the reaction $ep \rightarrow e'\pi^+ n$. For generating these events we used the Fortran based event generation package Pythia 6.4.28. The reason we used the Fortran based package instead of the newer C++ based package is because we needed the cross-sections for electro-production which have been removed in the C++ version.

To build the event generation software use the following steps:

1. Move to `Event_Generation` directory
   `cd Neutron_Efficiency/Event_Generation`
2. Run `make`

This should produce an executable file called *inelastic1*. Running `./inelastic1 -h` will output a help message that list all available flags and their default values. This information is shown in the table below.

Flag	Description
-o	Output file name (default: out.dat)
-n	Number of events to generate (default: 20)
-n_print	Number of events between print statements (default: 5)
-theta_min	Minimum electron angle in degrees (default: 0)
-theta_max	Maximum electron angle in degrees (default: 90)
-Q2_min	Minimum Q2 value (default: 3.1)
-Q2_max	Maximum Q2 value, negative value here means Q2 max is inactive (default: 18.0)
-pair_pro	Add this flag to allow for diquark-antidiquark production, this flag doesn't take a following argument
-seed	Seed for RNG, allowed values $0 \geq$ seed ≥ 900000000 (default: 19780503)
-v	Add this flag to set the output to be more verbose, this flag doesn't take a following argument

Just running `./inelastic1` will will run the event generator with the default values and put the output in a file called out.dat. All output files are formated in the LUND format. Along with the generated output file, *inelastic1* will also output the total number of events and the total number of 3-particle events (the ones we care about) in the output file. When generating events for CLAS12 simulation purposes we set `-theta_min=4` and `-theta_max=42`. An example command is shown below:

```
inelastic1 -n 10000 -n_print 1000 -theta_min 5 -theta_max 45 -Q2_max 20.0
-o nde.dat -seed 562849321
```

It should also be noted that if *inelastic1* is run on Fedora 25 or higher using the gfortran compiler, then you will probably get the following error:
`Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_DIVIDE_BY_ZERO IEEE_UNDERFLOW_FLAG IEEE_DENORMAL`
This is normal and is a problem with gfortran and Fedora 25 but it does not affect the output of *inelastic1*.

3 Simulation

All simulation was done with *gemc*. Install directions for *gemc* can be found at `https://www.jlab.org/12gev_phys/packages/sources/ceInstall/devel_install.html`. The gcard we used can be found under in the software download from Github in `Neutron_Efficiency/Simulation` and is called NDE.gcard. We ran *gemc* as follows:

```
gemc -gcard=NDE.gcard -INPUT_GEN_FILE="LUND,<input_file.dat>" -N=<number_of_events>
-USE_GUI=0
```

This produces the file NDE.evio.

4 Convert to HIPO and Reconstruct

To convert to HIPO and run the reconstruction you will need the coatjava package which can be found here `https://github.com/JeffersonLab/clas12-offline-software/releases`. We used version 4a.3.0 for the following conversion, reconstruction, and analysis. When converting to HIPO be sure to use the `evio2hipo` command as follows:

```
evio2hipo -r 11 -t -1.0 -s 1.0 -o <output_file.hipo> <input_file.evio>
```

The `-t` sets the strength percentage for the torus field and the `-s` does the same thing for the solenoid field. If these options are changed then you will either get particles not bending the right way in the magnetic fields or not bending enough.

For reconstructing the data we used the `notsouseful-util` tool that is also part of the coatjava package. We ran the command as follows:

```
notsouseful-util -i <input_file.hipo> -o <output_file.hipo> $SERVICES
```

Where `$SERVICES` is defined in our `.cshrc` file as:

```
setenv SERVICES 'org.jlab.service.dc.DCHBEngine
org.jlab.service.htcc.HTCCReconstructionService org.jlab.service.ftof.FTOFEngine
org.jlab.service.ltcc.LTCCEngine org.jlab.service.ec.ECEngine
org.jlab.service.eb.EBEngine org.jlab.service.dc.DCTBEngine
org.jlab.service.eb.EBEngine'
```

5 Groovy Analysis

For the analysis we developed the groovy script `neutronEfficiencyTBT.groovy` located in `Neutron_Efficiency/Neutron_Efficiency_Analysis`. This script uses the time based tracking reconstruction banks to determine the number of reconstructed particles. If there are any more or less than two reconstructed particles, the event is skipped. However, if there are exactly two reconstructed particles, then the script will calculate the square of the missing mass for the two particles. If the missing mass squared is more than 1.3 $(GeV/c^2)^2$, then we skip the event and move on to the next one. Otherwise, the script then calculates the missing momentum for the neutron and test

to see if the neutron's path intersects with the fiducial volume of the Electromagnetic Calorimeter (EC). If it does, then the neutron is counted as reconstructed; otherwise, the event is skipped. Once we know the path of the neutron intersects with the fiducial volume of the EC we search through the ECAL::Clusters bank to see if we find any hits within 1.5° of the predicted intersection point. If we do find a hit, then the neutron is counted as found. The NDE is finally calculated as the ratio between the number of reconstructed and found neutrons from each event. It should also be noted that the missing mass cut at 1.3 $(GeV/c^2)^2$ is a little high. This is because it also calculates the NDE at missing mass cuts of 1.2, 1.1, 1.0, 0.98, 0.95, 0.92, and 0.90 $(GeV/c^2)^2$ so that we can see how changing the cut effects the NDE. To run the script set the `JYPATH` environment variable as shown below and then run the following command in the directory containing the script:

```
setenv JYPATH <PATH_TO_GIT_PACKAGE>/Neutron_Efficiency/Histograms
run-groovy neutronEfficiencyTBT.groovy <PATH_TO_INPUT_FILE/input_file.hipo>
```

Once the script is done it will print out the average NDE across all momentum with error and it produces a histogram file called `nde_histograms.hipo`. To view all of the histograms run the `NDEHistViewer.groovy` script like follows:

```
run-groovy NDEHistViewer.groovy nde_histograms.hipo
```

The viewer can also show specific histograms or plots by providing any combination of the flags in the following table. The list of flags can also be shown by the viewer by running `run-groovy ndeHistViewer.groovy -h`.

Flag	Description
-theta	Show theta plots
-P	Show momentum plots
-MM	Show missing mass plots
-accep	Show acceptance plots
-NDE	Show NDE plots
-EC	Show EC hit plots

6 Extras

There are a few files included in the Github package that are included because either they are convenient or because were designed generally so that they may be used in other analysis projects.

The files `epiplusSkimmer.groovy`, `neutronEfficiencySkimmed.groovy`, and `ndeHistAdder.groovy` are included for convenience. The `epiplusSkimmer.groovy` script will go through a HIPO event file, find all of the $ep \to e'\pi^+n$ events and put them in a new HIPO file. To run the script use the below command:

```
run-groovy epiplusSkimmer.groovy -i <input_File.hipo> -o <output_File.hipo>
```

There is also a `-a` flag that can be used with the skimmer which will make it run the analysis on the skimmed file after it is done. The `neutronEfficiencySkimmed.groovy` script preforms the same NDE analysis as `neutronEfficiencyTBT.groovy` except it expects an already skimmed file as input.

Because it expects a skimmed file, `neutronEfficiencySkimmed.groovy` doesn't check for a time based tracking bank and doesn't have the same 2 particle cut that `neutronEfficiencyTBT.groovy` does. Thus if a non-skimmed file is given as input, the script may crash or produce poor results.

Finally, `ndeHistAdder.groovy` can be used to add all of the histograms together and perform the NDE calculation for the error bar plots. The general command to use it is as follows:

```
run-groovy ndeHistAdder.groovy <summed_files_base_name> <number_of_files_to_sum>
```

Now for a little more explanation. All files that are to be summed need to have the same base name followed by a four digit number designating the number of the file (starting at 0000 and increasing consecutively) and they should all have the `.hipo` extension. For example, suppose I have four files that I want to sum. Then I can name them as `histograms_0000.hipo`, `histograms_0001.hipo`, `histograms_0002.hipo`, and `histograms_0003.hipo`. Now to sum them, run the following command in the directory containing the four files.

```
run-groovy ndeHistAdder.groovy histograms_ 4
```

This tells the adder to look for HIPO files starting with 'histograms_' and it should sum the first 4 of them. In general, it will store the summed histograms in `<summed_files_base_name>_Total.hipo` so in this case they would be in `histograms__Total.hipo`

Everything in the `Histograms` sub-directory is written to be general and can be used in other analysis projects to maintain histograms and plots. To use them in other analyses, make sure to set the JYPATH environment variable to point to the `Histograms` directory and make the following changes. In the file `HistHandler.groovy`, the constant `DIR_NAME` should be set to a string describing the analysis (in this case we set it to `neutrons`). This is the only thing that should be change in `HistHandler.groovy`. The remaining changes need to be made in `Hist1D.groovy`, `Hist2D.groovy`, and `Graph.groovy`. Any histograms or plots required should be listed in one of these three files as follows:

1. `Hist1D.groovy`
   This file is used to list all 1-dimensional histograms using the $H1F$ class in the *coatjava* package. To add a new histogram simply add a line with the below syntax.

   ```
   <histogram_name> (<number_of_bins>, <x_minimum>, <x_maximum>, <x_axis_title>,
   <line_color>, <histogram_title>, <y_axis_title>)
   ```

   The number of bins and x minimum and maximum are required arguments but everything else is optional. That being said the order of the arguments does matter so you can't set a number of bins, x minimum and maximum, and a line color without also setting an x-axis title. To list multiple histograms make sure each entry is separated by a comma and that the list is terminated by a semi-colon.

2. `Hist2D.groovy`
   This file is used to list all 2-dimensional histograms using the $H2F$ class in the *coatjava* package. To add a new histogram simply add a line with the below syntax.

   ```
   <histogram_name> (<number_of_x_bins>, <x_minimum>, <x_maximum>, <number_of_y_bins>,
   ```

<y_minimum>, <y_maximum>, <x_axis_title>, <y_axis_title>, <histogram_title>)

The number of x and y bins and the x and y minimum and maximum are required arguments but everything else is optional. As above, the order of the arguments matters. To list multiple histograms make sure each entry is separated by a comma and that the list is terminated by a semi-colon.

3. Graph.groovy
   This file is used to list all point plots with error bars using the *GraphErrors* class in the *coatjava* package. To add a new graph simply add a line with the below syntax.

   <graph_name> (<marker_color>, <x_axis_title>, <y_axis_title>, <graph_title>)

   Graph entries don't have any required arguments but if arguments are given the order of the arguments matters as above. To list multiple graphs make sure each entry is separated by a comma and that the list is terminated by a semi-colon.

As for the color options, the following colors are available to choose from: black, red, green, blue, yellow, magenta, light blue, purple, and dark green.

Once all of the histograms and graphs have been added to the appropriate files, add the following lines to the main analysis script to setup up the histograms and get handles to access them.

```
import HistHandler;
HistHandler handler = new HistHandler();
TDirectory histFile = handler.initializeHist();
H1F[] histograms1D = handler.get1DHist();
H2F[] histograms2D = handler.get2DHist();
GraphErrors[] graphs = handler.getGraphs();
```

From here its easy to access a histogram or graph to add data. For example, say I had the following histogram in my Hist1D.groovy file:

```
hexampleHist (100, 0, 10, "example x title", "light blue", "Example Histogram", "example
y title")
```

Then to add data to it just do the following:

```
histograms1D[Hist1D.hexampleHist.ordinal()].fill(<data>)
```

By doing this the code is still self documenting since it is easy to see what histogram is being accessed but all of the creation and maintenance is abstracted away. Finally, to store the histograms in an output file simply put histFile.writeFile("<output_file_name.hipo") at the end of the main analysis script.