Using EIC event generators and conducting fast detector simulation with *eic-smear*

TABLE OF CONTENTS

Introduction

This document will show you how to run the Pythia6, Pythia8, Djangoh, and BeAGLE event generators for EIC collisions and how to analyze the produced generator-level results. The generator output can then be passed into either fast detector simulation or into full detector simulation. This document gives information on how to use the *eic-smear* fast detector simulation package. In the near future, it will also give information on how to run full detector simulations. All this work can be done within the standard EIC software environment.

You can run the simulations using either the SDCC (BNL) computer cluster, or, if you do not have a SDCC computer account, by using a Singularity container. (N.B. Creating a SDCC account takes some time and is not needed for most studies. So, the Singularity container is the best way to start.)

Once you have the Singularity container installed (see Accessing the Singularity container section) or have access to your SDCC account (see Getting a SDCC (BNL) Account section), then you need to get the files simulation files that will be used throughout this document. Clone this GitHub repository:

https://github.com/cipriangal/eicGenTutorials

Accessing the Singularity Container

The EIC singularity container can be installed from here:

https://github.com/ECCE-EIC/Singularity

This works best on Linux. (“Option-1” is preferable here, as “option-2” requires the user to download a deprecated version of the container and manually install LHAPDF data sets.) If you have MacOS or Windows, it is better to use the Virtual Box method described here:

https://github.com/eic/Singularity/blob/master/VirtualBox.md

If you are using the Virtual Box, you may need to copy an updated version of the *singularity_shell.sh* script as described in the link above. (Note that you will not need to use the *setup.sh* script that comes with the Virtual Box installation.)

After you have the Singularity container or Virtual Box correctly installed and a terminal open, follow the instructions here to enter the container and setup the EIC environment:

https://github.com/cipriangal/eicGenTutorials#eicgentutorials

Getting a SDCC (BNL) Account

If your work requires access to SDCC (BNL) computer resources, you need to first get a SDCC account. For creating a SDCC (RACF) computer account, this website has the most up-to-date information.

Once you have a SDCC account, take a look here and here to get started using the account. Then you will need to set up the remote environment (see here). Depending on which specific type of account you have (*eic* or *sphenix*, for example), the machines that you will work on and the directories where you will save your data may differ. If you have an *eic* account, for example, look at this page for information on the *eic* file system.

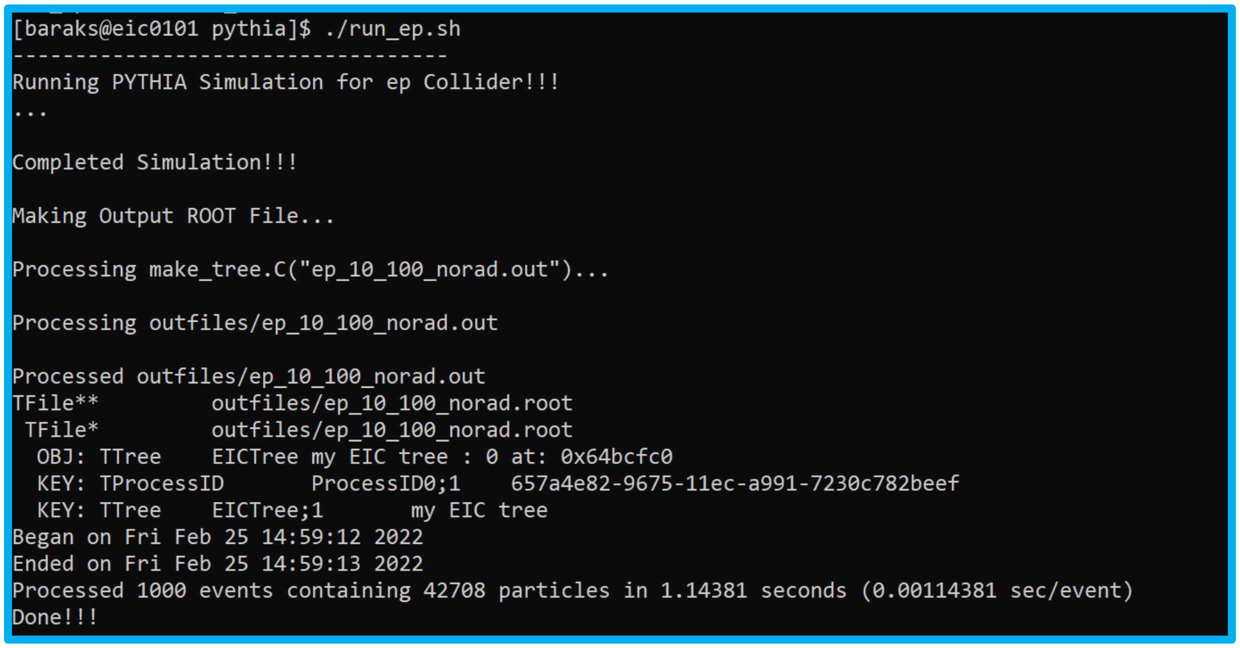
After you complete all this, you can then run all simulations described in this document. In addition, you can run Batch farm jobs.

Running *Pythia6*

In order to run the Pythia6 simulation for a 10 GeV electron scattering off a 100 GeV proton, do the following starting at the top level of our Github repository

```
> cd pythia
> ./run_ep.sh
```

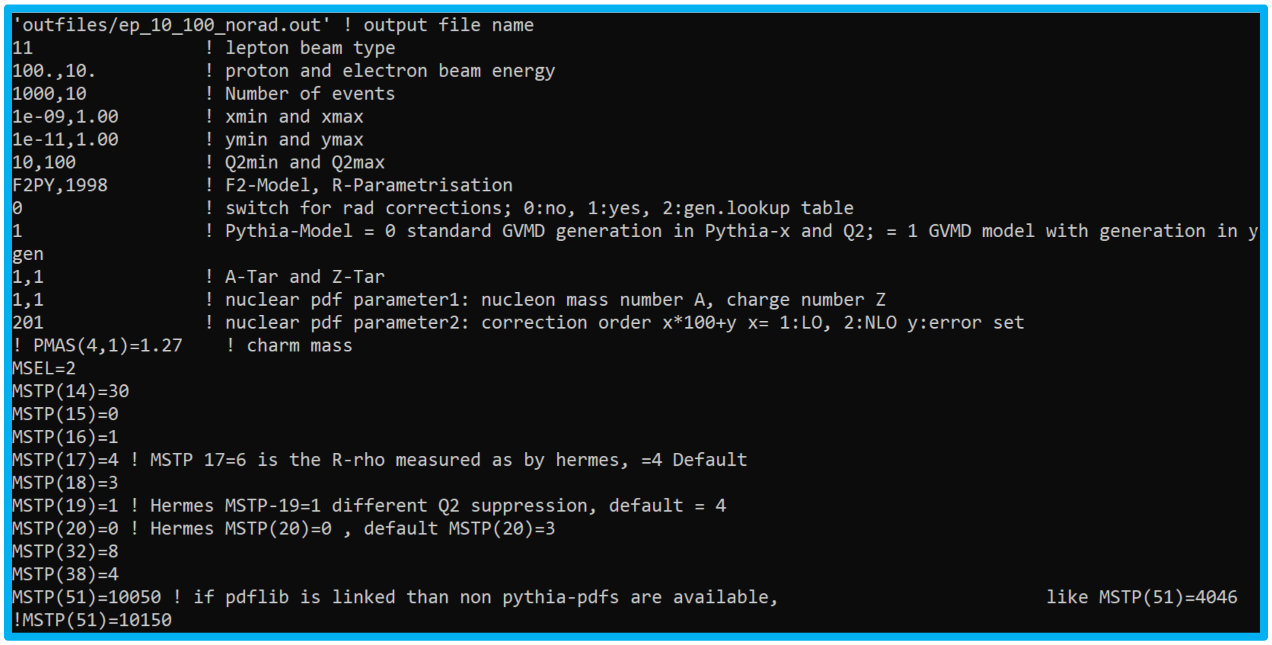
The '>' represents the terminal prompt and should not be typed. If you run the simulation successfully, you should see the following output on the screen (number of particles, etc. may be different):



```
[baraks@eic0101 pythia]$ ./run_ep.sh
-----------------------------------
Running PYTHIA Simulation for ep Collider!!!
...

Completed Simulation!!!

Making Output ROOT File...

Processing make_tree.C("ep_10_100_norad.out")...

Processing outfiles/ep_10_100_norad.out

Processed outfiles/ep_10_100_norad.out
TFile**         outfiles/ep_10_100_norad.root
 TFile*         outfiles/ep_10_100_norad.root
  OBJ: TTree    EICTree my EIC tree : 0 at: 0x64bcfc0
  KEY: TProcessID       ProcessID0;1    657a4e82-9675-11ec-a991-7230c782beef
  KEY: TTree    EICTree;1       my EIC tree
Began on Fri Feb 25 14:59:12 2022
Ended on Fri Feb 25 14:59:13 2022
Processed 1000 events containing 42708 particles in 1.14381 seconds (0.00114381 sec/event)
Done!!!
```

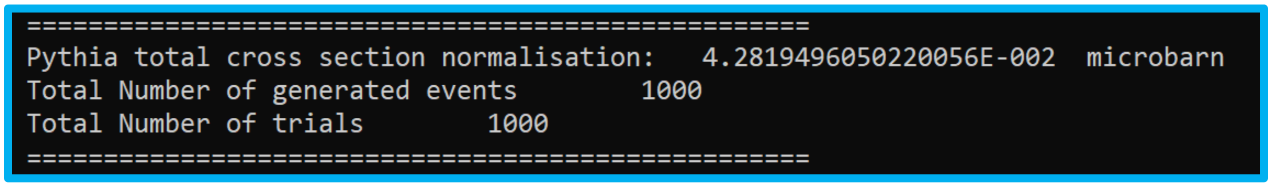
What did we do by running the above script? We took a *Pythia6* card called *input.data_noradcor.eic* and used it as an input to the simulation. This created three output files – an output text file called *ep_10_100_norad.out* in the outfiles subdirectory, an output ROOT file called *ep_10_100_norad.root* in the outfiles subdirectory, and a log file called *ep_10_100_norad.log* in the logfiles subdirectory. The names of the files indicates that the simulation is for a 10 GeV electron scattering off a 100 GeV proton, with no QED Radiative effects included. In addition, note that, for electron-proton scattering, *Pythia6* only simulates unpolarized neutral-current Deep Inelastic Scattering (DIS).

Let's look at each of these four files. Part of the input card is shown below. There are many parameters to tune, but usually all you'll need to modify are the beam energies, the number of events to simulate, and the kinematic limits.



```
'outfiles/ep_10_100_norad.out' ! output file name
11                 ! lepton beam type
100.,10.           ! proton and electron beam energy
1000,10            ! Number of events
1e-09,1.00         ! xmin and xmax
1e-11,1.00         ! ymin and ymax
10,100             ! Q2min and Q2max
F2PY,1998          ! F2-Model, R-Parametrisation
0                  ! switch for rad corrections; 0:no, 1:yes, 2:gen.lookup table
1                  ! Pythia-Model = 0 standard GVMD generation in Pythia-x and Q2; = 1 GVMD model with generation in y
gen
1,1                ! A-Tar and Z-Tar
1,1                ! nuclear pdf parameter1: nucleon mass number A, charge number Z
201                ! nuclear pdf parameter2: correction order x*100+y x= 1:LO, 2:NLO y:error set
! PMAS(4,1)=1.27     ! charm mass
MSEL=2
MSTP(14)=30
MSTP(15)=0
MSTP(16)=1
MSTP(17)=4 ! MSTP 17=6 is the R-rho measured as by hermes, =4 Default
MSTP(18)=3
MSTP(19)=1 ! Hermes MSTP-19=1 different Q2 suppression, default = 4
MSTP(20)=0 ! Hermes MSTP(20)=0 , default MSTP(20)=3
MSTP(32)=8
MSTP(38)=4
MSTP(51)=10050 ! if pdflib is linked than non pythia-pdfs are available,                         like MSTP(51)=4046
!MSTP(51)=10150
```

Next, consider the output log file. This will tell you various information about the setting used in the simulation, what input PDF parameterization was used, and how many events were successfully run. Most importantly, as shown below, the log file will contain the total cross section for the scattering process (within the kinematic limits). Dividing the number of events generated by the total cross section will give you the total simulated luminosity.



```
===================================================
Pythia total cross section normalisation:   4.2819496050220056E-002  microbarn
Total Number of generated events        1000
Total Number of trials        1000
===================================================
```

The two files in the outfiles subdirectory give the event listing – that is, for each event, it gives you the all particles that are generated and their momenta and creation point, as well as event-level information such as the physics process. Both files contain the exact same information – it is just that one is a text file, and the other is a ROOT file. The text file is the direct output of the *Pythia6* simulation. Information on the meaning of each variable in the text file can be found here.

The ROOT file is created in our running script above (*run_ep.sh*) by using the *make_tree.C* code. This created ROOT file contains a TTree called EICTree, the structure of which is described in detail here. (N.B. This *EICTree* ROOT file structure is defined in the *eic-smear* package, but is separate from any of the fast smearing implemented in that package.) We will perform several analyses using this ROOT file in the Analyzing Generated output — examples section below.

Running *DJANGOH*

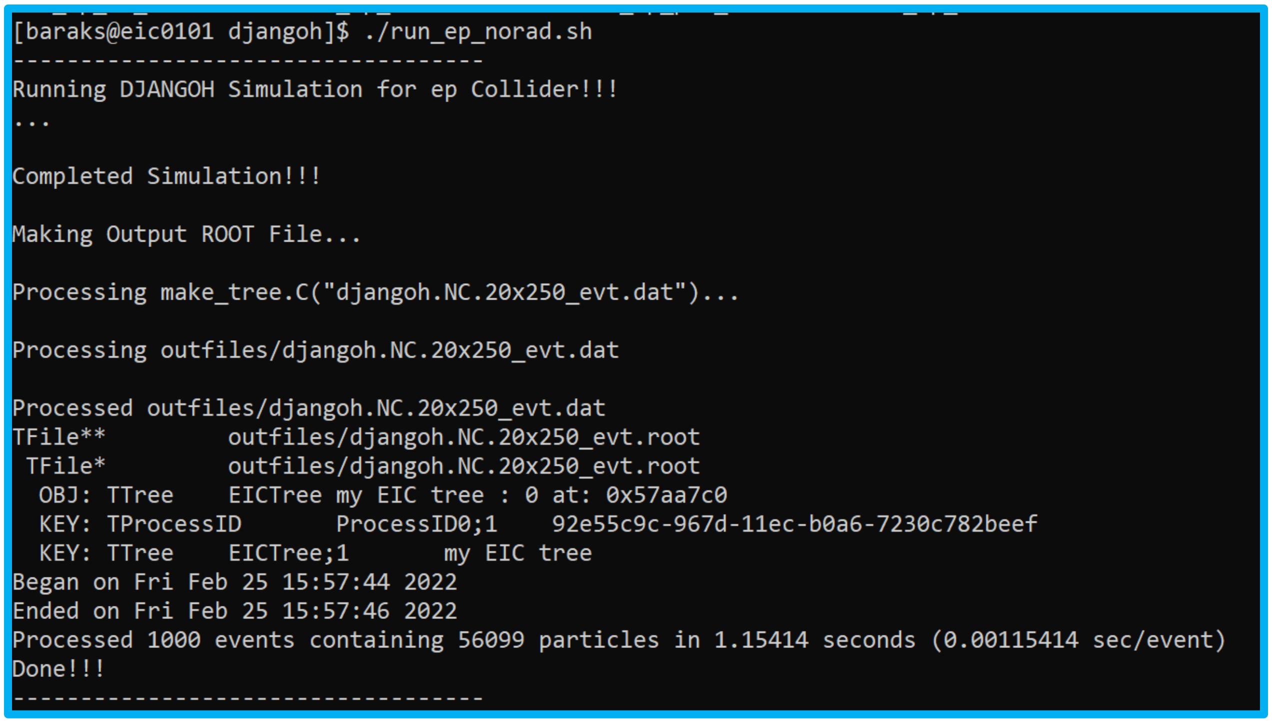
The DJANGOH event generator also lets us simulate electron-proton collisions as well, with a few unique capabilities. If you start in the top level of our Github repository and go into the djangoh subdirectory, you will find four different running scripts:

1. run_ep_norad.sh – Neutral current unpolarized electron-proton DIS with no QED radiative effects
2. run_ep_rad.sh – Neutral current unpolarized electron-proton DIS including QED radiative effects
3. run_ep_cc_norad.sh – Charge current unpolarized electron-proton DIS with no QED radiative effects
4. run_ep_pol_norad.sh – Neutral current electron-proton DIS with no QED radiative effects but including longitudinal polarization for both the electron and proton beams

To run the first simulation above, make sure you are in the djangoh subdirectory. Then simply do

```
> ./run_ep_norad.sh
```

If you run the simulation successfully, you should see the following output on the screen:



```
[baraks@eic0101 djangoh]$ ./run_ep_norad.sh
-----------------------------------
Running DJANGOH Simulation for ep Collider!!!
...

Completed Simulation!!!

Making Output ROOT File...

Processing make_tree.C("djangoh.NC.20x250_evt.dat")...

Processing outfiles/djangoh.NC.20x250_evt.dat

Processed outfiles/djangoh.NC.20x250_evt.dat
TFile**         outfiles/djangoh.NC.20x250_evt.root
 TFile*         outfiles/djangoh.NC.20x250_evt.root
  OBJ: TTree    EICTree my EIC tree : 0 at: 0x57aa7c0
  KEY: TProcessID       ProcessID0;1    92e55c9c-967d-11ec-b0a6-7230c782beef
  KEY: TTree    EICTree;1       my EIC tree
Began on Fri Feb 25 15:57:44 2022
Ended on Fri Feb 25 15:57:46 2022
Processed 1000 events containing 56099 particles in 1.15414 seconds (0.00115414 sec/event)
Done!!!
-----------------------------------
```

Similar to the *Pythia6* simulation, this creates both output files and a logfile. In particular, the output ROOT file *djangoh.NC.20x250_evt.root* located in the outfiles subdirectory. This ROOT file has the same structure as the one created by the Pythia6 simulation and can be analyzed the same way.

Running *BeAGLE*

The *BeAGLE* event generator allows us to simulate electron-nucleus scattering, including certain nuclear effects such as nPDFs, nuclear geometry, final-state cold nuclear effects, and the evaporation of the residual nucleus. Note that this generator continues to undergo significant development.

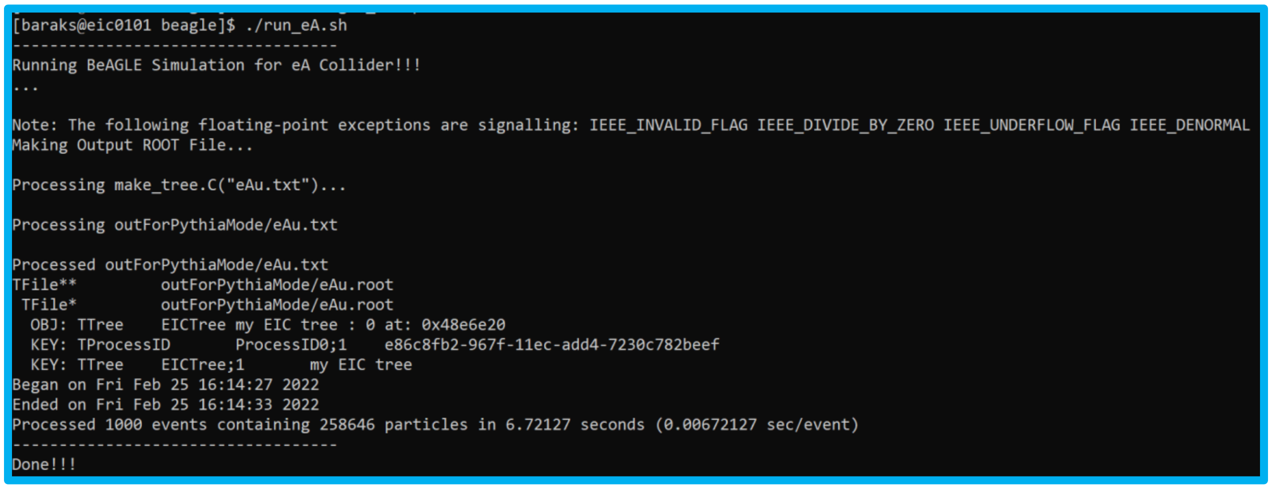
If you want to run the *BeAGLE* simulation package, start in the top level of our Github repository and do the following:

```
> cd beagle
> source beagle_setup.sh (or beagle_setup.csh if on C shell)
```

This sets additional environmental variables needed to run *BeAGLE*. Then run the simulation:

```
> ./run_eA.sh
```

If you run *BeAGLE* correctly, you should see the following output on the screen:



```
[baraks@eic0101 beagle]$ ./run_eA.sh
-----------------------------------
Running BeAGLE Simulation for eA Collider!!!
...

Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_DIVIDE_BY_ZERO IEEE_UNDERFLOW_FLAG IEEE_DENORMAL
Making Output ROOT File...

Processing make_tree.C("eAu.txt")...

Processing outForPythiaMode/eAu.txt

Processed outForPythiaMode/eAu.txt
TFile**         outForPythiaMode/eAu.root
 TFile*         outForPythiaMode/eAu.root
  OBJ: TTree    EICTree my EIC tree : 0 at: 0x48e6e20
  KEY: TProcessID       ProcessID0;1    e86c8fb2-967f-11ec-add4-7230c782beef
  KEY: TTree    EICTree;1       my EIC tree
Began on Fri Feb 25 16:14:27 2022
Ended on Fri Feb 25 16:14:33 2022
Processed 1000 events containing 258646 particles in 6.72127 seconds (0.00672127 sec/event)
-----------------------------------
Done!!!
```

(I confirmed with the *BeAGLE* experts that exception that prints to the screen can be ignored.) Again, the created files are similar to the simulations that we ran previously. Most importantly, there is a ROOT file called *eAu.root* in the outForPythiaMode subdirectory that has the same *EICTree* structure.

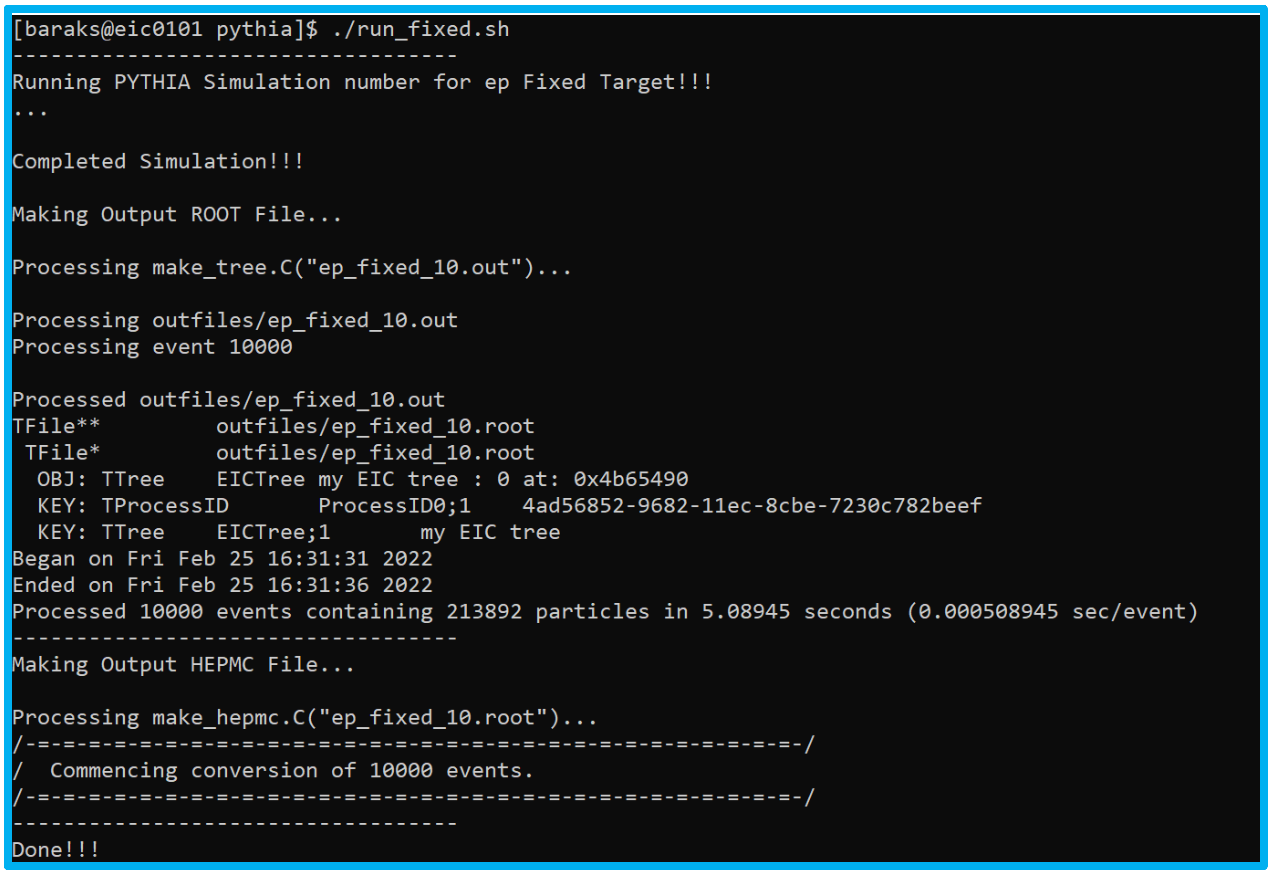
Converting EICTree ROOT files to the HepMC3 format

The ROOT files that we have created using the above three event generators are very useful for conducting generator-level analysis, as we'll see in the Analyzing Generated output — examples section below. In addition, these ROOT files can be directly used as input for the fast smearing package *eic-smear*. However, many of our current detector simulation packages want the generated simulation files to be in the *HepMC* format. Thankfully, there is a converter that allows us to convert the EICTree ROOT files into the HepMC3 format (or HepMC2, if desired).

We will demonstrate this using the *Pythia6* generator. Go back into the pythia subdirectory in our Github repository and run the included fixed-target electron-proton scattering simulation:

```
> ./run_fixed.sh
```

You should see the following printed to the screen:



```
[baraks@eic0101 pythia]$ ./run_fixed.sh
-----------------------------------
Running PYTHIA Simulation number for ep Fixed Target!!!
...

Completed Simulation!!!

Making Output ROOT File...

Processing make_tree.C("ep_fixed_10.out")...

Processing outfiles/ep_fixed_10.out
Processing event 10000

Processed outfiles/ep_fixed_10.out
TFile**         outfiles/ep_fixed_10.root
 TFile*         outfiles/ep_fixed_10.root
  OBJ: TTree    EICTree my EIC tree : 0 at: 0x4b65490
  KEY: TProcessID       ProcessID0;1    4ad56852-9682-11ec-8cbe-7230c782beef
  KEY: TTree    EICTree;1       my EIC tree
Began on Fri Feb 25 16:31:31 2022
Ended on Fri Feb 25 16:31:36 2022
Processed 10000 events containing 213892 particles in 5.08945 seconds (0.000508945 sec/event)
-----------------------------------
Making Output HEPMC File...

Processing make_hepmc.C("ep_fixed_10.root")...
/-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-/
/  Commencing conversion of 10000 events.
/-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-/
-----------------------------------
Done!!!
```

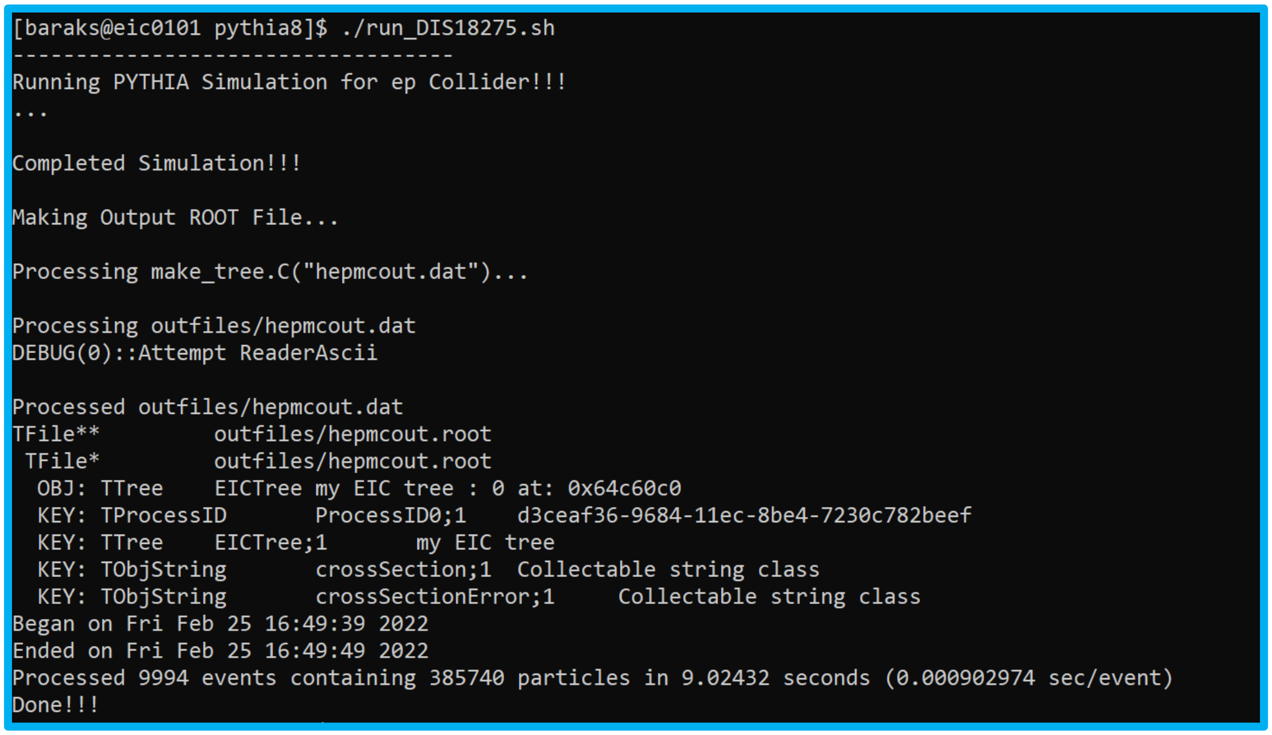
As can be seen, compared to our previous *Pythia6* simulation, there is one additional step. Here a HepMC file (*ep_fixed_10.hepmc*) is also created in the outfiles subdirectory.

Running *Pythia8*

We can also simulate electron-proton collisions using the *Pythia8* event generator. In order to run the simulation, start at the top level of our Github repository and then do the following:

```
> cd pythia8
> make
> ./run_DIS18275.sh
```

If you run the simulation correctly, you should see the following output printed to the screen:



```
[baraks@eic0101 pythia8]$ ./run_DIS18275.sh
----------------------------------
Running PYTHIA Simulation for ep Collider!!!
...

Completed Simulation!!!

Making Output ROOT File...

Processing make_tree.C("hepmcout.dat")...

Processing outfiles/hepmcout.dat
DEBUG(0)::Attempt ReaderAscii

Processed outfiles/hepmcout.dat
TFile**         outfiles/hepmcout.root
 TFile*         outfiles/hepmcout.root
  OBJ: TTree    EICTree my EIC tree : 0 at: 0x64c60c0
  KEY: TProcessID       ProcessID0;1    d3ceaf36-9684-11ec-8be4-7230c782beef
  KEY: TTree    EICTree;1       my EIC tree
  KEY: TObjString       crossSection;1  Collectable string class
  KEY: TObjString       crossSectionError;1     Collectable string class
Began on Fri Feb 25 16:49:39 2022
Ended on Fri Feb 25 16:49:49 2022
Processed 9994 events containing 385740 particles in 9.02432 seconds (0.000902974 sec/event)
Done!!!
```

This *Pythia8* simulation will directly generate a HepMC3 file in the outfiles subdirectory. It will also create a log file in the logfiles subdirectory, which contains information such as the total cross section. As the HepMC file is a bit difficult to use for generator-level studies, we run the converter mentioned above ‘in reverse’ and create a ROOT file with the same *EICTree* structure as the other generators.

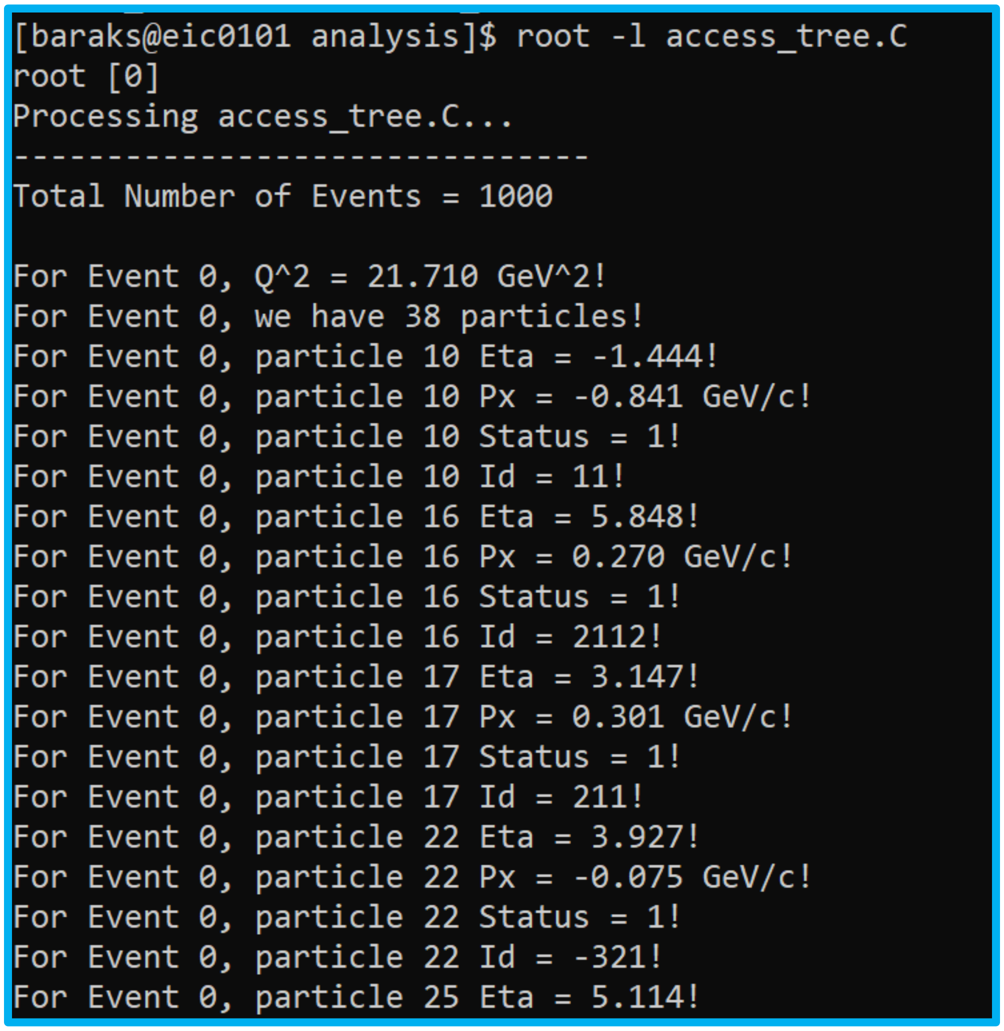
Analyzing Generated output — examples

The output from these four event generators is usually passed into either full or fast detector simulations. However, we can do some interesting analysis using just the generator-level files themselves. We’ll use the first *Pythia6* simulation that we ran for these examples.

Let’s start by going into the analysis subdirectory. Then run the following script to access the *EICTree* structure of the generated ROOT file and print some information to the screen:

```
> root -l access_tree.C
```

You should see the following information (actual particles and numbers will be different) print to the screen:



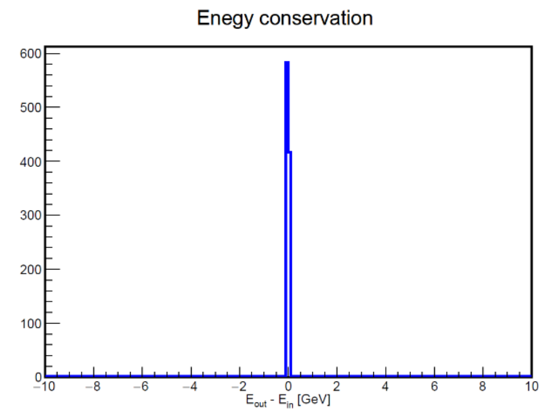
```
[baraks@eic0101 analysis]$ root -l access_tree.C
root [0]
Processing access_tree.C...
-------------------------------
Total Number of Events = 1000

For Event 0, Q^2 = 21.710 GeV^2!
For Event 0, we have 38 particles!
For Event 0, particle 10 Eta = -1.444!
For Event 0, particle 10 Px = -0.841 GeV/c!
For Event 0, particle 10 Status = 1!
For Event 0, particle 10 Id = 11!
For Event 0, particle 16 Eta = 5.848!
For Event 0, particle 16 Px = 0.270 GeV/c!
For Event 0, particle 16 Status = 1!
For Event 0, particle 16 Id = 2112!
For Event 0, particle 17 Eta = 3.147!
For Event 0, particle 17 Px = 0.301 GeV/c!
For Event 0, particle 17 Status = 1!
For Event 0, particle 17 Id = 211!
For Event 0, particle 22 Eta = 3.927!
For Event 0, particle 22 Px = -0.075 GeV/c!
For Event 0, particle 22 Status = 1!
For Event 0, particle 22 Id = -321!
For Event 0, particle 25 Eta = 5.114!
```

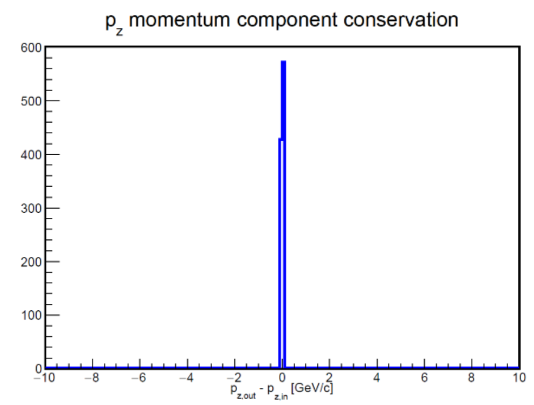
Open the above code in a text editor and look through it. See if you can understand how each piece of information is being accessed what each means.

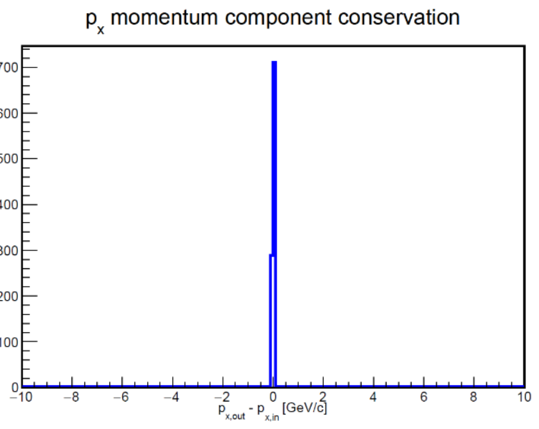
One fundamental property of the physics we are studying is the conservation of energy and momentum. We can test whether the generator respects this by comparing the total energy and momentum of the beam with the total energy and momentum of the produced final-state particles. Run this code:

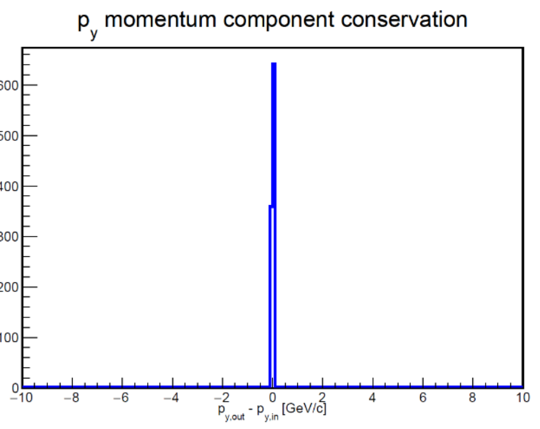
```
> root -l -b -q conservation_check.C
```

This will create an output pdf file in the plots subdirectory. (Note that I run with the '-b' option, which means the plots won't show on the screen. This makes the running quicker.) If we look at the pdf file, we have the following plots which show the difference between the incoming and outgoing energy or component of momentum:







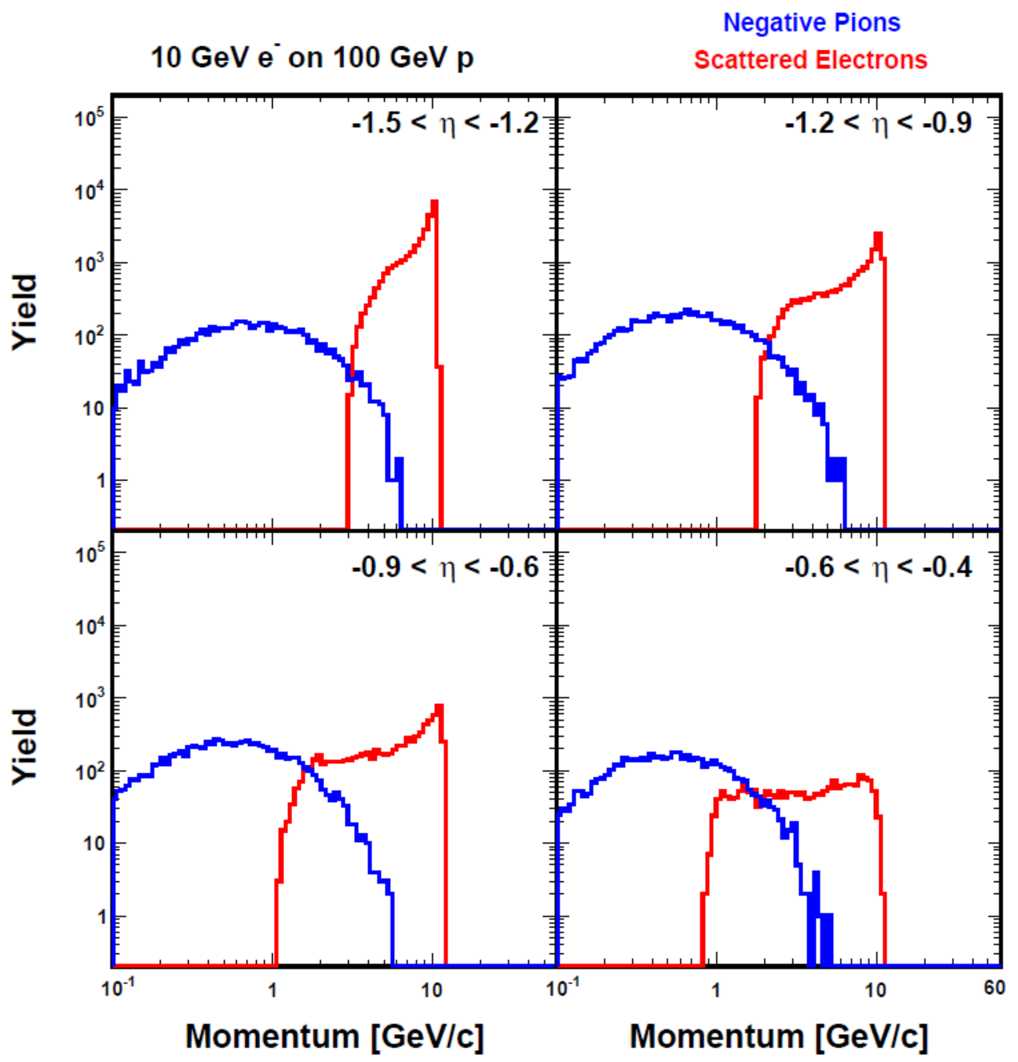


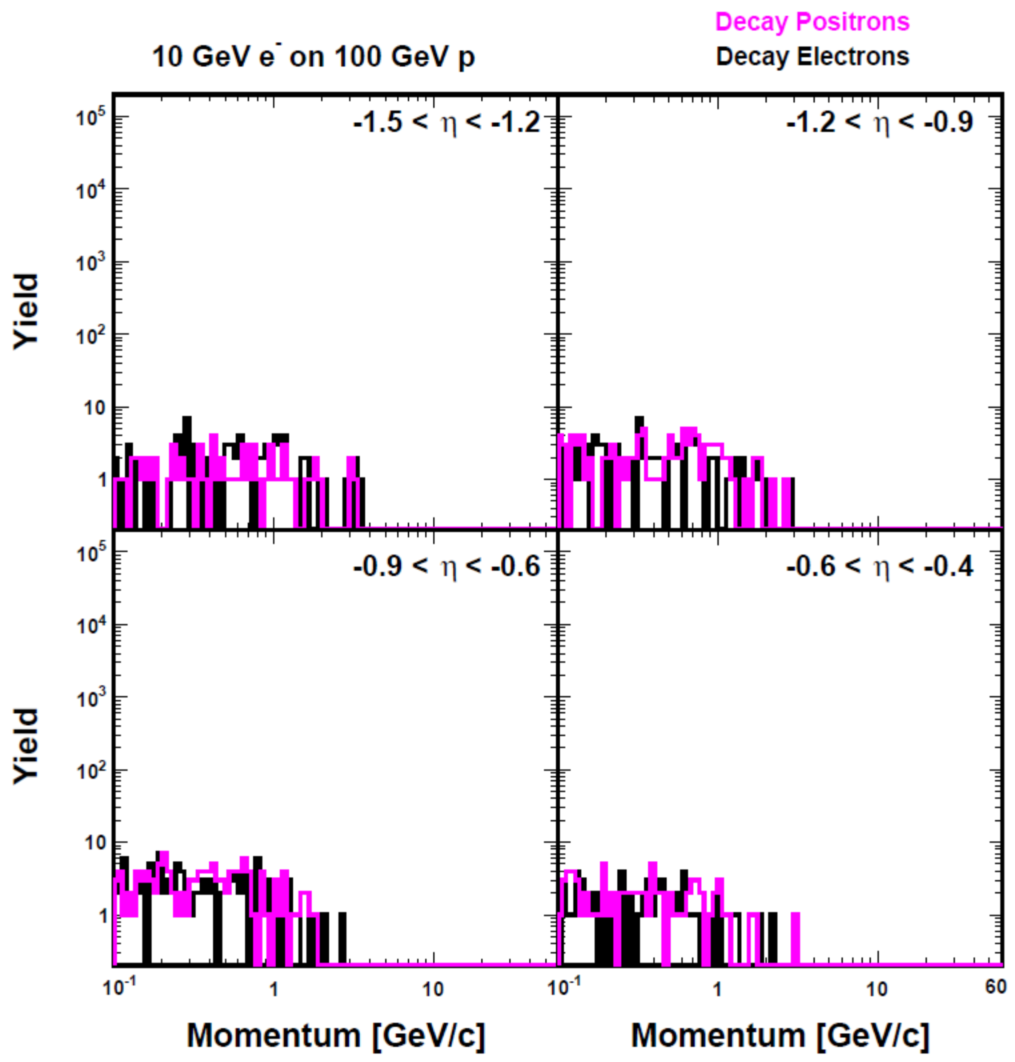
We see that the generator satisfies the expected conservation requirements. Once again, take a look at the code and make sure you understand how all the relevant information is being acessed and used.

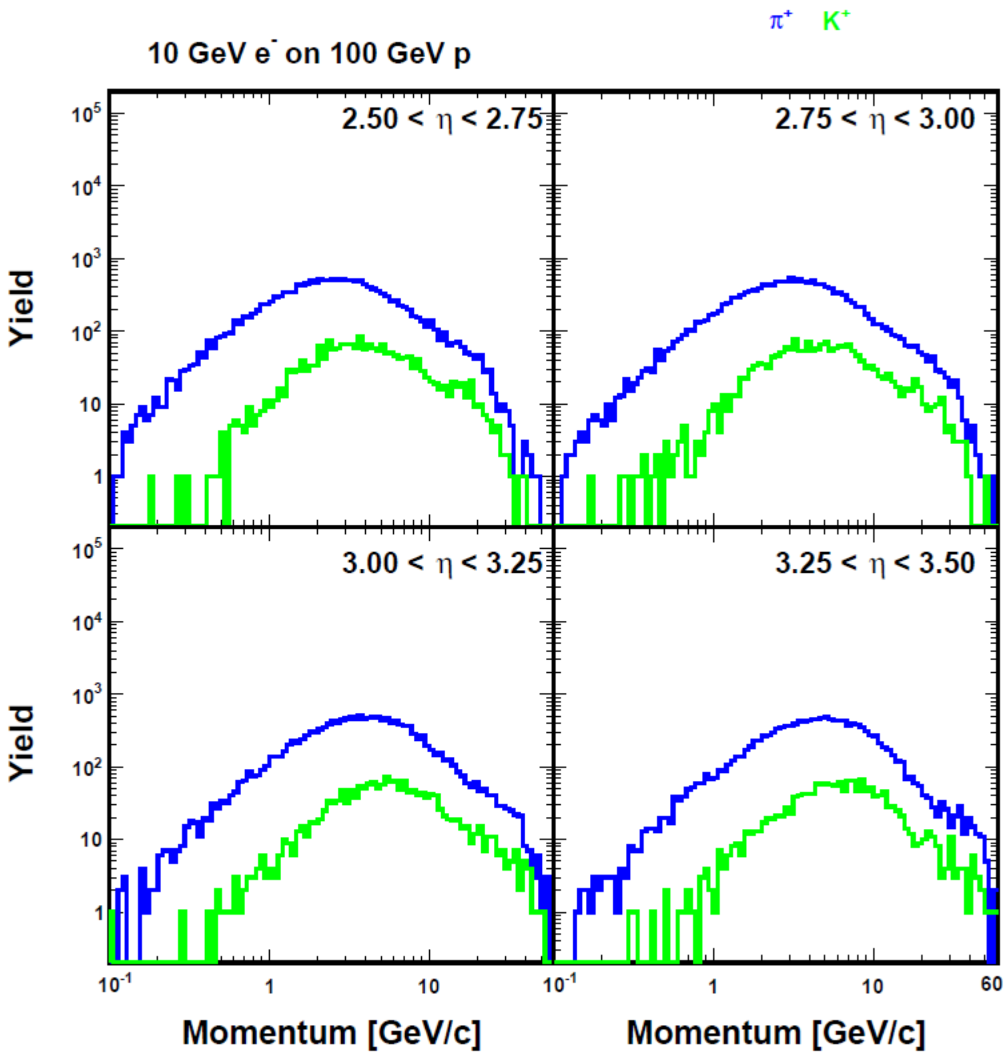
Next, we will take a more complex example where we plot the momentum spectra of various particles in different angular bins. For this example to have reasonable statistics, I had to go back to the *Pythia6* event generator and run 100,000 events. You may even want to run 1million events, but this will probably take ~1hr on a local computer. In the analysis subdirectory, do the following:

```
> cd compiled
> make
> ./particle_spectra -l -b -q
```

This will once again create a set of plots saved in a pdf file in the plots subdirectory. You should see the following set of plots:


Negative Pions
Scattered Electrons
10 GeV e$^-$ on 100 GeV p
$-1.5 < \eta < -1.2$
$-1.2 < \eta < -0.9$
$-0.9 < \eta < -0.6$
$-0.6 < \eta < -0.4$
Yield
Yield
Momentum [GeV/c]
Momentum [GeV/c]
10^5
10^4
10^3
10^2
10
1
10^{-1}
1
10
60


10 GeV e$^-$ on 100 GeV p
Decay Positrons
Decay Electrons
$-1.5 < \eta < -1.2$
$-1.2 < \eta < -0.9$
$-0.9 < \eta < -0.6$
$-0.6 < \eta < -0.4$
Yield
Momentum [GeV/c]
10^{-1}
1
10
60
10^5
10^4
10^3
10^2

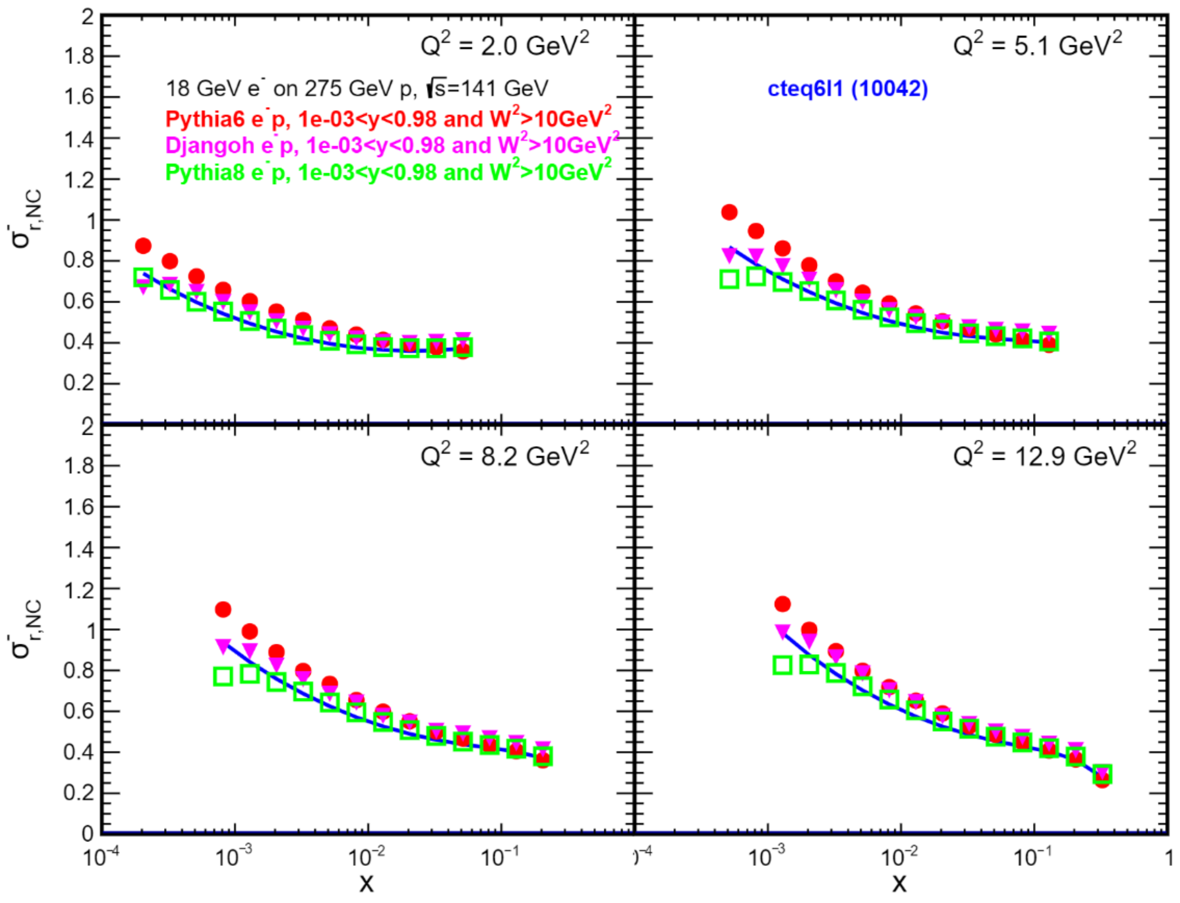


Here you see the momentum spectra of various particles in different angular bins. Think about what these plots tell us.

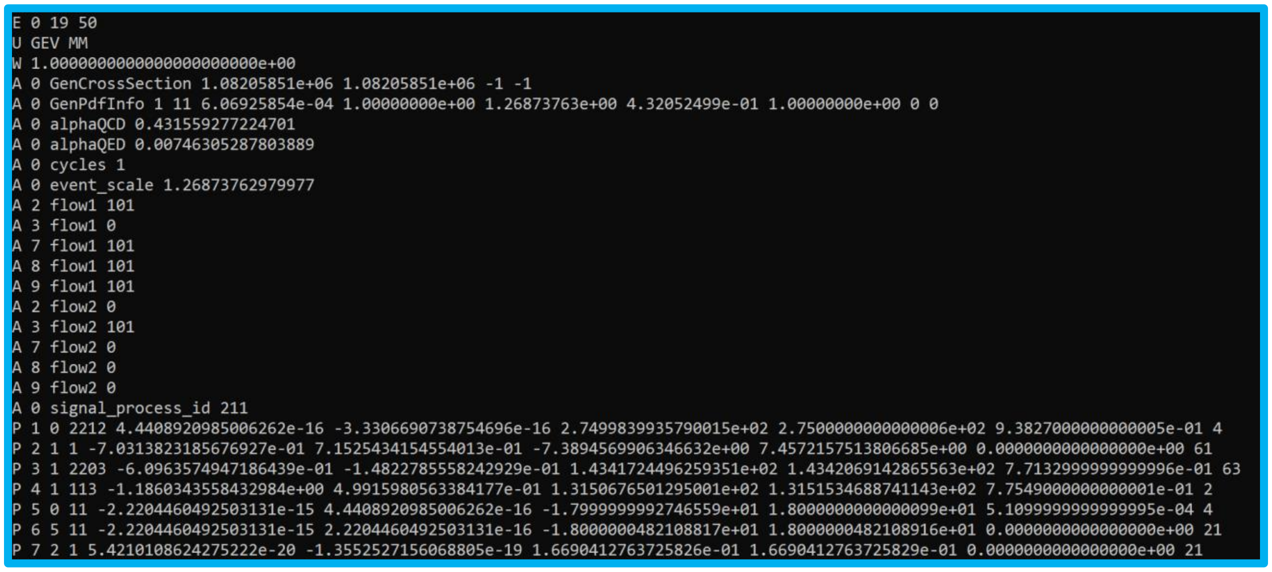
If you want to study further analyses conducted using this *EICTree* structure, take a look at this repository:

https://github.com/JeffersonLab/dis-reconstruction

The analysis scripts won't work right out-of-the-box, but you should be able to modify the codes to suit your needs. One useful code located there is comparing the inclusive cross section in the generators to a calculation. If you run this cross section analysis code, you will see the following plot:



Finally, take a look at the *HepMC3* file created by *Pythia8*, for example. You will see something like this printed for each event:

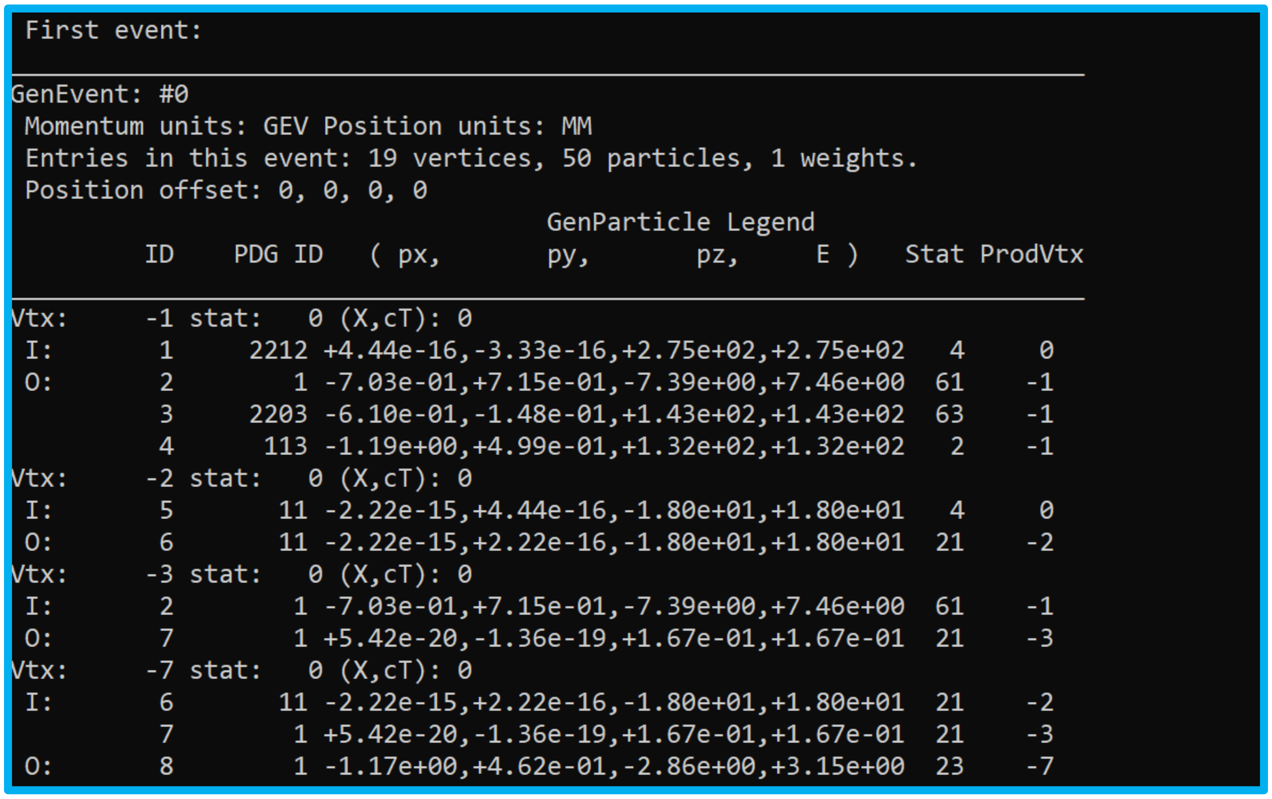


```
E 0 19 50
U GEV MM
W 1.0000000000000000000000e+00
A 0 GenCrossSection 1.08205851e+06 1.08205851e+06 -1 -1
A 0 GenPdfInfo 1 11 6.06925854e-04 1.00000000e+00 1.26873763e+00 4.32052499e-01 1.00000000e+00 0 0
A 0 alphaQCD 0.431559277224701
A 0 alphaQED 0.00746305287803889
A 0 cycles 1
A 0 event_scale 1.26873762979977
A 2 flow1 101
A 3 flow1 0
A 7 flow1 101
A 8 flow1 101
A 9 flow1 101
A 2 flow2 0
A 3 flow2 101
A 7 flow2 0
A 8 flow2 0
A 9 flow2 0
A 0 signal_process_id 211
P 1 0 2212 4.4408920985006262e-16 -3.3306690738754696e-16 2.7499839935790015e+02 2.7500000000000006e+02 9.3827000000000005e-01 4
P 2 1 1 -7.0313823185676927e-01 7.1525434154554013e-01 -7.3894569906346632e+00 7.4572157513806685e+00 0.0000000000000000e+00 61
P 3 1 2203 -6.0963574947186439e-01 -1.4822785558242929e-01 1.4341724496259351e+02 1.4342069142865563e+02 7.7132999999999996e-01 63
P 4 1 113 -1.1860343558432984e+00 4.9915980563384177e-01 1.3150676501295001e+02 1.3151534688741143e+02 7.7549000000000001e-01 2
P 5 0 11 -2.2204460492503131e-15 4.4408920985006262e-16 -1.7999999992746559e+01 1.8000000000000099e+01 5.1099999999999995e-04 4
P 6 5 11 -2.2204460492503131e-15 2.2204460492503131e-16 -1.8000000482108817e+01 1.8000000482108916e+01 0.0000000000000000e+00 21
P 7 2 1 5.4210108624275222e-20 -1.3552527156068805e-19 1.6690412763725826e-01 1.6690412763725829e-01 0.0000000000000000e+00 21
```

This output seems very complicated to understand. To see what is happening, start at the top of our Github repository and do the following:

> cd analysis/HepMC
> make
> ./HepMC3_fileIO ../../pythia8/outfiles/hepmcout.dat hepmcout_copy.dat

This will parse the input HepMC3 file and make an identical copy called *hepmcout_copy.dat*. That is not so interesting in itself, but what gets written to the screen for the first event is quite useful to study. We see the following printed to the screen:



```
 First event:
________________________________________________________________________________
GenEvent: #0
 Momentum units: GEV Position units: MM
 Entries in this event: 19 vertices, 50 particles, 1 weights.
 Position offset: 0, 0, 0, 0
                                        GenParticle Legend
         ID    PDG ID   ( px,       py,        pz,      E )   Stat ProdVtx
________________________________________________________________________________
Vtx:     -1 stat:   0 (X,cT): 0
 I:       1     2212 +4.44e-16,-3.33e-16,+2.75e+02,+2.75e+02   4     0
 O:       2        1 -7.03e-01,+7.15e-01,-7.39e+00,+7.46e+00  61    -1
          3     2203 -6.10e-01,-1.48e-01,+1.43e+02,+1.43e+02  63    -1
          4      113 -1.19e+00,+4.99e-01,+1.32e+02,+1.32e+02   2    -1
Vtx:     -2 stat:   0 (X,cT): 0
 I:       5       11 -2.22e-15,+4.44e-16,-1.80e+01,+1.80e+01   4     0
 O:       6       11 -2.22e-15,+2.22e-16,-1.80e+01,+1.80e+01  21    -2
Vtx:     -3 stat:   0 (X,cT): 0
 I:       2        1 -7.03e-01,+7.15e-01,-7.39e+00,+7.46e+00  61    -1
 O:       7        1 +5.42e-20,-1.36e-19,+1.67e-01,+1.67e-01  21    -3
Vtx:     -7 stat:   0 (X,cT): 0
 I:       6       11 -2.22e-15,+2.22e-16,-1.80e+01,+1.80e+01  21    -2
          7        1 +5.42e-20,-1.36e-19,+1.67e-01,+1.67e-01  21    -3
 O:       8        1 -1.17e+00,+4.62e-01,-2.86e+00,+3.15e+00  23    -7
```

This shows all the particles in the event and their creation and end points. The same information is contained in the *HepMC3* file, but in a somewhat more difficult-to-read format.

One last interesting thing to notice is that the *HepMC3* file contains the total cross section information. This is given for each event. As more events are simulated, knowledge of the total cross section improves, and the cross section given in the last event – which is the same as the one written to the log file – is the most accurate.

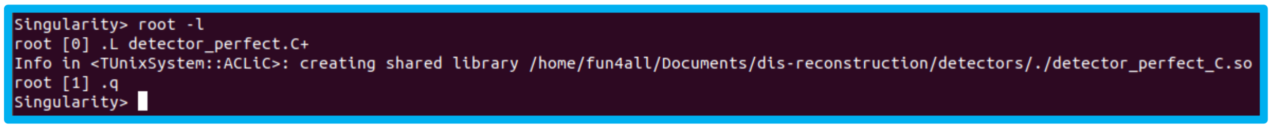
This completes the generator part of this document. The remainder of this document discusses the fast-smearing package *eic-smear*.

Compiling *eic-smear* detector configurations

In order to run the output of the event generators through the fast simulation *eic-smear* framework, you will need to first create a library for a given detector configuration. In our GitHub repository, two example configuration exist. Do the following to make the detector libraries:

```
> cd detectors
> root -l
> .L detector_perfect.C+
>.q
>root -l
>.L detector_central.C+
```

If you build the first detector correctly, for example, you should see the following on the screen (path to library will be different):



```
Singularity> root -l
root [0] .L detector_perfect.C+
Info in <TUnixSystem::ACLiC>: creating shared library /home/fun4all/Documents/dis-reconstruction/detectors/./detector_perfect_C.so
root [1] .q
Singularity>
```

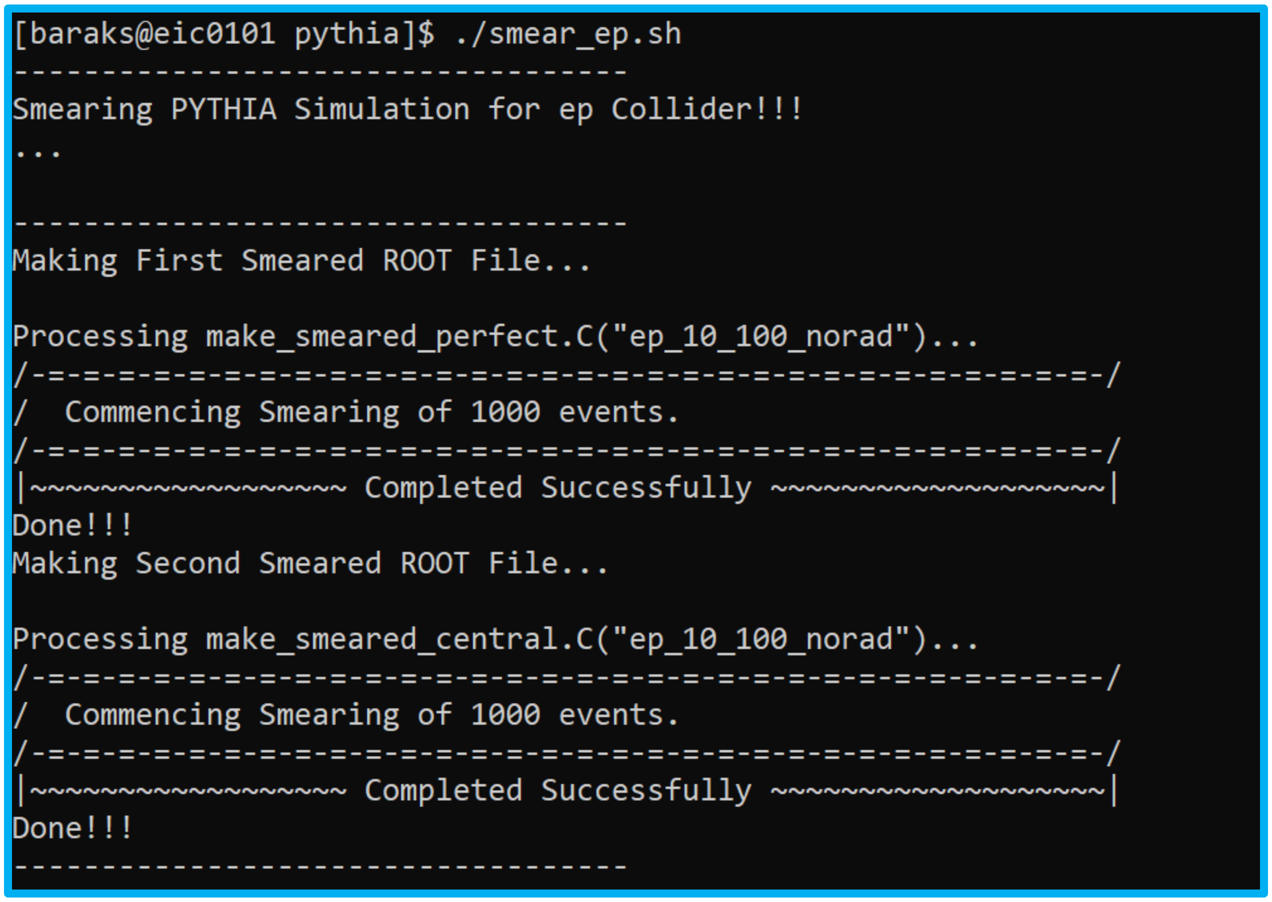
Each of these files defines a detector acceptance and resolution parametrization. Running a generator-level *EICTree* ROOT file through one of these 'detectors' produces a smeared output file that mimics the response of a real detector.

Creating Smeared ROOT files

In order to run the generated events through a fast-smearing configuration, go back into our pythia subdirectory, and do the following:

```
> ./smear_ep.sh
```

If you run the smearing correctly, two 'smeared' ROOT files will appear in the outfiles subdirectory, and you will see the following output on the screen:



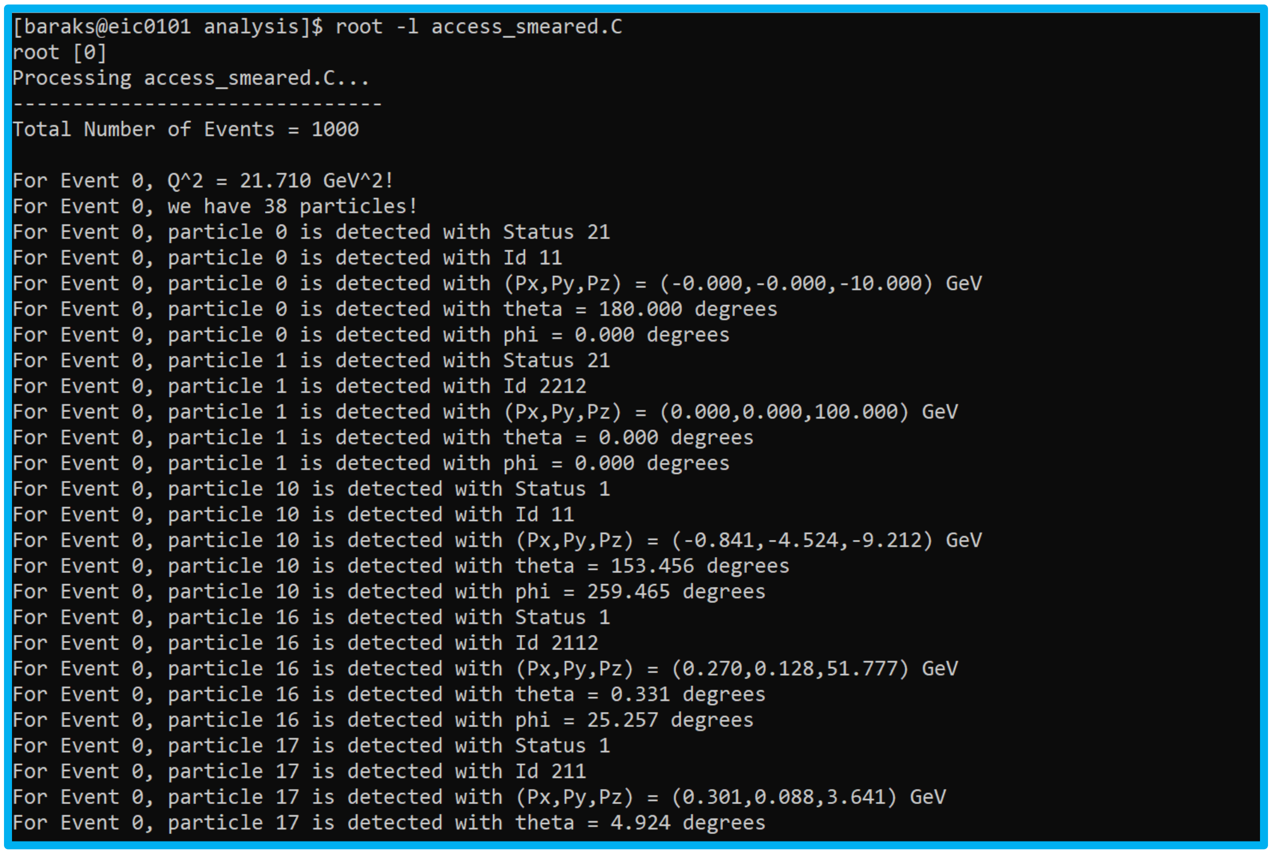
```
[baraks@eic0101 pythia]$ ./smear_ep.sh
-----------------------------------
Smearing PYTHIA Simulation for ep Collider!!!
...

-----------------------------------
Making First Smeared ROOT File...

Processing make_smeared_perfect.C("ep_10_100_norad")...
/-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-/
/  Commencing Smearing of 1000 events.
/-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-/
|~~~~~~~~~~~~~~~~~~ Completed Successfully ~~~~~~~~~~~~~~~~~~~|
Done!!!
Making Second Smeared ROOT File...

Processing make_smeared_central.C("ep_10_100_norad")...
/-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-/
/  Commencing Smearing of 1000 events.
/-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-/
|~~~~~~~~~~~~~~~~~~ Completed Successfully ~~~~~~~~~~~~~~~~~~~|
Done!!!
-----------------------------------
```

Analyzing Smeared output — examples

To analyze the output of the fast smearing, start in the top level of our Github repository and do the following:

> cd analysis
> root -l access_smeared.C

Like the generator-level script *access_tree.C*, this one prints out the reconstructed momenta and energies of particles within the detector acceptance. If you run the code correctly, you will see an output similar to this printed to the screen:

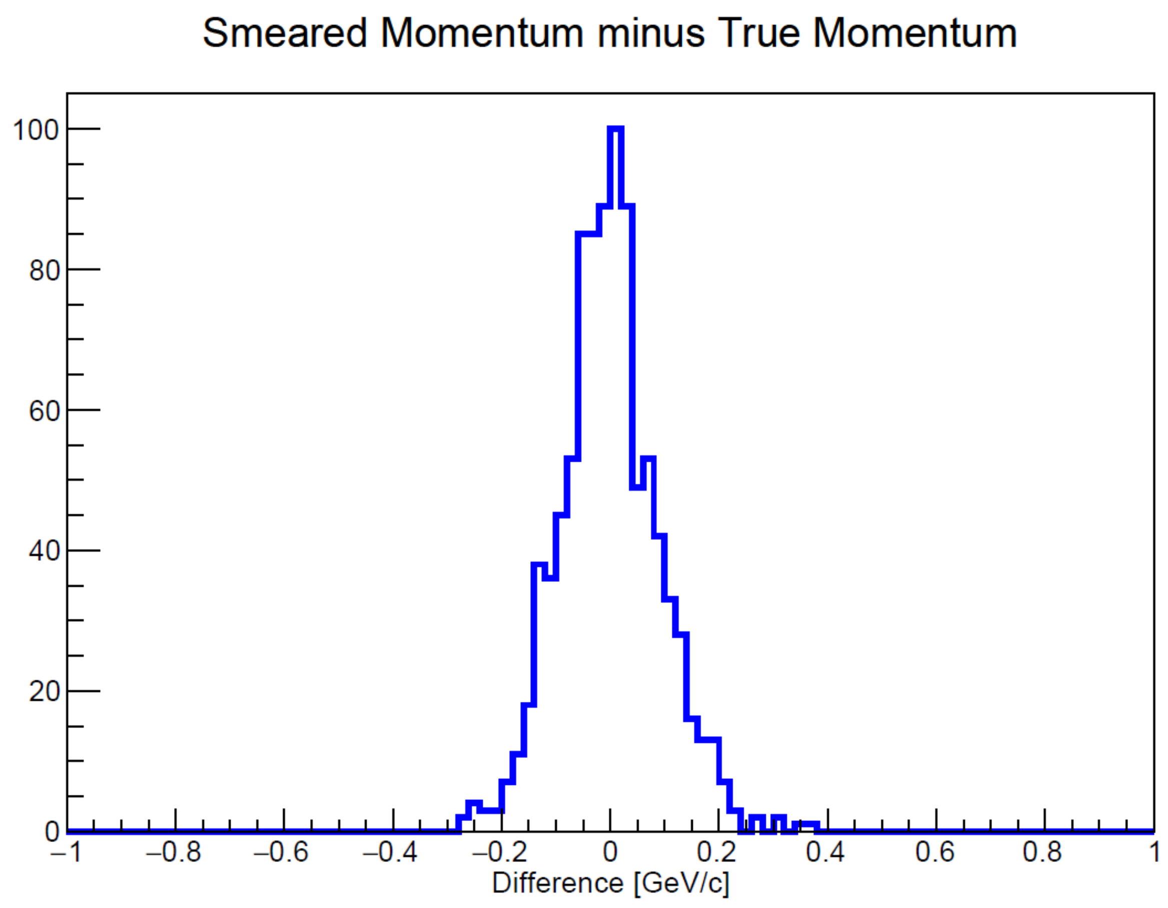


```
[baraks@eic0101 analysis]$ root -l access_smeared.C
root [0]
Processing access_smeared.C...
-------------------------------
Total Number of Events = 1000

For Event 0, Q^2 = 21.710 GeV^2!
For Event 0, we have 38 particles!
For Event 0, particle 0 is detected with Status 21
For Event 0, particle 0 is detected with Id 11
For Event 0, particle 0 is detected with (Px,Py,Pz) = (-0.000,-0.000,-10.000) GeV
For Event 0, particle 0 is detected with theta = 180.000 degrees
For Event 0, particle 0 is detected with phi = 0.000 degrees
For Event 0, particle 1 is detected with Status 21
For Event 0, particle 1 is detected with Id 2212
For Event 0, particle 1 is detected with (Px,Py,Pz) = (0.000,0.000,100.000) GeV
For Event 0, particle 1 is detected with theta = 0.000 degrees
For Event 0, particle 1 is detected with phi = 0.000 degrees
For Event 0, particle 10 is detected with Status 1
For Event 0, particle 10 is detected with Id 11
For Event 0, particle 10 is detected with (Px,Py,Pz) = (-0.841,-4.524,-9.212) GeV
For Event 0, particle 10 is detected with theta = 153.456 degrees
For Event 0, particle 10 is detected with phi = 259.465 degrees
For Event 0, particle 16 is detected with Status 1
For Event 0, particle 16 is detected with Id 2112
For Event 0, particle 16 is detected with (Px,Py,Pz) = (0.270,0.128,51.777) GeV
For Event 0, particle 16 is detected with theta = 0.331 degrees
For Event 0, particle 16 is detected with phi = 25.257 degrees
For Event 0, particle 17 is detected with Status 1
For Event 0, particle 17 is detected with Id 211
For Event 0, particle 17 is detected with (Px,Py,Pz) = (0.301,0.088,3.641) GeV
For Event 0, particle 17 is detected with theta = 4.924 degrees
```

Next, go into the compiled subdirectory and see if there is an executable called *analyze_smeared* there. If not, type ‘make’. Then do the following:

> ./analyze_smeared -l -b -q

This will create an output file in the plots subdirectory called *analyze_smeared.pdf*. This file contains a set of plots showing the difference between the reconstructed particle kinematics and the true generated quantities. For example, this plot shows the difference between the reconstructed and true momentum magnitude for the accepted scattered electron:



Note how in the above code, there are two loops – one over the generated particles and a second one over the smeared (reconstructed) particles. For more complex analyses, it is better to use a single loop over both the generated and smeared particles, as is done here. This works because the smeared particles have the same 'index' as the generated particles.

Author: Barak Schmookler (baraks@ucr.edu)