TennGen_v1 Write Up

TennGen is a data driven random number generator which simulates the jet background in heavy ion collisions. The jet background has features which match those in heavy ion collisions:

1) A low p_T particle spectrum generated from fits of data (**Phys. Lett.,B720:52–62, 2013**) to a Boltzmann-Gibbs Blast Wave.

$$\frac{d^2N}{dp_T dy} = N p_T \int_0^1 r' dr' \left(\sqrt{m^2 + p_T^2}\right) \cdot$$
$$I_0\left(\frac{p_T \sinh\left(\tanh^{-1}\left(\beta_s r'^n\right)\right)}{T_{kin.}}\right) \cdot$$
$$K_1\left(\frac{\sqrt{m^2 + p_T^2}\cosh\left(\tanh^{-1}\left(\beta_s r'^n\right)\right)}{T_{kin.}}\right)$$

2) An azimuthal distribution of particles modeled by a Fourier decomposition. A a realistic 2nd order asymmetry parameter (v_2) dependence on particle p_T is used. In addition, realistic p_T dependence of the 1st,3rd,4th, and 5th asymmetry parameters (v_1, v_3, v_4, v_5) is also implemented. The dependence comes from polynomial fits to data (**JHEP, 09:164, 2016**). Furthermore, TennGen also contains semi-realistic correlations between event planes.

$$\frac{dN}{d\phi} = \frac{N_0}{2\pi}\sum_{n=1}^{5} 2v_n \cos(n(\phi - \Psi_n))$$

3) A uniformly random pseudo-rapidity distribution (assuming a reasonable range at mid-rapidity, this is a good approximation which becomes worse at more forward rapidities).

4) A reasonable event multiplicity (at mid-rapidity) determined from data (**Phys. Rev., C88:044910, 2013** & **Phys. Rev. Lett., 106(CERN-PH-EP-2010-071. CERN-PH-EP-2010-071):032301**).

5) A realistic centrality dependence of all the particle properties mentioned above.

TennGen is based on fits to LHC Run 1 Pb+Pb collisions (see references in bold). TennGen contains NO jets from any hard scatterings. That is to say there are no correlations present in TennGen that result from hard scatterings (or mini-jets, resonances/decays). TennGen is also unique in that it allows the user to include or exclude harmonics (v_n) to make up the azimuthal distribution as they desire. If all v_n are excluded then a uniformly random azimuthal distribution is thrown. TennGen does contain event by event fluctuations due to randomly distributed particles. Finally, TennGen does allow the user to vary the centrality. The user can choose the following bins: [0-5 %], [5-10 %], [10-20 %], [20-30 %], [30-40 %], [40-50 %].

TennGen:

- does not contain event by event *multiplicity* fluctuations. For each event, the amount of particles thrown by TennGen is **constant**. However, TennGen does contain fluctuations due to random distribution of particles in coordinate space. In addition, the particle multiplicity does vary realistically with centrality.

- is not based on a full-fledged physics simulation. It is a random number generator which produces correlations in particle properties that match those of the event background in data.

- does not any kind of physics related to jets form hard scatterings. The particles that it throws do NOT react in any way to the presence of particles from hard scatterings (e.g. from another MC generator). There is no quenching, broadening, or suppression present in TennGen.

- produces only $\pi^{+/-/0}$,$K^{+/-}$, and $p/\bar{p}$. No other particle species are thrown.

- should only be used at MID-RAPIDITY.

Assumptions in TennGen:

1) Boltzmann-Gibbs Blast Wave fits can be extrapolated down to 0 GeV/c in p_T .
2) Boltzmann-Gibbs Blast Wave fits can be extrapolated up to 100 GeV/c in p_T .
3) The π^0 Boltzmann-Gibbs Blast Wave fit, v_n vs. p_T dependence, and multiplicity can all be taken to be the same as that of the π^-.
4) The v_n polynomial fit can be extrapolated below the lowest measured p_T values, unless they result in a 0 value
5) The v_n are set to be 0 when the fit results in a $v_n < 0$ (except for v_1 which is allowed to be < 0).
6) The v_n fitting polynomials are set to be equal to their functional form evaluated at the highest measured p_T value
7) The variation in measured per-event particle multiplicity is ignored
8) The pseudo-rapidity distribution is uniformly random and can be scaled up for wider acceptance.
9) Event event planes (Ψ_2, Ψ_4) are assumed to be 0 for all events. This is consistent with **PhysRevC.90.024905.**
10) Odd event planes (Ψ_1, Ψ_3, Ψ_5) are chosen from uniform random distributions from 0 to 2π per event which are non-correlated with each other. This is consistent with **PhysRevC.90.024905.**
11) v1 is assumed to take form: $v_1 = v_2 - 0.01$ which is consistent with **Phys-RevC.86.014907**.

<u>**Running TennGen**</u>

In the github there are several files (including):

- maindriver.C
- TennGen_Test_Macro.C
- TennGen.cxx
- TennGen.h

to run, make sure you are using ROOT5. Then in the ROOT environment:

```
>> root maindriver.C
```

<u>**Using the TennGen class:**</u>

TennGenn.cxx is compiled in maindriver.C. You must compile this class and include the TennGen.h header file in any code you run an instance of the TennGen class in.

TennGen_Test_Macro.C is run by maindriver.C. This file contains the actual instance of the TennGen class. Please read over this code carefully to understand how TennGen is used. All of the following lines are contained in the example TennGen_Test_Macro.C They must be called in the same order they are called in TennGen_Test_Macro.C

To use TennGen in your code you must do the following:

1) declare an instance of the class:

```
class TennGen;
```

2) declare a constructor:

```
TennGen *bkgd = new TennGen();
```

3) Set centrality bin:

```
bkgd->SetCentralityBin(cent_bin);
```

cent_bin is an integer: 0 → [0-5 %], 1 → [5-10 %], 2 → [10-20 %], 3 → [20-30 %], 4 → [30-40 %], 5 → [40-50 %].

4) Set a random seed (this is absolutely CRUCIAL):

```
bkgd->SetRandomSeed(seed5);
```

5) Set a harmonics flag (this determines which harmonics combination you want TennGen to determine the azimuthal spectrum by):

```
bkgd->SetHarmonics(HF);
```

HF is an integer: (0 : v1 - v5) , (1 : v2 - v5) , (2: v3 - v5) , (3: v1 - v4) , (4: v1 - v3) , (5: uniform dN/dphi no harmonics) , (6 : v1 - v2 , v4 - v5) , (7 : v1 - v3 , v5) , (8 : v1 , v3 - v5) , (9 : v1 only) , (10 : v2 only) , (11 : v3 only) , (12 : v4 only) , (13 : v5 only)

6) Set an Pseudo-Rapidity Range (be sure to keep it to mid-rapidity, 0.9 means -0.9 to 0.9):

```
bkgd->SetEtaRange(0.9);
```

7) Tell it whether or not you want to print out QA histograms (kTRUE yes, kFALSE no):

```
bkgd->PrintOutQAHistos(kTRUE);
```

8) Initialize the background:

```
bkgd->InitializeBackground();
```

9) In order to access the particle generated by TennGen declare a TClonesArray:

```
TClonesArray *BKGD = bkgd→GetBackground();
TMCParticle* bob = (TMCParticle*)BKGD→At(N);
```

where N is a looping index and should be called in a loop. TennGen gives the user a TClonesArray of TMCParticles.

10) In order to refresh the background TennGen throws to get a different event:

```
BKGD->Clear();
TClonesArray *BKGD = bkgd→GetBackground();
```

11) Remember to close out QA histograms with the following:

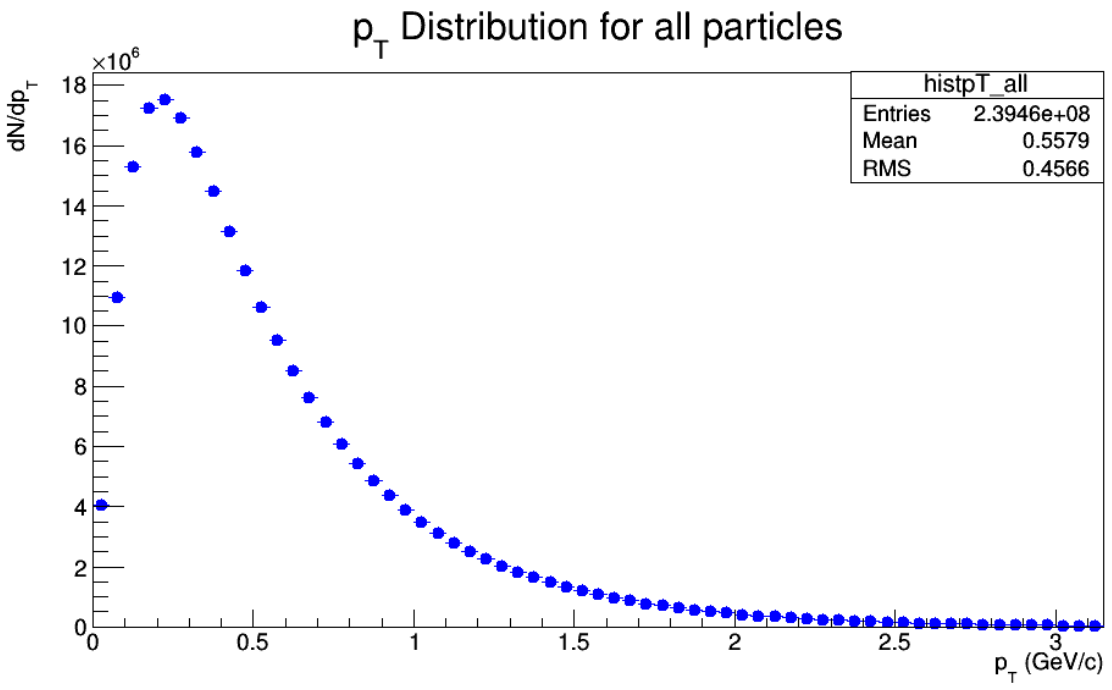
```
bkgd->CloserFunction();
```

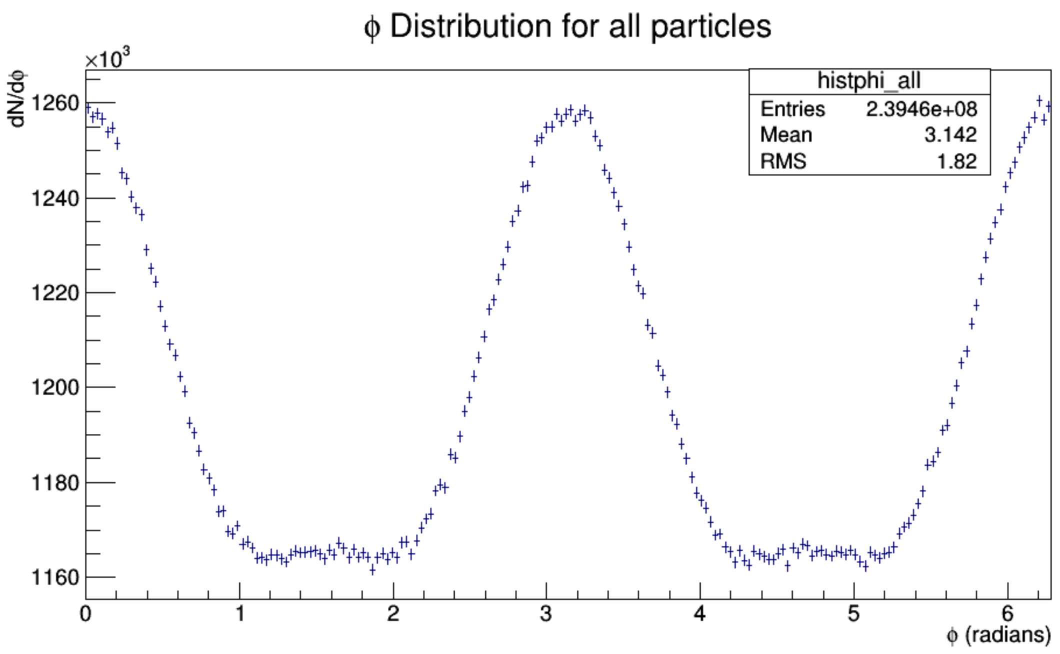
12) And finally the destructor:

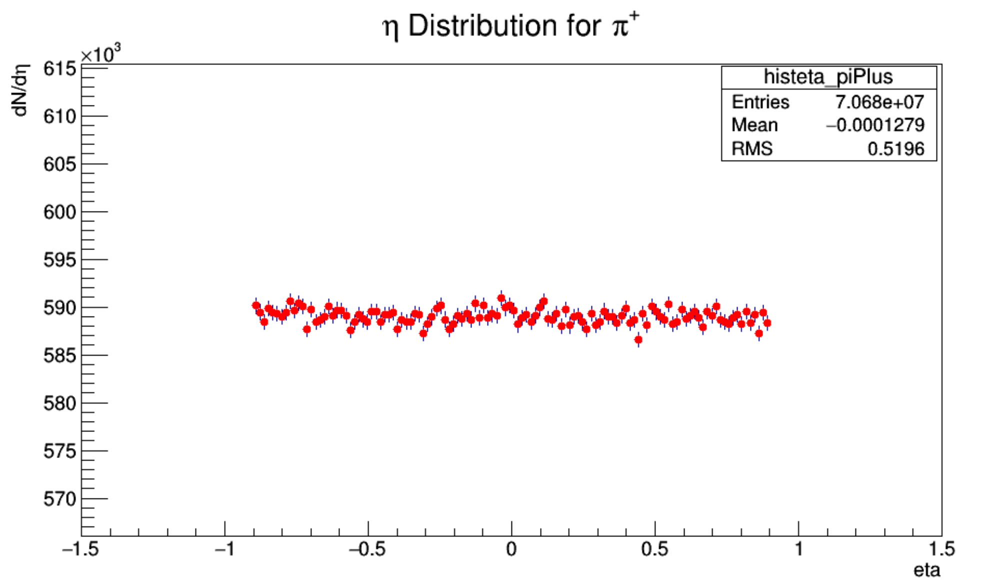
```
bkgd->~TennGen();
```

TennGen Output

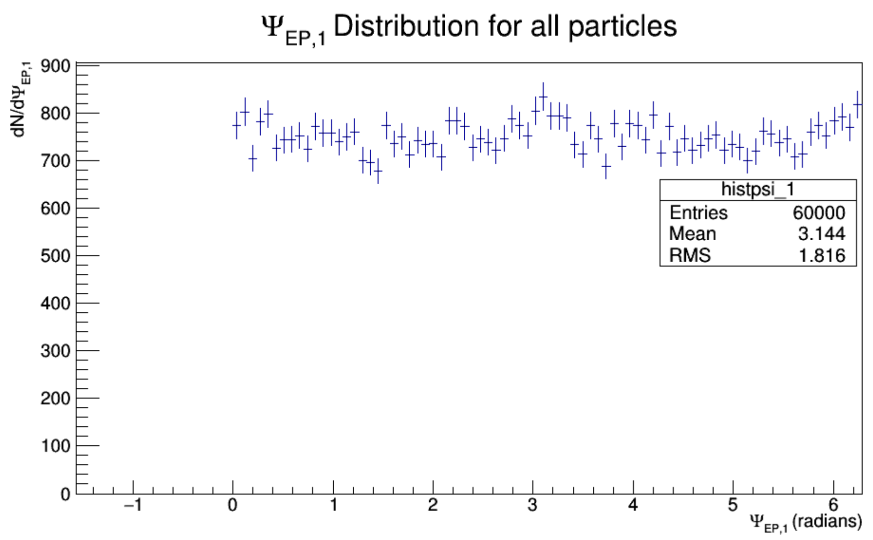
Below are some plots with examples of how particles generated with TennGen look like for 0-5 % centrality for v_1-v_5 (full harmonics) and a pseudo rapidity range of [-0.9,0.9]. Note this has been generated from a large event pool. Event planes shown are RECONSTRUCTED planes using qVectors method from TennGen output.



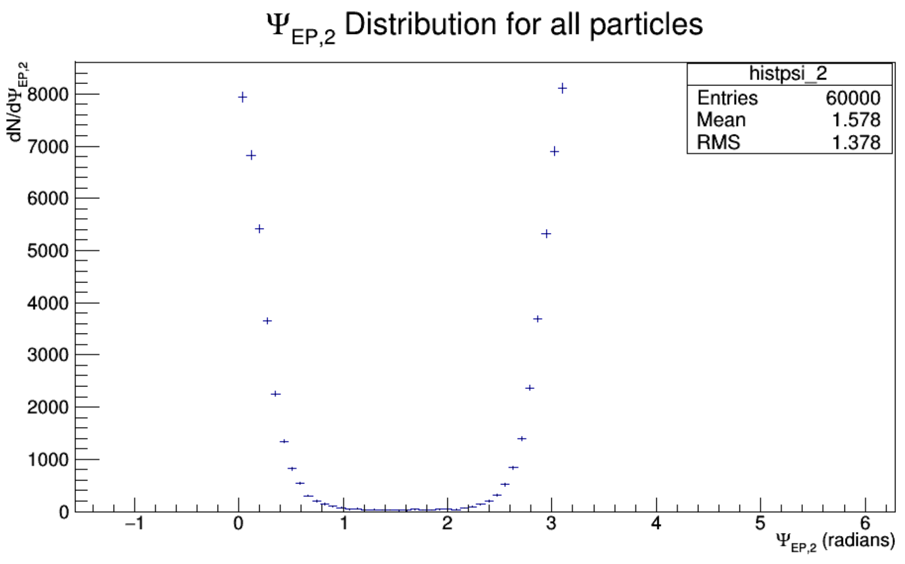




η Distribution for π+
dN/dη
×10³
histeta_piPlus
Entries 7.068e+07
Mean −0.0001279
RMS 0.5196
eta



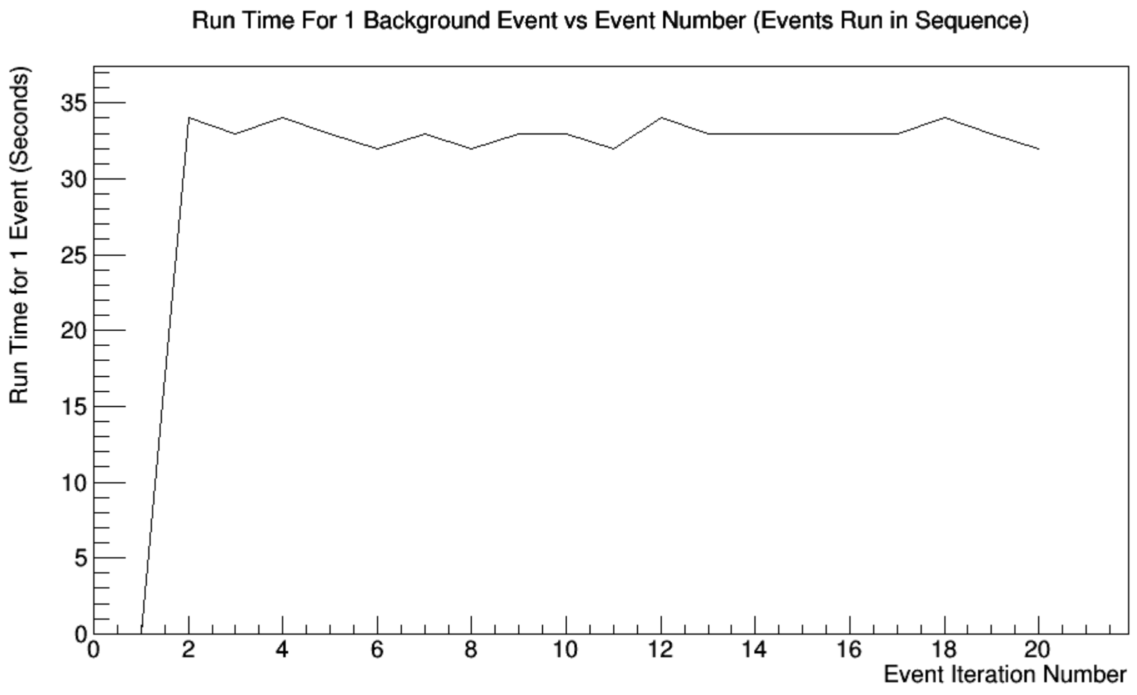
Ψ_EP,1 Distribution for all particles
dN/dΨ_EP,1
histpsi_1
Entries 60000
Mean 3.144
RMS 1.816
Ψ_EP,1 (radians)



Ψ_EP,2 Distribution for all particles
dN/dΨ_EP,2
histpsi_2
Entries 60000
Mean 1.578
RMS 1.378
Ψ_EP,2 (radians)

<u>TennGen performance</u>:

The below graph shows the time it takes to generate a TennGen event for 0-5 % full harmonic background over 20 events. This was run on a 4 core 2.50 GHz processor with 8 GB of RAM.

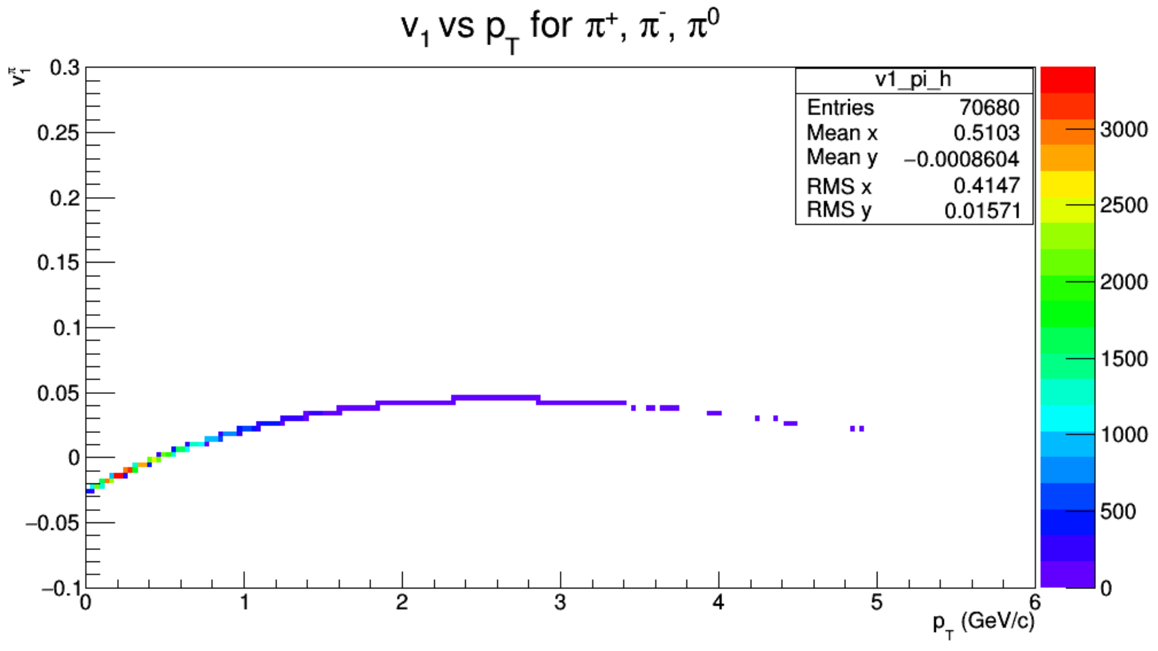


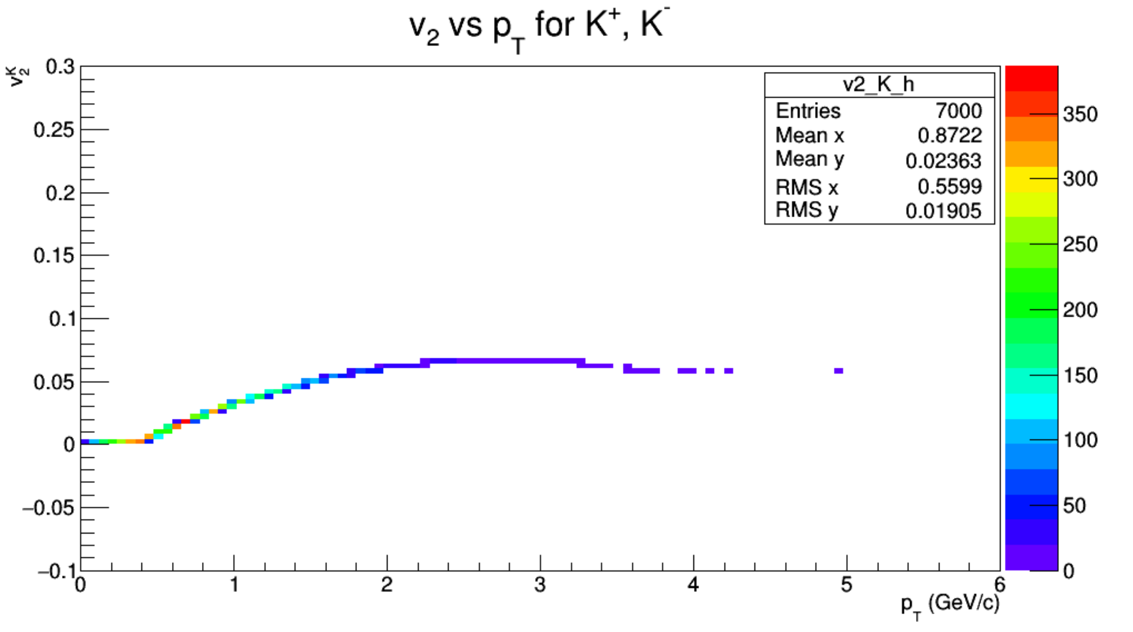
TennGen QA:

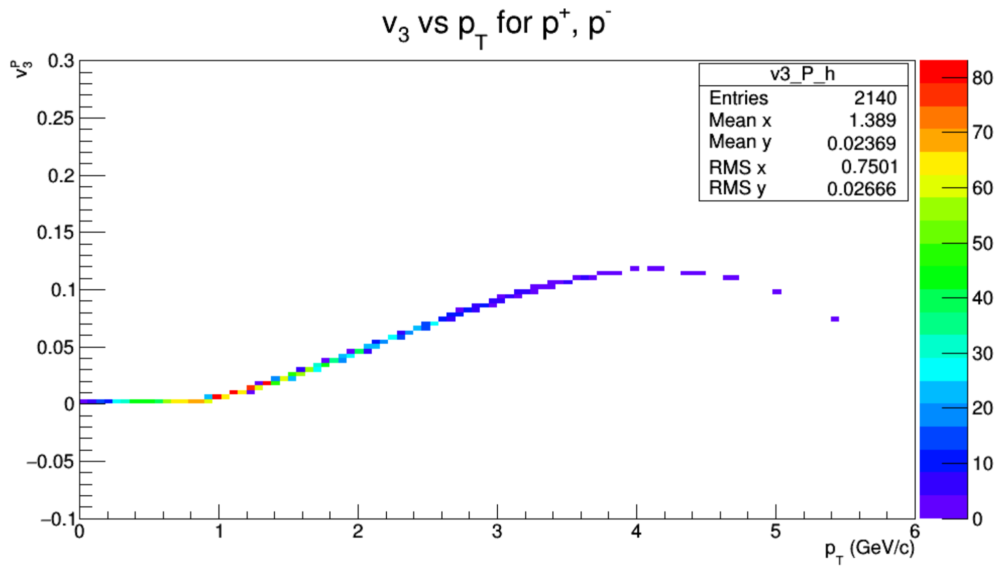
How can the end-user understand the v_n vs p_T relationship ? If the user enables the QA histograms using:

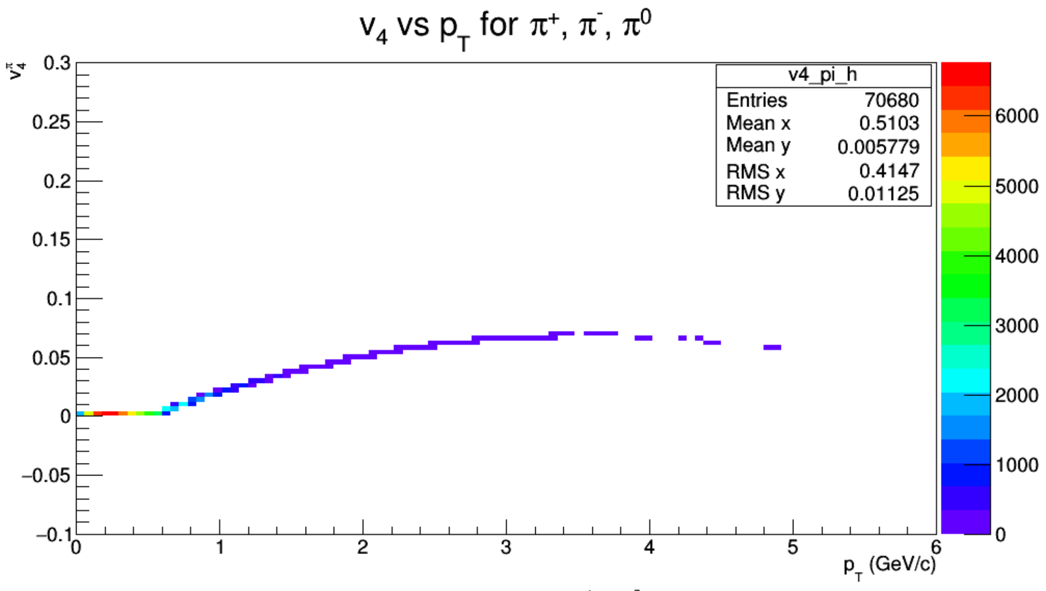
```
bkgd->PrintOutQAHistos(kTRUE);
```

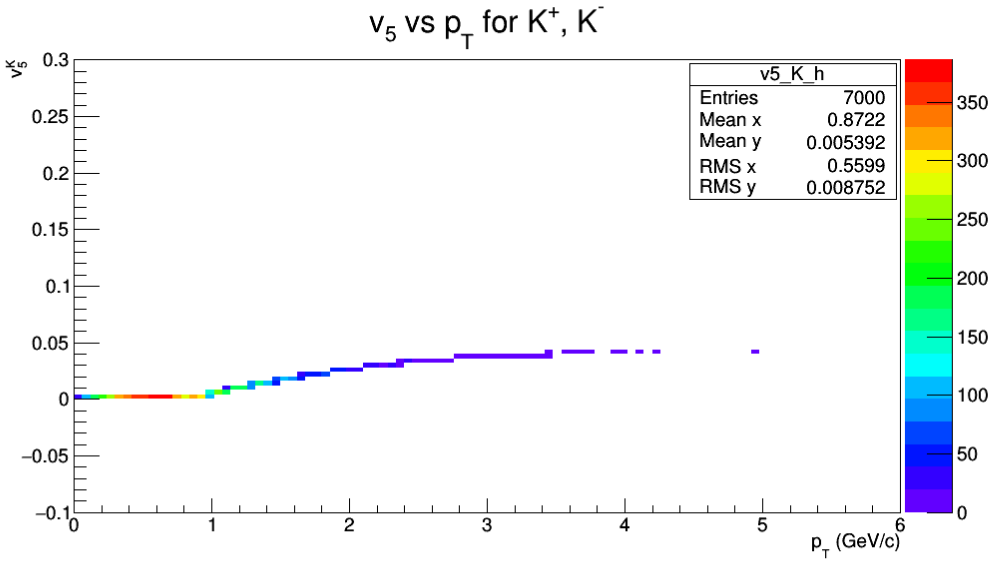
QA histograms are produced that show the user the v_n vs p_T curve for each particle thrown. These are saved to a .root file (Background_Generator_QA_file.root) Below is an example for 20 events:



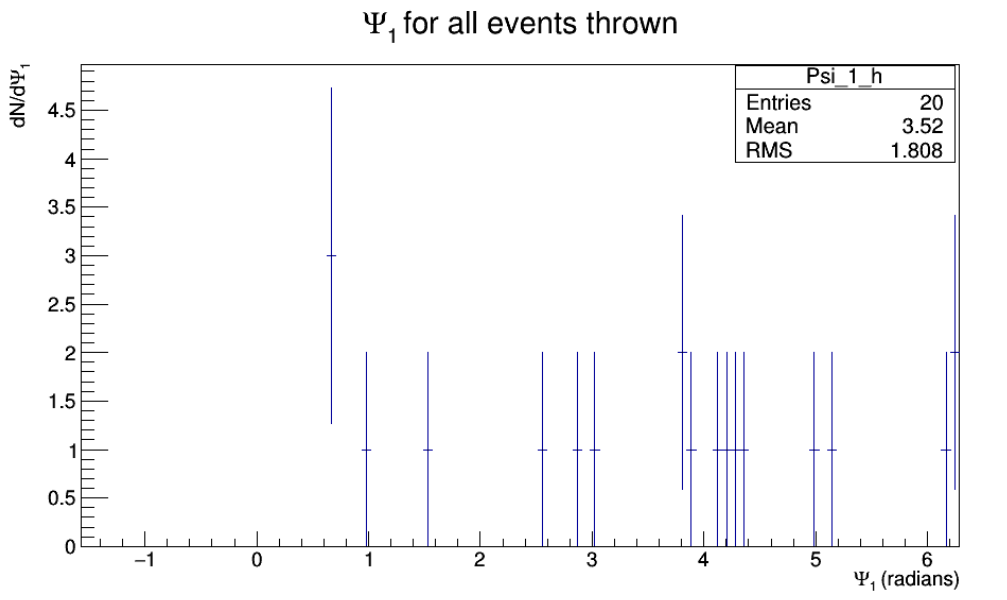


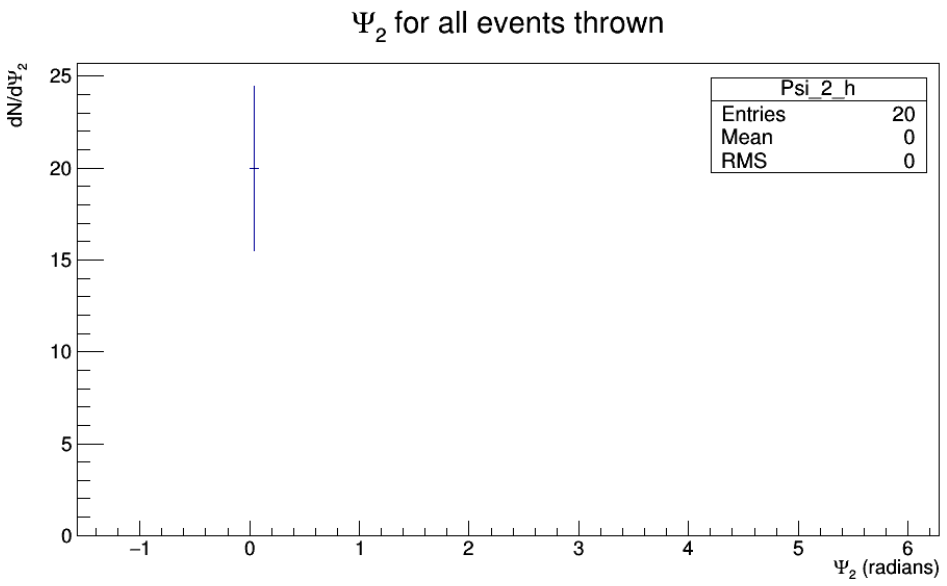

v_3 vs p_T for p^+, p^-
v_3^P
v3_P_h
Entries 2140
Mean x 1.389
Mean y 0.02369
RMS x 0.7501
RMS y 0.02666
p_T (GeV/c)
0.3 0.25 0.2 0.15 0.1 0.05 0 −0.05 −0.1
0 1 2 3 4 5 6
0 10 20 30 40 50 60 70 80


v_4 vs p_T for π^+, π^-, π^0
v_4^π
v4_pi_h
Entries 70680
Mean x 0.5103
Mean y 0.005779
RMS x 0.4147
RMS y 0.01125
p_T (GeV/c)
0.3 0.25 0.2 0.15 0.1 0.05 0 −0.05 −0.1
0 1 2 3 4 5 6
0 1000 2000 3000 4000 5000 6000

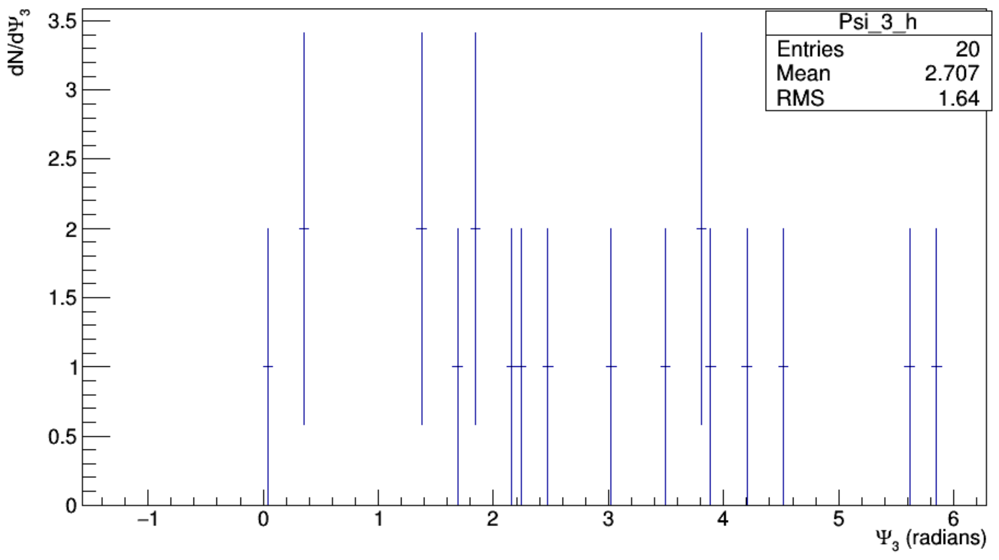

v_5 vs p_T for K^+, K^-
v_5^K
v5_K_h
Entries 7000
Mean x 0.8722
Mean y 0.005392
RMS x 0.5599
RMS y 0.008752
p_T (GeV/c)
0.3 0.25 0.2 0.15 0.1 0.05 0 −0.05 −0.1
0 1 2 3 4 5 6
0 50 100 150 200 250 300 350

In addition to the v_n histograms, we also have the actual thrown event planes save out the QA file (as opposed to the reconstructed ones shown earlier in this write up) in the case the user wants to check how these are being thrown. Below are more examples for 20 events.

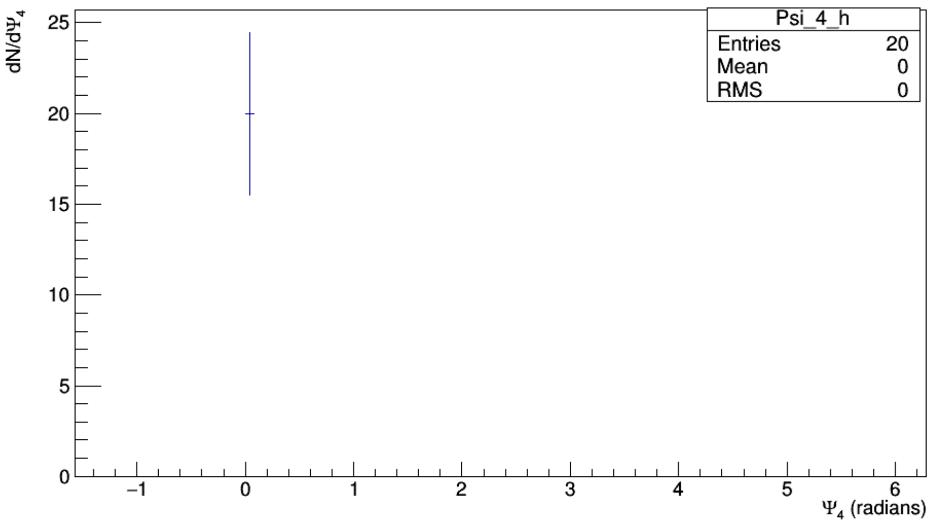




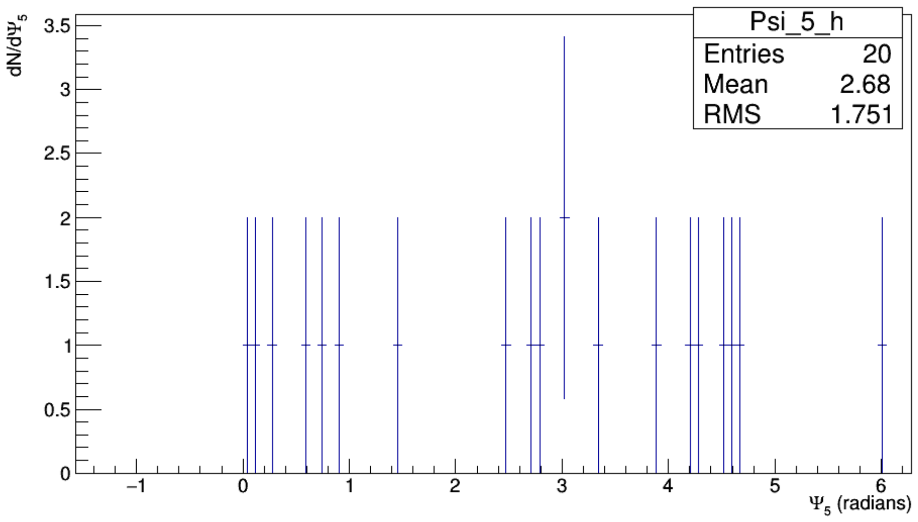
Ψ_3 for all events thrown



Ψ_4 for all events thrown



Ψ_5 for all events thrown


Psi_5_h
Entries 20
Mean 2.68
RMS 1.751
$dN/d\Psi_5$
Ψ_5 (radians)