Using RIVET to measure the non-perturbative corrections in Monte Carlo event generators

James Cowley

March 2015

University College London
Department of Physics and Astronomy

Abstract

A method is presented for obtaining the non-perturbative correction factors from Monte Carlo event generators using the RIVET toolkit. Instead of running the generator with particular processes switched off in order to measure the necessary jet cross sections, a RIVET analysis is written which can extract these from the event record. The analysis is performed on samples (all using the AU2-CT10 tune and centre-of-mass energy $\sqrt{s}$ = 7 TeV) from three Monte Carlo generators: PYTHIA8, POWHEG1+PYTHIA8 and POWHEG2+PYTHIA8. The measured non-perturbative (NP) and parton shower (PS) correction factors for these generators are found to be in agreement with results obtained in previous studies by switching off selected processes at generator-level. These results also present an additional opportunity to study the new generator POWHEG2 and compare its non-perturbative correction factors to those of POWHEG1. The PS correction factor measured in POWHEG2 is found to be larger (by a fraction on the order of 10%) than that measured in POWHEG1 in the low p_T region ($p_T \sim 50$ GeV).

Contents

1 Introduction

Hadron collisions, such as those performed at the Large Hadron Collider (LHC), are understood in terms of interacting partons (i.e. quarks and gluons). Due to colour confinement, these partons cannot be observed independently and hadronise into colourless particles which are then observed in the detector.

To bridge the gap between the unobservable parton-level interactions and the data available from the detector at hadron-level, collisions are simulated using Monte Carlo (MC) event generators.

The resulting events can then be passed through a simulated detector, which produces a signal of the same format as that produced by real detector, allowing for direct comparison.

1.1 Monte Carlo methods

MC generators (reviewed in detail in Ref [1]) calculate cross sections based on matrix elements which have been computed from perturbation theory. However, these are only available with the accuracy of the first few orders, since higher order contributions are lengthy to derive. Integrating the matrix elements over the phase space to obtain the final probabilities is also very computationally expensive, especially when higher order contributions are included.

As a result, general-purpose MC generators such as PYTHIA [2] and HERWIG [3] can calculate all major processes with leading order (LO) precision, but some processes are available for modelling in next-to-leading order (NLO) generators such as POWHEG [4] or MC@NLO [5]. Greater precision can be obtained by using a NLO generator to compute the hard scatter and passing this to a LO generator to complete the parton shower and hadronisation stages.

1.2 Non-perturbative corrections

The hard initial parton interactions involving large momentum transfers are well described by perturbation theory, due to the asymptotic freedom of QCD quanta [6]. However, the non-perturbative nature of the softer interactions between partons involved in the subsequent showering and hadronisation stages requires the use of models incorporating free parameters which can be tuned to match the data.

The relation between the perturbative hard processes and the softer interactions of particles involved in parton showers and hadronisation is quantified by non-perturbative correction factors. These can be measured using MC event generators and vary depending on the specific generator and the tune used.

The correction factors are defined as follows. For events employing NLO hard scattering (e.g. from POWHEG), the parton shower (PS) correction factor is defined as

$$C_{NLO}^{PS} = \frac{\sigma_{NLO+PS}}{\sigma_{NLO}} \tag{1}$$

where σ denotes the inclusive[1] jet cross section, and the subscripts NLO and PS refer to the stage up to which the event is generated (after the NLO hard interaction and after the parton shower, respectively).

The non-perturbative (NP) correction factor between the parton shower and the hadronic final state is defined [7] [8] as

$$C_{NLO}^{NP} = \frac{\sigma_{NLO+PS+HAD+MPI}}{\sigma_{NLO+PS}} \tag{2}$$

where the additional subscripts HAD and MPI refer to the hadronisation and multiple parton interaction steps, respectively.

The combined correction from the hard parton interaction to the final state hadron jets is then the product of the terms in equations (1) and (2):

$$K_{NLO} = C_{NLO}^{PS} \cdot C_{NLO}^{NP} = \frac{\sigma_{NLO+PS+HAD+MPI}}{\sigma_{NLO}} \tag{3}$$

The double-differential jet cross section at NLO is then corrected for PS effects by

$$\frac{\partial^2\sigma_{theo}}{\partial p_T \partial y} = \frac{\partial^2\sigma_{NLO}}{\partial p_T \partial y} \cdot C_{NLO}^{PS} \tag{4}$$

and for the combined PS and NP effects by

$$\frac{\partial^2\sigma_{theo}}{\partial p_T \partial y} = \frac{\partial^2\sigma_{NLO}}{\partial p_T \partial y} \cdot C_{NLO}^{PS} \cdot C_{NLO}^{NP} \tag{5}$$

The correction factors for LO events (e.g. from PYTHIA-only) are defined similarly:

$$C_{LO}^{PS} = \frac{\sigma_{LO+PS}}{\sigma_{LO}} \tag{6}$$

$$C_{LO}^{NP} = \frac{\sigma_{LO+PS+HAD+MPI}}{\sigma_{LO+PS}} \tag{7}$$

$$K_{LO} = C_{LO}^{PS} \cdot C_{LO}^{NP} = \frac{\sigma_{LO+PS+HAD+MPI}}{\sigma_{LO}} \tag{8}$$

[1] Here, "inclusive" refers to the fact that every jet from each event is considered in the measurement

$$\frac{\partial^2\sigma_{theo}}{\partial p_T \partial y} = \frac{\partial^2\sigma_{LO}}{\partial p_T \partial y} \cdot C_{LO}^{PS} \quad (9)$$

$$\frac{\partial^2\sigma_{theo}}{\partial p_T \partial y} = \frac{\partial^2\sigma_{LO}}{\partial p_T \partial y} \cdot C_{LO}^{PS} \cdot C_{LO}^{NP} \quad (10)$$

The difference between the NP factors C_{NLO}^{NP} and C_{LO}^{NP} arises from their differing definitions of the hard process (the p_T cut-off [2] for multiple parton interactions is different for LO and NLO [9]), and the PS factors C_{NLO}^{PS} and C_{LO}^{PS} differ as a direct result of the NLO treatment of the initial hard partons.

2 Obtaining the non-perturbative corrections

These correction factors have previously been studied using the LO generators PYTHIA6 and HERWIG in [10] and [11], and also using POWHEG at NLO interfaced with PYTHIA6 in [7] and [9].

This investigation differs in that the correction factors are obtained from POWHEG interfaced with PYTHIA8 rather than with PYTHIA6 (this entirely new PYTHIA is built in C++ instead of FORTRAN [12]).

Additionally, whereas previous studies have obtained the correction factors by comparing the output of MC generators with various components switched off in order to isolate the relevant cross sections, here the cross sections are measured by analysis of the complete event record using the RIVET toolkit [13].

2.1 Monte Carlo samples

This analysis was performed on centrally-available samples from the MC12 collection. Specifically, these were the events with run numbers 185491 (POWHEG+PYTHIA8) and 147905 (PYTHIA8 only).

Due to restrictions on the computing power available to the author on the UCL HEP system [14] and time constraints on the project itself, only a relatively small portion of these datasets is analysed here.

Approximately 2.4×10^5 events are analysed from each of the POWHEG+PYTHIA8 and PYTHIA8 samples, several orders of magnitude smaller than the total event populations of the entire datasets.

An additional 10^5 events were made available for analysis by the authors of a parallel project[3], generated using the new POWHEG2 (with PYTHIA8 shower).

All samples analysed here employ the ATLAS underlying event tune AU2 [15] with parton distribution function CT10 and a centre-of-mass energy $\sqrt{s}$ =7 TeV [16].

2.2 Defining the cross sections

By observation of equations (4, 5, 9 and 10), the various correction factors can be obtained by division of the relevant double-differential inclusive jet cross sections.

The jet cross section is defined using the anti-k_t jet clustering algorithm implemented by the FASTJET package [17] within RIVET.

This algorithm is infra-red safe and produces jets whose shape is unaffected by soft radiation [18] due to the fact that it prioritises clustering together the highest p_T entities (particles and pseudojets).

The anti-k_t algorithm defines a distance d_{ij} between two entities i and j as

$$d_{ij} = \min({p_{T,i}}^{-2}, {p_{T,j}}^{-2}) \frac{{\Delta R_{ij}}^2}{R^2} \quad (11)$$

where R is a parameter chosen by the user to define the desired jet radius (in this investigation, two values of R are used: 0.4 and 0.6),

$${\Delta R_{ij}}^2 = \left(y_i - y_j\right)^2 + \left(\phi_i - \phi_j\right)^2 \quad (12)$$

the rapidity y is

$$y = \frac{1}{2}\ln\left(\frac{E + p_z c}{E - p_z c}\right) \quad (13)$$

and ϕ is the azimuthal angle about the z (beam) axis.

2.3 Measuring the cross sections in RIVET

Jets are collected using anti-k_t on particles from three stages of the event record. The inclusive jet double-differential cross sections are obtained by plotting the jets at each stage as a function of p_T in six equally spaced bins of $\Delta|y| = 0.5$ up to a maximum absolute rapidity of $|y| = 3$ and requiring $p_T > 25$ GeV.

A full account of the source code for the analysis procedure summarised below is provided in the appendices and is also available online at [19].

The partons resulting from the hard scattering process are identified by status code[4], where a value of 23 denotes an outgoing par-

[2] The transverse momentum p_T is defined as the component of momentum perpendicular to the beam axis.

[3] See acknowledgements: Turner, B. and Chen, Y.

[4] PYTHIA8 uses status codes based on the HEPEVT standard. These are fully enumerated at [23].

ton from the main process. These are then clustered into jets using anti-k_t. This is necessary for comparison with the later stages wherein a single jet would be formed by the tightly focussed offspring of any group of very nearby partons.

The particles resulting from the parton shower stage are identified by status code in the range 60 → 70. They are then filtered to avoid double counting parents and children, so that only the outgoing particles from the parton shower (i.e. the immediate ancestors of the partons prepared for hadronisation) are collected into jets.

The final state particles are collected using RIVET's `FinalState` projection [20], which requires a status code of 1, and are then clustered into jets as above.

2.4 Dividing the cross sections

The histograms produced by RIVET are stored in the YODA format [21]. This is converted to a ROOT file so that further operations can be carried out using ROOT macros.

Although the source code for the macros written for this stage is longer than the combined code for both RIVET analyses, it contains little of interest to physics and is comprised mostly of functions for automating the processes of dividing, formatting and plotting the histograms.

Hence, only the sections of interest that are discussed below have been included in the appendices. All of the ROOT source code written for this study is available in its entirety online at [19].

The event collections in the POWHEG and POWHEG2 samples contain events across the full range of p_T analysed, whereas the samples from PYTHIA8 are separated into different p_T "slices", called JZXW where X refers to the number of the slice.

These facilitate analysis of large numbers of events at high p_T, where the unaltered jet energy spectrum would result in few entries.

Of the eight slices available (JZ0W → JZ7W), five slices JZ1W → JZ5W with leading $\hat{p}_T$ = 0, 40, 100, 250 and 600 GeV were analysed separately and then recombined within ROOT.

This necessitated a reweighting of each histogram using the cross sections and filter efficiencies available at [22]. These values are programmed into the source code of the ROOT function written to handle this stage, included in Appendix 3. This function extracts the histograms from the file produced by RIVET, applies the weights and saves them each to a new file. Then it uses ROOT's `hadd` program to merge these into one file. Thereafter, this file can be treated in the same way as the results from the other samples.

Dividing the hadron-level inclusive jet cross section distribution by the parton shower inclusive jet cross section distribution then yields the NP correction factor for each $p_T$and y bin. Similarly, dividing the parton shower distribution by the hard parton distribution provides the PS correction factor.

3 Results

3.1 NP correction factor from PYTHIA8 samples

Fig. 1 contains plots of the hadron jet and parton shower jet cross sections for anti-k_t parameters R = 0.4 and R = 0.6 in the central region ($0.0 < |y| < 0.5$) and in the forward region ($2.5 < |y| < 3.0$), from the analysis of the events produced at LO using PYTHIA8, using a logarithmic scale on the y-axis to improve visibility. Also shown is the calculated NP correction factor within each of the six rapidity bins.

Immediately obvious is the fact that the change from R = 0.4 to R = 0.6 has little effect on either the hadron jet or parton shower jet distributions, and hence does not have a significant effect on the calculated NP correction factors.

Because the populations of each kind of jet remain extremely close, the value of the NP correction stays ≈ 1 for all the rapidity bins and does not vary enough to be significant against the statistical fluctuations, particularly for p_T values between 100 and 300 GeV.

Around 75 GeV there is large fluctuation between rapidity bins, which may be related to the sudden drop in both hadron and parton shower jet populations in this same region, since the lower (pre-weighting) populations in these bins give rise to greater statistical variation. Similar behaviour occurs again above 350 GeV, where it can also be understood as the result of the lower populations this high above the leading $\hat{p}_T$ of the JZ5W slice. These fluctuations vary the NP factor widely both above and below 1 and do not produce an overall trend.

This effect might be mitigated by including the JZ0W slice ($\hat{p}_T$ = 0 GeV) in the analysis, so this is recommended for further study.

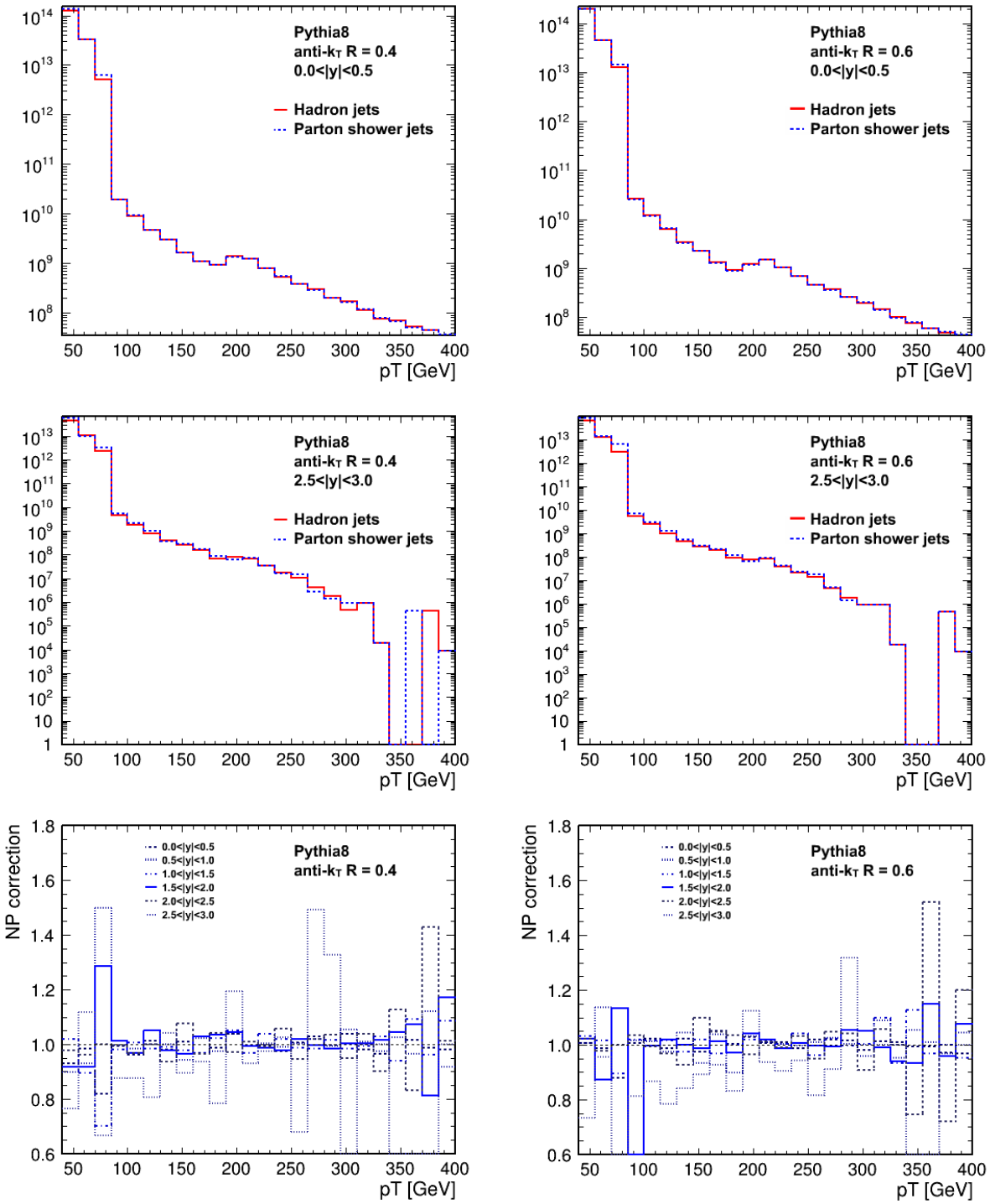

Figure 1: Results from the NP correction factor analysis of events from PYTHIA8 using anti-k_t R = 0.4 and R = 0.6 on the left and right, respectively. At the top are plots of the inclusive jet cross sections for hadron-level jets overlaid on parton shower jets in the central region ($0.0 < |y| < 0.5$) and below these are the same plots for the forward region ($2.5 < |y| < 3.0$). At the bottom are the NP corrections for each of the six rapidity bins. Additional separate plots for each of the rapidity bins are available online at [19].

3.2 PS correction factor from PYTHIA8 samples

Fig. 2 contains plots of the parton shower jet and hard parton jet cross sections for anti-k_t parameters R = 0.4 and R = 0.6 in the central region ($0.0 < |y| < 0.5$) and in the forward region ($2.5 < |y| < 3.0$), from the analysis of the events produced at LO using PYTHIA8. A logarithmic scale is used again for the y-axes to make the plot clear. Also shown is the calculated PS correction factor within each of the six rapidity bins.

The region around 75 GeV behaves similarly as in Fig. 1. The smooth rise in jet populations with decreasing p_T flattens off around 200 GeV and then rises steeply below 100 GeV.

While coding and debugging this analysis, it was noticed that, whereas the events within the other JZXW slices each involve 2 or 3 hard partons with status code 23 which can be clustered into jets, the events from the JZ1W slice did not.

An initial, superficial investigation (by printing particle information to the screen) yielded the discovery that although the events in this slice lacked hard partons with status code 23, they appeared to have 2 or 3 relatively hard ($p_T \approx 50$ GeV) partons with status 33, which denotes them as outgoing particles from the initial-state showers [23].

Collecting these particles into jets for the JZ1 slice smoothed out this section of the combined p_T distributions, but it could not be determined why the event records for this particular slice do not preserve the initial hard partons.

Since status 33 partons are produced by the early stages of the parton shower, they are not an appropriate population to measure the LO cross section, which is defined in the absence of parton shower effects.

Overall, the PS correction factor is shown generally to rise at lower p_T. Below 100 GeV, the parton shower jet populations rise steeply as they do in Fig. 1, but due to the lack of status 23 partons in the low p_T JZ1W slice the measured hard parton jet cross section does not rise in the same way, and the resulting PS correction factor becomes extremely large, ($\sim 10^4$).

A difference in behaviour is shown with different anti-k_t parameters. For R = 0.6 the rise of the PS correction factor with lower p_T becomes steeper than with R = 0.4. This is similar to the results obtained in [10] and [11] using PYTHIA6.

3.3 NP correction factor from POWHEG1+PYTHIA8 samples

Fig. 3 shows the results of the NP correction factor analysis at NLO using POWHEG1 interfaced with PYTHIA8 in a similar manner to Figs. 1 and 2.

The difference between rapidity ranges is apparent in that the forward region ($2.5 < |y| < 3.0$) displays much fewer jets at high p_T than the central region, as seen before by [7], [10] and [11]. This behaviour is to be expected because a particle with a given energy will have a smaller component of its momentum perpendicular to the beam axis when it has a higher absolute rapidity.

The jet distributions are smooth curves, unlike the PYTHIA8 distributions which contained discontinuous behaviour due to the use and (potentially error-prone) combination of p_T slices.

There is very slight statistical fluctuation in the measure NP correction factor around the central value of 1. This variation becomes more severe at higher $p_T$where there are fewer jets and slight differences in population become more significant.

The large statistical variation makes comparison with the results of [7] and [9] difficult, as these found the NP correction factor to rise by only ~0.1 above 1 at low p_T. However, the result obtained here is consistent with the possibility of reproducing those measurements with a larger sample size.

3.4 PS correction factor from POWHEG1+PYTHIA8 samples

Fig. 4 shows the results of the PS correction factor analysis at NLO using POWHEG1 interfaced with PYTHIA8, arranged similarly to the previous figures.

The suppression of high $p_T$jets in the forward region appears again, and for the same reasons.

The PS correction factor is found to rise above 1 at low $p_T$in a smooth curve which drops to 1 at around 100 GeV and flatten out, staying below 1 for high p_T. This is very similar to the PS correction measurement obtained by [7] for POWHEG1 interfaced with PYTHIA6.

Also similar to [7] is the tendency for the PS correction factor to drop further below 1 in the forward region.

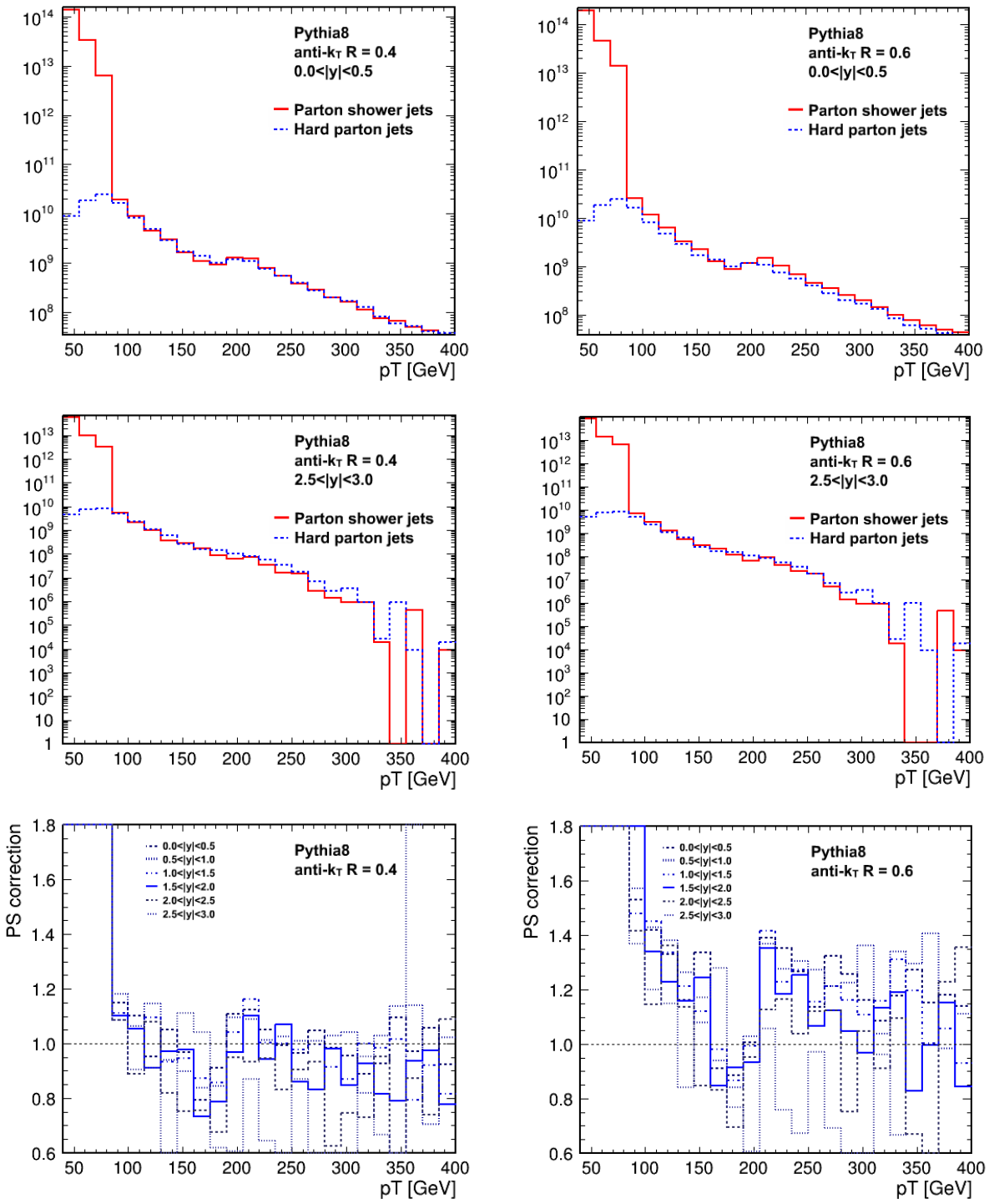

Figure 2: Results from the PS correction factor analysis of events from PYTHIA8 using anti-k_t R = 0.4 and R = 0.6 on the left and right, respectively. At the top are plots of the inclusive jet cross sections for parton shower jets overlaid on hard parton jets in the central region ($0.0 < |y| < 0.5$) and below these are the same plots for the forward region ($2.5 < |y| < 3.0$). At the bottom are the PS corrections for each of the six rapidity ranges. Additional separate plots for each of the rapidity bins are available online at [19].

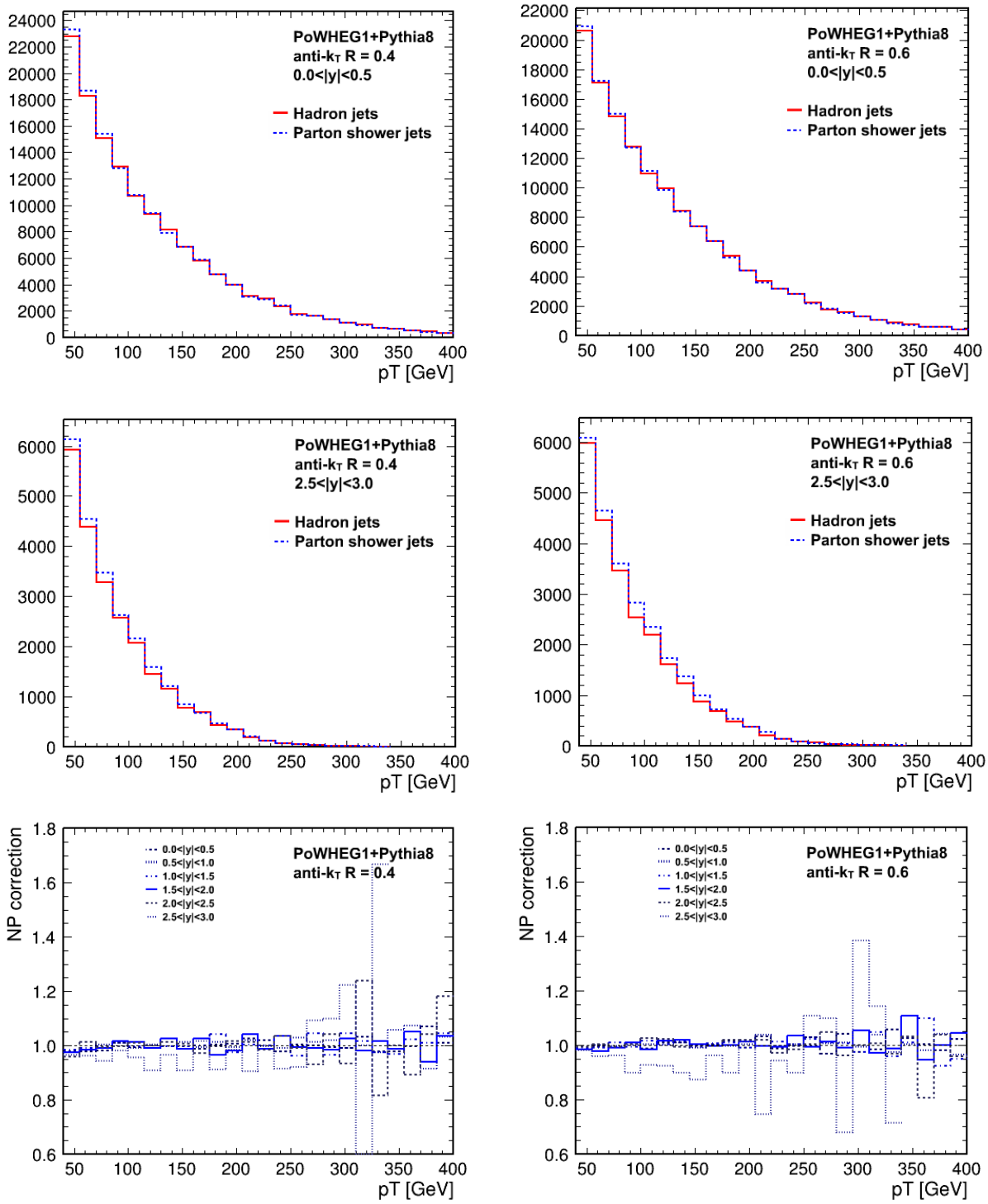

Figure 3: Results from the NP correction factor analysis of events from POWHEG1+PYTHIA8 using anti-k_t R = 0.4 and R = 0.6 on the left and right, respectively. At the top are plots of the inclusive jet cross sections for hadron-level jets overlaid on parton shower jets in the central region ($0.0 < |y| < 0.5$) and below these are the same plots for the forward region ($2.5 < |y| < 3.0$). At the bottom are the NP corrections for each of the six rapidity ranges. Additional separate plots for each of the rapidity bins are available online at [19].

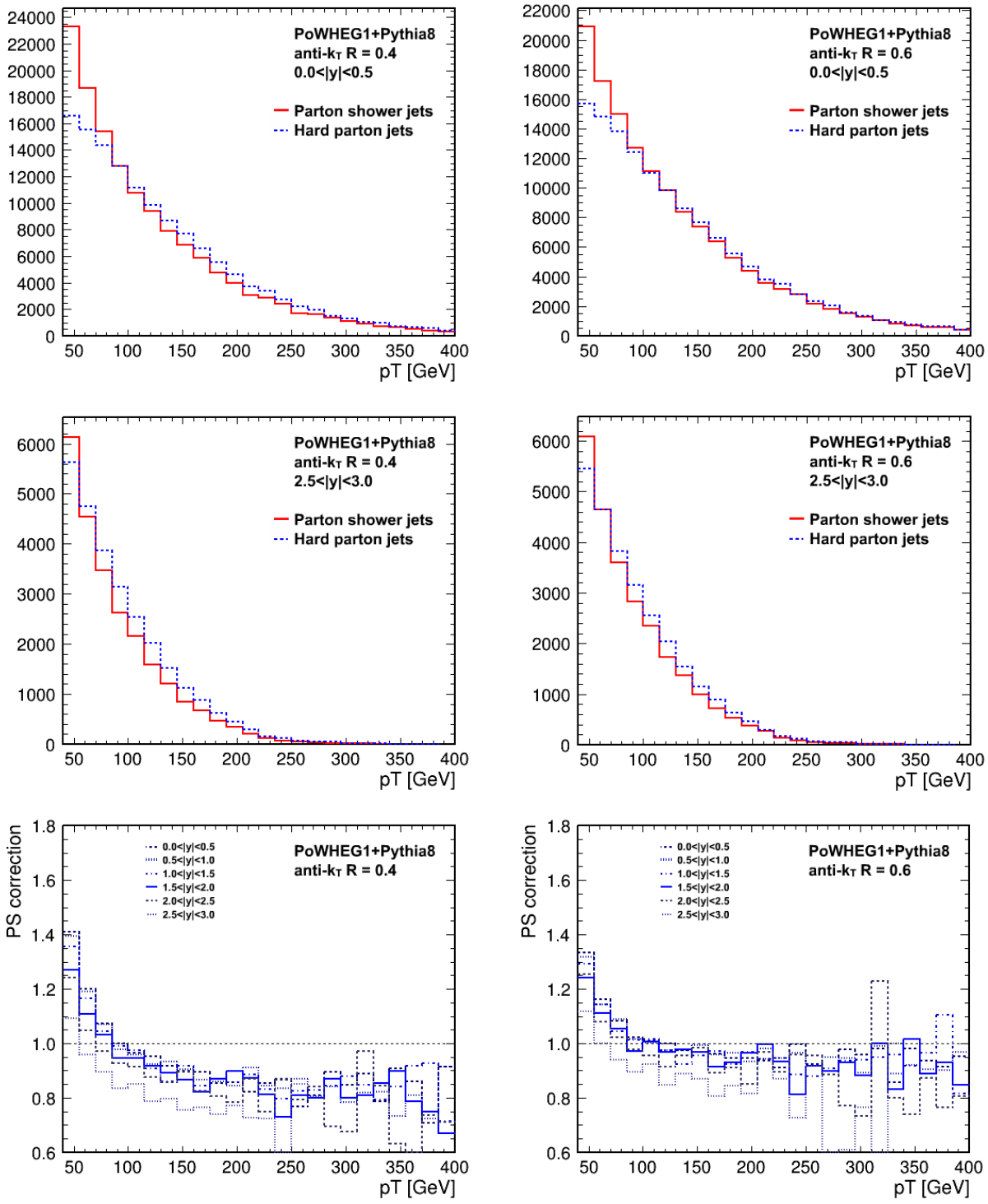

Figure 4: Results from the PS correction factor analysis of events from POWHEG1+PYTHIA8 using anti-k_t R = 0.4 and R = 0.6 on the left and right, respectively. At the top are plots of the inclusive jet cross sections for parton shower jets overlaid on hard parton jets in the central region ($0.0 < |y| < 0.5$) and below these are the same plots for the forward region ($2.5 < |y| < 3.0$). At the bottom are the PS corrections for each of the six rapidity ranges. Additional separate plots for each of the rapidity bins are available online at [19].

The anti-k_t R parameter is shown to have an effect on the PS correction factor, in that for R = 0.6 the distribution is flatter (closer to 1 at all p_T) than for R = 0.4.

This is potentially because the larger jet radius means more low p_T particles are collected into the parton shower jets, raising the overall jet's p_T, so the entire parton shower jet distribution shifts towards higher p_T, raising the measured PS factor closer to 1 at high p_T and mitigating its rise at low p_T.

3.5 NP correction factor from POWHEG2+PYTHIA8 samples

Fig. 5 shows the results of the NP correction factor analysis at NLO using POWHEG2 interfaced with PYTHIA8.

The similarity between Figs. 3 and 5 demonstrate a similarity between POWHEG versions 1 and 2. The measured NP correction factors for the two generators are both so close to 1 that the random variations are larger than any discernible trend and the anti-k_t R parameter appears to have as little effect on the NP factor in POWHEG2 as it does in POWHEG1.

There are not expected to be major deviations between the NP correction factor measured in POWHEG1 and that of POWHEG2, since this is defined in terms of the parton shower and hadronisation, both of which are handled by PYTHIA8.

Any difference between POWHEG1 and POWHEG2 may become apparent in a comparison of their PS correction factors.

3.6 PS correction factor from POWHEG2+PYTHIA8 samples

Fig. 6 gives plots for the analysis which measured the PS correction factor at NLO in POWHEG2 interfaced with PYTHIA8 with the same layout as the previous figures.

The results from POWHEG2 match those from POWHEG1 extremely closely (both in terms of the jet cross section shapes and the general variation of the PS correction factor with p_T). However, there is one significant difference between their behaviours in the low $p_T < 100$ GeV region.

In the POWHEG1 results, the hard parton jet cross section drops away from the parton shower jet cross section only slightly. They both continue to rise at low p_T, but the hard parton cross section rises less steeply.

By contrast, in this $p_T$region the hard parton jet cross section actually *decreases* in the events from POWHEG2. This results in an even steeper rise in the PS correction factor at low p_T.

The PS correction factor from POWHEG2 rises to ~1.5 at 50 GeV, whereas the factor from POWHEG1 only reaches 1.4, an increase of around 7%.

However, above 100 GeV the behaviour of the PS correction factors from both versions of POWHEG are the same, insofar as the plots produced by this analysis can show.

The measured difference in PS correction at low p_T demonstrates some difference in the handling of the NLO hard scattering process by POWHEG between the two versions.

It is unsurprising that differences would show up in the low p_T region because it is in this regime that non-perturbative effects are most significant, making the higher order terms calculated at NLO more significant and so any difference in POWHEG's handling of these terms becomes more noticeable.

4 Conclusion

Three generators (PYTHIA8, POWHEG1 and POWHEG2) have been investigated and their NP and PS correction factors determined.

Within the statistical variation in the results, the factors obtained were found to be in agreement with the values measured previously for earlier versions of these generators (PYTHIA6 and POWHEG1).

A difference was found between POWHEG versions 1 and 2 in the PS correction factor at low p_T: it is slightly larger in POWHEG2 than in POWHEG1.

It is recommended that these analyses be repeated with a larger sample size to reduce the statistical uncertainties and make more precise comparisons possible. Additionally, it is recommended that all eight JZXW slices for the PYTHIA8 samples be analysed and the absence of hard partons in the JZ1W slice should be investigated further.

5 Acknowledgements

Thanks are due to Ben Turner and Yuchen Chen for their provision of the POWHEG2 sample analysed here, to Alex Martyniuk and Stefan Richter for providing technical assistance with the UCL HEP computing systems, and to Mario Campanelli for supervising the project.

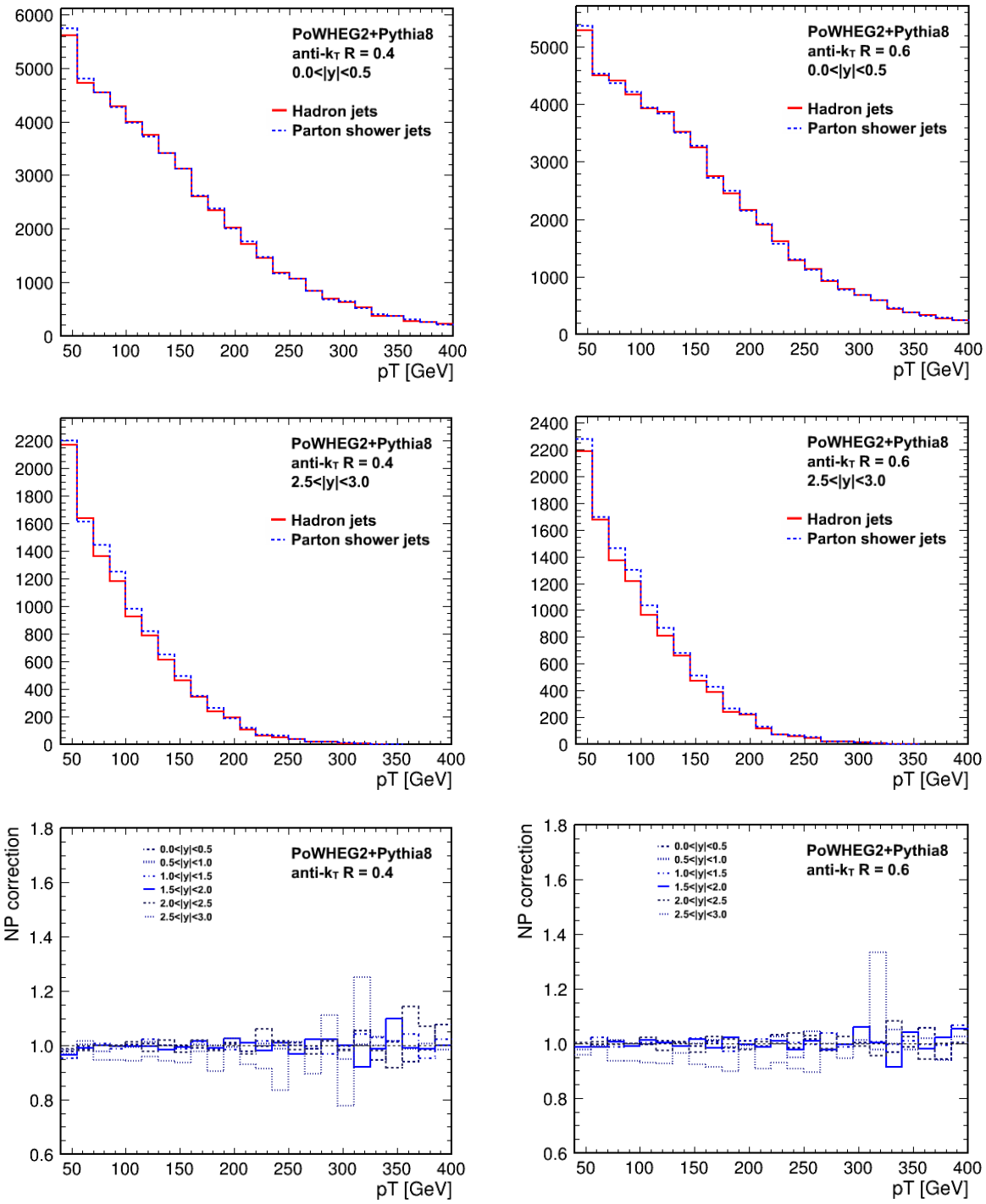

Figure 5: Results from the NP correction factor analysis of events from POWHEG2+PYTHIA8 using anti-k_t R = 0.4 and R = 0.6 on the left and right, respectively. At the top are plots of the inclusive jet cross sections for hadron-level jets overlaid on parton shower jets in the central region ($0.0 < |y| < 0.5$) and below these are the same plots for the forward region ($2.5 < |y| < 3.0$). At the bottom are the NP corrections for each of the six rapidity ranges. Additional separate plots for each of the rapidity bins are available online at [19].

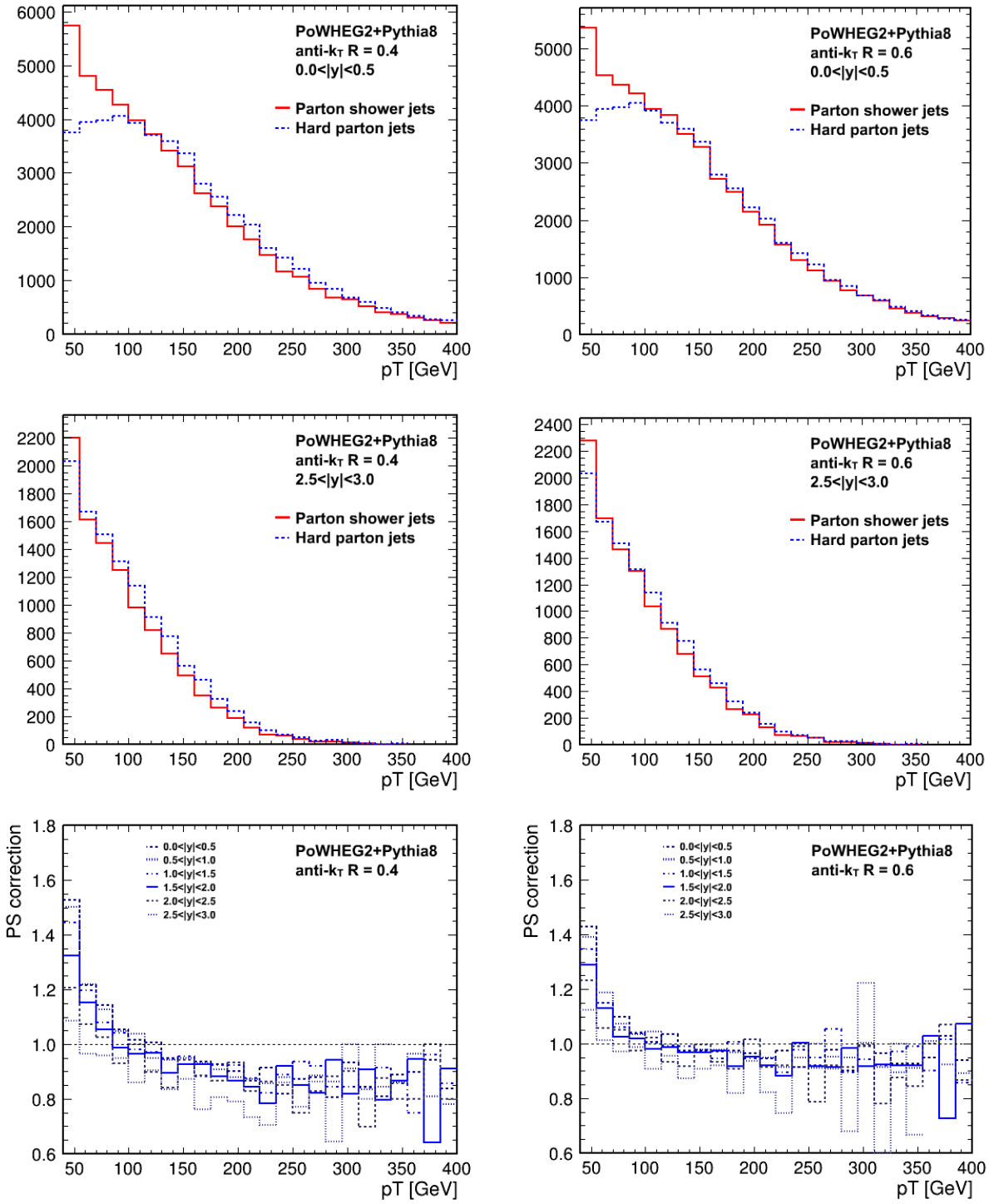

Figure 6: Results from the PS correction factor analysis of events from POWHEG2+PYTHIA8 using anti-k_t R = 0.4 and R = 0.6 on the left and right, respectively. At the top are plots of the inclusive jet cross sections for parton shower jets overlaid on hard parton jets in the central region ($0.0 < |y| < 0.5$) and below these are the same plots for the forward region ($2.5 < |y| < 3.0$). At the bottom are the PS corrections for each of the six rapidity ranges. Additional separate plots for each of the rapidity bins are available online at [19].

Appendix 1: RIVET source code for the PS analysis

This RIVET analysis collects the hard partons from the initial scattering process as well as the results of the parton shower into jets using the anti-k_t algorithm and plots their p_T distributions in different rapidity bins.
An electronic copy is available online at **www.ucl.ac.uk/~zcaph29/jets**

PS_correction.cc

```
// C++
// by James Cowley, UCL
// plots pT distributions of hard partons and jets from parton shower

#include "Rivet/Analyses/MC_JetAnalysis.hh"
#include "Rivet/Projections/FinalState.hh"
#include "Rivet/Projections/FastJets.hh"
#include "Rivet/Tools/ParticleIdUtils.hh"
#include "HepMC/GenParticle.h"

namespace Rivet {
  using namespace Cuts;

  class PS_correction : public Analysis {

  public:
    int evNum;       // counter for tracking # of events
    double maxR;     // jet clustering R parameter
    double pTmin;    // minimum pT for all analysed jets
    double pTmax;    // maximum pT for analysed jets
    double maxAbsY;  // maximum rapidity cut
    string evType;   // type of event to analyse

    PS_correction() : Analysis("PS_correction"){}

    // ********************************************************************************************
    //  INITIALIZE ANALYSIS
    // ********************************************************************************************
    void init() {
      evNum   = 0;
      maxR    = 0.6;
      pTmin   = 25*GeV;
      pTmax   = 400;
      maxAbsY = 3.0;
      evType  = "HAS_PARTONS";

      // histogram properties
      string xl = "pT [GeV]";                    // x axis label
      string yp = "# of hard partons";           // y axis label for hard partons
      string yh = "# of parton shower jets";     // y axis label for parton shower jets
      int nb = 25;                               // number of bins (25*15 = 375)
      double binMin = 25;                        // minimum pT bin edge
      double binMax = 400;                       // maximum pT bin edge

      std::ostringstream rtos;
      rtos << std::setprecision(1) << maxR;
      string rt = rtos.str();                    // rparameter string for titles
      string rtp = "Hard Parton pT (R = "+rt+")";  // histogram title for hard parton jet pT
      string rth = "Parton Jet pT (R = "+rt+")";   // histogram title for parton shower jet pT
      string nmh = "END_SHOWER_PARTON_JET_pT";     // histogram name for parton shower jet pT
      string nmp = "HARD_PARTON_pT";               // histogram name for hard parton jet pT

      // hard parton histos:
      _h_pjetpt       = bookHisto1D(nmp,        nb, binMin, binMax, rtp+" (all y)",       xl, yp);
      _h_pjetpt_00_05 = bookHisto1D(nmp+"_0.5", nb, binMin, binMax, rtp+" (0.0<|y|<0.5)", xl, yp);
      _h_pjetpt_05_10 = bookHisto1D(nmp+"_1.0", nb, binMin, binMax, rtp+" (0.5<|y|<1.0)", xl, yp);
      _h_pjetpt_10_15 = bookHisto1D(nmp+"_1.5", nb, binMin, binMax, rtp+" (1.0<|y|<1.5)", xl, yp);
      _h_pjetpt_15_20 = bookHisto1D(nmp+"_2.0", nb, binMin, binMax, rtp+" (1.5<|y|<2.0)", xl, yp);
      _h_pjetpt_20_25 = bookHisto1D(nmp+"_2.5", nb, binMin, binMax, rtp+" (2.0<|y|<2.5)", xl, yp);
      _h_pjetpt_25_30 = bookHisto1D(nmp+"_3.0", nb, binMin, binMax, rtp+" (2.5<|y|<3.0)", xl, yp);

      // parton shower histos:
      _h_hjetpt       = bookHisto1D(nmh,        nb, binMin, binMax, rth+" (all y)",       xl, yh);
      _h_hjetpt_00_05 = bookHisto1D(nmh+"_0.5", nb, binMin, binMax, rth+" (0.0<|y|<0.5)", xl, yh);
      _h_hjetpt_05_10 = bookHisto1D(nmh+"_1.0", nb, binMin, binMax, rth+" (0.5<|y|<1.0)", xl, yh);
      _h_hjetpt_10_15 = bookHisto1D(nmh+"_1.5", nb, binMin, binMax, rth+" (1.0<|y|<1.5)", xl, yh);
      _h_hjetpt_15_20 = bookHisto1D(nmh+"_2.0", nb, binMin, binMax, rth+" (1.5<|y|<2.0)", xl, yh);
      _h_hjetpt_20_25 = bookHisto1D(nmh+"_2.5", nb, binMin, binMax, rth+" (2.0<|y|<2.5)", xl, yh);
      _h_hjetpt_25_30 = bookHisto1D(nmh+"_3.0", nb, binMin, binMax, rth+" (2.5<|y|<3.0)", xl, yh);
    }

    // ********************************************************************************************
    //  ANALYSE EVENT
    // ********************************************************************************************
    void analyze(const Event& event) {
      double weight = 1; // alternatively = event.weight();
      evNum++;
      if(evNum > 250){ cout<<"'"<<evType<<"' analysis, R = "<<maxR<<", pTmin = "<<pTmin<<", event #"<<evNum<<endl; }

      // ********************************************************************************************
      //  COLLECT PARTONS
      // ********************************************************************************************
      std::vector<HepMC::GenParticle*> allParticles = particles(event.genEvent());
      Particles hardPartons, endPartons;
```

```
085
086       // get the partons
087       foreach(GenParticle* gp, allParticles){
088         int thisStatus = gp->status();
089
090         if(thisStatus == 23){ // hard partons
091           Particle p(gp);
092           hardPartons.push_back(p);
093         }
094
095         if(60 < thisStatus && thisStatus < 70){ // parton shower partons
096           Particle p(gp);
097           bool isEndParton = true;
098           Particles pcs = p.children();
099           foreach(Particle c, pcs){
100             double cStat = c.genParticle()->status();
101             if(60 < cStat && cStat < 70){
102               isEndParton = false;
103               break;
104             }
105           }
106           if(isEndParton){ endPartons.push_back(p); }
107         }
108       }
109
110       // ***************************************************************************************************
111       //  CLUSTER PARTON JETS WITH ANTI-KT
112       // ***************************************************************************************************
113       FastJets hardPartonJets(FastJets::ANTIKT, maxR); // apply anti-kT to hard partons
114       FastJets softPartonJets(FastJets::ANTIKT, maxR); // apply anti-kT to partons from shower
115       hardPartonJets.calc(hardPartons);  // jets from hard partons
116       softPartonJets.calc(endPartons);   // jets after parton shower
117
118       // apply cuts to jets
119       Jets cutHardPartonJets = hardPartonJets.jetsByPt(pTmin, pTmax, -maxAbsY, maxAbsY);
120       Jets cutSoftPartonJets = softPartonJets.jetsByPt(pTmin, pTmax, -maxAbsY, maxAbsY);
121
122       // ***************************************************************************************************
123       //  PLOT HARD PARTON JET PROPERTIES
124       // ***************************************************************************************************
125       foreach(Jet hj, cutHardPartonJets){
126         double thispT = hj.pT();
127         double absY = hj.absrapidity();
128
129         // populate hard parton jet pT histograms:
130         _h_pjetpt->fill(thispT, weight);
131              if (absY < 0.5){ _h_pjetpt_00_05->fill(thispT, weight); }
132         else if (absY < 1.0){ _h_pjetpt_05_10->fill(thispT, weight); }
133         else if (absY < 1.5){ _h_pjetpt_10_15->fill(thispT, weight); }
134         else if (absY < 2.0){ _h_pjetpt_15_20->fill(thispT, weight); }
135         else if (absY < 2.5){ _h_pjetpt_20_25->fill(thispT, weight); }
136         else if (absY < 3.0){ _h_pjetpt_25_30->fill(thispT, weight); }
137       } // end of parton jet loop
138
139
140       // ***************************************************************************************************
141       //  PLOT PARTON SHOWER JET PROPERTIES
142       // ***************************************************************************************************
143       foreach(Jet sj, cutSoftPartonJets){
144         double thispT = sj.pT();
145         double absY = sj.absrapidity();
146
147         // populate parton shower jet pT histograms:
148         _h_hjetpt->fill(thispT, weight);
149              if (absY < 0.5){ _h_hjetpt_00_05->fill(thispT, weight); }
150         else if (absY < 1.0){ _h_hjetpt_05_10->fill(thispT, weight); }
151         else if (absY < 1.5){ _h_hjetpt_10_15->fill(thispT, weight); }
152         else if (absY < 2.0){ _h_hjetpt_15_20->fill(thispT, weight); }
153         else if (absY < 2.5){ _h_hjetpt_20_25->fill(thispT, weight); }
154         else if (absY < 3.0){ _h_hjetpt_25_30->fill(thispT, weight); }
155       } // end of hadron jet loop
156
157     } // end of analyze() function
158
159     // ***************************************************************************************************
160     //  FINALISE EVENT
161     // ***************************************************************************************************
162     void finalize() {
163       // normalization would happen here
164     }
165
166   private:
167     // parton jet histos:
168     Histo1DPtr _h_pjetpt;
169     Histo1DPtr _h_pjetpt_00_05; Histo1DPtr _h_pjetpt_05_10; Histo1DPtr _h_pjetpt_10_15;
170     Histo1DPtr _h_pjetpt_15_20; Histo1DPtr _h_pjetpt_20_25; Histo1DPtr _h_pjetpt_25_30;
171
172     // hadron jet histos:
173     Histo1DPtr _h_hjetpt;
174     Histo1DPtr _h_hjetpt_00_05; Histo1DPtr _h_hjetpt_05_10; Histo1DPtr _h_hjetpt_10_15;
175     Histo1DPtr _h_hjetpt_15_20; Histo1DPtr _h_hjetpt_20_25; Histo1DPtr _h_hjetpt_25_30;
176   };
177
178   // The hook for the plugin system
179   DECLARE_RIVET_PLUGIN(PS_correction);
180 }
```

Appendix 2: RIVET source code for the NP analysis

This RIVET analysis collects the results of the parton shower as well as the final state hadrons into jets using the anti-k_t algorithm and plots their p_T distributions in different rapidity bins. An electronic copy is available online at **www.ucl.ac.uk/~zcaph29/jets**

NP_correction.cc

```
// C++
// by James Cowley, UCL
// plots pT distributions of hadron and parton jets

#include "Rivet/Analyses/MC_JetAnalysis.hh"
#include "Rivet/Projections/FinalState.hh"
#include "Rivet/Projections/FastJets.hh"
#include "Rivet/Tools/ParticleIdUtils.hh"
#include "HepMC/GenParticle.h"

namespace Rivet {
  using namespace Cuts;

  class NP_correction : public Analysis {

  public:
    int evNum;       // counter for tracking # of events
    double maxR;     // jet clustering R parameter
    double pTmin;    // minimum pT for all analysed jets
    double pTmax;    // maximum pT for analysed jets
    double maxAbsY;  // maximum rapidity cut
    string evType;   // type of event to analyse

    NP_correction() : Analysis("NP_correction"){}

    // ********************************************************************************************************
    //  INITIALIZE ANALYSIS
    // ********************************************************************************************************
    void init() {

      evNum   = 0;
      maxR    = 0.6;
      pTmin   = 25*GeV;
      pTmax   = 400;
      maxAbsY = 3.0;
      evType  = "HAS_PARTONS";

      // histogram properties
      string xl = "pT [GeV]";                 // x axis label
      string yp = "# of parton shower jets";  // y axis label for partons
      string yh = "# of hadron jets";         // y axis label for jets
      int nb = 25;                            // number of bins
      double binMin = 25;                     // minimum pT bin edge
      double binMax = 400;                    // maximum pT bin edge

      std::ostringstream rtos;
      rtos << std::setprecision(1) << maxR;
      string rt = rtos.str();                       // rparameter string for titles
      string rtp = "Parton Jet pT (R = "+rt+")";    // histogram title for parton jet pT
      string rth = "Hadron Jet pT (R = "+rt+")";    // histogram title for hadron jet pT
      string nmh = "HADRON_JET_pT";                 // histogram name for hadron jet pT
      string nmp = "PARTON_SHOWER_JET_pT";          // histogram name for parton jet pT

      // parton histos:
      _h_pjetpt       = bookHisto1D(nmp,        nb, binMin, binMax, rtp+" (all y)",       xl, yp);
      _h_pjetpt_00_05 = bookHisto1D(nmp+"_0.5", nb, binMin, binMax, rtp+" (0.0<|y|<0.5)", xl, yp);
      _h_pjetpt_05_10 = bookHisto1D(nmp+"_1.0", nb, binMin, binMax, rtp+" (0.5<|y|<1.0)", xl, yp);
      _h_pjetpt_10_15 = bookHisto1D(nmp+"_1.5", nb, binMin, binMax, rtp+" (1.0<|y|<1.5)", xl, yp);
      _h_pjetpt_15_20 = bookHisto1D(nmp+"_2.0", nb, binMin, binMax, rtp+" (1.5<|y|<2.0)", xl, yp);
      _h_pjetpt_20_25 = bookHisto1D(nmp+"_2.5", nb, binMin, binMax, rtp+" (2.0<|y|<2.5)", xl, yp);
      _h_pjetpt_25_30 = bookHisto1D(nmp+"_3.0", nb, binMin, binMax, rtp+" (2.5<|y|<3.0)", xl, yp);

      // hadron jet histos:
      _h_hjetpt       = bookHisto1D(nmh,        nb, binMin, binMax, rth+" (all y)",       xl, yh);
      _h_hjetpt_00_05 = bookHisto1D(nmh+"_0.5", nb, binMin, binMax, rth+" (0.0<|y|<0.5)", xl, yh);
      _h_hjetpt_05_10 = bookHisto1D(nmh+"_1.0", nb, binMin, binMax, rth+" (0.5<|y|<1.0)", xl, yh);
      _h_hjetpt_10_15 = bookHisto1D(nmh+"_1.5", nb, binMin, binMax, rth+" (1.0<|y|<1.5)", xl, yh);
      _h_hjetpt_15_20 = bookHisto1D(nmh+"_2.0", nb, binMin, binMax, rth+" (1.5<|y|<2.0)", xl, yh);
      _h_hjetpt_20_25 = bookHisto1D(nmh+"_2.5", nb, binMin, binMax, rth+" (2.0<|y|<2.5)", xl, yh);
      _h_hjetpt_25_30 = bookHisto1D(nmh+"_3.0", nb, binMin, binMax, rth+" (2.5<|y|<3.0)", xl, yh);
    }

    // ********************************************************************************************************
    //  ANALYSE EVENT
    // ********************************************************************************************************
    void analyze(const Event& event) {
      double weight = 1; // alternatively = event.weight();
      evNum++;
      if(evNum > 250){ cout<<"'"<<evType<<"' analysis, R = "<<maxR<<", pTmin = "<<pTmin<<", event #"<<evNum<<endl; }

      // ********************************************************************************************************
      //  COLLECT ALL END-OF-SHOWER PARTONS
      // ********************************************************************************************************
      std::vector<HepMC::GenParticle*> allParticles = particles(event.genEvent());
      Particles endPartons;

```

```
087       // get the partons from the parton shower
088       foreach(GenParticle* gp, allParticles){
089         int thisStatus = gp->status();
090
091         // select partons from end of parton shower, before hadronisation
092         if(60 < thisStatus && thisStatus < 70){
093           Particle p(gp);
094
095           // need to prevent double counting of ancestors e.g. 61->63
096           bool isEndParton = true;
097           Particles pcs = p.children();
098           foreach(Particle c, pcs){
099             double cStat = c.genParticle()->status();
100             if(60 < cStat && cStat < 70){
101               isEndParton = false;
102               break;
103             }
104           }
105           if(isEndParton){ endPartons.push_back(p); }
106         }
107       }
108
109       // ***************************************************************************************************
110       //  CLUSTER PARTON & HADRON JETS WITH ANTI-KT
111       // ***************************************************************************************************
112       FinalState fs(-maxAbsY, maxAbsY);       // get final state particles using FinalState projection
113       fs.project(event);
114       Particles endHadrons = fs.particles();
115
116       // apply anti-kT to particles
117       FastJets allPartonJets(FastJets::ANTIKT, maxR);
118       FastJets allHadronJets(FastJets::ANTIKT, maxR);
119       allPartonJets.calc(endPartons);
120       allHadronJets.calc(endHadrons);
121
122       // apply cuts to jets
123       Jets cutPartonJets = allPartonJets.jetsByPt(pTmin, pTmax, -maxAbsY, maxAbsY);
124       Jets cutHadronJets = allHadronJets.jetsByPt(pTmin, pTmax, -maxAbsY, maxAbsY);
125
126       // ***************************************************************************************************
127       //  PLOT PARTON JET PROPERTIES
128       // ***************************************************************************************************
129       foreach(Jet pj, cutPartonJets){
130         double thispT = pj.pT();
131         double absY = pj.absrapidity();
132
133         // populate parton jet pT histograms:
134         _h_pjetpt->fill(thispT, weight);
135               if (absY < 0.5){ _h_pjetpt_00_05->fill(thispT, weight); }
136         else if (absY < 1.0){ _h_pjetpt_05_10->fill(thispT, weight); }
137         else if (absY < 1.5){ _h_pjetpt_10_15->fill(thispT, weight); }
138         else if (absY < 2.0){ _h_pjetpt_15_20->fill(thispT, weight); }
139         else if (absY < 2.5){ _h_pjetpt_20_25->fill(thispT, weight); }
140         else if (absY < 3.0){ _h_pjetpt_25_30->fill(thispT, weight); }
141       } // end of parton jet loop
142
143       // ***************************************************************************************************
144       //  PLOT HADRON JET PROPERTIES
145       // ***************************************************************************************************
146       foreach(Jet hj, cutHadronJets){
147         double thispT = hj.pT();
148         double absY = hj.absrapidity();
149
150         // populate hadron jet pT histograms:
151         _h_hjetpt->fill(thispT, weight);
152         if (absY < 0.5){ _h_hjetpt_00_05->fill(thispT, weight); }
153         else if (absY < 1.0){ _h_hjetpt_05_10->fill(thispT, weight); }
154         else if (absY < 1.5){ _h_hjetpt_10_15->fill(thispT, weight); }
155         else if (absY < 2.0){ _h_hjetpt_15_20->fill(thispT, weight); }
156         else if (absY < 2.5){ _h_hjetpt_20_25->fill(thispT, weight); }
157         else if (absY < 3.0){ _h_hjetpt_25_30->fill(thispT, weight); }
158
159       } // end of hadron jet loop
160     } // end of analyze() function
161
162     // ***************************************************************************************************
163     //  DEFINE FUNCTION TO COMPARE JETS BY R
164     // ***************************************************************************************************
165     double jetDeltaR(Jet jet1, Jet jet2){
166       double deltaY = jet1.rapidity() - jet2.rapidity();
167       double deltaPhi = jet1.phi() - jet2.phi();
168       double deltaR = sqrt(deltaY*deltaY + deltaPhi*deltaPhi);
169       return deltaR;
170     }
171
172     // ***************************************************************************************************
173     //  DEFINE FUNCTION TO DECIDE WHETHER TO ANALYSE AN EVENT
174     // ***************************************************************************************************
175     bool shouldAnalyse(Particles inputPartons, string need){
176       bool hasP = inputPartons.size() != 0;
177       if(need == "HAS_PARTONS" && hasP){ return true; }
178       else if (!hasP){ return false; }
179
180       bool hasG = false; // has gluons
181       bool hasQ = false; // has quarks
182       bool hasC = false; // has c quark
183       bool hasB = false; // has b quark
184       bool gOnly = inputPartons.size() > 0; // has only gluons (i.e. no quarks, unless no partons at all)
```

```
185       bool qOnly = inputPartons.size() > 0; // has only quarks (i.e. no gluons, unless no partons at all)
186
187       // decide the above booleans based on tests of all partons
188       foreach(Particle p, inputPartons){
189         int id = p.pdgId();
190         if(PID::isQuark(id)){
191           hasQ = true;
192           gOnly = false;
193                if(abs(id) == PID::CQUARK){ hasC = true; }
194           else if(abs(id) == PID::BQUARK){ hasB = true; }
195         }
196         else if(id == PID::GLUON){
197           hasG = true;
198           qOnly = false;
199         }
200       }
201
202       // test if the criterion of interest ("need") has been met
203       bool result =
204         (need == "HAS_GLUONS"  && hasG)  ||
205         (need == "HAS_QUARKS"  && hasQ)  ||
206         (need == "HAS_CQUARKS" && hasC)  ||
207         (need == "HAS_BQUARKS" && hasB)  ||
208         (need == "GLUONS_ONLY" && gOnly) ||
209         (need == "QUARKS_ONLY" && qOnly) ;
210
211       return result;
212     }
213
214     // ********************************************************************************************************
215     //  FINALISE EVENT
216     // ********************************************************************************************************
217     void finalize() {
218       // normalisation would happen here
219     }
220
221   private:
222     // parton jet histos:
223     Histo1DPtr _h_pjetpt;
224     Histo1DPtr _h_pjetpt_00_05; Histo1DPtr _h_pjetpt_05_10; Histo1DPtr _h_pjetpt_10_15;
225     Histo1DPtr _h_pjetpt_15_20; Histo1DPtr _h_pjetpt_20_25; Histo1DPtr _h_pjetpt_25_30;
226
227     // hadron jet histos:
228     Histo1DPtr _h_hjetpt;
229     Histo1DPtr _h_hjetpt_00_05; Histo1DPtr _h_hjetpt_05_10; Histo1DPtr _h_hjetpt_10_15;
230     Histo1DPtr _h_hjetpt_15_20; Histo1DPtr _h_hjetpt_20_25; Histo1DPtr _h_hjetpt_25_30;
231   };
232
233   // The hook for the plugin system
234   DECLARE_RIVET_PLUGIN(NP_correction);
235 }
```

Appendix 3: ROOT macro source code

This ROOT function applies the necessary scaling factors to each of the JZXW slices used in the PYTHIA8 sample analysis. An electronic copy of this and all other ROOT code written for this investigation is available online at **www.ucl.ac.uk/~zcaph29/jets**

```
001 void doallJZ(int minSlice=2, int maxSlice=3, string logopt="nolog", string yax="NPC", string numName="Numerator", string den-
Name="Denominator", string ratName="Ratio", Double_t ratbot, Double_t rattop){
002   cout<<"Getting histograms from JZ slices..."<<endl;
003
004   // cross sections and filter efficiencies from https://twiki.cern.ch/twiki/bin/view/AtlasProtected/JetStudies2012
005   Double_t cros[] = {7.2850*pow(10.0,7.0), 7.2850*pow(10.0,7.0), 2.6359*pow(10.0,4.0), 5.4419*pow(10.0,2.0),
006                      6.4453*pow(10.0,0.0), 3.9740*pow(10.0,-2.0), 4.1609*pow(10.0,-4.0), 4.0636*pow(10.0,-5.0)};
007   Double_t filt[] = {9.8554*pow(10.0,-1.0), 1.2898*pow(10.0,-4.0), 3.9939*pow(10.0,-3.0), 1.2187*pow(10.0,-3.0),
008                      7.0821*pow(10.0,-4.0), 2.1521*pow(10.0,-3.0), 4.6843*pow(10.0,-3.0), 1.4600*pow(10.0,-2.0)};
009
010   string haddArg = "hadd -f mergedJZs.root";
011
012   // go through all desired slices
013   for(int JZi = minSlice; JZi <= maxSlice; JZi++){
014     Double_t thisCros = cros[JZi];
015     Double_t thisFilt = filt[JZi];
016     Double_t thisWeight = thisCros/thisFilt;
017
018     // name input and output .root files:
019     std::ostringstream numStream;
020     numStream << JZi;
021     string inName = "rivetJZ"+numStream.str()+".root";
022     string outName = "weightedJZ"+numStream.str()+".root";
023     haddArg = haddArg+" "+outName;
024
025     // get histos from file
026     std::vector<TH1D*> theseHistosJZ = getHistos(inName);
027
028     // apply scaling factor to the histograms
029     cout<<"Scaling JZ histograms for "<<inName<<"..."<<endl;
030     std::vector<TH1D*> theseScaledJZ = jzScale(theseHistosJZ, thisWeight);
031
032     TFile * thisOut = new TFile(outName.c_str(), "RECREATE");
033     // save scaled histograms to .root file
034     for(int i = 0 ; i < theseScaledJZ.size() ; i++){
035       theseScaledJZ.at(i)->Write();
036     }
037
038     thisOut->Close();
039     cout<<"Saved all '"<<inName<<"' histograms to '"<<outName<<"'"<<endl;
040   }
041
042   cout<<"Merging scaled .root files with '"<<haddArg<<"'..."<<endl;
043   gSystem->Exec(haddArg.c_str());
044   cout<<"Performing division on mergedJZs.root..."<<endl;
045   rundiv("mergedJZs.root", logopt, yax, numName, denName, ratName, ratbot, rattop);
046   cout<<"Here, have your command line back."<<endl;
047 }
048
049 std::vector<TH1D*> jzScale(std::vector<TH1D*> jzHistos, Double_t fac){
050   std::vector<TH1D*> output;
051   for(int i = 0 ; i < jzHistos.size() ; i++){
052     TH1D * thisHisto = new TH1D(*jzHistos.at(i));
053     thisHisto->Scale(fac);
054     output.push_back(thisHisto);
055   }
056   return output;
057 }
```

References

[1] A. Buckley et al., "General-purpose event generators for LHC physics," *Physics Reports,* arXiv:1101.2599, 2011.

[2] T. Sjostrand, S. Mrenna and P. Skands, "PYTHIA 6.4 Physics and Manual," arXiv:hep-ph/0603175, 2006.

[3] M. Bahr et al., "Herwig++ Physics and Manual," *Eur.Phys.J.C58:639-707,2008,* arXiv:0803.0883 [hep-ph], 2008.

[4] C. Oleari, "The POWHEG-BOX," *Nucl.Phys.Proc.Suppl.205-206:36-41,2010,* arXiv:1007.3893 [hep-ph], 2010.

[5] S. Frixione et al., "The MC@NLO 4.0 Event Generator," arXiv:1010.0819 [hep-ph], 2010.

[6] D. J. Gross, "Twenty Five Years of Asymptotic Freedom," *Nucl.Phys.Proc.Suppl. 74 (1999) 426-446,* arXiv:hep-th/9809060, 1998.

[7] CMS Collaboration, "Constraints on parton distribution functions and extraction of the strong coupling constant from the inclusive jet cross section in pp collisions at sqrt(s) = 7 TeV," *Eur. Phys. J. C,* arXiv:1410.6765 [hep-ex], 2014.

[8] S. Dooling, P. Gunnellini, F. Hautmann and H. Jung, "Longitudinal momentum shifts, showering and nonperturbative corrections in matched NLO-shower event generators," *Physical Review D,* arXiv:1212.6164 [hep-ph], 2012.

[9] S. Dooling, P. Gunnellini, F. Hautmann and H. Jung, "Nonperturbative corrections and showering in NLO-matched event generators," arXiv:1304.7180 [hep-ph], 2013.

[10] ATLAS Collaboration, "Measurement of inclusive jet and dijet production in pp collisions at sqrt(s) = 7 TeV using the ATLAS detector," *Phys.Rev. D86 (2012) 014022,* arXiv:1112.6297 [hep-ex], 2011.

[11] CMS Collaboration, "Measurements of differential jet cross sections in proton-proton collisions at sqrt(s) = 7 TeV with the CMS detector," *Phys. Rev. D 87 (2013) 112002,* arXiv:1212.6660 [hep-ex], 2012.

[12] T. Sjostrand, S. Mrenna and P. Skands, "A Brief Introduction to PYTHIA 8.1," *Comput.Phys.Commun.178:852-867,2008,* arXiv:0710.3820 [hep-ph], 2007.

[13] A. Buckley et al., "Rivet user manual," arXiv:1003.0694v8 [hep-ph].

[14] "UCL-HEP," GridPP Wiki, [Online]. Available: https://www.gridpp.ac.uk/wiki/UCL-HEP. [Accessed 26 03 2015].

[15] ATLAS Collaboration, "Summary of ATLAS PYTHIA 8 tunes," *ATL-PHYS-PUB-2012-003,* 2012.

[16] J. Robinson, "Job options file for POWHEG with Pythia8," 14 04 2013. [Online]. Available: https://svnweb.cern.ch/trac/atlasoff/browser/Generators/MC12JobOptions/trunk/share/DSID147xxx/MC12.185491.Powheg_Pythia8_AU2_CT10_Dijet_muR1muF1_ISR_alphaSmatching.py. [Accessed 30 01 2015].

[17] M. Cacciari and G. P. Salam, "Dispelling the N^3 myth for the Kt jet-finder," *Phys.Lett.B641:57-61,2006,* arXiv:hep-ph/0512210v2, 2006.

[18] M. Cacciari, G. P. Salam and G. Soyez, "The anti-k_t jet clustering algorithm," *JHEP 0804:063,2008,* arXiv:0802.1189 [hep-ph], 2008.

[19] J. Cowley, "Jets Project Resources," [Online]. Available: www.ucl.ac.uk/~zcaph29/jets

[20] A. Buckley et al., "The Rivet MC Analysis System: FinalState Class Reference," [Online]. Available: https://rivet.hepforge.org/code/dev/a00182.html. [Accessed 10 3 2015].

[21] Doxygen, "YODA - Yet more Objects for Data Analysis: Main Page – Hepforge," [Online]. Available: https://yoda.hepforge.org/doxy/.

[22] ATLAS, "Jet Studies 2012," CERN, [Online]. Available: https://twiki.cern.ch/twiki/bin/view/AtlasProtected/JetStudies2012.

[23] T. Sjostrand, S. Mrenna and P. Skands, "Pythia8 Particle Properties," [Online]. Available: http://home.thep.lu.se/~torbjorn/pythia81html/ParticleProperties.html#status. [Accessed 10 03 2015].

[24] S. Alioli, K. Hamilton and E. Re, "Practical improvements and merging of POWHEG simulations for vector boson production," arXiv:1108.0909 [hep-ph], 2011.

[25] P. Nason, "Very preliminary study with new generator for ttbar production including NLO corrections to decays," 2014. [Online]. Available: https://indico.cern.ch/event/301787/session/3/contribution/36/material/slides/0.pdf.