A minimally invasive strategy for NNLO event files

Daniel Maître

Institute for Particle Physics Phenomenology , Ogden Centre for Fundamental Physics, Department of Physics, University of Durham, Science Laboratories, South Rd, DURHAM DH1 3LE, UNITED KINGDOM

E-mail: daniel.maitre@durham.ac.uk

Abstract. In this contribution we discuss the extension of the event file format proposed in Ref [1] from NLO to NNLO. We describe a strategy that minimises the changes needed to the NNLO generator code, at the cost of increasing the CPU cost. We expect the CPU cost for the production of the event files to be amortised by reusing the event files in many different analyses.

1. Introduction

Recent advances in the calculation of one-loop amplitudes have made Next-to-Leading Order (NLO) QCD predictions for high multiplicity processes achievable (see e.g. [2] for a review). While the calculations necessary for these predictions are tractable, they come at a high CPU cost. In order to amortise the cost of the matrix element calculation and its integration over the phase-space an event file format has been proposed [1] for NLO processes. The strategy to produce and use these event files is sketched in Fig. 1. The full generator is run for a loose set of cuts and all phase-space and matrix element information is stored in a file (called nTuple file in this contribution). At a later stage the event files can be re-read and effectively replace the generator. This process is much more efficient than running the generator again and much less error prone. In this contribution we will present a strategy to obtain event files for NNLO generators. At NNLO the complexity of the generators is much higher than at NLO and it would be much harder to isolate the information needed for the event files in the code. Instead we will change strategy and try to minimise the code changes needed in the generator.

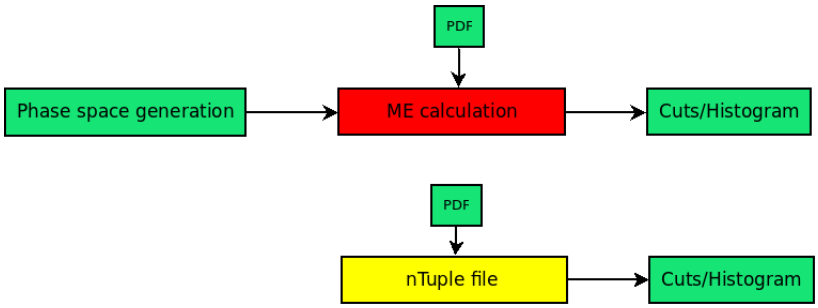

Figure 1. Overview of the production and usage of event files.

2. Advantages and disadvantages of event files

In a typical analysis the majority of the CPU resources are spent on the matrix element calculation. Other aspects of the analysis such as the PDF evaluation, cuts and jet algorithm tend to only take a small fraction of the time. Using event files allows to change details of the analysis after the generation, in particular the re-evaluation of the prediction with different PDF sets or scales would be extremely costly if one had to start the complete generation from the start. Beyond saving computation time this scheme allows researchers unfamiliar with the generator to use its results in a convenient fashion. In this scheme the generator expert have an opportunity to protect the users of their calculation from all the difficulties of running their program on a large scale. A significant effort is needed to validate a generator and ensure it reproduces established results after each change. Event files are a more persistent way of storing the result of a calculation and are easier to validate.

The main disadvantage of the event files is that their size is rather large. The usefulness of nTuples depends on the trade-off between storage and usage convenience. While considering the trade-offs between size and CPU time one should remember that generators have been optimised for CPU time and not optimised to produce the most efficient set of phase-space points form a storage point of view. The efficiency of the storage can generally be increased at the cost of CPU time.

3. Event files for NNLO

The event files have proven their usefulness at NLO, could they be useful NNLO too? The advantages and disadvantages are the same than at NLO, but the trade-offs are starker. While NLO event files only get uncomfortably large at large multiplicities NNLO calculations require a very large amount of phase-space points already at low multiplicities. On the other hand the CPU time for these calculations are much larger than at NLO so the potential savings are larger. To evaluate the potential of event files for NNLO generators a feasibility study [2, 3] was performed in the Les Houches workshop. For this study the authors used EERAD3 [4] to estimate the size of the event files that would be necessary to reproduce exiting results. The study's main result is shown in Fig. 2. The figure shows two storage strategies, trading some re-calculation at read-time for some storage saving, see ref. [2] for more details.

The main take away point of the feasibility study [2] is that event files could still offer a favourable size/CPU time trade off.

3.1. General structure of a NNLO generator

In order to design the least intrusive method of extracting the information we need from the NNLO generator we start form the schematic view displayed in Fig. 3. The calculation starts with the generation of the phase-space, including the parton momentum fractions in the case of hadronic initial state. This information is used in the matrix element calculation to calculate the weight for the event, this includes possible subtraction terms that are used to regulate infrared divergences. Finally the weights calculated are histogrammed according to observables calculated from the phase-space information. In the following we take the case of a hadron-hadron collider, cases with fewer hadronic initial states can be treated in a similar way.

3.2. Extracting the information

The structure shown in Fig. 3 already shows the places in the code where the phase-space information and the weights can be extracted, so that the intermediate step (the most costly part) can be replaced by reading from event files. Extracting information from the generator as it produces the NNLO cross section allows us the get the weight for a given phase-space point for the scale and PDF chosen by the generator at the time of the generation. We would like to have the flexibility to generate the weight as it would have been produced with a different scale choice

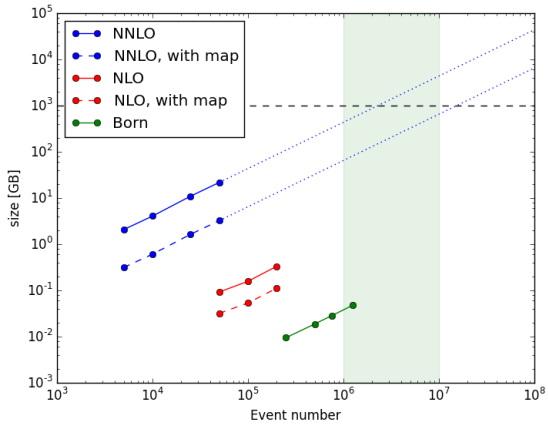

Figure 2. Size of the event files as a function of the number of events. The shaded green line represents the approximate number of events needed to match existing results. The horizontal dashed line at a size of a Terabyte is a subjective "pain threshold" where the size of the nTuples starts getting uncomfortable.

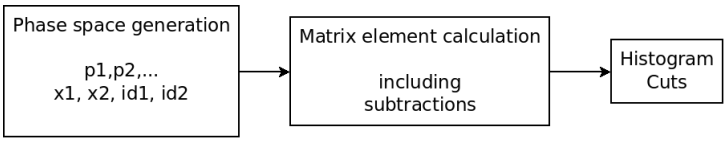

Figure 3. Schematic view of a generic generator.

or different PDF set so we need to decompose the weight ω we intercept at the histogramming stage into more specialised components. To improve the clarity of the presentation we also assume the factorisation and renormalisation scales are equal. The general form of the weight is given by

$$\omega = \alpha_S^n \, pdf(x_1, id_1) \, pdf(x_2, id_2) \times \Big(c_0 + c_1 \log(\mu^2) + c_2 \log^2(\mu^2) + \ldots \Big) \ . \tag{1}$$

In order to reconstruct the weight for a different PDF and scale choice we need to extract

- initial state flavours id_1, id_2
- momentum fractions x_1, x_2
- factorisation and renormalisation scales $\mu_F = \mu_R = \mu$
- The coefficients of the scale logarithms c_0, c_1, ...

The best place to extract this information is in the PDF procedure. Most generators use the LHAPDF [5] library to evaluate the PDFs, we will assume that is it the case here. If we observe the arguments passed to the LHAPDF functions we can infer the values of x_1, x_2, and μ. Because of the correlation between the PDF fit and the value of α_S used in the fit most generators also use the LHAPDF function for the evaluation of $\alpha_S(\mu)$.

Inferring the initial state flavours is more difficult, as one call to the PDF function returns the value of the PDF for all initial state flavours in a 13-element vector, we have no way to know which one the generator used (or whether multiple entries have been combined). To circumvent

this problem we introduce a strategy that we will reuse often for different purposes. The idea is to run several instances of the NNLO generator within slightly different environment and use the different weights obtained for each parallel instance to infer more information than is available from a single instance. To assert which of the 13 elements of the PDF vector are used in the calculation of the weight we can use 13 parallel instances of the NNLO generator, each with a modified version of the PDF function that returns zeros for all PDF values except for the one element, as sketched in Eq. 2.

$$\begin{aligned}
f_1(x) &= (*,0,0,0,0,0,0,0,0,0,0,0,0)\\
f_2(x) &= (0,*,0,0,0,0,0,0,0,0,0,0,0)\\
f_3(x) &= (0,0,*,0,0,0,0,0,0,0,0,0,0)\\
f_4(x) &= (0,0,0,*,0,0,0,0,0,0,0,0,0)\\
f_5(x) &= (0,0,0,0,*,0,0,0,0,0,0,0,0)\\
f_6(x) &= (0,0,0,0,0,*,0,0,0,0,0,0,0)\\
f_7(x) &= (0,0,0,0,0,0,*,0,0,0,0,0,0)\\
f_8(x) &= (0,0,0,0,0,0,0,*,0,0,0,0,0)\\
f_9(x) &= (0,0,0,0,0,0,0,0,*,0,0,0,0)\\
f_{10}(x) &= (0,0,0,0,0,0,0,0,0,*,0,0,0)\\
f_{11}(x) &= (0,0,0,0,0,0,0,0,0,0,*,0,0)\\
f_{12}(x) &= (0,0,0,0,0,0,0,0,0,0,0,*,0)\\
f_{13}(x) &= (0,0,0,0,0,0,0,0,0,0,0,0,*)
\end{aligned} \tag{2}$$

This method is quite expensive, in particular if one wants to identify pairs of PDF contributions as one would need to run $13 \cdot 13 = 169$ parallel versions of the generator, but it can be useful to identify potential PDF combinations in a preliminary run. In practice it is best to run two parallel instances of the generator, one with the normal PDFs and one with all the PDFs replaced by one. The ratio between the weights gives the product of the PDFs and we can compare this ratio with a table of possible combinations.

This strategy works well if the weight consists of only one PDF combination evaluated at a single pair of momentum fractions x_1, x_2. This is not always the case, especially in the subtraction method where subtraction contributions can be evaluated at different values of x_1 and x_2. In the case of multiple PDF pairs for the same weight we can use the same strategy and use instances of the generator with PDFs that are set to zero at different times. Eq. 3 illustrates this strategy. A first instance runs with each second call to the PDF set to 0, the second instance leaves the PDF vector untouched but sets the values left untouched by the first instance to zero instead. Two additional instances are run in parallel that project out the same PDF call than the first two, but replace the (non zero) values in the vector by 1. Building the ratio between the first and third instances and the second and fourth instances allows to obtain the PDF with argument x_1 or x_1' in isolation.

$$\begin{aligned}
\omega_1 &= c_1 \; pdf(x_1,x_2,id_1,id_2,Q) + c_2 \; 0\\
\omega_2 &= c_1 \; 0 + c_2 \; pdf(x_1',x_2',id_1',id_2',Q)\\
\omega_1^{pdf=1} &= c_1 \; 1 + c_2 \; 0\\
\omega_2^{pdf=1} &= c_1 \; 0 + c_2 \; 1
\end{aligned} \tag{3}$$

The coefficient of the different power of scale logarithms can be obtained by running different instances of the NNLO program with different scale settings. The power of n of $\alpha_S(\mu)$ in Eq. 1

is usually known from the settings in the generator but if not, or if it is changing during the run we can identify it by running two parallel instances, one with α_S set to unity. Comparing the ratio of the weights with the value of α_S returned by LHAPDF we can infer n as

$$n = \frac{\log(w) - \log(w|_{\alpha_S=1})}{\log(\alpha_S)} \, .$$

3.3. Implementation details

To implement the strategy described above we introduce an intermediary library (called "impersonator" in Fig. 4) between the generator and the LHAPDF library that allows to set some values in the PDF vector to zero or unity. This library also reports the values of the PDF vector and α_S to be used in the comparison between generator instances. Fig. 4 shows the interactions between the generator the LHAPDF impersonator and the LHAPDF library. The only modifications needed in the generator is one line of code to report the weight where the histogramming occurs and one line of code to report the momentum configuration when the cuts are evaluated. The impersonator can be implemented in a way that is transparent to the generator, so no changes have to be made to the interface between the generator and LHAPDF. One should note here that it is important that apart from the PDFs and α_S returned to the generator everything else should be equal between instances, this means for example that no integration optimisation should be made based on the weights calculated by the instances, as this optimisation would be different for the instance using the real PDF values and those with the PDF values changed to zero or unity. In practice this means that a first warmup integration with original PDFs has to be performed and should be frozen for the event file information collection runs.

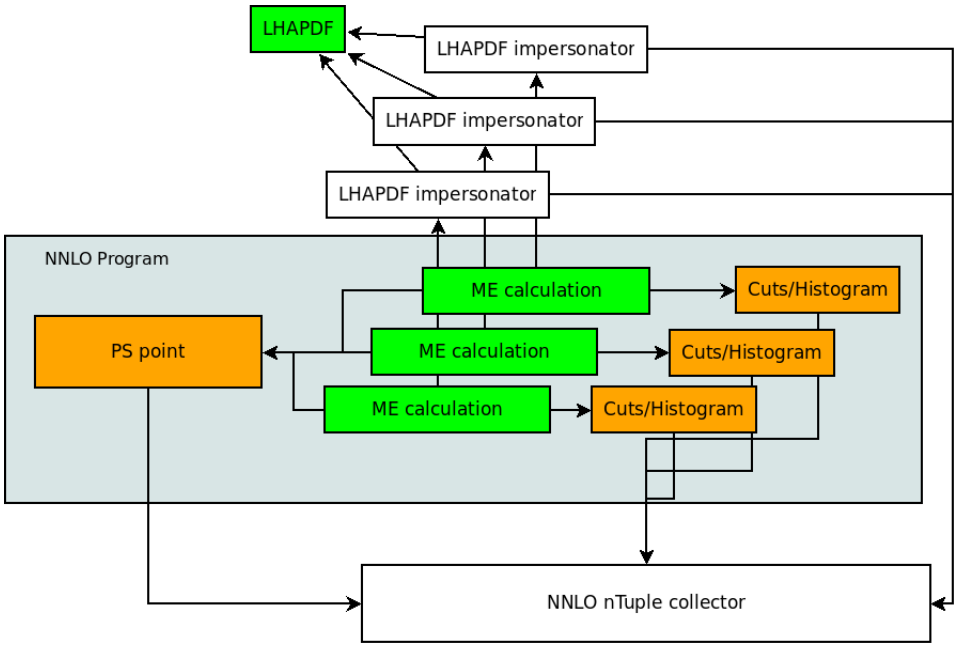

Figure 4. Schematic view of the information collection strategy: the generator is run with the same phase-space points in different environments. These different environments are implemented by the additional layer called "impersonator" in the figure. The weights, PDF and phase-space information is collected by an outside observer called "collector" in the figure.

When working with actual implementations of NNLO generators other complications arise. Most generators implement some caching of phase-space or PDF values which can interfere with our inference of the information we need to gather for the event files. It is easy to check that the information gathered is correct by running (yet another) instance with a different PDF and scale choice and checking that the information we reconstructed is able to reproduce the weights of the control instance.

4. Conclusion

We presented a strategy to collect information to build event files for processes calculated at NNLO accuracy. This method could also be applied to NLO. The strategy makes the clear choice of using some redundant CPU time in order to simplify the implementation of the information gathering. It is clear that this approach can not compete with an implementation where the generator authors extract the information directly from their code. This could be quite difficult and if attempted, the process could be aided by using the method described in this contribution as a check that the information provided in a direct way matches the empirically determined data. It should also be noted that the information extracted with the procedure above can be directly applied to construct `fastNLO/APPLgrid` [6, 7] tables for fast evaluation of histograms for PDF fits.

References

[1] Bern Z, Dixon L, Febres Cordero F, Höche S, Ita H *et al.* 2014 *Comput.Phys.Commun.* **185** 1443–1460 (*Preprint* 1310.7439)

[2] Andersen J R *et al.* 2016 *9th Les Houches Workshop on Physics at TeV Colliders (PhysTeV 2015) Les Houches, France, June 1-19, 2015* (*Preprint* 1605.04692) URL http://arxiv.org/pdf/1605.04692.pdf

[3] Maitre D, Heinrich G and Johnson M 2016 *PoS* **LL2016** 016 (*Preprint* 1607.06259)

[4] Gehrmann-De Ridder A, Gehrmann T, Glover E W N and Heinrich G 2007 *JHEP* **11** 058 (*Preprint* 0710.0346)

[5] Buckley A, Ferrando J, Lloyd S, Nordstrm K, Page B, Rfenacht M, Schnherr M and Watt G 2015 *Eur. Phys. J.* **C75** 132 (*Preprint* 1412.7420)

[6] Britzger D, Rabbertz K, Stober F and Wobisch M (fastNLO) 2012 *Proceedings, 20th International Workshop on Deep-Inelastic Scattering and Related Subjects (DIS 2012)* pp 217–221 (*Preprint* 1208.3641) URL http://inspirehep.net/record/1128033/files/arXiv:1208.3641.pdf

[7] Carli T, Clements D, Cooper-Sarkar A, Gwenlan C, Salam G P, Siegert F, Starovoitov P and Sutton M 2010 *Eur. Phys. J.* **C66** 503–524 (*Preprint* 0911.2985)