CERN-TH-2023-238, Nikhef 2023-026, OUTP-23-16P

Introduction to the PanScales framework, version 0.1

Melissa van Beekveld[1], Mrinal Dasgupta[2], Basem Kamal El-Menoufi[2,3], Silvia Ferrario Ravasio[4], Keith Hamilton[5], Jack Helliwell[6], Alexander Karlberg[4], Rok Medves[6], Pier Francesco Monni[4], Gavin P. Salam[6,7], Ludovic Scyboz[3,6], Alba Soto-Ontoso[4], Gregory Soyez[8], Rob Verheyen[5]

1 Nikhef, Theory Group, Science Park 105, 1098 XG, Amsterdam, The Netherlands
2 Department of Physics & Astronomy, University of Manchester, Manchester M13 9PL, United Kingdom
3 School of Physics and Astronomy, Monash University, Wellington Rd, Clayton VIC-3800, Australia
4 CERN, Theoretical Physics Department, CH-1211 Geneva 23, Switzerland
5 Department of Physics and Astronomy, University College London, London, WC1E 6BT, UK
6 Rudolf Peierls Centre for Theoretical Physics, Clarendon Laboratory, Parks Road, University of Oxford, Oxford OX1 3PU, UK
7 All Souls College, Oxford OX1 4AL, UK
8 Université Paris-Saclay, CNRS, CEA, Institut de physique théorique, 91191, Gif-sur-Yvette, France

mbeekvel@nikhef.nl, mrinal.dasgupta@manchester.ac.uk, basem.el-menoufi@monash.edu, silvia.ferrario.ravasio@cern.ch, keith.hamilton@ucl.ac.uk, jack.helliwell@physics.ox.ac.uk, alexander.karlberg@cern.ch, pier.monni@cern.ch, gavin.salam@physics.ox.ac.uk, ludovic.scyboz@monash.edu, alba.soto.ontoso@cern.ch, gregory.soyez@ipht.fr

Abstract

In this article, we document version 0.1 of the PanScales code for parton shower simulations. With the help of a few examples, we discuss basic usage of the code, including tests of logarithmic accuracy of parton showers. We expose some of the numerical techniques underlying the logarithmic tests and include a description of how users can implement their own showers within the framework. Some of the simpler logarithmic tests can be performed in a few minutes on a modern laptop. As an early step towards phenomenology, we also outline some aspects of a preliminary interface to Pythia8.3, for access to its hard matrix elements and its hadronisation modules.

The code is available from https://gitlab.com/panscales/panscales-0.X

Contents

1 Introduction

Parton showers lie at the core of the majority of experimental and phenomenological studies in collider physics. At the LHC, they connect the electroweak and TeV momentum scales of hard-scattering processes, where the relevant degrees of freedom are perturbative quarks and gluons, with the non-perturbative physics of hadrons at scales of a few hundred MeV. As such, parton showers account for physics across several orders of magnitude in momentum scales. In QCD, large logarithms typically appear in the presence of large momentum scale hierarchies, which have to be resummed to all orders in the strong coupling to obtain physically sensible results. One of the frontiers of the development of parton showers is to understand, demonstrate and improve their logarithmic accuracy, with analytic resummations providing crucial inputs, as well as reference results for comparison.

This paper documents the first public release of a new parton showering code, PanScales, version 0.1. It has been developed as part of a series of articles [1–9] investigating how to design parton shower algorithms that provide controlled and verifiable logarithmic accuracy, together with parallel analytical work on approaches to resummation at higher logarithmic accuracy and their connection with parton showers [10–17]. Several

other groups have also recently been working on the question of logarithmic accuracy in showers, see e.g. Refs. [18–25].

This PanScales release includes two main NLL-accurate parton showers, PanGlobal and PanLocal. They have had their next-to-leading-logarithmic (NLL) accuracy tested for e^+e^- [1] and colour-singlet production in pp [5,6] collisions, as well as Deep Inelastic Scattering (DIS) and Vector Boson Fusion (VBF) processes [8]. They include state-of-the-art handling of subleading colour corrections, which for many processes and observables allows for full colour accuracy at LL (and often beyond) [2,5]. They also include the treatment of both collinear and soft spin correlations [3–5], and first steps towards matching [7], as well as elements towards NNLL accuracy [9] (the latter two just for e^+e^- collisions). Finally, the codebase contains early versions of features that are yet to be discussed in physics research papers, in particular an interface with the Pythia8.3 event generator [26], which can be used to provide hard-process generation and hadronisation. Despite the inclusion of the interface to Pythia8.3, the code is not yet at a stage of maturity that is suitable for extensive comparisons to experimental data. This is notably because of the absence of finite quark-mass effects, the need for further work on matching with higher (fixed) order effects, as well as tuning of the non-perturbative parameters of the shower and of any hadronisation model with which it is used (see e.g. Refs. [27–29]).

This manuscript is structured as follows. Section 2 focuses on basic usage of the code, illustrating: the build procedure (section 2.1); stand-alone event generation (section 2.2); the use of the code for carrying out basic logarithmic tests of parton showers (section 2.3); usage with a preliminary interface to Pythia8.3 (section 2.4); and details for carrying out validation of the code and building it with higher numerical precision (section 2.5). Section 3 illustrates some of the techniques that underlie the logarithmic tests, discussing both double-logarithmic global event shape observables (section 3.1) and single-logarithmic non-global observables (section 3.2). Section 4 gives a brief discussion of how to use the PanScales framework to implement a new shower, which provides a relatively straightforward way to gain access to the colour, spin-handling and logarithmic-accuracy testing facilities. We close in section 5 with an outlook.

2 Basic usage

The PanScales code requires a C++14 compiler and a Fortran 95 compiler, the GSL library, CMake ($\geq$ 3.7) and, for some scripts, `Python` ($\geq$ 3.6) with `matplotlib` installed. Some features (higher-precision builds) require the MPFR [30] and QD [31] libraries (see section 2.5 for details). It includes several third party codes, notably `fjcore` [32] for jet finding and `hoppet` [33] for PDF handling and the Catch2 library for unit-testing (see the `3rdPartyCode.md` file for further details). The code can also be linked to LHAPDF [34] and to Pythia8.3. The PanScales code is released under the GNU GPL v3 license.

2.1 Downloading and building the code

PanScales can be obtained from the git repository

```
git clone --recursive https://gitlab.com/panscales/panscales-0.X
```

This comes with the following directory structure:

- `shower-code/`: main shower code and some simple examples
- `pythia-interface/`: code for the PanScales interface to Pythia8.3 [26]
- `analyses/nll-validation/`: code for the full set of NLL validation tests

- helpers/: contains the double_exp numerical type and a copy of fjcore [32]
- submodules/: contains HOPPET [33], CmdLine (for command-line processing and help) and AnalysisTools (for run steering and histogramming)
- scripts/: a variety of helper scripts
- unit-tests/: code for low-level unit tests

To build the code and examples in double-precision, do the following

```
cd panscales-0.X/shower-code
../scripts/build.py -j
```

The scripts/build.py script uses CMake to organise the build, which by default is placed in the build-double/ subdirectory. Advanced use of CMake is described in the BUILD.md file.

2.2 Standalone event generation

To run showering for the $e^+e^- \to q\bar{q}$ process and analyse the events, do

```
build-double/example-ee -shower panglobal -beta 0 -process ee2qq \
      -physical-coupling -rts 91.1876 -nev 100000 \
      -out example-results/example-ee.dat
```

This will take about 5−10 seconds, and produce an output file with histograms for a range of event shapes. Runs are generally configured with command-line options, for example with the first line indicating the use of the PanGlobal shower. The shower ordering variable is $v = k_t e^{-\beta_{\rm PS}|\eta|}$, with k_t and $\eta = -\ln\tan\theta/2$ respectively the transverse momentum and pseudorapidity of the emission with respect to its parent. The choice $\beta_{\rm PS} = 0$ (set with the -beta 0 command-line argument) corresponds to transverse-momentum ordering.

The available command-line options can be explored with the -h flag, and command-line options can also be placed in a card-file and read with the -argfile card-file.txt option. For convenience, the main options are also listed in an OPTIONS.md file (any executable can be made to generate such a file by adding -markdown-help to the command line). More details on the PanScales interface can be found by examining the example-ee.cc file. Much of the code also has doxygen documentation, which can be obtained by running doxygen from the shower-code/ directory.

Note that in standalone mode, as given above, currently all events have unit weight, i.e. in order to recover a physical cross-section from the above run the histograms should be multiplied by the appropriate cross section for the hard process. Furthermore, in most cases the Born event is a fixed configuration rather than being sampled over.

A final comment is that the event record (in the class panscales::Event, Appendix A.1) holds only the particles as they appear after all showering, rather than containing all intermediate steps as in some other codes. The reason for this is that for showers with global recoil, storing all intermediate steps would take memory of order n^2 for an n-particle event.[1] Access to the event record at intermediate steps can however be enabled, cf. Appendix A.2. By default the code prints the first event, but this can be changed to print N events with the -max-print N option.

Similar examples for $pp \to Z$ and DIS are given respectively in the example-pp.cc and example-dis.cc, with further explanations in an EXAMPLES.md file. Note that the above examples do not by default integrate over the hard-process kinematics. That functionality is instead available through the interface with Pythia8.3 (cf. section 2.4).

[1] Even showers with nominally local recoil sometimes involve global event changes, for example with initial-state branching in the PanLocal shower.

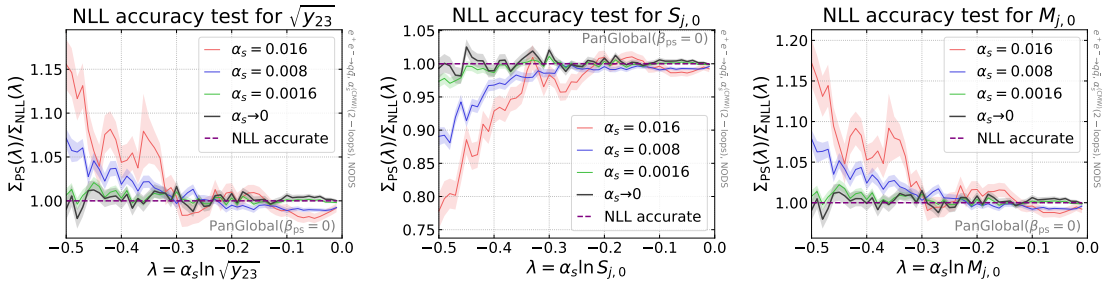


Figure 1: Example results from tests of NLL accuracy of the PanGlobal ($\beta_{\rm PS} = 0$) shower for three global event-shape observables, the Cambridge algorithm y_{23} and the Lund $\beta_{\rm obs} = 0$ sum and maximum observables, corresponding respectively to the three panels. Each panel shows the ratio of the shower cumulative-distribution result to the NLL calculation. Each coloured line corresponds to a specific value of α_s, while the black line gives the extrapolation to $\alpha_s = 0$. The results are shown as a function of the maximum allowed value of $\lambda = \alpha_s \ln O$, where O is the observable.

2.3 Logarithmic tests

Within the `shower-code/` directory, the `example-global-nll-ee.py` script, and associated `example-global-nll-ee.cc` program, illustrate how to carry out a basic test of the NLL accuracy of a shower for global event-shape observables. It can be used as

```
./example-global-nll-ee.py --njobs NJOBS --shower panglobal
```

and will test the NLL accuracy of the e^+e^- PanGlobal shower with $\beta_{\rm PS} = 0$ (i.e. k_t ordered), for event shapes like the Cambridge-algorithm y_{23} [35] and Lund observables [1]. The Lund observables measure either the sum or the maximum of the $k_{ti}e^{-\beta_{\rm obs}|\eta_i|}$ for primary Lund-plane declusterings i [36]. The NLL test works with $\beta_{\rm obs} = 0$, i.e. examining just the relative transverse momenta (k_t) of the declusterings.

The `--njobs` flag takes an integer which should be equal to the number of cores that one wishes to run on. Depending on the machine, the script takes around 15 to 50 CPU minutes, i.e. a few minutes of wall-time on a modern multi-core machine. On completion, for each observable, it will produce plots such as those in Fig. 1, showing the $\alpha_s \to 0$ limit of the ratio $\Sigma_{\rm shower}(\lambda)/\Sigma_{\rm NLL}(\lambda)$, where $\Sigma(\lambda)$ is the cross section for $\alpha_s \ln O$ to be smaller than λ, with O being the value of the observable. The plot is given as a function of λ, as is standard for NLL tests [1]: for an NLL-correct shower, the $\alpha_s \to 0$ limit of the ratio will be equal to 1. The user can examine the results for a non-NLL shower by replacing `panglobal` on the command line with `dipole-kt`, which provides a standard k_t-ordered dipole shower [5], much like those [37–39] in standard public tools.

The `shower-code` directory also includes an `example-nonglobal-nll-ee.py` script for testing non-global logarithms in the context of energy flow into an angular slice. It can be used as follows

```
./example-nonglobal-nll-ee.py --njobs NJOBS --shower panglobal
```

The script again generates a plot with the ratio to the NLL (single-logarithmic) result.

The above command lines serve mainly to illustrate the more straightforward logarithmic accuracy tests and provide only a subset of the functionality required for a full set of tests. For example, the global event-shape tests in the `example-global-nll-ee.py` script are limited to $\beta_{\rm PS} = \beta_{\rm obs} = 0$. Further discussion of considerations for logarithmic tests is given in section 3.

2.4 Usage with Pythia8

While we do not yet recommend the PanScales code for phenomenological production purposes, those wishing to start exploring such applications may try the interface with the Pythia8.3 [26] generator code, which we have tested with version 8.3.10. This enables the use of Pythia8.3 to generate the hard process,[2] as well as for hadronisation, while PanScales carries out the parton showering at the scales in between. It also enables access to Pythia8.3 I/O, e.g. for outputting HepMC files [40].

To compile and run the code, enter the `pythia-interface/` directory and run[3]

```
./get-pythia.sh
../scripts/build.py --build-lib -j --with-lhapdf
```

The main example program simulates Drell-Yan production. The default invocation requires the CT14lo set [41] from LHAPDF [34], which needs to be installed on the system e.g. by running `lhapdf install CT14lo`.[4] Running

```
build-double/main-dy -physical-coupling -lhapdf-set CT14lo \
      -shower panglobal -nev 1e5 -out main-dy.dat
```

will generate 10^5 Drell-Yan events and histogram the Z rapidity, mass and transverse momentum. It does not (yet) include matching to higher-order matrix elements, so kinematic distributions such as the transverse momentum of the colour singlet p_{tZ} are sensible only in the resummation region, i.e. at low p_{tZ}.

Note that the above run will produce warnings concerning x regions where the PDF set is badly behaved. Most other PDF sets have issues with negative or zero parton distribution functions, especially at large x, that cause the PanScales code to throw an exception after some number of events. Ultimately, we intend to make PanScales more tolerant of ill-behaved PDFs, but at this stage of development we have taken the approach that it is safer to abort the run than to continue generating events when a clear problem has arisen, even if only a rare occurrence.

The same directory contains a range of other examples that can be run with Pythia8.3, and the header of each example illustrates how to use it. We also provide an example of interfacing to Rivet [42], in which case one needs to make sure Rivet is installed and the code is compiled with it (through `--cmake-options="-DWITH_RIVET=on"`). This allows for an easy comparison with data, although we should note that none of the PanScales showers are currently tuned, nor do they include the effects of quark masses. We comment further at the end of this subsection.

The PanScales-Pythia8.3 interface transfers the event into the Pythia8.3 event record after each shower emission. This is done to have access to the full Pythia8.3 functionality in the future, i.e. interleaving multi-parton interactions (MPI) with showering of the hard process. Note that at this moment, we do not have the functionality to run with MPI, but hadronisation can be added through the flag `-hadron-level`.

The above examples all use the same PanScales event-loop framework as in the main `shower-code/` directory. We also distribute examples with a standard Pythia8.3 structure. These are to be found in the `main-pythia02.cc` and `main-pythia06.cc` files, which

[2] So far only a limited set of processes is supported in the interface, e.g. because of the setup of subleading-colour information for the hard process, which is currently handled manually.

[3] Users who wish to reuse an existing Pythia8.3 installation should see `pythia-interface/README.md` for instructions. For users who already have a version of Pythia8.3 installed but also run the `get-pythia.sh` script, care should to be taken about conflicts between the two versions of Pythia8.3 (e.g. library paths for dynamic linking leading to inconsistent versions).

[4] The code can also be used without LHAPDF, instead using a replacement toy PDF set. See `pythia-interface/README.md` for details.

are based on the corresponding `main02.cc` and `main06.cc` examples from the PYTHIA8.3 distribution. Note that only a restricted set of options is supported in this form, which can be found at the end of `PanScalesPythiaModule.cc`.

In thinking about the interface between the PanScales showers and Pythia's hadronisation, it is perhaps useful to comment briefly on the interplay between hadronisation and the parton shower's logarithmic resummation. Consider an event with a hard scattering scale Q and some infrared and collinear (IRC) safe observable (e.g. a jet rate) that is sensitive to a transverse-momentum scale $p_t \ll Q$. Any discussion in terms of quarks and gluons implicitly comes with a requirement that the transverse momentum scale p_t be much larger than $\Lambda_{\rm QCD} \sim 1\,\text{GeV}$. We expect that hadronisation (and associated tuning parameters, such as those of the Lund string model) will act primarily at scales of the order of $\Lambda_{\rm QCD}$. Thus for the observable at scale p_t, the relative impact of hadronisation should be suppressed by a power of $\Lambda_{\rm QCD}/p_t$. Consequently hadronisation will not modify the resummation properties of the shower. Of course in practice one may also be interested in physics at a few GeV or in quantities that are not IRC safe. In those cases, the interplay between shower resummation and hadronisation may be non-trivial: for example more accurate shower resummation may provide a better initial condition for hadronisation models. A more detailed study of these questions would be interesting, but is beyond the scope of this manual.

2.5 Code validation and more advanced builds

To validate that the code is generating expected results, enter the `shower-code/validation/` directory and run

```
./validate-showers.py -j
```

which runs a range of validation tests in parallel across all available cores. This takes a total of about 300−1000 CPU seconds on a typical laptop, carrying out of the order of 100 separate short runs, each with different settings, verifying that they give histograms that are identical to those in a set of reference files. It is possible to carry out validation runs with larger numbers of events, but one should be aware that there can be differences due to varying floating-point behaviours across different hardware. Additionally, lower-level unit tests can be found in the `unit-tests/` directory. These are based on the Catch2 library and are mainly used for checks of foundational classes, such as the main `Momentum` class (see the `README.md` file in that directory for further details). The unit tests and a subset of the validation tests can also be run with the continuous integration script `scripts/CI-build-and-test.py`.

It is sometimes useful to build the main code in different numerical precisions, e.g. for logarithmic tests that probe very disparate energy scales and angles. For this, the general build script has an option `--builds X` which essentially invokes `cmake` with a suitable set of configuration options. Specifically, one runs

```
../scripts/build.py --builds X [-j]
```

where `X` is a space-separated list that contains one or more of the following options: `double`, `ddreal`, `qdreal`, `doubleexp`, `mpfr4096`.

The `ddreal` and `qdreal` options require at least version 2.3.23 of the QD library [31] and have precisions of about twice and four times that of a double type, with speeds that are about 10 and 200−300 times slower.

The `doubleexp` type was developed specifically for the PANSCALES project and has the same relative precision as `double`, but a much larger range of exponents (stored in an

additional 64-bit integer), which is useful when exploring finite values of $\alpha_s \ln v$ with very small values of α_s and correspondingly large values of $\ln v$. It is about 3−10 times slower than `double`.

The `mpfr4096` type is based on the MPFR library and has 4096 bits for the mantissa, i.e. about 75 times higher than double precision, or equivalently a little over 1000 decimal digits of precision. It is about 1500−2500 times slower than `double` (using version 4.2.1 of the library).

For most purposes, the `double` and `doubleexp` types are sufficient, notably when used with methods that track the differences between directions of dipole ends in addition to the actual momenta of the corresponding partons [2]. That tracking can be enabled at run time with the `-use-diffs` option and has only a modest speed penalty, of the order of 10%. The higher precision types are, however, important when developing new showers and testing the correctness of any parts of the code that carry out the dedicated calculations with direction differences.

3 Further details for logarithmic tests

In this section we provide some insight into features of the code that facilitate shower logarithmic accuracy tests, together with a more detailed discussion of some of the underlying methodology than has been given in previous work.

Some of the discussion below concerns tests that go beyond the simple ones illustrated in section 2.3. Code for these more advanced tests is to be found in the directory `analyses/nll-validation/` with usage explanations in the corresponding `README.md` file.

3.1 Global event shapes

We start by considering the global event shape tests of section 2.3 and specifically examine the commands that are run by the `example-global-nll-ee.py` script. For each of several α_s values, that script executes one or more commands of the following kind

```
build-double/example-global-nll-ee -Q 1.0 -shower panglobal -beta 0.0 \
      -alphas 0.0016 -nloops 2 \
      -lambda-obs-min -0.5 -lnkt-cutoff -327.5 -dynamic-lncutoff -15 \
      -weighted-generation -nev-cache 75000.0 \
      -spin-corr off -use-diffs -nev 750000 -rseq 11 \
      -out example-results/lambda-0.5-alphas0.0016-rseq11.res
```

The second line indicates the value of $\alpha_s(Q)$ in the $\overline{\text{MS}}$ scheme (working in units of $\sqrt{s} \equiv Q = 1$) and that a two-loop running coupling is to be used. The next two lines contain two separate critical elements, which we discuss below in more detail. Further options are `-spin-corr off`, which turns off spin correlations (leaving them on adds about 50% to the run time). The `-use-diffs` option turns on tracking of direction differences for higher numerical precision. It is not strictly necessary for this example, but the speed cost is relatively limited, at about 10%. The `-rseq 11` argument specifies the random sequence that is used.

The choice of α_s values (cf. also Fig. 1) involves a balance between several considerations.

1. CPU usage differs across values of α_s. For example taking too small a value of α_s (and correspondingly large logarithms) requires the use of the `doubleexp` type, which is slower. Furthermore, in some cases, smaller α_s values bring slower Monte Carlo convergence (notably when one needs to use the methods of Section 3.1.3).

2. At order N^{p}LL one is looking for residual effects of $\mathcal{O}(\alpha_s^{p-1})$. For NLL tests, we have $p = 1$ and so we are measuring an $\mathcal{O}(1)$ effect, i.e. independent of α_s. But for, say, an NNLL test, one would be looking to measure $\mathcal{O}(\alpha_s)$ effects, and taking values of α_s that are too small would make it more difficult to identify those effects.

3. The ability to take the shower cutoff sufficiently small that cutoff effects are negligible, while also not encountering the Landau pole in the coupling. This complicates the use of larger values of α_s, because one approaches the Landau pole more rapidly. CPU usage then increases significantly, notably because of the higher particle multiplicities in each event.

4. The size of residual contributions that are beyond the order of extrapolation for the $\alpha_s \to 0$ limit. This favours taking smaller values of α_s.

As concerns the last of these points, one should estimate a systematic error for the extrapolation, for example by using two different sets of α_s points. While this is not done for Fig. 1, it is a part of all tests in `analyses/nll-validation/` that rely on extrapolation of $\alpha_s \to 0$.

Two further critical elements that we now need to discuss in more detail are (1) a dynamic generation cutoff, which is necessary to prevent event particle multiplicities from becoming intractable and (2) weighted event generation, which is necessary in order to explore regions with large Sudakov suppression.

3.1.1 Dynamic generation cutoff

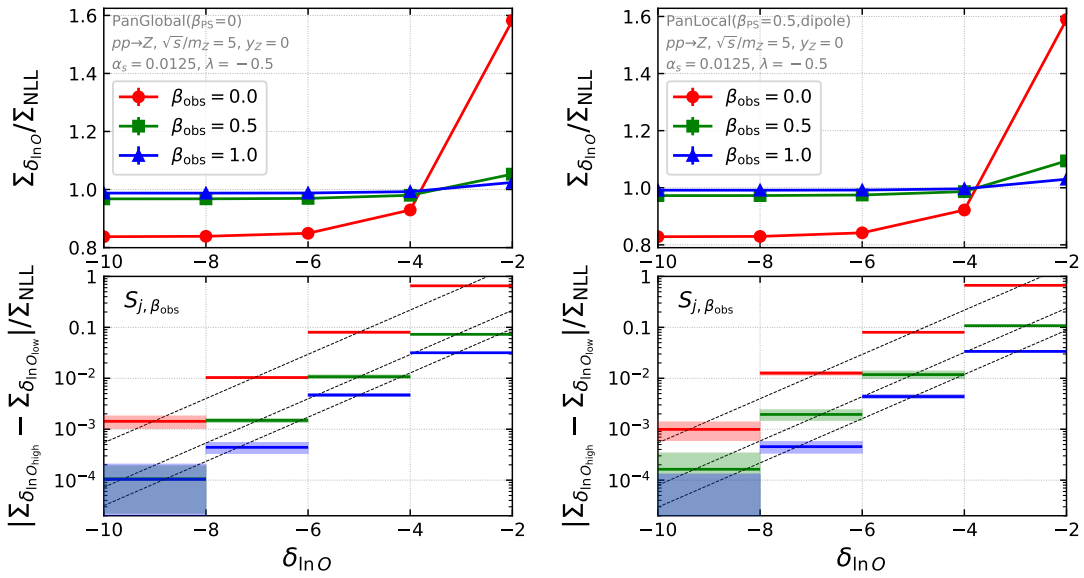
Let us discuss the following part of the command line:

```
-lambda-obs-min -0.5 -lnkt-cutoff -327.5 -dynamic-lncutoff -15
```

This indicates that the minimal value of $\lambda = \alpha_s \ln v$ is -0.5, which corresponds to $\ln k_t/Q = -312.5$ with $\alpha_s = 0.0016$. The choice `-lnkt-cutoff -327.5` allows the shower to run somewhat below the k_t scale associated with the minimal value of λ. This is important, because multiple emissions break the immediate relation between the shower scale and the observable, e.g. because of shower recoil effects, and because the observable sums contributions from multiple emissions. These effects are properly accounted for only if the shower is allowed to evolve sufficiently far below the single-emission scale that is equivalent to the smallest value of the observable that is probed.

If we were to straightforwardly run with the very small cutoff indicated above, then the average event particle multiplicity, n, would be prohibitively large. Specifically, it scales as $\ln n = |\lambda_c|\sqrt{2C_A/(\pi\alpha_s)} + \mathcal{O}(1)$ [43], where $\lambda_c = \alpha_s \ln k_{t,\text{cutoff}}/Q$. For the parameters above, this would give $n \sim 10^8$. To resolve this issue, in addition to the fixed k_t cutoff we also use a dynamic cutoff. We track the shower scale v_1 of the first shower emission. Knowing that we are restricting our attention to $\beta_{\text{obs}} = \beta_{\text{PS}}$ observables and that standard global event shapes are recursively infrared-and collinear safe [44], we can be sure that emissions with much lower values of v will not modify the event shapes. Accordingly, when the shower reaches an $\ln v$ scale that is sufficiently far below that of the first emission, we simply stop showering. That choice is specified by the `-dynamic-lncutoff -15` argument, which stops the showering when $\ln v < \ln v_1 + \delta_{\ln v} = \ln v_1 - 15$. This ensures that the multiplicity is kept limited, e.g. for the above run it averages to about 20.

To validate this approach, the dependence of the normalised cross section $\Sigma(\lambda)$ on the size of the dynamic cutoff is shown in Fig. 2a (red line, corresponding to a k_t-like observable, i.e. $\beta_{\text{obs}} = 0$). It is illustrated for the $pp \to Z$ process, as carried out for the studies of Ref. [6], and one observes good convergence of $\Sigma(\lambda)$ as the dynamic cutoff $\delta_{\ln v}$



(a) (b)

Figure 2: Dependence of $\Sigma(\lambda)$ on the size, $\delta_{\ln O}$, of the parton-shower dynamic cutoff. The results are shown for the $S_{j,\beta_{\rm obs}}$ class of observables in Drell-Yan production, which sum $p_{ti}e^{-\beta_{\rm obs}|\eta_i|}$ over all final-state jets i (with p_t defined as the transverse momentum with respect to the beam). The dependence on the size of the dynamic cutoff size is shown normalised to the NLL prediction for Σ, for each of three values of $\beta_{\rm obs}$: 0 (red), 0.5 (green) and 1.0 (blue). Two showers are shown (a) PanGlobal with $\beta_{\rm PS} = 0$ and (b) PanLocal (dipole) with $\beta_{\rm PS} = 0.5$. The upper panels show the ratio of Σ to the NLL result, showing the convergence as $\delta_{\ln O}$ is made more negative (note that the ratio is not expected to go to 1, because of the finite value of α_s). The lower panels show the difference between $\Sigma/\Sigma_{\rm NLL}$ results for successive $\delta_{\ln O}$ values, again normalised to the NLL result, giving a clearer view of the degree of convergence. The dashed lines help illustrate that the behaviour is consistent with exponential dependence on $\delta_{\ln O}$.

is taken below -9 (for the $\beta_{\text{PS}} = \beta_{\text{obs}}$ case discussed, $\delta_{\ln v} \equiv \delta_{\ln O}$ in the plot). The value of -15 that we use above leaves a comfortable margin.

The above procedure is sufficient when the angular power β_{PS} in the definition of the shower ordering variable coincides with the angular power β_{obs} in the observable. When $\beta_{\text{PS}} \neq \beta_{\text{obs}}$ a more elaborate procedure is needed. As a starting point, we need to be able to evaluate the order of magnitude of the contribution of each emission i to the observable, $O_{\text{approx},i}$. Given knowledge of β_{PS} and β_{obs}, this can be determined from the value of $\ln v$ and the (approximate) rapidity of the emission. Specifically we take it to have the form

$$O_{\text{approx},i} = c_v \ln v_i + c_b \ln b_i + \text{const.} \tag{1}$$

where $\ln b_i$ is a rapidity-like variable (cf. Section 4), the c_v and c_b are coefficients that are independent of the dipole kinematics, and the remaining constant term does depend on the dipole kinematics.[5] In general Eq. (1) will agree with the true contribution to the observable from emission i to within a factor of order 1. We then determine if the following condition holds

$$\ln O_{\text{approx},i} < \max_{j<i}\{\ln O_{\text{approx},j}\} + \delta_{\ln O} \tag{2}$$

where on the right-hand side the max operation runs over prior accepted emissions j and $\delta_{\ln O}$ generalises the $\delta_{\ln v}$ discussed above. If Eq. (2) holds, then emission i is discarded and showering continues. Showering is subsequently stopped when $\ln v$ is sufficiently small such that for all emission rapidities one can be sure that Eq. (2) will always hold.

Note that for $\beta_{\text{PS}} \neq \beta_{\text{obs}}$ this procedure is guaranteed to be safe only for showers that respect the PanScales condition that a given emission does not impact other emissions far in the Lund plane.[6] Results are illustrated in Fig. 2, for various combinations of β_{PS} and β_{obs}, showing the relative change in Σ between successive pairs of values of $\delta_{\ln O}$, corresponding to the extremities of each horizontal bar. One sees a behaviour that is consistent with an exponentially vanishing effect as $\delta_{\ln O}$ becomes more negative. Again, a choice of -15 should be more than adequate for NLL logarithmic tests. Note that the need for a more sophisticated dynamic veto and cutoff is not the only challenge that arises with $\beta_{\text{PS}} \neq \beta_{\text{obs}}$. Further considerations are discussed below in section 3.1.3.

3.1.2 Weighted event generation for $\beta_{\text{PS}} = \beta_{\text{obs}}$

Now we turn to the following part of the command line at the beginning of section 3.1

```
-weighted-generation -nev-cache 75000.0
```

This is needed to address the fact that Σ becomes infinitesimally small in the limit $\alpha_s \to 0$ for fixed $\lambda = \alpha_s \ln v$, specifically $\ln \Sigma \sim \lambda/\alpha_s$. To get a more concrete sense of the challenge, consider that $\Sigma \sim 10^{-103}$ for $\lambda = -0.5$ and $\alpha_s = 0.0016$ as in the command line at the start of section 3.1. With unweighted event generation, it would take orders of magnitude longer than the age of the universe to explore that region.

We address this challenge by greatly enhancing the number of events whose first emission has an extremely small value of $\ln v$, assigning a suitable weight to those events so as to reproduce the correct final Σ distribution.

We divide the full evolution range $[\ln v_{\text{max}}, \ln v_{\text{min}}]$ into a set of n consecutive bins, each defined by their upper boundaries, $\ln v_i^+$. Writing the shower Sudakov form factor between two scales v and v' as $\Delta(v, v')$ (for the Born event), we precompute the part of

[5] Our choice for $O_{\text{approx},i}$ can be evaluated with the help of a function in the base class for shower implementations, `ShowerBase::Element::lnobs_approx(...)`.

[6] This condition is not satisfied for standard dipole showers. For $\beta_{\text{obs}} \neq \beta_{\text{PS}}$ this results in super-leading logarithmic terms [1] and such terms would not be fully reproduced with the above dynamic veto procedure.

the Sudakov form factor associated with each bin.[7] The precomputed Sudakov can be inspected in the output file.

For each event, we choose a generation bin i randomly, with a probability p_i that we take proportional to $\ln v_i^+ - \ln v_{i+1}^+$. The shower then starts from scale v_i^+. If the first emission scale is above v_{i+1}^+, the shower continues down to the dynamic cutoff as explained in section 3.1.1. If the first emission scale is below v_{i+1}^+, the emission is discarded and showering is restarted from scale v_i^+, repeating this until one generates an emission above v_{i+1}^+. The weight assigned to the event is

$$w = \frac{1}{p_i}\left(\Delta(v_{\max}, v_i^+) - \Delta(v_{\max}, v_{i+1}^+)\right). \tag{3}$$

where the factor $1/p_i$ compensates for the likelihood of choosing the window and the second factor accounts for the Sudakov form-factor probability of having the first emission between v_i^+ and v_{i+1}^+.

In practice we choose the bins such that $\ln v_i^+$ scales as $-\sqrt{i}$, which in a fixed-coupling approximation ensures that the Sudakov form factor decreases by a similar factor from one bin to the next. We take the total number of bins to be $n = \ln \Delta(v^+, v^-)/\ln(u)$ where u is a number of $\mathcal{O}(1/2)$, so that the probability of each successive bin is about half that of the previous bin. This ensures that each attempt at starting the showering has a $\sim 50\%$ chance of generating an emission within the given window, while also ensuring that the event weight tracks the actual first-emission Sudakov probability to within about a factor of two.

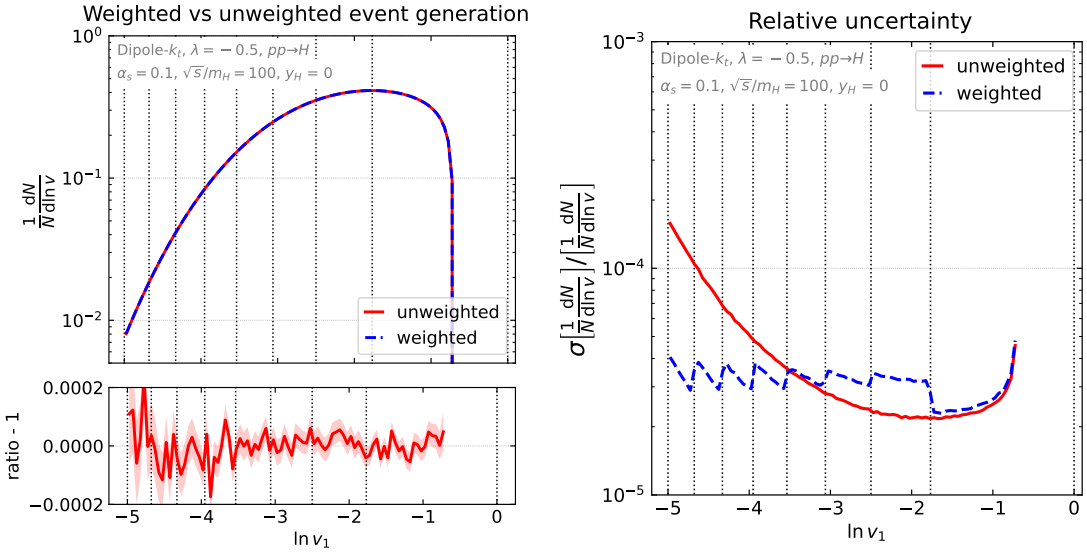
Fig. 3a shows a validation of the correctness of the procedure for a physical value of the coupling, $\alpha_s = 0.1$, where it is straightforward to obtain high accuracy with both weighted and unweighted approaches. The plot shows the distribution of $\ln v_1$ for the first emission when the generation is unweighted (red solid lines) and weighted (blue dashed lines). The two approaches agree to within statistical errors. Fig. 3b shows the size of the relative statistical error in the two approaches, illustrating the superiority of weighted generation for small $\ln v_1$ values, and also illustrating that its statistical error is fairly independent of $\ln v_1$, with a mild sawtooth structure that lines up with the generation bin edges (indicated as vertical dotted lines).

3.1.3 Weighted event generation for $\beta_{\mathbf{ps}} \neq \beta_{\mathbf{obs}}$

The above weighted event-generation procedure is very powerful when the Lund contour of the observable lines up with that of the shower evolution variable, i.e. $\beta_{\text{obs}} = \beta_{\text{PS}}$. If this is not the case, a given value of the observable receives contributions from different evolution windows with often widely differing weights, significantly worsening statistical convergence.

Let us start by explaining the structure of our approach, which is illustrated also with the help of Fig. 4. Ultimately, we want to generate all possible events such that a given observable O is below some threshold, e.g. $\alpha_s \ln O = \lambda < -0.5$. As a first step will consider how to generate events with a condition on an approximation to the observable, $O_{\max} \equiv \max_i\{O_{\text{approx},i}\}$, with the $O_{\text{approx},i}$ as defined in Eq. (1). Specifically, we will consider how to generate events such that $O_{\max}$ is in some range $O^- < O_{\max} < O^+$, i.e. the green band in Fig. 4, labelled "approximate observable window". Then we will show how to use event-samples in different approximate-observable windows so as to obtain a

[7] This can be done either with a Monte Carlo integration (the default) or with Gaussian quadrature. The `-nev-cache` argument indicates the number of events used for the MC integration in each bin. We have found that a suitable choice is about 10% of the total number of events that one wishes to generate.



(a) (b)

Figure 3: Validation of the weighted event generation. We show the results for Dipole-k_t for $pp \to H$ collisions with $\sqrt{s}/m_H = 100$ and $y_H = 0$. We take $\alpha_s = 0.1$ and target $\lambda = -0.5$. (a) The distribution of $\ln v_1$ values for the first emission, with and without weighted generation, illustrating that they agree. The bottom panel shows the ratio between the weighted and unweighted results minus 1, where the statistical uncertainty is indicated with the red band. (b) The relative statistical uncertainty in the $\ln v_1$ distribution for both unweighted and weighted shower generation.

set of events with a constraint on the actual observable, exploiting the fact that O and $O_{\max}$ differ at most by a factor of order one.

The complication that we face in generating an event sample with a constraint on $O_{\max}$ is that the shower contours (blue lines of Fig. 4) do not align with observable contours (e.g. edges of the green band). However, knowing the scaling of the observable, it is possible to analytically work out the largest shower scale, v_a, that can generate an emission with approximate observable-value O^+. We therefore start the showering from that scale v_a, with an initial weight equal to $w = \Delta(v_{\max}, v_a)$, where $v_{\max}$ is the largest kinematically accessible scale. We then need to ensure that the showering does not generate any emissions with $O_{\text{approx},i} > O^+$. The simplest approach would be to veto every event that has any of the emissions i contributing such that $O_{\text{approx},i} > O^+$. This would be correct, but would in general lead to a very small fraction of surviving events. That survival fraction corresponds roughly to the Sudakov form factor associated with the area between the top of the green band (O^+) and the thick blue v_a line in Fig. 4.

To work around this problem, we employ a procedure that relates to an approach first introduced in Ref. [45] in the context of multi-jet merging. Specifically, we use an enhancement $C_{\text{enh}} > 1$ for the probability of generating individual emissions. If an emission i has $O_{\text{approx},i} > O^+$, the emission is discarded, the event is kept, but the event weight w is multiplied by a factor such that

$$w \to w \, \frac{C_{\text{enh}} - 1}{C_{\text{enh}}}. \tag{4}$$

Emissions with $O_{\text{approx},i} \leq O^+$ (including those with $O_{\text{approx},i} \leq O^-$) are instead accepted with probability $1/C_{\text{enh}}$, with the event weight unchanged. Once v goes below some value

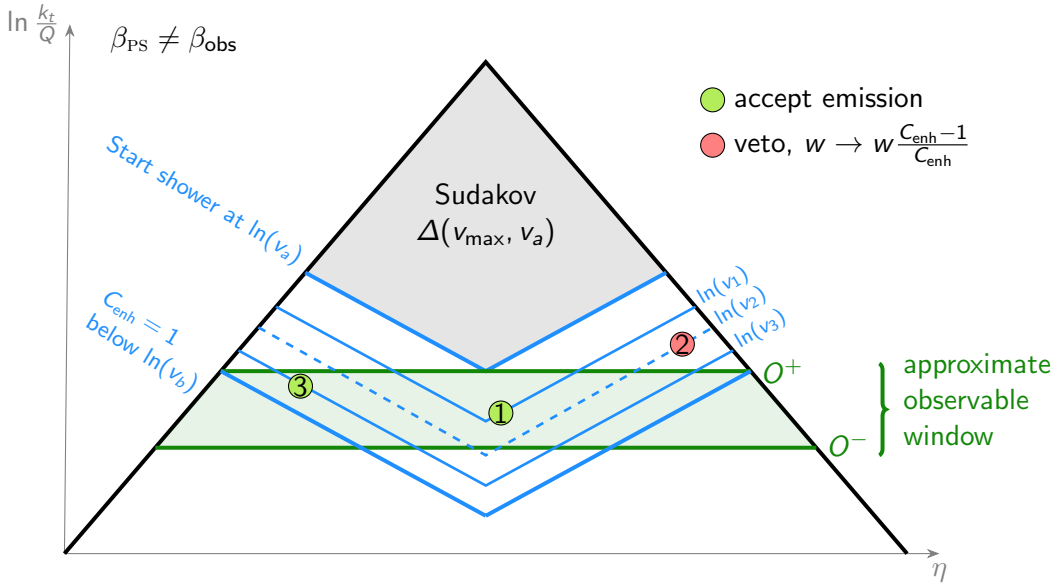


Figure 4: Illustration of some of the main steps in the weighted event-generation approach that is used for $\beta_{PS} \neq \beta_{obs}$, for a specific target observable window. Beyond what is shown in the figure, one important element is that if, after showering, there are no emissions in the approximate observable window, the event is discarded. The approach is additionally supplemented with the dynamic generation cutoff of section 3.1.1.

v_b such that no further emission can have $O_{\text{approx},i} > O^+$, the enhancement factor is set equal to 1 and the shower continues down to the scale of the dynamic cutoff. After statistical averaging this gives exactly the same result as a uniform-weight approach that discards every event with any emissions $O_{\text{approx},i} > O^+$ [45].[8] Finally, events are only accepted if at least one emission has an approximate observable value satisfying $O_{\text{approx},i} > O^-$. Together with the preceding steps, this guarantees $O^- < O_{\text{max}} < O^+$.

The final step is that for a given value of the constraint on the actual observable (e.g. $\alpha_s \ln O < \lambda = -0.5$), we need to add together several contiguous $O^\pm$ windows above the constraint, plus one final window without a lower bound (i.e. $O^- = 0$). Note that Fig. 4 shows only one of the windows. The need for several windows is because the actual value of the observable can be smaller than the approximate value of the observable by some $\mathcal{O}(1)$ factor. The number of windows that is needed depends on both the shower and the observable. Typically if the actual observable can be substantially below the approximate observable, one needs a greater number of windows above the value of the true observable constraint. One should always check that the highest window contributes minimally to the result, i.e. less than the total statistical uncertainty after summing all windows.

Let us close with a comment on event weights. Within a given approximate observable window, from one event to the next Eq. (4) will have been applied a different number of

[8] One way of understanding this is to think that one starts with a number of replicas of the shower that will evolve in parallel. The number of replicas is equal to C_{enh}, and the enhancement of the emission probability can be interpreted as being equivalent to the total probability of emissions occurring in any of the replicas. Concentrating specifically on emissions with $O_{\text{approx},i} > O^+$, the first time any of the replicas generates an emission with $O_{\text{approx},i} > O^+$, that replica is simply discarded, leaving $C_{\text{enh}} - 1$ replicas. The factor $\frac{C_{\text{enh}}-1}{C_{\text{enh}}}$ in Eq. (4) is simply the ratio of surviving to original replicas. As the shower continues, the remaining weight $\frac{C_{\text{enh}}-1}{C_{\text{enh}}}$ is then shared back out across C_{enh} replicas again, and so the procedure continues.

times, reflecting the number of emission that have been discarded in that event. There will therefore be a spread of event weights. The width of the event weight distribution gets smaller as C_{enh} is made larger, but there is an associated slow-down of the showering, because of the increased emission probability. The optimal choice for C_{enh} involves some balance between these two aspects. In practice we use $C_{\text{enh}} = 20$ for showers with $\beta_{\text{PS}} = 0$, and $C_{\text{enh}} = 100$ for showers with $\beta_{\text{PS}} = 0.5$.

In its current form, the code (`analyses/nll-validation/shower-global-obs.cc`) performs separate runs for each of several windows and the python script that manages the calculations (`run-nll-tests.py`) adds the results together. A command line that sets up the use of the above algorithm is, for instance

```
build-double/shower-global-obs -Q 1.0 -no-spin-corr  -nloops 2 -alphas 0.008 \
      -shower panglobal -beta 0.0 -colour NODS \
      -lambda-obs -0.5 -beta-obs 0.5 -lnkt-cutoff -77.5 \
      -strategy RouletteEnhanced -enhancement-factor 20.0 \
      -ln-obs-buffer 3.5 -nln-obs-div 7 -veto-buffer -15.0 \
      -use-diffs  -rseq 11 -nev 70000 \
      -out nll-tests-tmp/panglobal00-fapx0.5-lambda0.5-as0.008-NODS-rseq11.dat
```

The second and third lines set up a shower and an observable class with different β_{PS} and β_{obs} values. The fourth line sets up the strategy described above and the associated value of the enhancement factor, corresponding to C_{enh}. The fifth line indicates that 7 approximate observable windows are explored, extending to $e^{3.5}$ times above the target value of the observable. The code performs runs in each of the 7 separate approximate-observable windows, producing one output file for each (Fig. 4 corresponds to a single window). The `-veto-buffer -15` argument on the fifth line sets the size of the dynamic veto and cutoff, as explained for $\beta_{\text{PS}} \neq \beta_{\text{obs}}$ at the end of section 3.1.1. If a user wishes to explore new observables or new showers, they are strongly advised to manually verify that the contribution to the cross section from the highest window is sufficiently suppressed relative to the total result. This can be done by examining the output files from the above command, there being one file per observable window.

3.2 Single-logarithmic observables, e.g. non-global logarithms

An example of a single-logarithmic test, such as the transverse energy flow in a rapidity slice in $e^+e^- \to q\bar{q}$ collisions, can be performed with the `example-nonglobal-nll-ee.py` script in the `shower-code/` directory. This will execute the following command

```
build-doubleexp/example-nonglobal-nll-ee -Q 1.0 -shower panglobal -beta 0.0 \
      -nloops 2 -colour CATwoCF  \
      -slice-maxrap 1.0 -lambda-obs-min -0.5 \
      -alphas 1e-09 -lnkt-cutoff -501000000.0 -ln-obs-margin -11 \
      -strategy CentralRap -half-central-rap-window 10  \
      -spin-corr off -nev 10000 -rseq 11 \
      -out example-results/lambda-0.5-alphas1e-09-rseq11.res
```

The second line indicates that a 2-loop running coupling is to be used and that the colour scheme is a large-N_{C} scheme in which $C_A = 2C_F = 8/3$.[9] The third line indicates the size

[9] Recall that the PanScales showers are logarithmically accurate for non-global logarithms only in the large-N_{C} limit, though the subleading-colour schemes [2] are numerically close to the full-colour results [46,47]. The schemes of Ref. [2] can be obtained by replacing `CATwoCF` with `NODS` or `Segment`. The leading-colour test can also be performed with $C_F = \frac{1}{2}C_A = \frac{3}{2}$, using the option `CFHalfCA`. When running with small but finite values of α_s, one advantage of $C_A = 2C_F = 8/3$ over $C_F = \frac{1}{2}C_A = \frac{3}{2}$ is that the former has a greater cancellation between the C_A and n_f terms in the running-coupling β-function, which ensures that the Landau pole is further in the infrared for any given value of α_s.

of the rapidity slice in which the in-slice energy flow E_t will be measured, and sets the minimum value of $\lambda = \alpha_s \ln E_t/Q = -0.5$.

The fourth line indicates that we run at an infinitesimal value of the coupling, $\alpha_s(Q) = 10^{-9}$ (still working in units of $\sqrt{s} \equiv Q = 1$), with a sufficiently small k_t cutoff, $e^{-501000000}$, so as to cover the region down to $\lambda = -0.5$. The point of using such a small value of α_s is to avoid a substantial extrapolation to $\alpha_s = 0$, alleviating the need for runs at multiple values of α_s. It is physically possible to use such an infinitesimal value because the observable $\Sigma(\lambda, \alpha_s)$ is independent of α_s for $\alpha_s \to 0$. This is in contrast with the case of global event shapes where $\ln \Sigma(\lambda, \alpha_s) \sim \lambda/\alpha_s$. The huge range of orders of magnitude of momenta requires the use of the `doubleexp` type.

As with the case of global event-shape tests, a straightforward run with the above parameters would give multiplicities that are much too high to be managed (in fact, the situation is even worse, because of the much smaller value of α_s). The mitigation procedure is different in this case: the combination of arguments on the fifth line causes the shower to only generate emissions within a window where the absolute rapidity is less than 11 (with respect to the emitter or spectator). For showers that satisfy the PanScales conditions, as long as λ is not too large and the window is large enough, the results should be (and are) independent of the window size. Corresponding arguments also exist for an analogous modification of the shower to generate only hard-collinear emissions, as is relevant for many spin-correlation, fragmentation function and PDF evolution tests.

Note that for verifying next-to-single-logarithmic accuracy [12, 13, 48] for non-global logarithms [9] it is necessary to enable the double-soft matrix element corrections and associated virtual corrections (with the `-double-soft` command-line argument, available only for the PanGlobal showers with $\beta = 0$ and $\beta = 0.5$, for e^+e^- collisions). One also has to modify the above command line to run at multiple small but finite values of α_s, so as to be able to determine the first derivative of $\Sigma(\lambda, \alpha_s)$ with respect to α_s in the $\alpha_s \to 0$ limit. Doing so accurately requires considerable computer resources. Users who wish to explore this are advised, in a first instance, to run with the `-split-dipole-frame` option for the PanGlobal showers with $\beta_{\text{PS}} = 0$, which is the most computationally efficient setup.

4 Implementing a new shower

The PanScales framework allows for relatively straightforward addition of new dipole and antenna showers with alternative kinematic maps or ordering variables. This allows the user to leverage the existing code for colour and spin handling, as well as the infrastructure for logarithmic accuracy tests and the interface to Pythia8.3. As an example, our own toy implementation of the Pythia8.3 final-state shower involves about 250 lines of header (`ShowerToyPythia8.hh`), most of which are boilerplate code, and a further 250 lines in `ShowerToyPythia8.cc`, more than half of which are comments or blank lines. For a slightly more elaborate example that handles also initial-state radiation, the user may wish to look at the `ShowerDipoleKt` class.

Here we outline some of the aspects that a user should keep in mind in implementing their own new shower. Firstly, the code is in the `panscales` namespace. Inspecting the code, the user will see that rather than `double`, many variables are in `precision_type`: this corresponds to the precision that was chosen in the build step. Generically, `double` is used for logarithmic variables and acceptance probabilities, while `precision_type` is to be used for (non-logarithmic) kinematic variables and associated matrix-element calculations, where rounding errors and large exponents may be encountered.

Much of the core work of showering is carried out by the `ShowerRunner` class, which

is essentially agnostic as to the specific details of any given shower. The basic work flow of `ShowerRunner` is depicted in Fig. 5. The `ShowerRunner` class is constructed by passing a pointer to any class that derives from `ShowerBase`, whose role is to handle a small, well-defined set of shower-specific tasks, such as providing the acceptance probability and implementing the shower's kinematic map. The functions that the user should provide to the shower are also shown in Fig. 5. For concreteness, with this distribution, we have supplied two files, `ShowerUserDefined.hh` and `ShowerUserDefined.cc`, derived from `ShowerBase`, which one can modify. The functions that need to be touched are marked with "`USER-TODO`" in the code. The user-defined shower can then be run with the `-shower user-defined` command line argument. Note that the template is meant for e^+e^- showers. For additional features like initial-state radiation, matching or the implementation of double-soft currents, the reader should examine the structure of other existing showers.

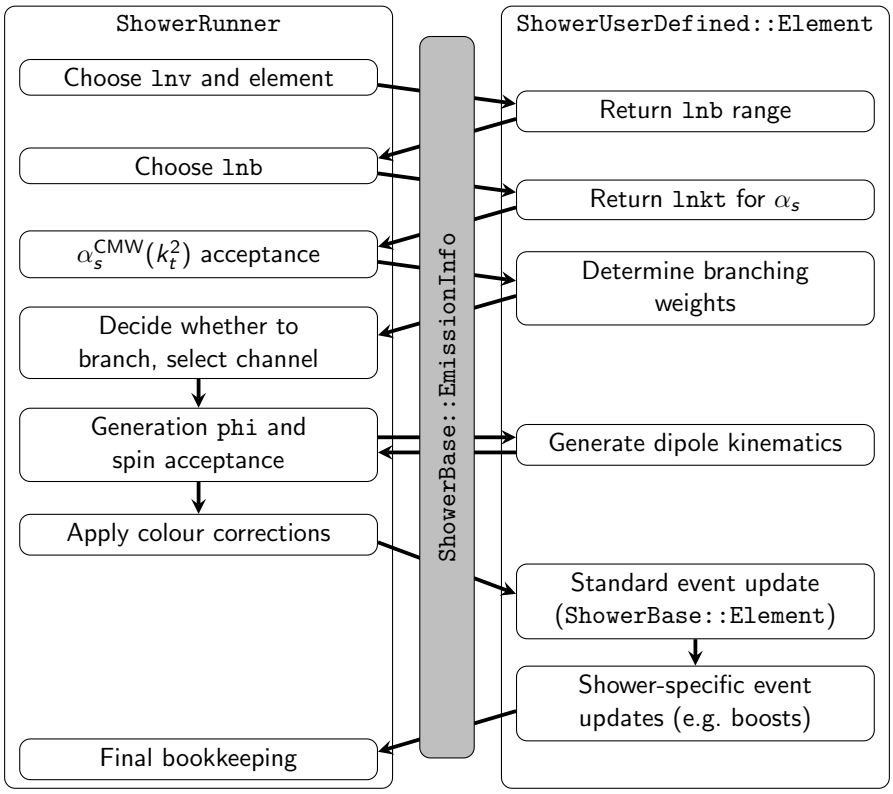


Figure 5: Simplified structure of a single trial for an emission in the shower. Steps on the left are performed by the centralised `ShowerRunner` class. Shower-specific steps, shown on the right, are the responsibility of the `ShowerUserDefined` implementation, as well as its sub-classes. The `EmissionInfo` sub-class stores all required information needed for the current branching, such as the splitting variables and constructed kinematics. The emitting dipole is accessible in the `Element` sub-class, which contains the majority of the shower-specific code. The matching, implementation of the spin correlations throughout the event evolution, and the double-soft splitting functionalities are not illustrated here.

The class `ShowerUserDefined` implements:

- Member functions that provide textual descriptive information about the shower.
- Member functions that provide structural information about the shower, notably whether the shower is a dipole shower (only the emitter splits), or an antenna shower

(both the emitter and spectator can split) via the function `only_emitter_splits()`, and `n_elements_per_dipoles()` (two for a dipole shower, one for an antenna). For showers with a global kinematic map, one where a dipole branching also impacts the kinematics of particles not in the dipole, the function `is_global()` should return `true` (more on this later).

- Member functions to create the two sub-classes `ShowerUserDefined::EmissionInfo`, and `ShowerUserDefined::Element`.

The two sub-classes encode the data associated with an emission and most of the implementation of the shower splitting:

- `ShowerUserDefined::EmissionInfo` derives from `ShowerBase::EmissionInfo`. A pointer to its base class is used by `ShowerRunner` as the main structure to keep track of the emission through the various steps of constructing that emission, and is generally passed to all shower-specific functions. Much of the required functionality is already present in the base class. Users may extend it to cache quantities computed at specific steps of a single emission that is trialled, so as to not reproduce them again at a later stage. An example would be the absolute k_t of an emission, or the longitudinal momentum fraction z, which might first be calculated in the early stages of Fig. 5 and then used later when generating the new dipole kinematics.

- `ShowerUserDefined::Element` derives from `ShowerBase::Element`. It is responsible for carrying out almost all of the shower-specific work. In case of multiple types of elements (i.e. II, IF and FF dipoles), one might choose to derive distinct element classes for each, typically each from an intermediate `ShowerUserDefined::Element` base class (see e.g. `ShowerDipoleKt`).

Let us focus here on the `Element` class. For a dipole shower, where each dipole end is associated with a distinct kinematic map, there will be two `Elements` per dipole, one for each end. For an antenna shower, there will be just one `Element` per dipole. The branching kinematics are governed by three variables:

- $\ln v$, the logarithm of the (dimensionful) ordering variable.

- $\ln b$, a logarithmic variable for the longitudinal degree of freedom of the emission. In the soft-collinear region of a back-to-back dipole it might, for example, map directly to a rapidity, or to $\ln z$ where z is the radiated particle's momentum fraction.

- ϕ, the azimuthal angle for the emission.

The main non-trivial member functions that need to be implemented are the following:

- `lnb_extent_const()` and `lnb_extent_lnv_coeff()`: for any given $\ln v$, `ShowerRunner` will take the $\ln b$ generation range to be taken equal to

$$\texttt{lnb\_extent\_const()} + \ln v \times \texttt{lnb\_extent\_lnv\_coeff()} \tag{5}$$

  where typically the first term would be a function of the dipole kinematics (as encoded in the corresponding `Element`) and the second term would not. The use of a simple analytic form for the extent facilitates the generation of the $\ln v$ distribution.

- `lnb_generation_range(lnv)`: for a given value of the evolution scale `lnv`, returns a `Range` object containing the limits of $\ln b$ generation. The difference between the upper and lower limits should coincide with the extent expected from the previous bullet point. Not all $\ln b$ values in the range need be kinematically valid.

- `lnv_lnb_max_density()`: returns the maximal emission density (for a $d\ln v\, d\ln b\, \frac{d\phi}{2\pi}$ measure). Typically this would be the soft-collinear limit of the emission density

$$\frac{dP}{d\ln v\, d\ln b\, d\phi/2\pi} = \frac{C_A \alpha_s^{\max}}{\pi} \tag{6}$$

  where $\alpha_s^{\max}$ is the maximal value that the strong coupling can take.[10]

- `lnkt_approx(lnv,lnb)`: returns the logarithm of the transverse momentum of an emitted parton with that $\ln v$, $\ln b$ combination. The result should be exact in the soft-collinear limit, but does not need to be exact in the soft large-angle or in the hard-collinear limits. It is typically used for the evaluation of $\alpha_s(k_t)$, as well as for some of the dynamic emission vetoing used for logarithmic accuracy tests.

- `eta_approx(lnv,lnb)`: similar, but returns the rapidity of the emitted parton in the soft-collinear limit, used in the computation of colour transition points and dynamic emission vetoing.

- `acceptance_probability(emission_info)`: sets information subsequently used by `ShowerRunner` in order to calculate the probability that the dipole splits. It makes use of the generation variables `lnv` and `lnb`, as cached in `emission_info`. Specifically it sets the following member variables in `emission_info`:

  - `emitter_weight_rad_gluon` : weight for the emitter to radiate a gluon;
  - `emitter_weight_rad_quark` : weight for the emitter to radiate a (anti-)quark;
  - `spectator_weight_rad_gluon` : weight for the spectator to radiate a gluon;
  - `spectator_weight_rad_quark` : weight for the spectator to radiate a (anti-)quark.

  The weights to radiate a gluon/quark depend on the splitting functions. Only for an antenna shower will the spectator weights be non-zero. They are to include a factor of the maximal allowed value of the strong coupling $\alpha_s^{\max}$, with the `ShowerRunner` class then accounting for an $\alpha_s(k_t)/\alpha_s^{\max}$ factor. In initial-state branchings, `ShowerRunner` also accounts for an appropriate PDF ratio factor. The user may choose to use the splitting functions implemented in `QCD.hh`. This can be done by calling `fill_dglap_splitting_weights`, which needs the longitudinal radiated momentum fraction z (where $z \to 0$ indicates the soft limit). In addition, when using spin correlations, the user should set

  - `z_radiation_wrt_emitter` : collinear momentum fraction with respect to the emitter;
  - `z_radiation_wrt_spectator`: collinear momentum fraction with respect to the spectator.

  The `acceptance_probability(...)` function returns a `bool`, where `false` indicates that the generation variables are definitely outside the kinematic limit. For most showers, if it returns `true` then the generation variables would normally be inside the kinematic limit. The emitter and spectator splitting probabilities are then used later in `ShowerRunner` to accept/reject a splitting, and choose the splitting channel.

[10] All current shower implementations have a private member pointer `Element::_shower`, and the base shower class has a `max_alphas()` member, as well as a `qcd()` member that supply access respectively to $\alpha_s^{\max}$ and QCD constants. Together these provide the information needed to calculate the maximum density.

- `do_kinematics(emission_info, rp)`: constructs the post-branching momenta of the emitter, the spectator and the newly radiated particle. For this, one may again use the cached generation variables `lnv`, `lnb` and `phi`, alongside any other quantity that the user stored in `emission_info`. The post-branching momenta should be stored in `emission_info` under the names `emitter_out`, `spectator_out` and `radiation`. This function returns a `bool`, where `true` indicates that the generation variables were inside the physical kinematic limit.[11] Note that the pre-branching emitter and spectator momenta should be taken from a variable `rp` of type `RotatedPieces`. This class is part of the framework for handling directional differences. It provides a rotated version of the dipole with one or other of its particles aligned along the z axis, which allows the user to retain high precision in the branching kinematics (specifically, small components along the x or y axis) without explicit knowledge of the underlying direction-difference structures. The direction-difference infrastructure then takes care of deducing the correct momenta and direction differences in the original frame.

- `update_event(...)`: In the `Element` base class, the `update_event(...)` member function takes care of replacing the pre-branching emitter and spectator particles with the post-branching ones, adds the radiated particle to the event, and takes care of some of the bookkeeping associated with colour handling. However, other particles in the event are by default not modified. Therefore, for showers with a global kinematic map, this function needs to implement additional operations on the rest of the event (e.g. a boost or rescaling). These would typically be preceded by an explicit call to `ShowerBase::Element::update_event(...)`. The event is then further processed by `ShowerRunner`, updating the event dipoles, colour and spin-density structure in addition to any caching associated with the event generation. Note that cached information associated with dipoles other than the newly-created dipole and the splitting dipole are by default not updated, unless the `ShowerUserDefined::is_global()` function returns true.

Once implemented, the new shower can be run by using the flag `-shower user-defined`.

If the user would like their shower to work with direction differences, they should inspect how this is implemented in existing showers. Aside from the `RotatedPieces` discussion above, calculations of dot products in determining dipole invariants should make explicit use of knowledge of direction differences (available from the `dirdiff_3_minus_3bar` member variable of `element.dipole()`), and `do_kinematics(...)` should use 3-vectors in its internal calculations to avoid triggering off-mass-shell errors. Furthermore if the shower carries out any global boosts, these need to be performed in a way that correctly boosts also the full set of dipole direction differences. The user is invited to inspect the code of existing showers for further details.

It is important also to test the correctness of the direction-differences implementation. Typically we do this by first running a double-precision build with a physically sensible range and verifying that results are identical with/without the `-use-diffs` argument. Then we create a build in the `doubleexp` type, running with `-use-diffs` and a logarithmic

[11] It is possible for the `acceptance_probability(...)` function to return true even outside the kinematic limits. In this case there are two possible avenues for imposing the kinematic limit. One is for the shower's `Element` class to overload the base-class member function `check_after_acceptance_probability(...)`, which gets called after α_s and PDF factors have been incorporated into the branching probabilities and those have been used to decide to continue with the emission generation. It is called before ϕ and the splitting channel are known. Alternatively `do_kinematics(...)` is called with knowledge of the ϕ value and channel, and can return `false` if the emission is outside the kinematics. The former is the only route that is currently valid in order for spin correlations to be correctly accounted for.

range of about 1000 (and correspondingly low α_s, so that the multiplicity stays of the order of 10−100) and compare the output to a build with the much slower `mpfr4096` type, running without `-use-diffs`. Again the results should be identical, though typically a few iterations are likely to be necessary to identify all sources of potential loss of precision. A final comment is that, by default, the logarithmic-accuracy tests of section 2.3 run in double precision, with a $\ln v$ range reaching about 300. However, for this to work, the shower should not ever do more than take the square of a momentum, otherwise the result will exceed the exponent that can be represented in double precision. If this is a problem, the user should modify a configuration flag in `example-global-nll-ee.py` so as to use `doubleexp` (which is somewhat slower).

5 Conclusions

This 0.1 series release of the PanScales code allows users to start exploring its features and techniques, notably for tests of logarithmic accuracy of parton showers. It also demonstrates an early version of the interface to Pythia8.3. While we do not yet recommend its use for phenomenological applications, we hope that this early release of the code will provide a foundation for exploring connections with other projects, so as to enable the wide ecosystem of collider physics tools to benefit from the validated logarithmic accuracy of the PanScales showers.

Acknowledgements

We thank Frédéric Dreyer for his contributions to the PanScales project and code during the initial stages of the project. We are grateful to Peter Skands and Silvia Zanoli for testing a pre-release version of the code and for helpful comments.

This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 788223), by a Royal Society Research Professorship (RP\R1\180112, GPS and LS, and RP\R\231001, GPS) and by the Science and Technology Facilities Council (STFC) under grants ST/T000864/1 (MvB, GPS), ST/X000761/1 (GPS), ST/T000856/1 (KH) and ST/X000516/1 (KH), ST/T001038/1 (MD) and ST/00077X/1 (MD). LS is supported by the Australian Research Council through a Discovery Early Career Researcher Award (project number DE230100867). The work of PM is funded by the European Union (ERC, grant agreement No. 101044599). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. We also thank each others' institutes for hospitality during the course of this work.

A Event record

A.1 Basic event information

Here we give an example event record, printed by default for the first event of any run (the first N events can be printed with the command-line option `-max-print N`).

```
iev = 0
Event weight = 1
Base (unshowered) event:
Q =              0            0            0       91.1876  pt =          0 y  =          0 phi=          0 m2 =     8315.18
    0            0            0      -45.5938       45.5938  pt =          0 y  =    -100000 phi=          0 m2 =           0  ID=   -1 IS=1 HS=0 DIS=0
    1            0            0       45.5938       45.5938  pt =          0 y  =     100000 phi=          0 m2 =           0  ID=    1 IS=2 HS=0 DIS=0
    2            0            0            0       91.1876  pt =          0 y  =          0 phi=          0 m2 =     8315.18  ID=   23 IS=0 HS=1 DIS=0
Showered event:
Q =              0            0            0       91.1876  pt =          0 y  =          0 phi=          0 m2 =     8315.18
    0            0            0      -570.915       570.915  pt =          0 y  =    -100000 phi=          0 m2 =           0  ID=   21 IS=1 HS=0 DIS=0
    1            0            0       520.429       520.429  pt =          0 y  =     100000 phi=          0 m2 =           0  ID=   21 IS=2 HS=0 DIS=0
    2      3.11993      3.07746            0       91.2928  pt =    4.38232 y  =          0 phi=   0.778545 m2 =     8315.18  ID=   23 IS=0 HS=1 DIS=0
    3     -2.47555     0.733957      -132.101       132.126  pt =    2.58206 y  =   -4.62822 phi=    2.85337 m2 =           0  ID=    1 IS=0 HS=0 DIS=0
    4     0.176626     -1.17714      8.23204       8.31765  pt =    1.19031 y  =    2.63215 phi=    4.86132 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
    5     0.225872    -0.693864     -10.6021       10.6272  pt =   0.729702 y  =   -3.3705 phi=     5.0271 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
    6     0.806285    0.0490162      3.51406        3.6057  pt =   0.807773 y  =    2.17635 phi=   0.060718 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
    7     -1.50297     0.278166     -110.974       110.985  pt =     1.5285 y  =   -4.97821 phi=    2.95859 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
    8     0.334135    -0.240379     -35.8056       35.8079  pt =   0.411616 y  =   -5.15895 phi=    5.65955 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
    9     0.284056     0.202652      87.4383        87.439  pt =   0.348935 y  =    6.21695 phi=   0.619678 m2 =           0  ID=   -1 IS=0 HS=0 DIS=0
   10    0.0393766     -0.98628     -55.6666       55.6753  pt =   0.987066 y  =   -4.72562 phi=    4.75229 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
   11     0.548843    -0.631906      1.05701       1.34826  pt =   0.836979 y  =    1.05562 phi=    5.42756 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
   12     0.547439    -0.663149     -4.74753       4.82478  pt =   0.859916 y  =   -2.40979 phi=     5.4025 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
   13     0.803506   -0.0658898       374.22        374.22  pt =   0.806203 y  =    6.83341 phi=    6.20137 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
   14     -2.90755     0.117353      -175.05       175.074  pt =    2.90992 y  =   -4.79016 phi=    3.10125 m2 =           0  ID=   21 IS=0 HS=0 DIS=0
```

The printout includes the hard event before showering, and the final fully showered event. The `ID` corresponds to PDG code for the particle, a non-zero value of `IS` indicates that the particle is in the initial state and `HS` is non-zero for particles that are considered to be part of the hard system (here, particle 2, which is the Z boson). The `DIS` variable is relevant only to DIS and VBF processes and in particular for the latter it indicates which of the two sides of the VBF process a given particle is associated with. From the point of view of an analysis, all particles with `IS` equal to zero are valid final-state particles. Within an analysis, the particles are available through `f_event.particles()`, which returns a vector of `Particle` objects. That same `f_event` object also can be used to access the event dipoles.

A.2 Access to step-by-step event information

As discussed in Section 2.2, by default PanScales does not store the event structure separately after each branching. However it is possible to ask for the corresponding information to be stored. This is achieved by calling `f_shower_runner.set_step_by_step_event_caching(true)` in the `user_startup` phase of an analysis. Once this is done, after each event the user can access `f_shower_runner->cached_step_by_step_event()` which has various enquiry functions, for example for retrieving the branching variables or the event at any given step of the shower. Note that this interface may still evolve.

For debugging purposes, there is also a compile-time option (`-DPSVERBOSE=on`) that can be passed to cmake. When enabled, running a shower provides very extensive output as to the shower's individual steps.

A final feature to be aware of is that in most cases the order of emissions is preserved in the event record, so the last particle in the event record corresponds to the last emission made by the shower. However, if running with double-soft corrections there is an internal "swap" procedure [9] that exchanges partons. In this case, the order of partons does not always correspond to the order of generation.

References

[1] M. Dasgupta, F. A. Dreyer, K. Hamilton, P. F. Monni, G. P. Salam and G. Soyez, *Parton showers beyond leading logarithmic accuracy*, Phys. Rev. Lett. **125**(5), 052002 (2020), doi:10.1103/PhysRevLett.125.052002, 2002.11114.

[2] K. Hamilton, R. Medves, G. P. Salam, L. Scyboz and G. Soyez, *Colour and logarithmic accuracy in final-state parton showers*, JHEP **03**, 041 (2021), doi:10.1007/JHEP03(2021)041, 2011.10054.

[3] A. Karlberg, G. P. Salam, L. Scyboz and R. Verheyen, *Spin correlations in final-state parton showers and jet observables*, Eur. Phys. J. C **81**(8), 681 (2021), doi:10.1140/epjc/s10052-021-09378-0, 2103.16526.

[4] K. Hamilton, A. Karlberg, G. P. Salam, L. Scyboz and R. Verheyen, *Soft spin correlations in final-state parton showers*, JHEP **03**, 193 (2022), doi:10.1007/JHEP03(2022)193, 2111.01161.

[5] M. van Beekveld, S. Ferrario Ravasio, G. P. Salam, A. Soto-Ontoso, G. Soyez and R. Verheyen, *PanScales parton showers for hadron collisions: formulation and fixed-order studies*, JHEP **11**, 019 (2022), doi:10.1007/JHEP11(2022)019, 2205.02237.

[6] M. van Beekveld, S. Ferrario Ravasio, K. Hamilton, G. P. Salam, A. Soto-Ontoso, G. Soyez and R. Verheyen, *PanScales showers for hadron collisions: all-order validation*, JHEP **11**, 020 (2022), doi:10.1007/JHEP11(2022)020, 2207.09467.

[7] K. Hamilton, A. Karlberg, G. P. Salam, L. Scyboz and R. Verheyen, *Matching and event-shape NNDL accuracy in parton showers*, JHEP **03**, 224 (2023), doi:10.1007/JHEP03(2023)224, 2301.09645.

[8] M. van Beekveld and S. Ferrario Ravasio, *Next-to-leading-logarithmic PanScales showers for Deep Inelastic Scattering and Vector Boson Fusion* (2023), 2305.08645.

[9] S. Ferrario Ravasio, K. Hamilton, A. Karlberg, G. P. Salam, L. Scyboz and G. Soyez, *Parton Showering with Higher Logarithmic Accuracy for Soft Emissions*, Phys. Rev. Lett. **131**(16), 161906 (2023), doi:10.1103/PhysRevLett.131.161906, 2307.11142.

[10] M. Dasgupta, F. A. Dreyer, K. Hamilton, P. F. Monni and G. P. Salam, *Logarithmic accuracy of parton showers: a fixed-order study*, JHEP **09**, 033 (2018), doi:10.1007/JHEP09(2018)033, 1805.09327.

[11] M. Dasgupta and B. K. El-Menoufi, *Dissecting the collinear structure of quark splitting at NNLL*, JHEP **12**, 158 (2021), doi:10.1007/JHEP12(2021)158, 2109.07496.

[12] A. Banfi, F. A. Dreyer and P. F. Monni, *Next-to-leading non-global logarithms in QCD*, JHEP **10**, 006 (2021), doi:10.1007/JHEP10(2021)006, 2104.06416.

[13] A. Banfi, F. A. Dreyer and P. F. Monni, *Higher-order non-global logarithms from jet calculus*, JHEP **03**, 135 (2022), doi:10.1007/JHEP03(2022)135, 2111.02413.

[14] M. Dasgupta, B. K. El-Menoufi and J. Helliwell, *QCD resummation for groomed jet observables at NNLL+NLO*, JHEP **01**, 045 (2023), doi:10.1007/JHEP01(2023)045, 2211.03820.

[15] R. Medves, A. Soto-Ontoso and G. Soyez, *Lund and Cambridge multiplicities for precision physics*, JHEP **10**, 156 (2022), doi:10.1007/JHEP10(2022)156, 2205.02861.

[16] R. Medves, A. Soto-Ontoso and G. Soyez, *Lund multiplicity in QCD jets*, JHEP **04**, 104 (2023), doi:10.1007/JHEP04(2023)104, 2212.05076.

[17] M. van Beekveld, M. Dasgupta, B. K. El-Menoufi, J. Helliwell and P. F. Monni, *Collinear fragmentation at NNLL: generating functionals, groomed correlators and angularities* (2023), 2307.15734.

[18] S. Höche, D. Reichelt and F. Siegert, *Momentum conservation and unitarity in parton showers and NLL resummation*, JHEP **01**, 118 (2018), doi:10.1007/JHEP01(2018)118, 1711.03497.

[19] G. Bewick, S. Ferrario Ravasio, P. Richardson and M. H. Seymour, *Logarithmic accuracy of angular-ordered parton showers*, JHEP **04**, 019 (2020), doi:10.1007/JHEP04(2020)019, 1904.11866.

[20] G. Bewick, S. Ferrario Ravasio, P. Richardson and M. H. Seymour, *Initial state radiation in the Herwig 7 angular-ordered parton shower*, JHEP **01**, 026 (2022), doi:10.1007/JHEP01(2022)026, 2107.04051.

[21] J. R. Forshaw, J. Holguin and S. Plätzer, *Building a consistent parton shower*, JHEP **09**, 014 (2020), doi:10.1007/JHEP09(2020)014, 2003.06400.

[22] Z. Nagy and D. E. Soper, *Summations of large logarithms by parton showers*, Phys. Rev. D **104**(5), 054049 (2021), doi:10.1103/PhysRevD.104.054049, 2011.04773.

[23] Z. Nagy and D. E. Soper, *Summations by parton showers of large logarithms in electron-positron annihilation* (2020), 2011.04777.

[24] F. Herren, S. Höche, F. Krauss, D. Reichelt and M. Schoenherr, *A new approach to color-coherent parton evolution*, JHEP **10**, 091 (2023), doi:10.1007/JHEP10(2023)091, 2208.06057.

[25] B. Assi and S. Höche, *A new approach to QCD evolution in processes with massive partons* (2023), 2307.00728.

[26] C. Bierlich *et al.*, *A comprehensive guide to the physics and usage of PYTHIA 8.3*, SciPost Phys. Codeb. **2022**, 8 (2022), doi:10.21468/SciPostPhysCodeb.8, 2203.11601.

[27] P. Skands, S. Carrazza and J. Rojo, *Tuning PYTHIA 8.1: the Monash 2013 Tune*, Eur. Phys. J. C **74**(8), 3024 (2014), doi:10.1140/epjc/s10052-014-3024-y, 1404.5630.

[28] G. S. Chahal and F. Krauss, *Cluster Hadronisation in Sherpa*, SciPost Phys. **13**(2), 019 (2022), doi:10.21468/SciPostPhys.13.2.019, 2203.11385.

[29] M. R. Masouminia and P. Richardson, *Hadronization and Decay of Excited Heavy Hadrons in Herwig 7* (2023), 2312.02757.

[30] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier and P. Zimmermann, *Mpfr: A multiple-precision binary floating-point library with correct rounding*, ACM Trans. Math. Softw. **33**(2), 13–es (2007), doi:10.1145/1236463.1236468.

[31] Y. Hida, X. S. Li and D. H. Bailey, *Quad-double arithmetic: Algorithms, implementation, and application*, In *15th IEEE Symposium on Computer Arithmetic*, pp. 155–162 (2000).

[32] M. Cacciari, G. P. Salam and G. Soyez, *FastJet User Manual*, Eur. Phys. J. **C72**, 1896 (2012), doi:10.1140/epjc/s10052-012-1896-2, 1111.6097.

[33] G. P. Salam and J. Rojo, *A Higher Order Perturbative Parton Evolution Toolkit (HOPPET)*, Comput. Phys. Commun. **180**, 120 (2009), doi:10.1016/j.cpc.2008.08.010, 0804.3755.

[34] A. Buckley, J. Ferrando, S. Lloyd, K. Nordström, B. Page, M. Rüfenacht, M. Schönherr and G. Watt, *LHAPDF6: parton density access in the LHC precision era*, Eur. Phys. J. C **75**, 132 (2015), doi:10.1140/epjc/s10052-015-3318-8, 1412.7420.

[35] Y. L. Dokshitzer, G. D. Leder, S. Moretti and B. R. Webber, *Better jet clustering algorithms*, JHEP **08**, 001 (1997), doi:10.1088/1126-6708/1997/08/001, hep-ph/9707323.

[36] F. A. Dreyer, G. P. Salam and G. Soyez, *The Lund Jet Plane*, JHEP **12**, 064 (2018), doi:10.1007/JHEP12(2018)064, 1807.04758.

[37] T. Sjostrand and P. Z. Skands, *Transverse-momentum-ordered showers and interleaved multiple interactions*, Eur. Phys. J. **C39**, 129 (2005), doi:10.1140/epjc/s2004-02084-y, hep-ph/0408302.

[38] S. Schumann and F. Krauss, *A Parton shower algorithm based on Catani-Seymour dipole factorisation*, JHEP **03**, 038 (2008), doi:10.1088/1126-6708/2008/03/038, 0709.1027.

[39] S. Platzer and S. Gieseke, *Coherent Parton Showers with Local Recoils*, JHEP **01**, 024 (2011), doi:10.1007/JHEP01(2011)024, 0909.5593.

[40] A. Buckley, P. Ilten, D. Konstantinov, L. Lönnblad, J. Monk, W. Pokorski, T. Przedzinski and A. Verbytskyi, *The HepMC3 event record library for Monte Carlo event generators*, Comput. Phys. Commun. **260**, 107310 (2021), doi:10.1016/j.cpc.2020.107310, 1912.08005.

[41] S. Dulat, T.-J. Hou, J. Gao, M. Guzzi, J. Huston, P. Nadolsky, J. Pumplin, C. Schmidt, D. Stump and C. P. Yuan, *New parton distribution functions from a global analysis of quantum chromodynamics*, Phys. Rev. D **93**(3), 033006 (2016), doi:10.1103/PhysRevD.93.033006, 1506.07443.

[42] C. Bierlich *et al.*, *Robust Independent Validation of Experiment and Theory: Rivet version 3*, SciPost Phys. **8**, 026 (2020), doi:10.21468/SciPostPhys.8.2.026, 1912.05451.

[43] S. Catani, Y. L. Dokshitzer, F. Fiorani and B. R. Webber, *Average number of jets in e^+e^- annihilation*, Nucl. Phys. **B377**, 445 (1992), doi:10.1016/0550-3213(92)90296-N.

[44] A. Banfi, G. P. Salam and G. Zanderighi, *Principles of general final-state resummation and automated implementation*, JHEP **03**, 073 (2005), doi:10.1088/1126-6708/2005/03/073, hep-ph/0407286.

[45] L. Lönnblad, *Fooling Around with the Sudakov Veto Algorithm*, Eur. Phys. J. C **73**(3), 2350 (2013), doi:10.1140/epjc/s10052-013-2350-9, 1211.7204.

[46] Y. Hatta and T. Ueda, *Resummation of non-global logarithms at finite N_c*, Nucl. Phys. **B874**, 808 (2013), doi:10.1016/j.nuclphysb.2013.06.021, 1304.6930.

[47] Y. Hatta and T. Ueda, *Non-global logarithms in hadron collisions at $N_c = 3$*, Nucl. Phys. B **962**, 115273 (2021), doi:10.1016/j.nuclphysb.2020.115273, 2011.04154.

[48] T. Becher, N. Schalch and X. Xu, *Resummation of Next-to-Leading Non-Global Logarithms at the LHC* (2023), 2307.02283.