Preprint typeset in JHEP style - PAPER VERSION

March 16, 2011

POWHEG BOX @ rev. ***************************

(release 1.0)

The POWHEG BOX user manual: common features

Simone Alioli

Deutsches Elektronen-Synchrotron DESY
Platanenallee 6, D-15738 Zeuthen, Germany
E-mail: simone.alioli@desy.de

Paolo Nason

INFN, Sezione di Milano-Bicocca, Piazza della Scienza 3, 20126 Milan, Italy
E-mail: Paolo.Nason@mib.infn.it

Carlo Oleari

Università di Milano-Bicocca and INFN, Sezione di Milano-Bicocca
Piazza della Scienza 3, 20126 Milan, Italy
E-mail: Carlo.Oleari@mib.infn.it

Emanuele Re

Institute for Particle Physics Phenomenology, Department of Physics
University of Durham, Durham, DH1 3LE, UK
E-mail: emanuele.re@durham.ac.uk

Abstract: This note documents the use of the package POWHEG BOX. In this document the user will find a description of the general features of the package. Issues related to specific processes are discussed in dedicated manuals available in the corresponding folders.

Keywords: POWHEG, Shower Monte Carlo, NLO.

Contents

1. Introduction

The `POWHEG BOX` program is a framework to implement Next-to-Leading-Order (NLO) calculations in Shower Monte Carlo (SMC) programs according to the `POWHEG` method. An explanation of the method and a detailed discussion of how the code is organized can be found in refs. [1, 2, 3]. The code is distributed according to the "MCNET GUIDELINES for Event Generator Authors and Users" and can be found in

`http://powhegbox.mib.infn.it`.

The latest version of the package can be downloaded trough SVN:

```
$ svn checkout --username anonymous --password anonymous
             svn://powhegbox.mib.infn.it/trunk/POWHEG-BOX
```

Previous revisions are available using the `[--revision n]` option. Periodic releases of the code will be available in `svn://powhegbox.mib.infn.it/tags`.

2. Organization of the package

In the POWHEG BOX directory (the main directory, from now on) there are the common source files, an include directory, a Docs directory and several other process folders, with a name reminiscent of the process to which they correspond: the Z folder contains the code to simulate the Drell-Yan process at NLO with POWHEG, Zj the code to simulate Z in association with 1 jet at NLO, and so on.

Each folder contains a Makefile and some specific source files which are compiled and linked with the common files to build an independent program. Therefore, an executable built in a process directory is a self-contained program: it needs only to be linked to dynamic libraries, according to the corresponding Makefile. In the following, we denote a generic process folder as proc_dir.

Once downloaded, an executable related to a given process can be built with the following commands:

```
$ cd POWHEG-BOX/proc_dir
$ make <target>
```

where the choice of the <target> depends upon the way one wants to interface the program with a Shower Monte Carlo. Specifically, <target> needs to refer to one of the targets contained in the Makefile. All the process directories contain in their Makefile instructions to build the following programs:

```
pwhg_main
lhef_analysis
main-HERWIG-lhef
main-PYTHIA-lhef
```

For some processes, other executables can be built. If this is the case, we refer to the specific process manual for more details. In the next section, we will briefly describe the purpose of the aforementioned programs.

From release 1.0, the POWHEG BOX program also includes a stand-alone PDF package, originally due to M.L. Mangano. In order to use this package, it is enough to set the Makefile variable PDF equal to native. When this is done, the program doesn't need to be linked against any external PDF library as LHAPDF, and therefore can be run out-of-the-box. Some data files needed for this package are in the POWHEG-BOX/pdfdata directory, and a symbolic link of the data file to the directory where the run is performed is needed. The name of the link should not include the .tbl or .dat ending of the data file (for example cteq6m.tbl should be linked to cteq6m).[1]

If instead one needs other PDF sets, the user is asked to have the LHAPDF library [4] installed on his/her system and to take care to insert its correct search path in the Makefile, or, simply, to add the path of the lhapdf-config executable to the $PATH environmental

[1]In several testrun directories the user will find the link already present.

variable. We remind that in case of linking against "dynamic" shared library, the correct LHAPDF library path should also be added to the $LD_LIBRARY_PATH environmental variable, otherwise run time errors may occur.

For some processes, the default analysis routine that comes with the package relies on jet algorithms, as can be inferred from the Makefile or by direct inspection of the pwhg_analysis.f file in proc_dir. When this is the case, in order to build the program the user needs to have the FASTJET library [5] installed. It is up to the user to correctly install it and to modify the Makefile accordingly. For most systems, adding the fastjet-config executable to the $PATH environmental variable is enough. A dummy analysis file not invoking FASTJET is also provided, for users that have problems linking the FASTJET library.

The Makefile is set up to use the compiler gfortran on Linux platforms. If one wishes to use g77, one should change the appropriate lines in the Makefile. The ifort compiler has also been tested for some specific processes.

For some processes, other special actions are needed to install the program. In those cases, we refer again to the specific process manuals.

3. Modes of operation

The main purpose of our implementations of the POWHEG method is to generate hard events that can then be fed into a SMC program for subsequent showering. To this aim, POWHEG BOX saves the hard event information according to the conventions of the Les Houches Interface for User Processes (LHIUP from now on) [6]. The SMC should also comply with these conventions (as is the case for PYTHIA and HERWIG) in order to be used in conjunction with the POWHEG BOX.

Having this in mind, we now briefly explain the purpose of the 4 programs we mentioned above.

- pwhg_main: when this executable is run, the program performs several steps, which are documented in detail in [3]. The final output is the file with extension .lhe (standing for Les Houches events), that contains the generated hard events, written according to the format described in ref. [7]. Several other files are produced where internal information are stored (the most important ones are .dat files), and some .top files with plots. Since this is the main program, a more detailed description on how to run the program, write or modify input cards and interpret the results is given later in this document.

- lhef_analysis: when this executable is run, the program performs an analysis of the events in the .lhe file. The purpose of this program is to allow a quick analysis of partonic events before showering them. Therefore it is useful mainly for developers rather than for users. The output is the file LHEF_analysis.top, where NLO distributions with (N)LL resummation of soft gluon effects are present.

- main-HERWIG-lhef and main-PYTHIA-lhef: these programs are used to feed the events stored in the .lhe file to the HERWIG or PYTHIA program, to obtain a full event simulation. The corresponding main sources (main-HERWIG.f and main-PYTHIA.f) are located in the main directory. However, some of the routines called therein need to be customized differently for different processes: for this reason, their source code has been placed in the process-dependent files setup-HERWIG-lhef.f and setup-PYTHIA-lhef.f.[2] These files are also important because they show how to call the template analysis at the end of the event generation. Having said this, we also recall that these drivers should be considered as templates to write an interface to use POWHEG within the experimental collaboration software, since the information in the .lhe files are already written according to a standard, documented format [7].

The typical procedure to simulate events with our POWHEG implementation is performed in two stages, which we briefly describe in the following:

1. **Generating the hard events**
   The first step of a POWHEG simulation is to generate and store the hard events in a file, which we call the *event file*. The format of the event file supported by POWHEG BOX is the "Standard format for Les Houches event files", documented in ref. [7]. Les Houches Event Files (LHEF from now on) generated with our program have the suffix .lhe. The program for the generation of LHEF can be built from a proc_dir with the command

   ```
   $ make pwhg_main
   ```

   The program must be executed in a directory where the file powheg.input is present. The only libraries needed by this program to work are LHAPDF and, in some cases, FASTJET. The event file is named pwgevents.lhe. By using an input card with a prefix, the user is given the possibility to change the name of this file and of all the other output files, as documented in sec. 4.

2. **Perform the parton shower**
   The subsequent step is to read the events and process them with the SMC. An example program that reads the event file, showers it with HERWIG and analyzes the results can be built as follows

   ```
   $ make main-HERWIG-lhef
   ```

   A similar program, named main-PYTHIA-lhef, is provided for PYTHIA, and can be built with the command

   ```
   $ make main-PYTHIA-lhef
   ```

[2] Notice that this structure has been introduced from release 1.0, whereas in previous versions all the driver files were placed in the processes subdirectories, and were named main-HERWIG-lhef.f and main-PYTHIA-lhef.f

A version of PYTHIA and HERWIG is included in the POWHEG BOX package. These can be substituted by the user's favorite version. In the case of HERWIG, the appropriate include files should also be substituted.

3.1 The analysis routines

The file pwhg_analysis.f contains a template analysis, that one can take as a starting point for more complex analysis. It uses pwhg_bookhist, the histogramming package of M.L. Mangano with minor modifications, and it produces topdrawer outputs. The routines in pwhg_analysis.f are adequate for both fortran HERWIG and PYTHIA since they rely on the standard common blocks of ref. [8]. If the user wants to use other analysis routines, he/she can simply modify the pwhg_analysis.f file or write his/her own.[3]

4. Input parameters

When generating the hard events, the pwhg_main program needs to set some physical and some technical parameters. Some of these parameters are mandatory, some other are not. Moreover, there are some parameters which are needed only for some processes. In the POWHEG BOX an independent facility to set these input parameters is available. All parameters are stored in a file, named powheg.input. If the file powheg.input is not present, the program asks the user to enter a prefix, and then looks for the file <prefix>-powheg.input. In this case, all the output files created by POWHEG BOX in the current run will carry the prefix <prefix>- instead of pwg.

Examples of powheg.input files can be found in the testrun subdirectory of proc_dir. The format of these files is as follows

1. Lines are no more than 100 characters long.

2. Empty (blank) lines are ignored

3. If a # or a ! appears at any point in a line, the part of the line starting from the # or ! symbol up to its end is blanked.

4. An entry has the format:
   `name   value`
   usually followed by a ! and a comment to clarify the meaning of the variable. The name keyword has no more than 20 characters, and value is an integer or floating point number.

5. A maximum of 100 keywords are allowed.

The order in which the lines are written is not important, although it is a good practice to put the mandatory tokens at the beginning of the file.

[3]During the integration stage and/or during the generation of the event file, the possibility to perform a NLO analysis or an analysis at the level of the POWHEG output, before interfacing to the shower, is left to the user. In these cases, the analysis is executed with the string WHCPRG set to 'NLO'.

The input parameters are read by the (`real * 8`) function `powheginput(string)`, whose source code is in the file `powheginput.f`. The statement

```
rvalue=powheginput('myparm')
```

returns the value of the token `myparm` stored in `powheg.input`. If the token is not found in the input file, a message is printed, and the program is stopped. The file is read only once, on the first invocation of the function `powheginput`, and token-value pairs are stored in internal arrays, so that subsequent calls to `powheginput` are relatively fast. The statement

```
rvalue=powheginput('#myparm')
```

also returns the value of the token `myparm`. However, in case the token `myparm` is not present, the program does not stop, and returns the value -10^6. This is used for optional keywords, that are given a default value when absent.

The file `powheginput.f` is a stand-alone code, and can be linked to any program. In this way, a SMC that is reading an event file may get parameters of the `POWHEG BOX` run, if needed.

In the rest of this section we describe the typical lines of an input file.

4.1 Mandatory parameters

```
numevts 100000 !  number of events to be generated
ih1 1          !  hadron 1 type (1:  proton; -1:  antiproton)
ih2 1          !  hadron 2 type (1:  proton; -1:  antiproton)
lhans1 10050   !  pdf set for hadron 1 ( LHAGLUE number )
lhans2 10050   !  pdf set for hadron 2 ( LHAGLUE number )
ndns1 131 !  pdf for hadron 1 ( when using the hvqpdf package )
ndns2 131 !  pdf for hadron 2 ( when using the hvqpdf package )
```

The first entry is self-explanatory. The integers `ih1,ih2` and `lhans1,lhans2` (or `ndns1,ndns2`) characterize instead the hadron type and the PDF set used by `POWHEG BOX`. The hadron type in `ih1` and `ih2` can be 1 for a proton or -1 for an antiproton. When in the `Makefile` one sets `PDF=native`,[4] one of the 2 internal PDF sets is used (see sec. 2). In this case, the set is chosen accordingly to the value of the tokens `ndns1, ndns2`, where the hvqpdf numbering is assumed. Otherwise, if the `Makefile` variable `PDF` is set equal to `lhapdf`, then the program uses the LHAPDF library, and the set is chosen according to the tokens `lhans1, lhans2`. In this case, the numbering scheme is that of the LHAGLUE interface, leaving the possibility of re-evaluate pdf's on the fly (using number corresponding to .LHpdf file) or to interpolate from a previously calculated grid (number corresponding to .LHgrid file), as explained in ref. [4]. In the example above, 10050 corresponds to the central value of

[4]From release 1.0 this is the default behaviour of the program as it is distributed.

the `CTEQ6M` set in this latter case.

```
ebeam1 7000 !  energy of beam 1 in GeV
ebeam2 7000 !  energy of beam 2 in GeV
```

These are the energies of the two beams given in GeV. We assume that beam 1 and 2 move along the third axis in the positive and negative direction respectively.

```
!  Parameters to allow or not the use of stored data
use-old-grid 1
use-old-ubound 1
```

The meaning of these tokens requires a little knowledge of the operation of `POWHEG BOX`. Before events can be generated, the program goes through an initialization stage. It first computes the importance sampling grid for the computation of the inclusive cross section. The importance sampling data is stored in the file `pwgxgrid.dat`. Then, the integral is computed, and an upper bounding envelope is found for the unweighted generation of the underlying Born configurations. This envelope, together with the importance sampling data and the cross section are stored in the file `pwggrid.dat`. Afterwords, a grid for the normalization of the upper bounding function for the generation of radiation is computed. The normalization grid is stored in the file `pwgubound.dat`.

The generation of the grids is time consuming, but the time spent in this calculation is negligible in a normal run, when hundreds of thousands of events are generated. On the other hand, sometimes it is useful (for example, when debugging an analysis program) to skip the grid generation stage. For this purpose, if the `use-old-grid` token is set equal to 1, and if `pwggrid.dat` exists and is consistent, it is loaded, and the old grid and old value of the cross section are used. Otherwise, first the program checks if a `pwgxgrid.dat` file exists. If it does, the file is loaded, and the generation of the importance sampling grid is skipped. Observe that the program does check the file for consistency with the current run, but the check is not exhaustive. The user should make sure that a consistent grid is used.
The token `use-old-ubound` has the same role as `use-old-grid`, but it applies to the step where the normalization of the upper bounding function is filled.
The following parameters are used to control the generation of these grids:

```
!  Parameters that control the grid for Born variables generation
ncall1 50000    !  number of calls for initializing the integration grid
itmx1 5         !  number of iterations for initializing the integration grid
ncall2 50000    !  number of calls for computing integral
itmx2 5         !  number of iterations for computing integral
foldcsi 1       !  number of folds on x integration
foldy 1         !  number of folds on y integration
foldphi 1       !  number of folds on phi integration
```

```
nubound 50000    !  number of calls to setup upper bounds for radiation
iymax 1          !  <=100, number of intervals in y grid to compute upper bounds
icsimax 1        !  <=100, number of intervals in csi grid
xupbound 2       !  increase upper bound for radiation generation by given factor
```

In the `proc_dir` directories there are example `powheg.input` files with the settings of these variables that have been found to be satisfactory for the specific parameters that have been used. The values of some of the tokens may be changed in the following cases:

- The integration does not seem to converge well, and the file `pwgbtlgrid.top` representing the importance sampling subdivision of each integration coordinate for each iteration `itmx1`, does not seem to converge well. In this case, `ncall1`, and if that does not work also `itmx1` should be increased.
- The integration results have large errors. One may try to increase `ncall2`, `itmx2`.
- The number of upper bound failure for the generation of the inclusive cross section, reported in `pwgcounters.dat`, is an important fraction of the total number of generated events. As a rule of thumb, one might expect that fraction to represent the error on the generated distribution. In this case, increase `ncall2`, `itmx2`.
- If the fraction of negative weights is large, one may perform a so-called "folded integration". This procedure has been described in refs. [2, 3, 9], to which we refer the reader more details. The procedure is enabled by setting at least one of the parameters `foldcsi`, `foldy`, `foldphi` to a value different from 1: allowed values are 1, 2, 5, 10, 25, 50. The speed of the program is inversely proportional to the product of these numbers, so that a reasonable compromise should be found. For processes where the fraction of negative weights in the $\bar{B}$ calculation is non negligible, the corresponding input card comes already with reasonable folding parameters set.
- If there are too many upper bound violations in the generation of radiation, one may increase `nubound`, and/or `xupbound`.
- If the efficiency in the generation of radiation is too small, one may try to increase `iymax`, `icsimax`.

In order to check whether any of these conditions occurs, the user should inspect the files `pwgstat.dat` and `pwgcounters.dat` at the end of the run, as explained in section 6. Information present in these files are also printed on the shell during the run.

The input cards of some processes need more parameters, such as the value of masses, widths, couplings, etc. Specific information can be found in the corresponding dedicated manual.

The tokens to control scale variations are not mandatory parameters. However, we describe them here:

```
facscfact 1 !  factorization scale factor:  mufact=muref*facscfact
renscfact 1 !  renormalization scale factor:  muren=muref*renscfact
```

Factorization and renormalization scale factors appearing here have to do with the computation of the inclusive cross section (i.e. the $\bar{B}$ function [1, 2, 3]), and can be varied by a factor of order 1 to study scale dependence. As usual, the value used as central value depends on the process at hand. The exact details for the process at hand can be found in the dedicated manual. The relevant `fortran` code can be found in the `set_fac_ren_scales` routine (`Born_phsp.f` file), which the experienced user can modify at his/her will and risk.

4.2 Optional parameters

In addiction to the mandatory parameters presented above, `POWHEG BOX` also accepts other parameters.
For some processes, the use of these parameters is actually needed (in some cases, they are mandatory!). In these cases, proper warnings and instructions are present in the dedicated manual, and the input files have the corresponding lines uncommented and with the tokens set to proper values.
In general, however, the user should not worry if these parameters are not present or are commented. This means that there is no need to use them, and the default values are used. In the following, we describe them, since they can be useful for a more advanced use of the program: who is not interested can safely skip this section.

```
QCDlambda5 0.25       !  for not equal pdf sets
ptsqmin 0.8           !  (default 0.8 GeV) minimum pt for generation of radiation
charmthr 1.5          !  (default 1.5 GeV) charm threshold for gluon splitting
bottomthr 5.0         !  (default 5.0 GeV) bottom threshold for gluon splitting
charmthrpdf 1.5       !  (default 1.5 GeV) pdf charm threshold
bottomthrpdf 5.0      !  (default 5.0 GeV) pdf bottom threshold
```

The first token can be used to set explicitly the value of Λ_{QCD} to a given value. In a standard run, there is no need to perform this operation (actually it may produce wrong results), because, by default, the value read from the PDF table is properly used. The other parameters are cutoff used in the programs: the first three are the cutoff for generating emission off light, c and b quarks respectively. They are also used to set heavy flavor thresholds in the strong coupling running. Instead, the last two parameters control the threshold values at which heavy flavor PDF's start to be nonzero.

```
withdamp 1   !  (default 0, do not use) use Born-zero damping factor
hfact 100     !  (default no dumping factor) dump factor for high-pt radiation:
              !  > 0 dampfac=h**2/(pt2+h**2)
```

These tokens control the separation of the full real matrix element in a singular and a

nonsingular part. Their use may be needed in presence of processes where the Born cross section vanishes in some phase-space region. They are also useful to test the behavior of the program in presence of large K-factors. Their exact meaning is explained in ref. [3].
For testing the correct behavior of the program and to obtain NLO distributions, we added other parameters that may also be useful for developers. The normal user is asked not to change them, since their invocation is time consuming and/or may cause some conflicts with other settings. If instead the user is interested in changing them, a detailed explanation of their behavior can be found on ref. [3].

```
testsuda 0       !  (default 0, do not test) tests the Sudakov FF by
                 !  numerical integration
testplots 0      !  (default 0, do not) do NLO and PWHG distributions
bornonly 0       !  (default 0) if 1 do Born only
smartsig 0       !  (default 1) remember equal amplitudes (0 do not remember)
withsubtr 0      !  (default 1) subtract real counterterms (0 do not subtract)
radregion 1      !  (default all regions) only generate radiation in the
                 !  selected singular region
iupperisr 1      !  (default 1) choice of ISR upper bounding functional form
iupperfsr 2      !  (default 2) choice of FSR upper bounding functional form
flg_debug 1      !  (default 0) write extra information on LHEF
```

Other technical parameters have been introduced for more specific debugging purposes: for example `par_diexp, par_2gsupp,jacsing` were relevant for dijets. As already stated, we refer to the process manual for details on other specific parameters.

5. Special modes of operation

In this subsection, we describe some special features that were added to our package to comply with experimental needs or to deal with complicated processes, namely $V + j$ and dijets.

5.1 Weighted-event generation

In normal conditions, the events generated by `POWHEG` are unweighted, i.e. all come with the same weight. At times, however, this feature has some drawbacks. In this subsection we describe how the generation of weighted events can be performed and we specify when this is particularly needed.

Two flags control the nature of the output in the `POWHEG BOX`: `withnegweights` and `bornsuppfact`.[5] If neither of these flags is set, events are output with weight 1, i.e. the `XWGTUP` variable is set to 1. The `IDWTUP` variable in the Les Houches interface is set to 3 in this case. The total cross section is stored in the `xsecup` variable. Negative weighted events are neglected with this choice.

[5]In old versions, the flag `bornsuppfact` was named `ptsupp`, which is now deprecated.

If `withnegweights` is set to 1 (true in our convention), negative weighted events are not discarded. The `IDWTUP` variable in the Les Houches interface is set to -4, and the weight of the event is set to its sign times the sum of the total cross section for positive weighted events plus the absolute value of the cross section for negative weighted events. In this way, the average value of `XWGTUP` equals the real cross section, as required by the Les Houches convention when the `IDWTUP` variable is set to -4. The variable `xsecup` always stores the real cross section.

If the `bornsuppfact` token is set, a suppression factor that depends upon the underlying Born configuration of each event is supplied with it. The cross section computed by the `pwhg_main` program is in this case not valid. It is the integral of the cross section times the suppression factor. Events are generated using this "fake" cross section, and thus are weighted with the inverse of the suppression factor. The `IDWTUP` variable in the Les Houches interface is set to -4. The weight of the event is in this case the sign, times the total cross section for positive weighted events plus the absolute value of the cross section for negative weighted events, times the inverse of the weighting factor. The weight factor is returned by the user routine `born_suppression`, that can use the value of the `bornsuppfact` token as a parameter to compute the suppression factor. Also in this case, the average value of the weight of the event is equal to the real cross section. This option can be active in conjunction with the `withnegweights` flag.

These flags have many uses. On one side, one might like to know where negative weighted events end up. Even if they constitute a small fraction, we may worry that they could end up in some tiny tail of some important distribution. One may also prefer to work with negative weight in cases when getting rid of them requires high folding numbers (the `foldcsi`, `foldy` and `foldphi` tokens), and thus has a high cost in computer time. The `bornsuppfact` feature can be use to enhance a region of phase space (like a high k_T tail) where it would be otherwise difficult to get high statistics. These features, however, become really useful for processes where the Born contribution itself is singular. The simplest examples are the $Z + \text{jet}$ and the dijet production processes. Here we discuss $Z + \text{jet}$. The dijet case is fully analogous.

5.1.1 Generation cut and Born suppression factor

The $Z + 1j$ process differs substantially from all processes previously implemented in `POWHEG`, in the fact that the Born diagram itself is collinear and infrared divergent. In all previous implementations, the Born diagram was finite, and it was thus possible to generate an unweighted set of underlying Born configurations covering the whole phase space. In the present case, this is not possible, since they would all populate the very low transverse momentum region. Of course, this problem is also present in standard Shower Monte Carlo programs, where it is dealt with by generating the Born configuration with a cut k_{gen} on the transverse momentum of the Z boson. After the shower, one must discard all events that fail some transverse momentum analysis cut k_{an} in order to get a realistic sample. The analysis cut k_{an} may be applied to the transverse momentum of the Z, or to the hardest jet. We assume here, for sake of discussion that the analysis cut is applied to the Z transverse momentum.

Taking $k_{\rm an} \gtrsim k_{\rm gen}$ is not enough to get a realistic sample. In fact, in an event generated at the Born level with a given $k_T < k_{\rm gen}$, the shower may increase the transverse momentum of the jet so that $k_T > k_{\rm an}$. Thus, the generation cut, even if it is below the analysis cut, may reduce the number of events that pass the analysis cut. Of course, as we lower $k_{\rm gen}$ keeping $k_{\rm an}$ fixed, we will reach a point when very few events below $k_{\rm gen}$ will pass the analysis cut $k_{\rm an}$. In fact, generation of radiation with transverse momentum larger than $k_{\rm gen}$ is strongly suppressed in `POWHEG`, and, in turn, radiation from subsequent shower is required to be not harder than the hardest radiation of `POWHEG`. Thus, given the fact that we want to generate a sample with a given $k_{\rm an}$ cut, we should choose $k_{\rm gen}$ small enough, so that the final sample remains substantially the same if $k_{\rm gen}$ is lowered even further.

A second option for the implementation of processes with a divergent Born contribution is also available. It requires that we generate weighed events, rather than unweighted ones. This is done by using a suppressed cross section for the generation of the underlying Born configurations:

$$\bar{B}_{\rm supp} = \bar{B} \times F(k_T), \tag{5.1}$$

where $\bar{B}$ is the inclusive NLO cross section at fixed underlying Born variables, and k_T is the transverse momentum of the vector boson in the underlying Born configuration. In this way $\bar{B}_{\rm supp}$ is integrable, and one can use it to generate underlying Born configurations according to its value. The generated event, however, should be given a weight $1/F(k_T)$ rather than a constant one, in order to compensate for the initial $F(k_T)$ suppression factor. With this method, events do not concentrate in the low k_T region. However, their weight in the low k_T region becomes divergent. After shower, if one imposes the analysis cut, one gets a finite cross section, since it is unlikely that events with small transverse momentum at the Born level may pass the analysis cut after shower. In fact, shower transverse momenta larger than the one present in the initial Born process must be suppressed in the Monte Carlo generator.

In recent `POWHEG BOX` revisions, both methods can be implemented at the same time. We wanted in fact to be able to implement the following three options:

- Generate events using a transverse momentum generation cut.
- Generate events using a Born suppression factor, and a small transverse momentum cut, just enough to avoid unphysical values of the strong coupling constant and of the factorization scale that appears in the parton density functions.
- Apply a Born suppression factor, and set the transverse momentum cut to zero. In this case the program cannot be used to generate events. It can be used, however, to produce NLO fixed order distributions, provided the renormalization and factorization scales are set in such a way that they remain large enough even at small k_T^Z. This feature is only used for the generation of fixed order distributions.

The generation cut is activated by setting the token `bornktmin` to the desired value in the `powheg.input` file. The Born suppression is activated by setting the token `bornsuppfact` to a positive real value. The process-specific subroutine `born_suppression` sets the suppression factor to $k_T^2/(k_T^2 + \text{bornsuppfact}^2)$. If `bornsuppfact` is negative, the suppression factor is set to 1.

In the `POWHEG` approach, negative weighted events can only arise if one is approaching a region where the NLO computation is no longer feasible. In our studies for the $Z + 1j$ process we approach this region at small transverse momentum. In order to better see what happens there, rather than neglecting negative weights (that is the default behavior of the `POWHEG BOX`), we have introduced a new feature in the program, that allows one to track also the negative weighted events. This feature is activated by setting the token `withnegweights` to 1 (true). If `withnegweights` is set to 1, events with negative weight can thus appear in the Les Houches event file. While we normally set the `IDWTUP` flag in the Les Houches interface to 3, in this case we set it to -4. With this flag, the SMC is supposed to simply process the event, without taking any other action. Furthermore, the `XWGTUP` (Les Houches) common block variable is set by the `POWHEG BOX` to the sign of the event times the integral of the absolute value of the cross section, in such a way that its average equals the true total cross section.

Notice that, if `withnegweights` is set and a Born suppression factor is also present, the events will have variable `XWGTUP` of either signs. In this case `XWGTUP` is set to the sign of the event, times the absolute value of the cross section, divided by the suppression factor `bornsuppfact`. Also in this case the average value of `XWGTUP` coincides withe the true total cross section. We preferred not to use the option -3 in case of signed events with constant absolute value. This option is advocated by the Les Houches interface precisely in such cases. However, the Les Houches interface does not provide a standard way to store the integral of the absolute value of the cross section, that would be needed to compute correctly the weight of the event. In fact, the `XSECUP` variable is reserved for the true total cross section. More specifically, if we have N events of either sign, they should be weighted with the sum of the positive plus the absolute value of the negative part of the cross section, in such a way that

$$\sum_{i=1}^{N} W_i \left(\sigma_{(+)} + |\sigma_{(-)}|\right) = N \left(\sigma_{(+)} - |\sigma_{(-)}|\right) = \quad N\sigma_{\text{NLO}}, \tag{5.2}$$

(where W_i are the sign of the event ± 1), because

$$\frac{\sum_i W_i}{N} = \frac{\left(\sigma_{(+)} - |\sigma_{(-)}|\right)}{\left(\sigma_{(+)} + |\sigma_{(-)}|\right)} \quad . \tag{5.3}$$

Weighted events are also useful if one wants to generate a homogeneous sample from relatively low up to very high transverse momenta. It is convenient in this case to pick a very large `bornsuppfact` value, of the order of the maximum transverse momentum one is interested in. The large momentum region will be more populated in this way.

5.2 The `pdfreweight` flag

Nowadays it is common practice to estimate uncertainties due to PDF's by reweighting each event with a weight equal to the ratio between the new and the reference PDF's values.

This is done also when using SMC programs. In particular, the evaluation of PDF's ratio is performed by saving for each event the momentum fractions of the incoming partons which are involved in the hard collision and the corresponding values of the reference PDF's. Strictly speaking, this procedure is not valid, since the result of a SMC does not depend linearly on PDF's, because these enter also in the Sudakov form factors. Nevertheless, it is common belief that this procedure should capture the dominant part of uncertainties due to PDF's also in SMC simulations.

In this respect, the output of `POWHEG` is affected by the same problems of SMC's, the dependence on PDF's being non-linear. Moreover, in `POWHEG`, the real contributions are evaluated with different structure functions and different Feynman' x's, so that another reason to doubt about this procedure is present.

Despite all the aforementioned caveat, we decided to make the information needed for the reweighting procedure available also in `POWHEG`, to give the opportunity to perform such studies.

In particular, when the token `pdfreweight` is set to 1, at the end of each event in the LHEF a line is added. The format is the following:

```
#pdf id1 id2 x1 x2 xmufact xf1 xf2
```

Apart from the `#pdf` tag, the other numbers refer to the two id's of the incoming partons that enter the Born process (PDG conventions), to the two momentum fractions of these partons, to the value of the factorization scale at which PDF's were evaluated and, finally, to the value of the PDF's times the momentum fractions. With this information, it should be straightforward to perform the reweighting procedure.

Before concluding this subsection, we would like to stress again that we cannot guarantee that this reweighting procedure gives the same results one would obtain by performing a completely new run, using different PDF's from the very beginning.

5.3 The `manyseeds` flag

The run time needed to produce a large sample of events may become significant, especially for complicated processes. To circumvent this problem, we introduced a feature that can be used for running the `POWHEG BOX` on several nodes of a cluster and optimize the efficiency.

The relevant flag is `manyseeds`. When it is set to 1, the `POWHEG BOX` program looks for a file named `pwgseeds.dat` and stops if this file is not found. The typical sequence of operations to perform parallel runs is then the following one.

1. Prepare a `powheg.input` file with the `manyseeds` flag is set to 1. Set the number of events `nev` to 0.

2. Prepare a file `pwgseeds.dat`, containing a sequence (one per line) of different random number seeds. For example: first line 1, second line 2, etc. (but any number will do).

3. Run the `pwhg_main` program. It will ask an integer for input. Input an integer. That integer is the line number of the random seed to be used for the current run. Assuming

that the number 17 is given as input to the pwhg_main program, the run will produce files named pwgxgrid.dat and pwggrid-0017.dat. The pwhg_main program can be run with different integers as input. Each run can be sent, for example, to a different node of a cluster.

At the end of this step, a bunch of pwggrid-[????].dat, pwgubound-[????].dat files and pwgNLO-[????].top files will be present in the run directory. The pwgNLO-[????].top files are statistically independent topdrawer histograms. They can be combined to provide a higher statistics NLO analysis of the current analysis routines.

If the subsequent runs are sent after the file pwgxgrid.dat was already produced, and if the flag use-old-grid is set to one, the importance sampling grid will be loaded from the pwgxgrid.dat. Otherwise it will be recreated.

4. Now set the nev token to a given number, make sure that the flag use-old-grid is set to one, and run a bunch of copies of the pwhg_main program, each with the same integers as input. The program will now load all the pwggrid-[????].dat and pwgubound-[????].dat that it can find, and combines them adequately, assuming that they are all statistically independent, and will start to generate events. The events will be in files pwgevent-[????].lhe, and they will all be independent statistically.

The sequence above is a quite simple two step procedure. It is useful, however, to better clarify the logic that the POWHEG BOX follows in this procedure.

There are 3 steps in the initialization phase of POWHEG. First of all, an importance sampling grid is determined. Let us call this stage ISG (Importance Sampling Grid). The second step is the calculation of the integrals, and the determination of an upper bounding envelope for the $\tilde{B}$ function, to be used for the generation of underlying Born configurations. We call this stage the UBB (Upper Bounds for underlying Born). As a third step, the upper bound normalization for radiation is determined. We call this stage UBR (Upper Bounds for Radiation).

First of all, we remark that, if the use-old-grid flag is not set to 1, no grid file is loaded. Similarly, if the use-old-ubound flag is not set to 1, no ubound file is loaded. In other words, if these flags are set, looking for a corresponding file will always yield a negative result. The reader should keep this in mind when reading the following procedure. If the flag manyseeds is set, the pwhg_main program asks for an integer. We will call cj this integer, between 0 and 9999. We denote with [cj] the corresponding string of four digits (leading digits are set to 0; thus if ic=1 [cj]=0001. We will denote as [????] any four digit string. The logic of grid loading in POWHEG BOX is as follows:

1. If a file pwggrid.dat exist and is consistent, this file is loaded, and steps ISG and UBB are skipped (go to 6).

2. If the above fails, if the manyseeds flag is not set, or if it is set and a file named pwggrid-[cj].dat already exists and is consistent, all files of the form pwggrid-[????].dat are loaded and suitably combined, and the steps ISG and UBB are skipped (go to 6).

3. If the above fails, if a file `pwgxgrid.dat` exist and is consistent, this file is loaded, and steps ISG is skipped (go to 5).

4. Step ISG is performed. The resulting grid is stored in the file `pwgxgrid.dat`. This step, whether the `manyseeds` flag is set or not, is performed using the default initial seed value (i.e. not the seed found at the `cj` line of the `pwgseeds.dat` file). In this way, all copies of the program being run will use the same importance sampling greed. This is mandatory if we want to combine results.

5. Step UBB is performed. If the `manyseeds` flag is set, this step is performed using the seed found at the `cj` line of the `pwgseeds.dat` file, and the result is stored in the file `pwggrid-[cj].dat`. Otherwise, the current seed value is used, and the result is stored in a file named `pwggrid.dat`.

6. If a file named `pwgubound.dat` exists and is consistent it is loaded. The UBR step is skipped (goto 10).

7. If a file named `pwgubound.dat` exists and is consistent it is loaded. The UBR step is skipped (goto 10).

8. If the above fails, if the `manyseeds` flag is not set, or if it is set and a file named `pwgubound-[cJ].dat` exists and is consistent, all files with names of the form `pwgubound-[????].dat` are loaded and combined. The UBR step is skipped (go to 10).

9. The UBR step is performed. If the `manyseeds` flag is not set, the result is stored in a file named `pwgubound.dat`. Otherwise, it is stored in the file `pwgubound-[cj].dat`.

10. Now `nev` events are generated. If the `manyseeds` flag is not set, the result is stored in a file named `pwgevents.lhe`. Otherwise, it is stored in the file `pwgevents-[cj].dat`.

This logic has the purpose to allow several possible combinations of actions. For example, one can use grids generated in parallel runs to produce events without using the `manyseeds` flag. Or one can use grids generated without the `manyseeds` flag for generating events in parallel with the `manyseeds` flag set.

6. Counters and statistics

Several results relevant to the interpretation of the output of the run are written into the files `pwgstat.dat` and `pwgcounters.dat`. The fraction of negative weights, the total cross section, the number of upper bound failures in the generation of the inclusive cross section, and the generation efficiency, together with failures and efficiency in the generation of hard radiation, are printed there. These are quite self-explanatory and we do not comment them any further. These numbers are sufficient to take action in case of problems, as explained in sec. 4.

7. Random number generator

`POWHEG BOX` uses the `RM48` random number generator, documented in the CERNLIB write-ups. This generator has default initialization. If a user wishes to start the program with different seeds, he/she should add lines similar to

```
!  Random number generator initializing parameters
iseed 6093726 !  initialize random number sequence
rand1 -1      !  initialize random number sequence
rand2 -1      !  initialize random number sequence
```

to the input card. This results in a call to the `rm48in(iseed,rand1,rand2)` subroutine that seeds the generator with the integer `iseed`, and skip the first `rand1+rand2*10**8` numbers, as documented in the CERNLIB manual. This can be useful if one wants to resume a previous run. In that case, one has simply to use as initializing values those reported in the `<prefix>-events.lhe` file. If instead one just wants to change the seed only, he/she can comment or skip the `rand1` and `rand2` lines in the input card.

We remind the reader that a change in the random number generator initialization affects the `POWHEG BOX` random number sequence, both in the generation of events and in NLO computation or upper bound searching, when the corresponding grids are not present. If the program is interfaced to a SMC, the user should also take care to initialize the seeds of the latter.

References

[1] P. Nason, "A new method for combining NLO QCD with shower Monte Carlo algorithms," JHEP **0411** (2004) 040 [arXiv:hep-ph/0409146].

[2] S. Frixione, P. Nason and C. Oleari, "Matching NLO QCD computations with Parton Shower simulations: the POWHEG method," JHEP **0711** (2007) 070 [arXiv:0709.2092 [hep-ph]].

[3] S. Alioli, P. Nason, C. Oleari and E. Re, "A general framework for implementing NLO calculations in shower Monte Carlo programs: the POWHEG BOX," JHEP **1006**, 043 (2010) [arXiv:1002.2581 [hep-ph]].

[4] M. R. Whalley, D. Bourilkov and R. C. Group, "The Les Houches accord PDFs (LHAPDF) and LHAGLUE," [arXiv:hep-ph/0508110].

[5] M. Cacciari and G. P. Salam, "Dispelling the N^3 myth for the k_t jet-finder," Phys. Lett. B **641**, 57 (2006) [arXiv:hep-ph/0512210].

[6] E. Boos *et al.*, "Generic user process interface for event generators," [arXiv:hep-ph/0109068].

[7] J. Alwall *et al.*, "A standard format for Les Houches event files," Comput. Phys. Commun. **176** (2007) 300 [arXiv:hep-ph/0609017].

[8] T. Sjöstrand et al., in "Z physics at LEP1: Event generators and software,", eds. G. Altarelli, R. Kleiss and C. Verzegnassi, Vol 3, pg. 327.

[9] P. Nason, "MINT: a Computer Program for Adaptive Monte Carlo Integration and Generation of Unweighted Distributions," arXiv:0709.2085 [hep-ph].