The POWHEG-BOX-W2jet and Z2jet manual

J.M. Campbell, R.K. Ellis, P. Nason and G. Zanderighi

1 Introduction

The `POWHEG-BOX-W2jet` and `Z2jet` programs [1] are NLO generators for the QCD production of $W^{\pm}$jj and Zjj events in hadronic collisions, with the vector boson decaying into leptons. They can be properly interfaced to Shower Monte Carlo programs. Off-shell effects are included in the calculation. The vector bosons are given a finite, fixed width. The CKM matrix is assumed diagonal.

This document describes the input parameters that are specific to this implementation, and also certain new features of the `POWHEG-BOX` that they use. The parameters that are common to all `POWHEG BOX` implementations are given in the `manual-BOX.pdf` document, in the `POWHEG-BOX/Docs` directory.

2 Running the program

Enter the process directory and do
`$ make pwhg_main`
to build the executable for the generation of the events. In the following, the input parameters to be entered in the `powheg.input` file are described.

3 Process specific input parameters

Parameters in `powheg.input` that are specific to Wjj production:

```
vdecaymodeW -11    !  decay mode of W+/W- (11=e-,13=mu-,15=tau-,
                   !                      -11=e+, -13=mu+, -15=tau+.)
mllmin     0       !  Minimum invariant mass for the lepton pair (default 0)
bwcutoff   15      !  How many W widths above and below its pole mass do we
                   !  allow for the virtuality of the W (default: absent)
```

Parameters in `powheg.input` that are specific to Zjj production:

```
vdecaymodeZ 11     !  Z decay mode: 11=e,12=nue, etc.,
mllmin     1       !  Min invariant mass for the lepton pair (default 1 GeV)
mllmax     140     !  Max invariant mass for the lepton pair (default sqrt(S))
bwcutoff   15      !  How many Z widths above and below its pole mass do we
                   !  allow for the virtuality of the Z (default: absent)
```

We remark that the restrictions on the invariant mass of the lepton pairs are all applied simultaneously.

The following parameters are the same for Wjj and Zjj:

```
runningscales 0    !  if /=1 or absent use MW, if 1 uses HT/2
bornsuppfact  20   !  Set the pt scale parameter for Born-zero damping factor
                   !  equal to 20 GeV.  If absent, no damping factor
ptborncut  0.01    !  pt of jets and their relative pt limited by ptborncut
                   !  default: 0.01 GeV
```

4 Option that activate the new POWHEG BOX features

The W2jet and Z2jet packages use a number of new features that will eventually become the default in an upcoming version 2 of the POWHEG BOX. Several new files, stored in the directory POWHEG-BOX/Version-pre2-1, replace the files in the POWHEG-BOX directory. The Makefile is structured in such a way that the new files have priority over the old ones. Here we describe these new features.

- MiNLO: the procedure of ref. [2]. Using MiNLO also inclusive quantities not requiring that the jets are resolved are computed with a certain accuracy.
  ```
  minlo    1      !  Activate the minlo feature (default absent)
  ```

- ```
  doublefsr 1     !  Activate doubling of singular region in final state
                  !  quark gluon splitting.  Default on for this process.
  ```
  This feature is described in ref. [3].

- ```
  olddij    1     !  Use old values for the d_ij functions (default 0)
  ```
  The new separation or regions described in ref. [1] is used by default. To fall back on the traditional approach, set this flag to 1.

- ```
  fastbtlbound    1     !  Use improved upper bounding envelope (default 0)
  ```
  This uses the method described in ref. [4].

- Reweighting: it is possible to run a process saving reweighting information in the event file, and then use this information to compute new weights corresponding, for example, to a different choice of the factorization and renormalisation scales. In order for reweighting to work one should always turn on the option
  `withnegweights 1`
  (on by default in Z2jet). In order to store reweighting information, one should include the line:
  `storeinfo_rwgt  1 !  Store reweighting information (default 1)`
  If this feature is present during the event generation, a line of reweighting information will appear at the end of each event in the event file, of the form
  `#rwgt 1 59 759.386766596014 100 1001 0`
  At the end of the run, one can change a few parameters in the powheg.input file, or even alter the POWHEG executable, in a way that affects the value of the cross section (typically one can change scale factors, PDF's, etc.), include the line
  `compute_rwgt 1    !  compute new weight`
  and run the program again in exactly the same directory that was used to produce the event files. Similar event files, with the -rwgt string in the name, will be produced, where at the end of each event a new weight is appended, of the form
  `#new weight,renfact,facfact,pdf1,pdf2 19335.9 1.0 1.0 10050 10050 lha`
  where the new weight, the new values for the renormalization and factorization scale factors, the pdf number for each hadron, and the pdf package that was used are reported. The reweighting run is much faster than the generation run. Thus, iterating this method, it is easy to append a bunch of lines with new weights to the end of each event, in order to perform uncertainty studies. Notice that if the program is run with no changed parameters, the new weight should equal the original weight (the third entry of the first line in the Les Houches event record) up to truncation errors.

Besides the features listed above, the method used to perform parallel runs has been completely overhauled and improved. In the following subsection we describe how it works.

4.1 Parallel runs

It is possible to split a POWHEG run into several parallel ones. However, since POWHEG runs in several stages, each stage has to be run until the end before the next stage is started, in order for the parallelization to be useful. We need the following flags:
  ```

```
manyseeds 1 !  get the seeds for the random number generator from the file
           !  pwgseeds.dat.
parallelstage  <m>  !  <m>=1...4, which level of parallel stage
xgriditeration <n>  !  <n> is the iteration level for the calculation of the
                    !  importance sampling grid improvement (relevant only for
                    !  parallelstage=1)
```

The `pwgseeds.dat` file contains a list of integer seeds, one per line. If the `manyseeds` flag is set, when the `pwhg_main` program starts, it asks for an integer number m. This number is used to select a line in the `pwgseeds.dat` file, and the integer found there is used to initialize the random number generator. In this way, several runs can be performed simultaneously in the same directory, all using different random number seeds. Furthermore, the integer m appears as a four digit integer (with leading zeros) in the name of all files that are generated by the corresponding run.

The `pwhg_main` program runs in four stages:

1. An importance sampling grid for the integration of the inclusive cross section is built and stored. This stage is run when `parallelstage=1`. The number of calls to the inclusive cross section is controlled by the input variable `ncall1`. If also a remnant cross section is present, the number of calls for the remnants is equal to `ncall1`, unless a parameter `ncall1rm` appears. Initially, this stage is called with `xgriditeration=1`. The information that is needed to compute the importance sampling grids are stored in the files named (assuming $m = 1$) `pwggridinfo-btl-xg1-0001.dat` for the $\tilde{B}$ function and `pwggridinfo-rmn-xg1-0001.dat` for the remnant. The quality of the grid for each single run is represented in the topdrawer files `pwg-xg1-0001-btlgrid.top`. These plots are not very significant, since what counts is the grid formed by assembling all the information contained in all the `pwggridinfo` files. These are generated in the subsequent step, are named `pwg-0001-btlgrid.top`, and are all equal among each other for all values of m that have been used.

   Observe that it is important that all `parallelstage=1` runs complete, in order to use all the produced grid information for building the grid.

   If after the `parallelstage=1` and `xgriditeration=1` run the grids do not look smooth enough, one or more iterations with `xgriditeration=2` and higher, can be attempted. At this stage, the relevant files are named `pwggridinfo-btl-xg2-0001.dat`, `pwggridinfo-rmn-xg2-0001.dat` and `pwg-xg2-0001-btlgrid.top`, and so on. Increasing `ncall1` also helps in getting more satisfactory grids.

2. This stage is performed after the `parallelstage=1` run, by setting `parallelstage=2`. The integral of the inclusive cross section is computed, and an upper bounding of the integrand, having the form of a multidimensional step function on the grid, is computed and stored. The number of calls used for this task is controlled by `ncall2` and `itmx2` (the total number of calls per run is `ncall2*itmx2`). Again, independent variables for the remnants are also available if needed, `ncall2rm` and `itmx2rm`. If the files with information on the importance sampling grid are missing, the program complains and stops. The integration and upper bounding envelope information is stored in files named `pwggrid-0001.dat`. If the `storemintupb` flag is set, auxiliary files named `pwgbtildeupb-0001.dat` and `pwgremnupb-0001.dat` are created and loaded with these information. Again, all processes run at this stage should be completed before going on. Only at the next stage these files are all loaded and assembled to get the cross section and upper bounding envelopes using the full statistics of the runs. Files named `pwgfullgrid-0001.dat` are then created. They are all equal among each other. On subsequent runs, if any `pwgfullgrid-0001.dat` is present, it is loaded in place of all the others, since it contains all the necessary information.

3. This is obtained by setting `parallelstage=3`. The upper bounding factors for the generation of radiation are computed at this stage. The number of calls (per run) is controlled by the variable `nubound`. The computed factors are stored in the files named `pwgubound-0001.dat`.

4. With `parallelstage=4`, the program starts generating events, that are stored in files named `pwgevents-0001.lhe`. The number of events per file is controlled by the `numevts` parameter in the `powheg.input` file.

When performing parallel runs, for very slow processes that require very many cpu's, the number of events per process may not be enough for building a valid upper bounding grid (see ref. [4]). In this case, it is convenient to set the flag `storemintupb=1`. In this way, the results of the calls to the inclusive cross section are all stored in files, and the upper bounding envelope is built at a later parallel stage, by reading and assembling all these files.

5 Files to inspect at the end of the run

The file `pwgstat.dat` (for single run), or `pwg-stY-0001-stat.dat`, with the stage number `Y` equal to 2 or 3, contain information on the value of the cross section. Notice that, in case of parallel runs, the stage 3 files are the final ones, since they refer to the result of the combination of all the stage 2 runs, while the stage 2 files only include the result of a single run. It is important to check that the cross section has the required precision.

At the end of the runs, the values of certain counters are printed in files named `pwgcounters.dat`, or `pwgcounters-stY-XXXX.dat`, where `Y` is the `parallelstage` value and `XXXX` stands for the four digits run number in case of parallel runs. It is important to check in the stage 4 files (`pwgcounters-st4-XXXX.dat`) that the number of upper bound violation in the generation of radiation and in the inclusive cross section are much smaller than the total number of events. They appears as follows:

```
upper bound failures in generation of radiation = 636.000000000000
upper bound failure in inclusive cross section = 340.0000000000000
```

These numbers should be of the order of a percent of the total number of events. If the number of failure for the inclusive cross section is too large, the number of calls `ncalls2` should be increased. If the failure in the generation of radiation is too large, then `nubound` should be increased. In this last case, one can also attempt to increase `xupbound`, with no need to rerun the stage of generation of the upper bounds for radiation (that is to say, stage 3).

The counter files also report the approximate value of the total number of seconds spent doing the virtual and real calculations. These numbers are significant only if the program is run at 100% speed. If the virtual time is large compared to the real time, it may be convenient to increase the folding parameters `foldcsi`, `foldy` and `foldphi`, that also reduces the fraction of negative weighted events (for the allowed values for these parameter see the `POWHEG BOX` manual).

6 Shower and analysis

The user can shower the Les Houches event with whatever software he likes. We provide elementary interfaces to `PYTHIA` and `Pythia8`. An interface to `HERWIG` can be easily developed on the basis of the examples in other `POWHEG BOX` processes. An enhancement of the previous histogramming package in the `POWHEG BOX` does now output the histograms associated with different weights, if present in the Les Houches event file. Assuming that all event files are merged into a single `pwgevents.lhe` files, in order to run the builtin analysis one proceeds as follows:

```
$ cd POWHEG-BOX/W2jet
$ make main-PYTHIA-lhef
$ cd test-lhc
$ ../main-PYTHIA-lhef
```

where `test-lhc` is the directory where the events are stored. The output files are
`pwgPOWHEG+PYTHIA-output.top.`
If more weights are present in the event file, files of the form
`pwgPOWHEG+PYTHIA-output-WY.top`
are produced, where `Y` is the weight ordering number in the event file. These files are gnuplot data files, and can be plotted with the gnuplot program.

The analysis can also be run in parallel. If the program does not find a `pwgevents.lhe` file, it prompts for a filename. If the file name has the form `pwgevents-XXXX.lhe`, then the output files have the name
`pwgPOWHEG+PYTHIA-output-XXXX-WY.top`
This is useful if one wants to speed up the analysis of parallel runs. It is up to the user, at the end, to combine the files. A fortran program `mergedata.f` is provided for this purpose in the `Version-pre2-1` directory. Its use is self explanatory.

7 Examples

In the directories `runs-paper` we have put the setup to perform parallel runs of the kind that we used for our paper [1]. This setup assumes that the run is to be performed interactively on a multi-core machine with 48 cores. The script `runpar.sh` executes all stages of the run. It is up to the user to adapt the run to his/her batch environment. The example of the Wjj takes about 2 hours for the preparation stage, and 10 hours for the generation of events, on a 48 cores machine. For the Zjj case, the preparation stage takes roughly eight hours, while the events are generated in about 75 hours.

Bibliography

[1] J. M. Campbell, R. K. Ellis, P. Nason, and G. Zanderighi, *W and Z bosons in association with two jets using the POWHEG method*, 1303.5447.

[2] K. Hamilton, P. Nason, and G. Zanderighi, *MINLO: Multi-Scale Improved NLO*, *JHEP* **1210** (2012) 155, [1206.3572].

[3] P. Nason and C. Oleari, *Generation cuts and Born suppression in POWHEG*, 1303.3922.

[4] T. Melia, P. Nason, R. Rontsch, and G. Zanderighi, W^+W^+ *plus dijet production in the POWHEGBOX*, *Eur.Phys.J.* **C71** (2011) 1670, [1102.4846].