Preprint typeset in JHEP style - PAPER VERSION

March 23, 2013
POWHEG BOX 1.0

The POWHEG BOX user manual: $Z + 2$ jets production via strong interactions

Emanuele Re

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

Abstract: This note documents the use of the package POWHEG BOX for the simulation of the $Z + 2$ jets production process. Results can be easily interfaced to shower Monte Carlo programs, in such a way that both NLO and shower accuracy are maintained. Please read carefully the manual before using this code.

Keywords: POWHEG, Shower Monte Carlo, NLO.

Contents

1. Introduction

The POWHEG BOX program is a framework for implementing NLO calculations in Shower Monte Carlo programs according to the POWHEG method. An explanation of the method and a 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 at the web page

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

In this manual, we will focus on the POWHEG implementation of the NLO corrections to $Z+$ 2 jets hadroproduction, where the $Z \to \ell^+\ell^-$ decay is taken into account fully. Source files can be found in the POWHEG-BOX/Zjj subdirectory, and some details of the implementation are described in ref. [4], where a comparison with LHC data was also carried out.

2. Installation

In order to run the POWHEG BOX program, we recommend the reader to start from the POWHEG BOX user manual, which contains all the information and settings that are common between all subprocesses. In this note we focus on the settings and parameters specific for the process at hand.

The program needs to be linked with an external code able to provide the one loop corrections, first computed in ref. [5]. There are currently several tools available to achieve this task, such as BlackHat, GoSam and MadLoop. The needed loop amplitudes are also contained in the MCFM code, against which the POWHEG implementation has been checked.

In the following we describe how to link BlackHat (BH) [6] to the POWHEG BOX program for the $Z + 2$ jets production process. Notice, however, that this is achieved using the Binoth-LH Accord (BLHA) [7] that one-loop programs (OLP) generically implement (or will implement): therefore, linking to another code other than BH should be relatively straightforward.

2.1 Installation of BlackHat

The BlackHat library can be downloaded from

https://blackhat.hepforge.org/trac/wiki

It has to be installed following the instructions available at the above web address. Notice in particular that the qd package has to be installed as well.

In order to link BH with POWHEG, BH has to be installed with the BLHA interface enabled: hence the configuration command `--enable-LHpythoninterface` has to be used at the configuration stage. After BH has been installed, it is useful to explicitly check that this interface has been created: an executable named LH_reader should be present in the bin directory of the BH installation folder. Standard checks to test the correct installation of BH can be also performed, as detailed in the BH manual.

As usual, at this point the user should also remember to (re)define the environmental variable LD_LIBRARY_PATH in order to include the absolute path of the BH lib directory. It could be useful to also update the PATH variable to include the bin directory where the executables blackhat-config and LH_reader are located.

2.2 Compiling POWHEG and linking with BlackHat

Before compiling the POWHEG code in the Zjj subdirectory, the user should edit the Makefile as follows:

1. Set the variables BHPATH and QDPATH to the absolute paths of the directories where qd and BH have been installed.
2. Set the variable VIRTUAL equal to BH.
3. Set the variable BH equal to true.

If other libraries as LHAPDF [8] or FastJet [9, 10] need to be used, as usual the other corresponding settings in the Makefile (and possibly the LD_LIBRARY_PATH variable) should be edited. Afterwards, it should be possible to compile the POWHEG code with the usual command

```
$ make pwhg_main.
```

The two fortran routines invoking the BH library are OLP_start and OLP_EvalSubprocess, and they are called in `virtual.f`. Their source code is part of the BH library. Therefore, if something went wrong with the BH installation or in the linking procedure between `POWHEG BOX` and BH, likely at this stage some error message regarding these two routines will be issued by the compiler and/or the linker.

3. Generation of events

As explained in the previous section, the executable is built with the command `make pwhg_main`. If the compilation was successful, all should be in place and ready to run the code. However, according to the BLHA interface, before the actual running stage, an initialization phase has to take place between `POWHEG` and the OLP program (BH in our case). This phase involves the creation of an order file by the MC program, by means of which the OLP code is queried about the availability of partonic subprocesses for which the 1-loop amplitudes are needed. This negotiation stage also involves exchange of the details of the physical parameters and the format relevant to return the 1-loop amplitudes. Details of this procedure can be found in ref. [7]. In the following we describe schematically how to go through this necessary stage. We also describe an important sanity check which we strongly recommend to always perform when BH is used.

3.1 MC-OLP Negotiation stage

In practice, when linking `POWHEG BOX` with BH, these are the steps to follow:

1. Enter the run directory

   ```
   $ cd rundir
   ```

   and edit the POWHEG input card (named as usual `<prefix>-powheg.input`) to enable the usage of the BLHA interface, by means of the flag `use-OLP-interface`:

   ```
   use-OLP-interface 1
   ```

2. Run the POWHEG program by doing

   ```
   $ ../pwhg_main
   ```

   and then insert the prefix of the input file when prompted.

3. The code should run for few seconds, and exit with the following message:

   ```
   WARNING: contract.lh not found.
   Run the OLP appropriate command to create it
   (for BH, the command is LH_reader)
   ```

At this stage a file named order.lh should be present in the run directory. Commented lines in this file begin with the # character.

4. Now the actual negotiation phase should take place. This is done by calling the BlackHat LH_reader executable:

   ```
   $ <path_to_BH>/bin/LH_reader order.lh contract.lh
   ```

   If this stage ends successfully, no warnings should be issued, and only a brief BH output message should be printed on the terminal. Crucially, a contract.lh file should have been created starting from the content of order.lh . Its format should read like

   ```
   # order file written by POWHEG-BOX
   ...
   <parameter> <numerical value> | OK
   ...
   # <in1 in2 ==> out3 out4 ...  outM>
   <[PDGcode1] [PDGcode2] -> [PDGcode3] ...  [PDGcodeM]> | 1 n
   ...
   # options
   # ...
   ```

   Lines starting with # in this file are are comments, and n should be a progressive integer.

Once this is done, by calling as usual the command pwhg_main, the standard full POWHEG run will start, according to the settings present in the input card. However, we suggest to first go through a very quick sanity check, as described in the next subsection.

3.2 Sanity Checks

After the contract.lh file has bee created, edit the input card such that the flag check_ref_amp is enabled:

```
check_ref_amp 1
```

Make also sure that the relevant physical parameters for this process are set exactly to the following values:

```
Zmass 91.1876
Zwidth 2.49
sthw2 0.23
alphaem 0.007763854599
```

Then, run pwhg_main. POWHEG BOX will check that the OLP results for the 1-loop amplitudes agree with reference hard-coded values. After few seconds, the program will stop, and an output message will be written on the terminal. Depending on its content, you will know whether the sanity check was succesfull or not.

If all the aforementioned stages were succesfull, you can remove or comment the token check_ref_amp from the input card, and you are ready for a normal run.

3.3 Other useful informations

It is possible to speed up the integration stage and/or the event-generation stage by splitting these steps in several short runs. When this is done, the program will produce, respectively, several grid files (ending *-grid.dat) and/or several event files (ending *.lhe). The full description of this procedure can be found in POWHEG-BOX/Docs/Manyseeds.pdf. In summary, to parallelize these stages, the code has first to be run setting in the input card "itmx2 0" and/or "numevts 0", respectively.

When this run is finished, all the files for subsequent runs will be ready, but no grids (or event files, respectively) will be present yet. At this point, the user should uncomment the option manyseeds in the input card:

```
manyseeds 1
```

and decide the number of integration points (events) for each short run, setting itmx2 and ncall2 (or respectively numevts) as desired. In order to start the next step, a file named <prefix>-seeds.dat has to be created, with the following format:

```
randomseed1
randomseed2
randomseed3
..
..
```

where each line has to be the integer that will be used to initialize the random numbers generator. At this point, by running the code with the usual command pwhg_main, the user will be asked not only the prefix of the input file, but also an integer. This integer corresponds to the line in <prefix>-seeds.dat that the program will read to initialize the random numbers generator. The program will use all the needed files already present (combining them properly). If we are at the integration (event generation) stage, the grids (event files) produced will contain in the name also the integer typed, so there is no risk of overwriting files.

4. Process specific input parameters

- The physical parameters relevant for this process are the Z-boson mass and width, the squared sine of the Weinberg angle ($\sin^2\theta_W$) and α_{em}. Their values are set with the tokens `Zmass, Zwidth, sthw2, alphaem`.

- 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:

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

  The default scale choices available are

$$\mu = m_Z,\, \hat{H}_T/2,\, E_{T,Z}\,,$$

  where $\hat{H}_T = \sqrt{m_Z^2 + p_{T,Z}^2} + p_{T,1} + p_{T,2}$, $p_{T,Z}$ is the transverse momentum of the Z-boson, and all the quantities are computed using the underlying-Born kinematics. They can be chosen by setting the token `scalechoice` equal to 1, 2 or 3 respectively. Other scale choices are possible, and the experienced user can change this setting modifying the `set_fac_ren_scales` routine.

- This implementation needs a suppression factor, to make the integration of the $\bar{B}(\Phi_n)$ function numerically finite. We refer the reader to [4] for more details. In practise, this is achieved multiplying the $\bar{B}(\Phi_n)$ with the suppression factor

$$F(\Phi_n) = \left(\frac{p_{T,1}^2}{p_{T,1}^2 + \Lambda_{p_\mathrm{T}}^2}\right)^{k_\mathrm{IS}} \left(\frac{p_{T,2}^2}{p_{T,2}^2 + \Lambda_{p_\mathrm{T}}^2}\right)^{k_\mathrm{IS}} \left(\frac{s_{1,2}}{s_{1,2} + \Lambda_m^2}\right)^{k_\mathrm{FS}}\,.$$

  $p_{T,1}$ and $p_{T,2}$ are the transverse momenta of the 2 outgoing light partons in the underlying-Born kinematics and $s_{1,2}$ is the invariant mass squared obtained from their momenta.

  The code will not work without this factor, and therefore the flag `bornsuppfact` should always be present and set to 1 in the input card.

  The function $F(\Phi_n)$ depends on the 4 parameters k_IS, k_FS, Λ_{p_T} and Λ_m. Their default values are $k_\mathrm{IS} = k_\mathrm{FS} = 2$, $\Lambda_{p_\mathrm{T}} = 10$ GeV and $\Lambda_m = 5$ GeV, and we suggest to use them, unless corners of the phase-space (far from where the cross-section is large) have to be populated with many events.

  To set these parameters to values different from the default ones, the tokens `supp_pt2j` and `supp_m2inv56` can be used in the input card. The numerical values are interpreted as (GeV)^2 quantities, hence they will change the value of $\Lambda_{p_\mathrm{T}}^2$ and Λ_m^2 respectively.

  As a consequence of the presence of a suppression factor, events will be weighted. We also stress that for this process the fraction of negative-weighted events will not be negligible. Therefore we strongly suggest to run the code with the flag `withnegweights` set to 1.

- When generating events with this implementation, negative weights appear. The fraction of negative weights in `POWHEG` can be reduced increasing `foldcsi`, `foldy`, `foldphi`. 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.

  Since running this implementation is already quite computationally demanding, we suggest to avoid using the folding procedure.

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] E. Re, "NLO corrections merged with parton showers for Z+2 jets production using the POWHEG method," JHEP **1210**, 031 (2012) [arXiv:1204.5433 [hep-ph]].

[5] Z. Bern, L. J. Dixon and D. A. Kosower, Nucl. Phys. B **513**, 3 (1998) [hep-ph/9708239].

[6] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres Cordero, D. Forde, H. Ita, D. A. Kosower and D. Maitre, Phys. Rev. D **78**, 036003 (2008) [arXiv:0803.4180 [hep-ph]].

[7] T. Binoth, F. Boudjema, G. Dissertori, A. Lazopoulos, A. Denner, S. Dittmaier, R. Frederix and N. Greiner *et al.*, Comput. Phys. Commun. **181**, 1612 (2010) [arXiv:1001.1307 [hep-ph]].

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

[9] M. Cacciari and G. P. Salam, Phys. Lett. B **641** (2006) 57 [hep-ph/0512210].

[10] M. Cacciari, G. P. Salam and G. Soyez, Eur. Phys. J. C **72** (2012) 1896 [arXiv:1111.6097 [hep-ph]].