Preprint typeset in JHEP style - PAPER VERSION

Bicocca-FT-07-10
GEF-TH-16/2007
Version 1.01, May 18, 2009

The POWHEG-hvq manual version 1.01

Stefano Frixione
INFN, Sezione di Genova, Italy
E-mail: `Stefano.Frixione@cern.ch`

Paolo Nason
INFN, Sezione di Milano Bicocca, Italy
E-mail: `Paolo.Nason@mib.infn.it`

Giovanni Ridolfi
Dipartimento di Fisica, Università di Genova
and INFN, Sezione di Genova, Italy
E-mail: `Giovanni.Ridolfi@ge.infn.it`

Abstract: This note documents the use of the POWHEG-hvq package, a generator for heavy flavour hadroproduction at next-to-leading order in QCD, that can be easily interfaced to shower Monte Carlo programs, in such a way that NLO and shower accuracy are both maintained.

Contents

1. Introduction

The POWHEG-hvq program is a hard event generator for heavy quark production in hadronic collisions. It is accurate at the next-to-leading order in QCD (NLO from now on), and it can be interfaced to Shower Monte Carlo programs like HERWIG and PYTHIA, in such a way that both the leading logarithmic accuracy of the shower and the NLO accuracy are maintained in the output. It is thus an alternative to the MC@NLO heavy flavour production program of ref. [1]. The code can be found in

`http://moby.mib.infn.it/~nason/POWHEG`.

The program is an implementation of the heavy flavour NLO cross sections of refs. [2, 3, 4], according to the formalism of refs. [5] and [6]. A detailed description of the implementation is given in ref. [7]. In the case of $t\bar{t}$ production, spin correlations in top decays are included with a method similar to the one discussed in ref. [8]. The relevant matrix elements for the spin correlations are the same used in the MC@NLO package [9], and were obtained using MadEvent [10]. Our decay package takes into account the finite width of the top quarks and of the W.

In this note, we give all the necessary information to run the program.

2. Installation

The program comes as a tarred-gzipped file `POWHEG-hvq.tar.gz`, with a set of patches `POWHEG-hvq-patch1`, `POWHEG-hvq-patch2`, etc., containing bug fixes and improvements. The current patch level is 4. It should be unpacked following the instructions in the file `AAAreadme-Patches`, or by simply executing it

```
$ sh AAAreadme-Patches
```

and then compiled

```
$ cd POWHEG-hvq
$ make <target>
```

where the choice of the target depends upon the way one wants to interface the program with a Shower Monte Carlo implementation. The `Makefile` is set up to use the compiler `g77` on Linux platforms. If one wishes to use `gfortran`, one should uncomment the appropriate lines in the `Makefile`. Notice that some versions of `gfortran` may not support the `idate` and `time` intrinsics. These are used in the `mbook.f` file, that in turn is used in the examples. Thus, one may also need to comment out the calls to `idate` and `time` in order to run the examples with `gfortran`.

3. Modes of operation

The program POWHEG-hvq generates hard events. The hard events can then be fed into a Shower Monte Carlo programs (SMC from now on) for subsequent showering. POWHEG-hvq saves the hard event information according to the Les Houches Interface for User Processes (LHIUP from now on) conventions [11]. The SMC should also comply with these conventions (as is the case for PYTHIA and HERWIG) in order to be used in conjunction with POWHEG-hvq.

The program can be run in three ways.

- POWHEG-hvq generates hard events, and store them in a file. A SMC reads the file and showers it.
- POWHEG-hvq is linked directly to the SMC. In this case the events are generated and immediately showered, without intermediate storage.
- POWHEG-hvq is run as a standalone program, and the produced hard events are analyzed without showering. The output yields in this case NLO distribution with LL resummation of soft gluon effects.

3.1 Storing the user events

The easiest way to interface POWHEG-hvq to a SMC is to simply store the hard events in a file, and in a separate run read the events and process them with the SMC. The format of the event file supported by POWHEG-hvq is the "Standard format for Les Houches event files", documented in ref. [12]. The program for the generation of the Les Houches Event Files (LHEF from now on) can be built with the command

```
$ make main-lhef
```

The event file is named `pwglhef.events` (a mechanism is provided to change the prefix of the file name, as documented in the next section). An example program that reads the event file, showers it with HERWIG and analyzes it the can be built as follows

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

A similar program, named `main-PYTHIA-lhef` is provided for `PYTHIA`. The HERWIG or PYTHIA program should be installed in the POWHEG-hvq directory. In the case of HERWIG, the appropriate include files should also be present. As can be evinced from the Makefile, the fortran files relevant for these examples are `main-HERWIG-lhef.f`, `herwig6510.f` (`main-PYTHIA-lhef.f`, `pythia6326.f` for the PYTHIA example), `mbook.f` and `analize-hvq.f`. The file `analize-hvq.f` is a minimal analysis program, provided as a starting example for more complex analysis. It is adequate for both HERWIG and PYTHIA (since it uses the standard common blocks of ref. [13]). It uses the histogramming package of M.L. Mangano, `mbook.f`, and it produces topdrawer output in the file `pwgoutput.top`. It can be used for charmed and bottomed hadron production, and for $t\bar{t}$ production. For charm and bottom, it analyses a few kinematic observables, looking only at $D^{\pm}, D^0$, $\overline{D}^0$ for charm, and $B^{\pm}, B^0$, $\overline{B}^0$ for bottom. For top pair production, the program looks for lepton pairs with transverse momenta above a given cut.

POWHEG-hvq also provides an output that is as close as possible to the MC@NLO user file format, so that users that have already implemented MC@NLO in their analysis frameworks should be able to run POWHEG-hvq interfaced to HERWIG with no extra work. The corresponding target is `main-mcatnlofl`. The event file name is `pwgmcatnlofmt.events`.

3.2 Interfacing POWHEG-hvq with a shower Monte Carlo program

One should create a main program that initializes the SMC to make it ready to accept a user process, and provide the following routines

```
subroutine UPINIT
call pwhginit
end

subroutine UPEVNT
call pwhgevnt
end
```

that are the only link to the POWHEG-hvq program. The main program should call the appropriate subroutines to run the SMC. If the SMC is compliant with the LHIUP, it will call the routines `UPINIT` and `UPEVNT` in order to initialize, and to actually generate the hard events. The routine `pwhinit` performs the initialization of POWHEG-hvq, setting up all the grids that are necessary for the efficient generation of the events, and it also initializes the process common block of the LHIUP. Each call to `pwhgevnt` causes the generation of one event, and its storage in the LHIUP event common block.

When using HERWIG, one should remove the dummy subroutines `UPINIT` and `UPEVNT` that are present in the HERWIG source file.

We provide two examples, that can be built with the commands

```
$ make main-HERWIG
```

and

```
$ make main-PYTHIA
```

The analysis program is the same one described in section 3.1.

3.3 POWHEG-hvq as a standalone program

In this case, the main program should have the structure

```
program MAIN
call pwhginit
do j=1,NEVENTS
    call pwhgevnt
c call some analysis routines here
    ...
enddo
c Print out results
...
end
```

No examples are provided. The analysis routines should make use of the information stored in the LHIUP, as documented in ref. [11].

4. Input parameters

POWHEG-hvq provides an independent facility to set the input parameters for the run. All parameters are stored in a file, named `powheg.input`. The format of the file 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.

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 files created by POWHEG-hvq in the current run will carry the prefix `<prefix>-` instead of `pwg`.

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

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

returns the value of token `myparm` stored in `powheg.input`. If the token is not found, a message is printed, and the program is halted. 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. With the statement

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

in case the token `myparm` is not present, the program does not stop, and returns the value -10^6. The file `powheginput.f` is a standalone code, and can be linked to any program. In this way, an SMC that is reading an event file may get parameters of the POWHEG-hvq run, if it needs too.

We document here a typical input file `powheginput.dat`:

```
!Heavy flavour production parameters

maxev 500000 !  number of events to be generated
ih1 1 !  hadron 1 type
ih2 -1 !  hadron 2 type
ndns1 191 !  pdf for hadron 1
ndns2 191 !  pdf for hadron 2
```

The integer `ih1,ih2` and `ndns1,ndns2` characterize the hadron type and pdf used in POWHEG-hvq. The numbering scheme is documented in the file `hvqpdfpho.f`. In that file, in the routine `PRNTSF`, all pdf sets available and the corresponding set number are listed. In particular, 191 corresponds to `MRST2002`. The hadron type in `ih1,ih2` is 0 for a nucleon (i.e. the average of a proton and a neutron), 1 (-1) for a proton (antiproton), 2 (-2) for a neutron (antineutron), and 3 (-3) for a π^+ (π^-). Thus `ih1=1,ih2=-1` corresponds to proton-antiproton collisions.

```
ebeam1 980 !  energy of beam 1
ebeam2 980 !  energy of beam 2
```

We assume that beam 1 and 2 move along the third axis in the positive and negative direction respectively.

```
qmass 175 !  mass of heavy quark in GeV
facscfact 1 !  factorization scale factor:  mufact=muref*facscfact
renscfact 1 !  renormalization scale factor:  muren=muref*renscfact
bbscalevar 1 !  if 0 use muref=qmass in Bbar calculation
```

Factorization and renormalization scale factors appearing here have to do with the computation of the inclusive cross section (i.e. the $\bar{B}$ function [5, 6, 7]), and can be varied by a factor of order 1 in both directions to study scale dependence. Normally the reference scale is set equal to the transverse mass of the heavy quark in the rest frame of the $q\bar{q}$ system. If `bbscalevar` is set to 0, the reference scale is chosen equal to the heavy quark mass. Other choices require a modification of the the subroutine `setscalesbb` in the file `physpar-hvq.f`.
The following is only needed if the quark is a top, and we want it to decay with the inclusion of spin correlations

```
topdecaymode 20000 !  an integer of 5 digits representing the decay mode.
```

The value of the token is formed by five digits, each representing the maximum number of the following particles in the (parton level) decay of the $t\bar{t}$ pair: $e^\pm$, $\mu^\pm$, $\tau^\pm$, $u^\pm$, $c^\pm$. Thus, for example, 20000 means the $t \to e^+\nu_e b$, $\bar{t} \to e^-\bar{\nu}_e\bar{b}$, 22222 means all decays; 10011 means one goes into electron or antielectron, and the other goes into any hadron, or one goes into up or antiup and the other goes into charm or anticharm; 00022 means fully hadronic; 00011 means fully hadronic with a single charm; 00012 fully hadronic with at least one charm. If all digits are 0, neither the t nor the $\bar{t}$ are decayed. This scheme is flexible enough to cover most cases, except for the semileptonic decays. These are achieved by also setting the optional input variable

```
semileptonic 1 !  filter out only semileptonic decays.
```

Thus, for example, with `topdecaymode 10011` and `semileptonic 1` we get only events with one top going to $e^\pm$, and the other into any hadron. Values that imply impossible decays should cause the program to halt.
In case `topdecaymode` is different from 0, more parameters are needed for the decay kine-

matics, and are used exclusively for decays

```
tdec/wmass 80.4    !  W mass for top decay
tdec/wwidth 2.141  !  W width
tdec/bmass 5       !  b quark mass in t decay
tdec/twidth 1.31   !  top width
tdec/elbranching 0.108  !  W electronic branching fraction
tdec/emass 0.00051 !  electron mass
tdec/mumass 0.1057 !  mu mass
tdec/taumass 1.777 !  tau mass
tdec/dmass 0.100   !  d mass
tdec/umass 0.100   !  u mass
tdec/smass 0.200   !  s mass
tdec/cmass 1.5     !  charm mass
tdec/sin2cabibbo 0.051  !  sine of Cabibbo angle squared
```

Spin correlations in the decay are implemented, and effects due to the finite width of the top and of the W are also accounted for.
The following parameters control the operation of the POWHEG-hvq program:

```
!  Parameters to allow-disallow use of stored data
use-old-grid 1
use-old-ubound 1
```

The meaning of this flag require a little knowledge of the operation of POWHEG-hvq. Before the program starts generating events, the integral of the inclusive cross section is computed, and a grid is set up for the generation of Born-like configuration. Similarly, in the generation of hard radiation a grid is computed to get an upper bounding function to the radiation probability. The generation of the grids is time consuming, but the time spent in this way is negligible in a normal run with hundreds of thousands of events being generated. On the other hand, it is useful (for example, when debugging an analysis program) to skip the generation stage. For this purpose, the grid for the generation of Born-like kinematics is stored in the file `pwggrid.dat`.
If `use-old-grid` is 0, and `pwggrid.dat` exists and is consistent, it is loaded, and the old grid and old value of the cross section are used. Otherwise, the grid is generated. 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 flag `use-old-ubound` has the same role for the upper bounding array that is used in the generation of radiation.
The following parameters are used to control the grids generation

```
!  parameters that control the grid for Born variables generation
ncall1 10000 !  number of calls for initializing the integration grid
```

```
itmx1 5 !  number of iterations for initializing the integration grid
ncall2 100000 !  number of calls for computing integral and upper bound grids
itmx2 5 !  number of iterations for computing integral and upper bound grids
foldx 1 !  number of folds on x integration
foldy 1 !  number of folds on y integration
foldphi 1 !  number of folds on phi integration
!  Parameters that controll the generation of radiation
nubound 100000 !  number of bbarra calls to setup upper bounds for radiation
iymax 1 !<=10, number of intervals in y grid to compute upper bounds
ixmax 1 !<=10, number of intervals in x grid
xupbound 2 !  increase upper bound for radiation generation by given factor
```

They may be increased in the following cases

- If the integration results have large errors, one may try to increase `ncall1`, `itmx1`, `ncall2`, `itmx2`.
- If the fraction of negative weights is large, one may increase `foldx`, `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.
- If there are too many upper bound violation in the generation of radiation (see Section 6), one may increase `nubound`, and/or `xupbound`.
- If the efficiency in the generation of radiation is too small, one may try to increase `iymax`, `ixmax`.

In oder to check whether any of these conditions is met, the user should inspect the file pwgstat.dat at the end of the run, as illustrated in sec. 6.

5. Examples

Examples of `powheg.input` files are given in the directories `c-tev`, `b-tev`, `t-tev`, `tdec-tev` and `c-lhc`, `b-lhc`, `t-lhc`, `tdec-lhc`. In the examples in the `tdec-tev,tdec-lhc` directories the $t\bar{t}$ pair is decayed semileptonically into an electron and an antielectron by POWHEG-hvq. In the `t-tev,t-lhc` directories the top decay is handled by the SMC, according to its own parameters, but without including spin correlations. In all examples, the choice of the parameters that control the grid generation is such that a reasonably small fraction of negative weights is generated, even for the extreme case of charm production at the LHC, where a fraction of negative weight less than 3% is achieved.

6. Counters and statistics

Several results relevant to the interpretation of the output of the run are written to the file `pwgstat.dat`. The fraction of negative weights, the total cross section, the number of upper bound failure in the generation of the inclusive cross section, and the generation efficiency, together with failures and efficiency in the generation of radiation, are printed there. These numbers are sufficient to take action in case of problems.

A call to the subroutine `printstat` causes a printout of all `POWHEG-hvq` counters in `pwgstat.dat` file.

7. Using the PDF sets

POWHEG-hvq uses the pdf implementation of `hvqpdfpho.f`. In that file, in the routine `PRNTSF` all pdf sets available, and the corresponding set number are listed. In order to use a set, the corresponding data file must be present in the directory where the run is performed. If the file is not found a message is printed with the name of the missing file, and the program stops. In this case one should copy the missing file from the directory `pdfdata` to the current directory, or set up a symbolic link to it.

8. Random number generator

POWHEG-hvq uses the `RM48` random number generator, documented in the CERNLIB writeups. This generator has default initialization. If a user wishes to start the program with different seeds, he should call `rm48in(iseed,n1,n2)` (before calling the `pwhginit` routine) in order to seed the generator with the integer `iseed`, and skip the first `n1+n2*10**8` numbers, as documented in the manual. This affects the POWHEG-hvq random number sequence. If the program is interfaced to an SMC, the user should also take care to initialize its seeds.

Since patch level 4, one can also set the optional variable `randomseed` in the input file to an arbitrary integer value. The program itself will call `rm48in(randomseed,0,0)` to initialize random number generation.

References

[1] S. Frixione, P. Nason and B. R. Webber, "Matching NLO QCD and parton showers in heavy flavour production," JHEP **0308** (2003) 007 [arXiv:hep-ph/0305252].

[2] P. Nason, S. Dawson and R. K. Ellis, "The Total Cross-Section for the Production of Heavy Quarks in Hadronic Collisions," Nucl. Phys. B **303** (1988) 607.

[3] P. Nason, S. Dawson and R. K. Ellis, "The One Particle Inclusive Differential Cross-Section for Heavy Quark Production in Hadronic Collisions," Nucl. Phys. B **327** (1989) 49 [Erratum-ibid. B **335** (1990) 260].

[4] M. L. Mangano, P. Nason and G. Ridolfi, "Heavy quark correlations in hadron collisions at next-to-leading order," Nucl. Phys. B **373** (1992) 295.

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

[6] P. Nason and G. Ridolfi, "A positive-weight next-to-leading-order Monte Carlo for Z pair hadroproduction," JHEP **0608** (2006) 077 [arXiv:hep-ph/0606275].

[7] S. Frixione, P. Nason and G. Ridolfi, in preparation.

[8] S. Frixione, E. Laenen, P. Motylinski and B. R. Webber, "Angular correlations of lepton pairs from vector boson and top quark decays in Monte Carlo simulations," JHEP **0704** (2007) 081 [arXiv:hep-ph/0702198].

[9] S. Frixione and B. R. Webber, "The MC@NLO 3.3 event generator," arXiv:hep-ph/0612272.

[10] "MadEvent: Automatic event generation with MadGraph" F. Maltoni and T. Stelzer, JHEP 0302:027, 2003 [hep-ph/0208156]

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

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

[13] 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.