The W^+W^+jj POWHEG BOX manual

The Wp_Wp_j_j program is an implementation of the $W^+W^+ jj$ production cross section within the POWHEG BOX framework.

This document describes the input parameters that are specific to this implementation. The parameters that are common to all POWHEG BOX implementation are given in the manual-BOX.pdf document, in the POWHEG-BOX/Docs directory.

If you use this program, please quote [1], [2], [3].

Running the program

Download the POWHEG BOX, do

```
$ cd POWHEG-BOX/Wp_Wp_j_j
```

Running is most conveniently done in a separate directory, for instance do

```
$ mkdir testrun
```

The directory must contain the powheg.input file and a pwgseeds.dat file (see manual-BOX.pdf).

Before compiling make sure that:

- fastjet is installed and fastjet-config is in the path
- ifort and/or gfortran is in the path
- ifort/gfortran libraries are in the environment variable LD_LIBRARY_PATH

It is possible to compile both with ifort or gfortran at the same time, independently, retaining the objects files of each architecture. The executables itself are instead unique.

Create the main program by doing

```
$ make COMPILER=ifort pwhg_main
```

or

```
$ make COMPILER=gfortran pwhg_main
```

If you use LHAPDF, make sure that you have a version compatible with the compiler you are using, and insert the appropriate path for LHAPDFCONFIG in the Makefile.

The gfortran version we tested was 4.4.3. For older versions, the compiler may not recognize the -J option in F90/Makefile.gfortran. One may need to replace it with the -M option.

Enter the testrun directory:

```
$ cd testrun
```

A powheg.input, and a pwgseeds.dat file is present there. When executing

```
$../pwhg_main
```

```
enter which seed
```

the program will require to enter an index in the pwgseeds.dat file, that specifies the line number where the seed of the random number generator to be used for the run is stored. All results generated by the run will be stored in files named *-[index].*. When running on parallel CPU's, make sure that each parallel run has a different index.

The program must be run in several steps. Each new step requires the completion of the previous step.

The timings given in the following refer to the program compiled with `ifort`.

Step 1

Consists of a single run to generate the importance sampling grid. The grid must be generated with the option `fakevirt 1` in `powheg.input`, which means that the virtual term is replaced by a fake one proportional to the Born term.

One needs at least 1000000 events and 2 iterations. Set the following tokens in the `powheg.input` file:

```
ncall1 1000000

itmx1 2

ncall2 0

fakevirt 1
```

Run the program

```
$../pwhg_main

enter which seed
```

enter 1 or any other valid seed number.

It takes roughly 20 hours of CPU. By setting `ncall2 0` in the `powheg.input` file the program stops after the complition of this step.

Step 2

Runs in parallel can be performed now. Comment out the `fakevirt` token from powheg.input.

The runs must be performed in the same directory where Step 1 was performed.

The integration and upper bound for the generation of btilde can be performed with 50-100 runs with 2500-5000 calls. Set

```
ncall2 5000

itmx2 1
```

in `powheg.input`.

Folding numbers that are appropriate for runs at LHC energy are

```
foldcsi 5    !  number of folds on csi integration

foldy 5      !  number of folds on y integration

foldphi 10   !  number of folds on phi integration
```

Time is about 100 hours of cpu for each run with ncall2=5000.

Setting

```
nubound 0
```

in `powheg.input` causes the program to stop after the completion of this step.

In order to run, for example, 100 parallel processes do:

```
$../pwhg_main
```

`enter which seed`

enter an index for each run, (from 1 to 100). The pwgseeds.dat must contain at least 100 lines, each with a different seed.

Step 3

This step can be run in parallel. The number of processes cannot be larger than the one used in the previous step. The run must be performed in the same directory, after all processes in Step 2 are completed. Setting

`numevts 0`

`nubounds 100000`

takes roughly 7 hours per process.

The setting "numevts 0" causes the program to stop after completion of this step.

The parallel execution of the program is performed as in the previous step.

Step 4

Set

`numevts 100000`

(for example) and run in parallel. The number of processes cannot be larger than the one used in the previous step.

At this point, files of the form `pwgevents-[index].lhe` are present in the run directory.

Count the events:

```
$ grep '/event' pwgevents-*.lhe
```

Analyzing the events

It should be easy to feed the `*.lhe` events into generic shower Monte Carlo programs, within the analysis framework of each experiment. We also provide a sample analysis, that computes several histograms and stores them in topdrawer output.

Doing (from the `Wp_Wp_j_j` directory:

```
$ make lhef_analysis
```

```
$ cd testrun
```

```
$ ../lhef_analysis
```

analyzes the bare `POWHEG BOX` output, creating the topdrawer file `LHEF_analysis.top` at the end. The targets `main-HERWIG-lhef` and `main-PYTHIA-lhef` are instead used to perform the analysis on events fully showered using `HERWIG` or `PYTHIA`.

Bibliography

[1] T. Melia, P. Nason, R. Rontsch, and G. Zanderighi, *W^+W^+ plus dijet production in the POWHEGBOX*, 1102.4846. * Temporary entry *.

[2] T. Melia, K. Melnikov, R. Rontsch, and G. Zanderighi, *Next-to-leading order QCD predictions for W^+W^+jj production at the LHC*, *JHEP* **1012** (2010) 053, [1007.5313].

[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** (2010) 043, [1002.2581].