Published for SISSA by Springer

Received: *March 4, 2010*
Accepted: *May 10, 2010*
Published: *June 9, 2010*

A general framework for implementing NLO calculations in shower Monte Carlo programs: the POWHEG BOX

Simone Alioli,[a,b] **Paolo Nason,**[b] **Carlo Oleari**[c,b] **and Emanuele Re**[d,b]

[a] *Deutsches Elektronen-Synchrotron DESY, Platanenallee 6, D-15738 Zeuthen, Germany*
[b] *INFN, Sezione di Milano-Bicocca, Piazza della Scienza 3, 20126 Milan, Italy*
[c] *Università di Milano-Bicocca, Piazza della Scienza 3, 20126 Milan, Italy*
[d] *Institute for Particle Physics Phenomenology, Department of Physics, University of Durham, Durham, DH1 3LE, U.K.*

E-mail: simone.alioli@desy.de, Paolo.Nason@mib.infn.it, Carlo.Oleari@mib.infn.it, emanuele.re@durham.ac.uk

Abstract: In this work we illustrate the POWHEG BOX, a general computer code framework for implementing NLO calculations in shower Monte Carlo programs according to the POWHEG method. Aim of this work is to provide an illustration of the needed theoretical ingredients, a view of how the code is organized and a description of what a user should provide in order to use it.

Keywords: NLO Computations, Hadronic Colliders, QCD

ArXiv ePrint: 1002.2581

doi:10.1007/JHEP06(2010)043

JHEP06(2010)043

Contents

1 Introduction **1**

2 The format of the user subroutines **4**
- 2.1 Tagging parton lines 5
- 2.2 The Born phase space 6
- 2.3 The Born and Born-correlated squared amplitudes 7
- 2.4 The virtual amplitudes 8
- 2.5 The real squared amplitudes 9
- 2.6 Analysis routines 10

3 The singular regions **10**
- 3.1 Example 14

4 The $\tilde{B}$ function **17**
- 4.1 The `btilde` function 18
- 4.2 The Born cross section 19
- 4.3 The soft-virtual cross section 20
- 4.4 The collinear remnants 21
- 4.5 The real contribution to `btilde` 22
- 4.6 The `btildereal` subroutine 23
- 4.7 The subroutine `sigreal_btl` 24
- 4.8 The `flg_smartsig` flag 26
- 4.9 The soft, collinear and soft-collinear limit functions 27

5 Tuning the real cross section in POWHEG **29**

6 The initialization phase **30**

7 The generation of radiation **31**
- 7.1 Radiation from the $\bar{B}$ component 31
  - 7.1.1 Normalization of the upper bounding function 35
  - 7.1.2 The `gen_radiation` routine 36
  - 7.1.3 The `gen_rad_isr` and `gen_rad_fsr` routines 36
- 7.2 Remnant radiation 38
- 7.3 Completion of the event 39

8 The Les Houches interface for user processes **39**

9 Conclusions **41**

JHEP06(2010)043

A Soft contributions **42**
A.1 Soft phase space 42
A.2 One massless and one massive particle 44
A.3 Two massless particles 45
A.4 Two massive particles 45
A.4.1 Two massive particles with equal momenta 47

B Collinear limits **47**
B.1 Initial-state radiation 47
B.2 Final-state singularities 48

C Upper bounding functions for FSR **48**

D Upper bounding functions for ISR **49**

E Choice of scales **52**
E.1 Scales and couplings for the inclusive cross section 52
E.2 Scales and couplings for radiation 52

F Miscellaneous features of the code **53**
F.1 Checking the soft, collinear and soft-collinear limits 53
F.2 Names 53
F.3 Input variables 54
F.4 User files 54

JHEP06(2010)043

1 Introduction

The POWHEG method is a prescription for interfacing NLO calculations with parton shower generators. It was first suggested in ref. [1], and was described in great detail in ref. [2]. Until now, the POWHEG method has been applied to the following processes: Z pair hadroproduction [3], heavy-flavour production [4], e^+e^- annihilation into hadrons [5] and into top pairs [6], Drell-Yan vector boson production [7, 8], W' production [9], Higgs boson production via gluon fusion [10, 11], Higgs boson production associated with a vector boson (Higgs-strahlung) [11], single-top production [12], $Z + 1$ jet production [13], and, very recently, Higgs production in vector boson fusion [14]. Unlike MC@NLO [15], POWHEG produces events with positive (constant) weight, and, furthermore, does not depend on the Monte Carlo program used for subsequent showering. It can be easily interfaced to any modern shower generator and, in fact, it has been interfaced to HERWIG [16, 17] and PYTHIA [18] in refs. [3, 4, 7, 10, 12, 14].

The present work introduces a computer framework that implements in practice the theoretical construction of ref. [2]. We call this framework the POWHEG BOX. The aim of the POWHEG BOX is to construct a POWHEG implementation of an NLO process, given the following ingredients:

i. The list of all flavour structures of the Born processes.

ii. The list of all flavour structures of the real processes.

iii. The Born phase space.

iv. The Born squared amplitudes $\mathcal{B}$, the color correlated ones $\mathcal{B}_{ij}$ and spin correlated ones $\mathcal{B}_{\mu\nu}$. These are common ingredients of NLO calculations performed with a subtraction method.

v. The real matrix elements squared for all relevant partonic processes.

vi. The finite part of the virtual corrections computed in dimensional regularization or in dimensional reduction.

vii. The Born colour structures in the limit of a large number of colours.

With the exception of the virtual corrections, all these ingredients are nowadays easily obtained. A matrix element program (like MADGRAPH) can be used to obtain (iv) and (v). The colour-correlated and spin-correlated Born amplitudes are also generated automatically by programs like MadDipole [19] and AutoDipole [20]. Recent progress in the automatization of the virtual cross section calculation may lead to developments where even the virtual contribution (vi) may be obtained in a painless way [21–30]. Given the ingredients listed above, the POWHEG BOX does all the rest. It automatically finds all the singular regions, builds the soft and collinear counterterms and the soft and collinear remnants, and then generates the radiation using the POWHEG Sudakov form factor.

The purpose of this work is twofold. Our first aim is to complete here the theoretical work of ref. [2], by explaining several variants of the procedure illustrated there, that have turned out to be necessary to fulfill our goal. In doing so, we will refer often to formulae given in ref. [2], that is thus a prerequisite for reading the present work. Our second aim is to illustrate specifically how the code really works. It is true to some extent that well-written codes are self explanatory, and, in fact, we tried to write the POWHEG BOX as transparently as possible. However, what may not be so easy to understand from the code is its global organization. We believe that this document, together with the source code, could be used by others to understand the code up to the point of being able to modify it.

Strictly speaking, the present work is neither the documentation of the POWHEG BOX code, nor a description of its theoretical basis. So, for example, we do not include a rigorous description of all the subroutines in the code, and, as we already said, we refer to ref. [2] for the theoretical bases of our method. In general, one expects that a theoretical paper should include the description of how a given calculation has been performed. This normally includes the illustration of some algebraic steps, and, maybe, an indication of what part of the calculation was performed numerically, so that the reader should be able, on the basis of the given indications, to verify its content. In the present case, the calculation is performed relying heavily on computer algorithms, and the only realistic way to verify its content is to understand the code. Thus, here we explain the algorithms and give sufficient indications on the code structure for the reader to understand it and

JHEP06(2010)043

verify its correctness. We believe that a detailed documentation of the POWHEG BOX is not necessary for this purpose, since the details are better understood by directly studying the code. Furthermore, the code itself will unavoidably evolve with time, so that detailed documentation may not be so helpful after all. In summary, this paper should be simply seen as a description of our calculations, that, being performed essentially by a computer program, must, to some extent, coincide with a description of the program itself.

Researchers wishing to use the POWHEG BOX should not need, in principle, to study or understand this whole paper. They only need to know in which format they have to supply the ingredients that are listed above. These are summarized in section 2. Looking at the various implementation examples included in the code should also help with this task. In the near future, we will provide a manual that documents in detail the user interface. The remaining part of this work should be useful in order to understand better certain features that the POWHEG BOX provides, and that the user may need, or to implement new features that are not yet available.

The paper is organized as follows: in section 2 we report all the information needed to interface an NLO program to the POWHEG BOX. Thus, for example, we specify how the flavour structure of scattering processes is represented, how to specify their kinematics, and the format that the Born, virtual and real cross section subroutines must have. In section 3 we describe the algorithm used to generate all the singular regions of the process, and how these are represented in the computer code. A simple example is also discussed in detail. In section 4 we describe how the $\tilde{B}$ function is constructed. This function, when integrated over the radiation variables, yields the inclusive cross section at fixed underlying Born configuration, and thus is crucial for the first stage of the event generation, i.e. the generation of the underlying Born structure. The computation of $\tilde{B}$ is quite complex, so this section is divided into several subsections, dealing with the Born contribution, the soft-virtual contribution, the real contribution and its soft and collinear limits. In section 5 we describe the mechanism provided in POWHEG BOX for tuning the part of the real cross section that is dealt with using the shower Monte Carlo method, and how the remaining finite part is treated. This separation into a singular and a finite part, besides being useful for tuning the POWHEG output, also provides a method to improve generation efficiency in certain cases. In section 6 we give an overview of the initialization stage of POWHEG. At this stage the inclusive cross section is computed, and the appropriate grids are set up for the generation of the underlying Born configurations. In section 7 we describe the second stage of the event generation, i.e. the generation of radiation, and in section 8 we describe how the POWHEG-generated event is prepared for further showering by a standard shower Monte Carlo program. Finally, in section 9, we give our conclusions.

Several appendixes collect further analytical and technical details. In appendix A we report the formulae for the soft integrals that are used in the POWHEG BOX. In appendix B we report the collinear limits of the real cross section, that are used in the POWHEG BOX to build the collinear counterterms. In appendixes C and D we describe the upper bounding functions used in the generation of radiation, both for final-state and for initial-state radiation. In appendix E we describe how the renormalization and factorization scales are set in the POWHEG BOX, and how the strong coupling constant is computed. Finally, in

JHEP06(2010)043

appendix F, we give a discussion of a few miscellaneous topics that are useful for using and understanding the program.

2 The format of the user subroutines

By flavour structure of a process we mean the type of all incoming and outgoing particles. In the program, the flavour type is denoted by an integer number, so that the flavour structure is a list of integers. The ordering and numbering of particles follow the rules:

1. first particle: incoming particle with positive rapidity
2. second particle: incoming particle with negative rapidity
3. from the third particle onward: final-state particles ordered as follows
   - colourless particles first,
   - massive coloured particles,
   - massless coloured particles.

The flavour is taken incoming for the two incoming particles and outgoing for the final-state particles.

The flavour index is assigned according to the Particle Data Group conventions [31], except for the gluons, where 0 (rather than 21) is used.

Example: if we are interested in the associated production of a t quark and a vector boson Z plus two jets

$$pp \to Z\,t + 2 \text{ jets}\,, \tag{2.1}$$

then one of its contributing subprocesses is

$$b\,u \to Z\,t\,s\,g\,, \tag{2.2}$$

whose flavour structure, according to the previous rules, is given by

$$[5,\ 2,\ 23,\ 6,\ 3,\ 0]\,. \tag{2.3}$$

In QCD calculations, the colourless particles and the massive coloured particles will remain the same at the Born and NLO level. In the NLO calculation, the flavour structure of real graphs will have one more light parton in the final state. The virtual term, being the interference of the Born and of the one-loop amplitude, has the same flavour structure of the Born term.

In the `POWHEG BOX`, the header file `pwhg_flst.h`, in the `include` subdirectory, contains all arrays and parameters having to do with flavour structures. They depend upon the parameters `nlegborn` and `nlegreal` that are set to the number of legs (incoming plus outgoing) of the Born (or virtual) and real graphs. The user of the `POWHEG BOX` will have to set explicitly `nlegborn` in the `nlegborn.h` file, that is thus included before the `pwhg_flst.h` file in all program units that need to access the flavour structures. In the

JHEP06(2010)043

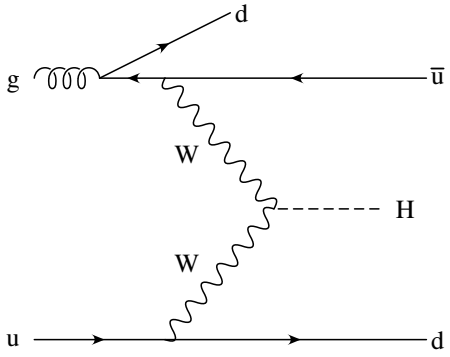

Figure 1. Example of NLO gluon-initiated correction to Higgs boson production in VBF: $gu \to H\bar{u}dd$.

example (2.2), the user should set `nlegborn=6`. The variable `nlegreal` is always set to `nlegborn+1`. The user should also set the variables `flst_nborn` and `flst_nreal` to the number of inequivalent flavour structures for the Born and real graphs, and should also fill the arrays

```
flst_born(k=1:nlegborn,j=1:flst_nborn)
flst_real(k=1:nlegreal,j=1:flst_nreal)
```

in an appropriate initialization subroutine, named `init_processes`. Notice that flavour structures that are equivalent under a permutation of final-state particles should never appear in the list. Thus, in the example of eqs. (2.2) and (2.3), either [5, 2, 23, 6, 3, 0] or [5, 2, 23, 6, 0, 3] may appear as flavour structures, but not both.

The user should also set the value of `flst_lightpart`, the position of the first light coloured parton. Then, in the example previously described, `flst_lightpart=5`.

2.1 Tagging parton lines

At times it is convenient to treat lines with the same flavour as if they were different.

One such example is Higgs boson production via vector boson fusion (VBF). The fermion lines attached to the vector bosons may be treated as being distinct. Of course, in W^+W^- fusion, they may also be effectively distinct, with the W^+ coming from a u quark turning into a d, and the W^- coming from a c quark turning into an s. Consider however the real graph depicted in figure 1. It corresponds to a gluon-initiated next-to-leading correction to VBF Higgs boson production: $gu \to H\bar{u}dd$. It is clear that the two d quarks in the final state have a very different role, and should be kept distinct. However, as far as the flavour combinatorics is concerned, they are considered identical in the `POWHEG BOX`, that assumes that the graphs are already symmetrized with respect to identical final-state particles. Thus, the combinatoric algorithm will generate two regions for this graph, corresponding to either d being collinear to the incoming gluon. In order to overcome this problem, the `POWHEG BOX` allows the possibility to attribute a tag to each line, so that lines with the same flavour but different tags will be treated differently from the combinatoric point of view. In the example at hand, one assigns the tags according to the scheme in

JHEP06(2010)043

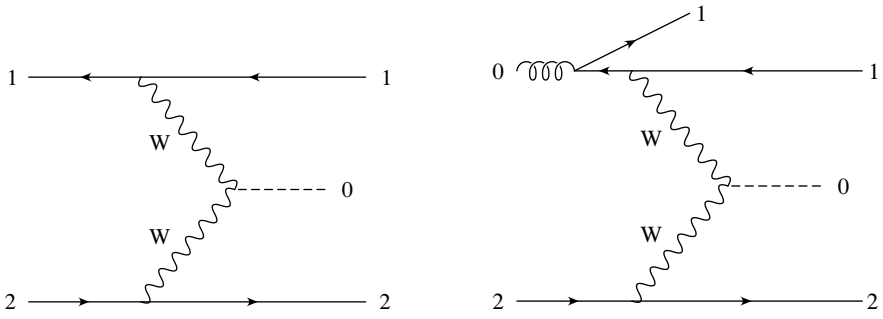

Figure 2. Tag assignment for the underlying Born graph $\bar{d}u \to H\bar{u}d$ and its gluon-initiated real diagram $gu \to H\bar{u}dd$.

figure 2. We arbitrarily assign a tag equal to zero to particles that we do not need to tag (the initial-state gluon and the produced Higgs boson). Within this scheme, the two final-state d quarks will be treated as different from the combinatoric point of view. Only the quark tagged as 1 in the real graph will generate a singular region, since if quark 2 were collinear to the incoming gluon, the associated underlying Born would have an incoming antiquark tagged as 2, and thus would not be present.

In the code, tagged lines are treated by generating an internal flavour index that replaces the real flavour index, in such a way that the internal flavour is different for lines with different flavour or different tag. The combinatorics is carried out with these internal flavour numbers. At the end, internal flavour numbers are replaced with the original real flavour numbers. From this point on, tags are ignored. The tags are set to zero during `POWHEG` initialization. If needed, the user's initialization routine should appropriately set the arrays `flst_borntags` and `flst_realtags`.

The task of changing the flavour indexes to account for different tags, and of changing them back to the original state, are carried out by the routines `mapflavours` and `unmapflavours`, that are called near the beginning and near the end of the `genflavreglist` routine, the subroutine that finds the different singular regions, as described in section 3.

2.2 The Born phase space

The Born phase space (provided by the user of the `POWHEG BOX`) is a subroutine named `born_phsp(xborn)` that fills the four-momenta of Born-process particles (both in the laboratory and in the center-of-mass frame), the Bjorken x of the two incoming partons, the minimal mass of the Born system and the phase space volume.

It receives as input `xborn`, an array of real numbers, in the range $[0, 1]$. If no resonances are present in the final state, the dimension of this array is $3\,n - 2$, n being the number of final-state particles, i.e. $n =$`nlegborn`-2 . In case some resonances are present, their virtuality will require one more variable, and the user will have to take account of them too, increasing accordingly the numbers of elements of this array.

In the paper, we will refer to this set of numbers as $X_{\rm born}$. We recall that the Born

JHEP06(2010)043

phase space $\mathbf{\Phi}_n$, defined in [2], is given by

$$d\mathbf{\Phi}_n = dx_\oplus\, dx_\ominus (2\pi)^4 \delta^4 \left(k_\oplus + k_\ominus - \sum_{i=1}^{n} k_i \right) \prod_{i=1}^{n} \frac{d^3 k_i}{(2\pi)^3 2k_i^0}\,. \tag{2.4}$$

The `born_phsp` routine should perform the following tasks:

1. Set `kn_pborn(mu=0:3,k=1:nlegborn)` and `kn_cmpborn(mu=0:3,k=1:nlegborn)`[1] to the Born momenta in the laboratory frame and in the center-of-mass (CM) frame. The Lorentz index $\mu = 0$ denotes the time component, $1, 2$ the transverse components x, y, and 3 the longitudinal component z. Set the variables `kn_xb1` and `kn_xb2` to the value of the parton momentum fraction $x_\oplus$ and $x_\ominus$. Set the variable `kn_sborn` to the squared CM energy of the Born process.

2. The array `kn_masses` should be filled with the masses of the legs of the process. Furthermore, the variable `kn_minmass` should be set to a fixed (i.e. independent upon the kinematics) lower bound on the mass of the final state. Thus, if no resonances are present, it is typically set to the sum of the masses of the final-state particles. If there are resonances, it will be set to the sum of the lower limits of the windows imposed around the resonances.

3. Set the variable `kn_jacborn` to the Jacobian

$$J_{\rm born} = \left| \frac{\partial \mathbf{\Phi}_n}{\partial X_{\rm born}} \right|. \tag{2.5}$$

2.3 The Born and Born-correlated squared amplitudes

The user of the `POWHEG BOX` should provide the routine

```
setborn(p(0:3,1:nlegborn),bflav(1:nlegborn),born,
        bornjk(1:nlegborn,1:nlegborn),bmunu(0:3,0:3,1:nlegborn)).
```

Given the four-momenta `p` and the flavour structure `bflav` of a Born subprocess, the routine should return the Born squared matrix element $2s_b\mathcal{B}$ in `born`, the colour correlated one in `bornjk` and the spin correlated one in `bmunu`. The flux factor $1/(2\,s_b)$ =`1/(2*kn_sborn)` (where s_b is the center-of-mass energy squared of the Born process) should not be included,[2] since it is supplied by the `POWHEG BOX`.

The colour correlated Born amplitude is defined in eq. (2.97) of ref. [2]. We report it here for completeness

$$2s_b\mathcal{B}_{ij} = -N \sum_{\substack{\text{spins}\\ \text{colours}}} \mathcal{M}_{\{c_k\}} \left(\mathcal{M}^\dagger_{\{c_k\}}\right)_{\substack{c_i\to c_i'\\ c_j\to c_j'}} T^a_{c_i,c_i'}\, T^a_{c_j,c_j'}\,. \tag{2.6}$$

Here $\mathcal{M}_{\{c_k\}}$ is the Born amplitude, and $\{c_k\}$ stands for the colour indexes of all external coloured particles in the amplitude. The suffix on the parentheses that enclose $\mathcal{M}^\dagger_{\{c_k\}}$

[1] All variables with the `kn_` prefix are defined in the header file `pwhg_kn.h`.

[2] In the notation of ref [2], $\mathcal{B}$ includes the flux factor

JHEP06(2010)043

indicates that the colour indexes of partons i, j are substituted with primed indexes in $\mathcal{M}^{\dagger}_{\{c_k\}}$. The factor N is the appropriate normalization factor including averages over initial spin and colour and symmetry factors. We assume summation over repeated colour indexes (c_k, c'_i, c'_j and a) and spin indexes. For gluons $T^a_{cb} = if_{cab}$, where f_{abc} are the structure constants of the $SU(3)$ algebra. For incoming quarks $T^a_{\alpha\beta} = t^a_{\alpha\beta}$, where t are the colour matrices in the fundamental representation (normalized as $\mathrm{Tr}[t\,t] = 1/2$). For antiquarks $T^a_{\alpha\beta} = -t^a_{\beta\alpha}$. It follows from colour conservation that $\mathcal{B}_{ij}$ satisfy

$$\sum_{i,i\neq j} \mathcal{B}_{ij} = C_{f_j} \mathcal{B}\,, \tag{2.7}$$

where i runs over all coloured particles entering or exiting the process, and C_{f_j} is the Casimir constant for the colour representation of particle j. The spin correlated Born squared amplitude $\mathcal{B}^{\mu\nu}_j$ is defined to be non-zero if the j^{th} Born leg is a gluon, and is basically the Born cross section obtained by leaving the gluon indexes of the j^{th} leg uncontracted. More precisely, we can write

$$\mathcal{B}^{\mu\nu}_j = N \sum_{\{i\},s_j,s'_j} \mathcal{M}\left(\{i\},s_j\right) \mathcal{M}^{\dagger}\left(\{i\},s'_j\right) \left(\epsilon^{\mu}_{s_j}\right)^* \epsilon^{\nu}_{s'_j}\,, \tag{2.8}$$

where $\mathcal{M}\left(\{i\},s_j\right)$ is the Born amplitude, $\{i\}$ represent collectively all remaining spins and colours of the incoming and outgoing particles, and s_j represents the spin of the j^{th} particle. The $\epsilon^{\mu}_{s_j}$ are polarization vectors, normalized as

$$\sum_{\mu,\nu} g_{\mu\nu} \left(\epsilon^{\mu}_{s_j}\right)^* \epsilon^{\nu}_{s'_j} = -\delta_{s_j s'_j}\,. \tag{2.9}$$

Thus

$$\sum_{\mu,\nu} g_{\mu\nu}\, \mathcal{B}^{\mu\nu}_j = -\mathcal{B}\,. \tag{2.10}$$

Notice that the Born squared amplitude is requested for each individual flavour structure of the contributing subprocesses. Many different flavour structures will return identical or proportional values of the Born cross section. For example $d\bar{d} \to Z$ is identical to $s\bar{s} \to Z$, and $u\bar{u} \to \gamma^*$ is proportional to $d\bar{d} \to \gamma^*$. The `POWHEG BOX` identifies these identical contributions initially, and stores the proportionality constants. When computing the Born cross section for all needed flavour structures, it computes only the minimum number of squared amplitudes it needs, and obtains the others using the proportionality relations found initially.

2.4 The virtual amplitudes

The user should provide a subroutine

`setvirtual(p(0:3,1:nlegborn),vflav(1:nlegborn),virtual)`,

that returns in `virtual` the finite part $\mathcal{V}_{\text{fin}}$ of the virtual cross section for the process with flavour structure `vflav` and external momenta `p`. The $\mathcal{V}_{\text{fin}}$ contribution is defined, in

JHEP06(2010)043

conventional dimensional regularization (after renormalization), by

$$\mathcal{V}_b = \mathcal{N}\,\frac{\alpha_{\rm S}}{2\pi}\left[\frac{1}{\epsilon^2}\,a\,\mathcal{B} + \frac{1}{\epsilon}\sum_{i,j} c_{ij}\,\mathcal{B}_{ij} + \mathcal{V}_{\rm fin}\right], \tag{2.11}$$

where the a and c_{ij} coefficients do not depend upon ϵ. Their explicit form can be found, for example, in ref. [32]. The normalization factor is

$$\mathcal{N} = \frac{(4\pi)^\epsilon}{\Gamma(1-\epsilon)}\left(\frac{\mu_{\rm R}^2}{Q^2}\right)^\epsilon. \tag{2.12}$$

Here it is important to remark that, in the conventional dimensional regularization, $\mathcal{B}$ and $\mathcal{B}_{ij}$ in formula (2.11) are evaluated in $(4-2\epsilon)$ dimensions, and thus do depend upon ϵ. In fact, all formulas for the soft contributions and the collinear remnants used in the POWHEG BOX are computed in the $\overline{\rm MS}$ scheme, dropping the $1/\epsilon^2$ and $1/\epsilon$ contributions written in terms of the $(4-2\epsilon)$-dimensional expression of the Born squared amplitudes and with the normalization factor of eq. (2.12) (see for example appendix A). Thus, if the virtual contribution is written as in formula (2.11), the divergent terms dropped in the POWHEG BOX cancel exactly the $1/\epsilon$ and $1/\epsilon^2$ terms in (2.11), leaving the finite contribution $\mathcal{V}_{\rm fin}$.

If the NLO calculation has been performed in the Dimensional Reduction (DR) scheme, in order to use it within the POWHEG BOX, we do the following. First of all, we assume that the NLO result is expressed in terms of the standard $\overline{\rm MS}$ coupling constant. If this is not the case, the appropriate straightforward corrections need to be applied. Then one defines $\mathcal{V}_{\rm fin}^{\rm DR}$ using the same formula (2.11) where now $\mathcal{B}$ and $\mathcal{B}_{ij}$ are evaluated in four space-time dimensions. In other words, $\mathcal{V}_{\rm fin}^{\rm DR}$ is defined as the zeroth order coefficient of the Taylor expansion of $\mathcal{V}_b^{\rm DR} 2\pi/(\alpha_{\rm S}\mathcal{N})$ in ϵ. The expression to be used in the POWHEG BOX is [33]

$$\mathcal{V}_{\rm fin} = \mathcal{V}_{\rm fin}^{\rm DR} - \mathcal{B}\sum_i \tilde{\gamma}(f_i)\,, \tag{2.13}$$

where $\tilde{\gamma}(g) = N_c/6$, and $\tilde{\gamma}(q) = (N_c^2-1)/(4N_c)$, with the index i running over all coloured massless partons of the amplitude and where N_c is the number of colours.

The scale Q in eq. (2.12) is completely arbitrary in this context. It may be chosen for convenience equal to s or equal to the renormalization/factorization scale. Within the POWHEG BOX it has been fixed to be equal to the renormalization scale $\mu_{\rm R}$, and so also the user should set it to this value.

2.5 The real squared amplitudes

Together with the Born and Born-correlated squared amplitudes, the user of the POWHEG BOX should provide the routine

```
setreal(p(0:3,nlegreal),rflav(1:nlegreal),amp2)
```

that computes, given the momenta `p` of the external particles, the squared amplitude[3] for the real process specified by the flavours `rflav`, stripped off by a factor $\alpha_{\rm S}/(2\pi)$.

[3] Averaged over spin and colours; as for the Born and the virtual contributions, also in this case the flux factor should not be included by the user.

JHEP06(2010)043

As for the Born contribution, also the real squared amplitude is requested for each individual flavour structure contributing to the real cross section. Many different flavour structures will return identical or proportional values for the real squared amplitude. The POWHEG BOX identifies these identical contributions initially, and stores the proportionality constants. When computing the real cross section for all needed flavour structures, it calculates only the minimum number of squared amplitudes it needs, and obtains the others using the proportionality relations found initially.

2.6 Analysis routines

In the analysis file, pwhg_analysis.f, the user provides her/his own analysis routine to plot kinematic distributions.

In order to have a unique analysis file, able to deal with events at the partonic or at the hadronic level (i.e. after the shower), we pass all the kinematics and properties of the particles via the HEPEVT common block. In this way, the analysis subroutine has only to have access to this common block and to the value of the differential cross section. This last subroutine is then completely independent from the program that has filled the common block, and can be called by PYTHIA or HERWIG too.

The POWHEG BOX, during the integration of the $\tilde{B}$ function, can produce plots with fixed NLO accuracy. This is done via a call to the routine analysis_driver, that fills the common block with the kinematic momenta. This routine receives as input parameters the value of the cross section at that specific kinematic phase-space point and the integer variable ikin. If ikin is set to 0, the event is treated as a Born-like one, and the nlegborn momenta in the kn_pborn kinematic common block are copied on the HEPEVT common block. Otherwise, the subroutine copies on the common block the nlegreal momenta stored in kn_preal. As final step, it invokes the routine analysis, that receives as input parameter only the value of the cross section.[4]

3 The singular regions

For each real flavour structure, the POWHEG method requires that one decomposes the real cross section into the sum of contributions that are divergent in one singular region only. In the notation of ref. [2] we write

$$R = \sum_{\alpha_r} R^{\alpha_r}. \tag{3.1}$$

Each α_r is thus associated with a single flavour structure, and a single singular region. Sometimes we will refer to it as α_r region (or alr region, which is the name of the variable that we often use to represent it in the code). The separation in eq. (3.1) is illustrated in detail in section 2.4 of ref. [2].

We have investigated two methods for obtaining such separation: the first one is based on the subtraction method proposed by Catani and Seymour (CS) [34], and the second

[4] In the POWHEG BOX package, we have included our own analysis routine, that uses TOPDRAWER as histogramming tool for our plots.

JHEP06(2010)043

one on the method proposed by Frixione, Kunszt and Signer (FKS) [35, 36]. It was found that the FKS subtraction method is better suited for our purpose, and so we focused upon it. The difficulties with the CS dipole method are due to the large number of dipoles and dipole types, and to the fact that it is not easy to separate the real cross section into a sum of terms having the same singularity structure of the CS dipoles. It is in fact not possible to simply weight the real cross sections with factors that vanish in all but one singular region, since for each singular region there are several CS dipoles, depending upon the choice of the spectator.

The FKS method is slightly more cumbersome than the CS method when counterterms are computed using the collinear and soft plus-distributions. It turns out, however, that this difficulty remains the same for all processes. The procedure to disentangle the plus-distributions can be coded simply in a general way once and for all.

In the FKS framework, singular regions are characterized as follows:

- A final-state parton i is becoming collinear to either initial-state partons $j = 1, 2$, or soft
- A final-state parton i is becoming collinear to a final-state parton j, or soft.

We will call i the *emitted or radiated parton*, and j the *emitter*. If we replace the pair emitted-emitter with a single parton of the appropriate flavour, we obtain the flavour configuration of the underlying Born.

Given the list of flavours of the real graphs, one is faced with the combinatoric problem of finding all singular regions. In the `POWHEG` framework, this task is carried out keeping in mind that we should be able to group easily all singular regions that share the same underlying Born. In order to ease this task, it is convenient to choose a standard ordering for the flavour structure of each `alr`. One can easily demonstrate that the flavour of all the `alr` can be ordered in such a way that the two following properties are satisfied:

Property 1 *The emitted parton is always the last parton.*

Property 2 *The underlying Born configuration, obtained by removing the emitted parton, and replacing the emitter with a parton of the appropriate flavour, is exactly equal to one of the Born flavour structures present in the list of Born processes* `flst_born`.

Property 2 is non-trivial, since in general the underlying Born structure obtained from a generic `alr` will be equivalent up to a permutation to a flavour structure present in the `flst_born` list.

It is clear that putting all the `alr` in a standard form with the properties 1 and 2 simplifies the `POWHEG` implementation. In fact, since real contributions sharing the same underlying Born are often grouped together in `POWHEG`, it is better if the underlying Born flavour structures are unique. This will be illustrated more clearly in the example described in section 3.1.

In case of initial-state collinear singularities, the emitter will be assigned the value 1 (2) to distinguish collinear emissions from incoming line 1 (2) (the $\oplus$ and $\ominus$ directions

JHEP06(2010)043

of ref. [2]). If the emitted parton is a gluon, the emitter will be assigned the conventional value 0, that means that both 1 and 2 may have emitted it. These distinctions are such to minimize the number of regions, maintaining however the fact that, to each region, we associate a unique underlying Born configuration.

The FKS framework, for final-state radiation (FSR), distinguishes between the emitter and the emitted parton (often called the FKS parton), in the fact that only the emitted parton leads to soft singularities. Thus, the emitter can be a quark, with the emitted parton being a gluon, but not viceversa. On the other hand, the emitter-emitted can be a quark-antiquark pair. In this case, it does not matter what we choose to be the emitted parton. By convention, we will always choose it to be the antiquark. If the emitter and radiated partons are both gluons, when computing the corresponding α_r, we supply a damping factor that removes the soft singularity of the emitter, with an appropriate compensating coefficient. For example, if we call $E_{\rm em}$ and E_r the CM energies of the emitter and emitted parton, the damping factor may be chosen equal to $E_{\rm em}/(E_{\rm em}+E_r)$, and an extra factor of 2 is supplied to account for the region where the role of emitter and emitted partons are exchanged.

In the `POWHEG BOX`, the task of finding all regions associated with a given flavour structure is performed by the routine

`find_regions(nleg,rflav,nregions,iregions)`

where `nleg=nlegreal` and `rflav(1:nlegreal)` is the input flavour structure. It returns in `nregions` the number of regions found, and, for every found region `j`, it returns the positions (in the string of flavours) of the emitter and of the emitted parton (arbitrarily ordered) in `iregions(1:2,j)`.

The algorithm for finding the final-state regions is the following:

1. Loop over all massless parton pairs `i=flst_lightpart:nlegreal`, `j=i+1:nlegreal`.

2. Check if `i` and `j` can come from the splitting of the same parton (i.e. if they have opposite flavours, or if they are both gluons, or one of them is a gluon).

3. If they cannot come from the same parton flavour, skip them.

4. If they can come from the same parton flavour, build up a flavour list with `nlegborn` elements, obtained from `rflav` by deleting partons `i,j` and adding a parton with the appropriate flavour (i.e. if `i,j` have opposite flavour, or are both gluons, add a gluon; if one is a gluon add the flavour of the other parton). Check if the newly built flavour list is an admissible flavour structure for the Born cross section. This is done by checking if the Born flavour structure at hand is equivalent, up to a permutation of the final-state partons, to any element of the list `flst_born`.

5. If the underlying Born flavour structure is valid, increase `nregions` and set

   `iregions(1,nregions) = i` and `iregions(2,nregions)=j`.

The initial-state regions are treated similarly. We check, for each final-state light coloured parton `j`, if it may come from the splitting of an initial-state parton. If it comes from the first (second) incoming line, then the number of regions `nregions` is increased and we set

JHEP06(2010)043

`iregions(1,nregions) = 1 (2)` and `iregions(1,nregions) = j`. If the emitted parton `j` is a gluon, then only one region is generated, and we set `iregions(1,nregions) = 0`.

The first task of the `POWHEG BOX` is to build the list of all α_r in the standard form specified in Properties 1 and 2. This task is performed by the subroutine `genflavreglist`, that, more specifically, does the following:

- It sets the variable `flst_nalr` to the total number of inequivalent singular regions α_r found.
- It sets the variable `flst_nregular` to the number of real flavour structures that do not have any singular region,[5] and it fills the array

  `flst_regular(k=1:nlegreal,alr=1:flst_nregular)`

  with the corresponding flavour structures. If `flst_nregular` is greater than 0, also the flag `flg_withreg` (defined in the header file `pwhg_flg.h`) is set to true.
- It fills the array `flst_alr(k=1:nlegreal,alr=1:flst_nalr)` with the flavour structure corresponding to the given `alr` region. The ordering is guaranteed to respect Properties 1 and 2.
- It sets the array `flst_emitter(alr)` to the emitter of the `alr` structure.
- It sets the `flst_mult(alr)` array to the multiplicity of the `alr`$^{\text{th}}$ structure. This number arises because regions may be found that are equivalent by permutations of the external legs. Only one is retained in this case, with the correct multiplicity.
- It sets the array `flst_uborn(k=1:nlegborn,alr)` to the flavour structure of the underlying Born of the `alr`$^{\text{th}}$ structure.
- It sets the array `flst_alr2born(alr)` to an index in the `flst_born` array, that points to the underlying Born flavour structure of the `alr`$^{\text{th}}$ region.
- It sets up a pointer structure from the underlying Born index to the set of `alr`'s that share it: it sets the array `flst_born2alr(0,jborn)` to the number of `alr` regions that have `jborn` as underlying Born, i.e. `flst_alr2born(alr)=jborn`, and sets `flst_born2alr(i,jborn)`, with `i=1:flst_born2alr(0,jborn)`, to the corresponding `alr` index.
- For each `alr`, a list of all singular regions for the corresponding flavour structure is also build. These are needed in order to compute $\mathcal{R}_{\alpha_r}$, as we will see further on. The array element `flst_allreg(1,0,alr)` is set to the number of singular regions of the flavour structure of the `alr`$^{\text{th}}$ region, i.e. `flst_allreg(1,0,alr)=nregions` of that particular `alr`. Then `flst_allreg(i=1:2,k=1:nregions,alr)` is set to the list of pairs of indexes characterizing the emitter and the emitted parton for each singular region.

[5] An example of such configurations is given by the $q\bar{q} \to Hg$ flavour structure in Higgs boson production via gluon fusion.

JHEP06(2010)043

jborn	processes	flst_born
1	$sc \to gud$	$[3, 4, 0, 2, 1]$
2	$gu \to \bar{s}sc$	$[0, 2, -3, 3, 4]$

Table 1. Flavour structure of two Born subprocesses and the corresponding POWHEG BOX notation, in the third column.

In order to perform this task, the subroutine genflavreglist loops through the flst_real list of real flavour structures, calling for each of them the routine find_regions. Each region found is first transformed with a permutation, in such a way that the emitted parton is always the last in the list. In the case of final-state radiation, the emitter is also moved near the emitted parton (i.e. at the nlegreal-1 position) with a permutation. At this stage, one also makes sure that, if the emitter-emitted pair is made by a quark (antiquark) and a gluon, the gluon is always the emitted parton, and if the pair is a quark and antiquark, the antiquark is always the emitted parton (if this is not the case, emitter and emitted partons are exchanged). The lists flst_alr and flst_emitter are updated accordingly. Once this procedure is completed, the alr list is complete, but each element may appear more than once. The list is thus searched for equivalent elements, it is collapsed in such a way that each element appears only once, and a multiplicity factor flst_mult is setup to keep track of how many occurrences of a given contribution are present. At the end of this procedure, only inequivalent alr's remain, with an associated multiplicity factor. But it may still happen that the same underlying Born configuration may appear in different alr with different ordering. At this stage the alr list is scanned again. If at any point one finds and underlying Born that differs from one appearing in the flst_born list, the flavour structure of the current arl (and its underlying Born flavour structure) are permuted, in such a way that one recovers the same ordering of one of the elements appearing in the flst_born list. In case of final-state radiation, the emitter may end up to be no longer the nlegreal-1 leg of the process, and so, the array flst_emitter is updated accordingly.

3.1 Example

Due to the intrinsic complexity of the procedure for finding the singular regions, it might be useful to examine it on a simple example. Let us consider a process that, at the Born level, has only two flavour structures (flst_nborn=2) and five coloured massless partons (nlegborn=5 and flst_lightpart=3). In the second column of table 1, we list two partonic Born subprocesses, with the corresponding POWHEG BOX flavour structure flst_born.

Suppose now that the real-process contributions are only the four ones in table 2, so that flst_nreal=4. Since the real contributions have one more parton with respect to the Born diagrams, we have also nlegreal=6. Note that we deliberately have not included subprocesses such as $sg \to gud\bar{c}$, $gg \to \bar{s}sc\bar{u}$ and $gu \to \bar{s}scg$, in order to keep the example short.

The subroutine find_regions is called on each flavour structure of the list flst_real. This subroutine returns the list of emitter-radiated pairs, and their total number. For example, when the first flavour structure $[3, 4, 0, 2, 1, 0]$ is passed to the subroutine, this

JHEP06(2010)043

jreal	processes	flst_real
1	$sc \to gudg$	$[3,4,0,2,1,0]$
2	$sc \to s\bar{s}ud$	$[3,4,3,-3,2,1]$
3	$gc \to gud\bar{s}$	$[0,4,0,2,1,-3]$
4	$du \to d\bar{s}sc$	$[1,2,1,-3,3,4]$

Table 2. Flavour structure of four real subprocesses and the corresponding POWHEG BOX notation, in the third column.

alr	iregions	flst_emitter	flst_alr
1	(3,4)	5	$[3,4,1,0,2,0]$
2	(3,5)	5	$[3,4,2,0,1,0]$
3	(3,6)	5	$[3,4,2,1,0,0]$
4	(4,6)	5	$[3,4,0,1,2,0]$
5	(5,6)	5	$[3,4,0,2,1,0]$
6	(0,3)	0	$[3,4,2,1,0,0]$
7	(0,6)	0	$[3,4,0,2,1,0]$
8	(3,4)	5	$[3,4,2,1,3,-3]$
9	(1,6)	1	$[0,4,0,2,1,-3]$
10	(1,3)	1	$[1,2,-3,3,4,1]$

Table 3. List of all the 10 singular regions found up to this stage of the program, of the emitter-radiated pairs, of the position of the emitter, flst_emitter, and of the corresponding flavour structure, flst_alr.

returns the list of 7 pairs: $\{(3,4),(3,5),(3,6),(4,6),(5,6),(0,3),(0,6)\}$. The $(3,4)$ pair means that the gluon in the third position can be emitted by the up quark in the fourth position, and that, by removing this gluon, the so obtained underlying Born is a valid Born, since its flavour structure is equivalent (up to a permutation of the final-state lines) to the first flavour structure in flst_born. The $(0,3)$ pair represents the singular region associated with gluon 3 being emitted by both initial-state partons, and, once removed, we get a valid flavour Born. When the subroutine is called on the second flavour structure in flst_real, $[3,4,3,-3,2,1]$, it returns a single pair $(3,4)$, meaning that the $s\bar{s}$ pair can come from the splitting of a gluon, and the diagram obtained by replacing the $s\bar{s}$ pair by a gluon is a valid Born. Similarly the call of find_regions on the third element of flst_real returns the pair $(1,6)$ that signals the fact that this diagram is compatible with the splitting of an initial-state gluon into an $s\bar{s}$ pair.

In total we have then 10 singular regions, so that, at this stage of the program, flst_nalr=10. We have listed them in the second column of table 3. The flavour structure of these singular regions is then saved in the array flst_alr after a suitable permutation of the final-state partons, such that the emitted parton is in the last position (nlegreal), and, in case of final-state radiation, the emitter is in the nlegreal-1 position. At the same

JHEP06(2010)043

alr	flst_alr	flst_mult	flst_uborn
1	$[3,4,1,0,2,0]$	2	$[3,4,1,0,2]$
2	$[3,4,2,0,1,0]$	2	$[3,4,2,0,1]$
3	$[3,4,2,1,0,0]$	1	$[3,4,2,1,0]$
4	$[3,4,2,1,0,0]$	2	$[3,4,2,1,0]$
5	$[3,4,2,1,3,-3]$	1	$[3,4,2,1,0]$
6	$[0,4,0,2,1,-3]$	1	$[3,4,0,2,1]$
7	$[1,2,-3,3,4,1]$	1	$[0,2,-3,3,4]$

Table 4. List of all the 7 singular regions found, of their flavour structure, `flst_alr`, of the multiplicity of each singular region, `flst_mult`, and of the underlying Born flavour, `flst_uborn`.

time, the position of the emitter is recorded in the array `flst_emitter` (see the third and fourth column of table 3).

At this stage, the list of singular regions is scanned, and, if two elements are equivalent (up to a permutation of the final-state partons) and have equal emitted and radiated parton, one of them is removed from the list and the multiplicity factor `flst_mult` of that singular region is increased. By referring to table 3, the fourth `alr` flavour list, $[3,4,0,1,2,0]$, is equivalent to the first one, $[3,4,1,0,2,0]$, so that it is removed and the multiplicity factor of the first singular region is set to 2. This is illustrated in table 4, where the fifth and the seventh singular regions of table 3 have been removed and the multiplicity factors of the second and fourth singular region in table 4 are set to 2.

In the last column of table 4, we collect the underlying Born flavour structures, `flst_uborn`, corresponding to each `alr` singular region. Each of these underlying Born must be equivalent (up to a permutation of the final-state partons) to one of the Born in the list of valid Born flavour structures, `flst_born`.

At this point, we scan the list of the underlying Born flavour structures, `flst_uborn`, and permute the flavours in such a way to obtain exactly one element of the `flst_born` list. Correspondingly, we reorder the flavours of the corresponding `alr` singular region. For example, consider the first element of `flst_uborn`, $[3,4,1,0,2]$. This element becomes equal to $[3,4,0,2,1]$ (the first element of the `flst_born` list), only after the exchange of the element in the third position with the one in the forth position ($1 \leftrightarrow 0$) followed by the exchange of the element in the fourth position with the one in the fifth position ($2 \leftrightarrow 1$). We perform the same exchanges on the first element of `flst_alr`, i.e. $[3,4,1,0,2,0]$, and we obtain $[3,4,0,2,1,0]$, and, after these permutations, the emitter is the parton in the fourth position, so that `flst_emitter=4`. By performing this task on every underlying Born flavour structure, we obtain the second and third column of table 5.

As final tasks, we fill the arrays with the pointers to go from an `alr` region to its underlying Born flavour structure and viceversa. The first 6 elements in the `flst_alr` list have the first Born flavour structure as underlying Born, and only the last region has the second Born flavour structure, so that the elements of the array `flst_alr2born` are as in the fourth column of table 5. Viceversa, 6 `alr` singular regions have the first Born flavour structure

JHEP06(2010)043

alr	flst_alr	emi	a2b	nreg	flst_allreg
1	$[3,4,0,2,1,0]$	4	1	7	$\{(3,4),(3,5),(3,6),(4,6),(5,6),(0,3),(0,6)\}$
2	$[3,4,0,2,1,0]$	5	1	7	$\{(3,4),(3,5),(3,6),(4,6),(5,6),(0,3),(0,6)\}$
3	$[3,4,0,2,1,0]$	3	1	7	$\{(3,4),(3,5),(3,6),(4,6),(5,6),(0,3),(0,6)\}$
4	$[3,4,0,2,1,0]$	0	1	7	$\{(3,4),(3,5),(3,6),(4,6),(5,6),(0,3),(0,6)\}$
5	$[3,4,3,2,1,-3]$	3	1	1	$\{(3,6)\}$
6	$[0,4,0,2,1,-3]$	1	1	1	$\{(1,6)\}$
7	$[1,2,-3,3,4,1]$	1	2	1	$\{(1,6)\}$

Table 5. List of the flavours of the singular regions after they have been reordered so that the corresponding underlying Born is equal to one of the flavour subprocess in the `flst_born` list. From the third column on: the position `emi` of the emitter (`emi=flst_emitter`), the underlying Born flavour structure pointer `a2b` (`a2b=flst_alr2born`), the number of singular regions `nreg` associated with that particular `alr` region (`nreg=flst_allreg(1,0,alr)`) and the list of the corresponding emitter-radiated pairs, `flst_allreg(1:2,1:nreg,alr)`.

jborn	flst_born2alr(0,jborn)	flst_born2alr
1	6	$\{1,2,3,4,5,6\}$
2	1	$\{7\}$

Table 6. In the second column, the number of singular `alr` regions that have the `jborn`$^{\text{th}}$ Born subprocess as underlying Born and the corresponding list of these `alr` regions.

as underlying Born, while the seventh has the second one, so that `flst_born2alr(0,1)=6` and `flst_born2alr(0,2)=1` (see table 6). The list of these singular regions is shown in the third column: `flst_born2alr(1:6,1)`$=\{1,2,3,4,5,6\}$ and `flst_born2alr(1:1,2)`$=\{7\}$.

Finally, for every flavour structure in `flst_alr`, we build the list of all the associated singular regions. This is done by simply calling again `find_regions` on every element of the second column of table 5. For every `alr`, the total number of singular regions, `nreg`, is saved in `flst_allreg(1,0,alr)` (see the fifth column of table 5), and all the pairs of emitted-radiated partons is saved in the array `flst_allreg(1:2,1:nreg,alr)` (see sixth column).

4 The $\tilde{B}$ function

In order to generate an event, `POWHEG` first generates a Born kinematic and flavour configuration, with a probability proportional to (see eq. (4.13) in ref. [2])

$$\bar{B}(\mathbf{\Phi}_n)\, d\mathbf{\Phi}_n = \left[\sum_{f_b} \bar{B}^{f_b}(\mathbf{\Phi}_n)\right] d\mathbf{\Phi}_n, \tag{4.1}$$

JHEP06(2010)043

where

$$\bar{B}^{f_b}(\mathbf{\Phi}_n) = [B(\mathbf{\Phi}_n) + V(\mathbf{\Phi}_n)]_{f_b} + \sum_{\alpha_r \in \{\alpha_r | f_b\}} \int \left[d\Phi_{\rm rad}\, \hat{R}(\mathbf{\Phi}_{n+1}) \right]_{\alpha_r}^{\bar{\Phi}_n^{\alpha_r} = \mathbf{\Phi}_n} + \sum_{\alpha_\oplus \in \{\alpha_\oplus | f_b\}} \int \frac{dz}{z}\, G_\oplus^{\alpha_\oplus}(\mathbf{\Phi}_{n,\oplus}) + \sum_{\alpha_\ominus \in \{\alpha_\ominus | f_b\}} \int \frac{dz}{z}\, G_\ominus^{\alpha_\ominus}(\mathbf{\Phi}_{n,\ominus})\,. \tag{4.2}$$

The square brackets with a suffix represent a context: everything inside is relative to that suffix. The symbol f_b labels a Born flavour structure. Thus, due to this context notation, B and V in the square bracket refer to the contribution of the Born and soft-virtual cross section having the flavour structure f_b. The suffix in the first summation represents a sum over all α_r that have f_b as underlying Born. The corresponding square bracket under the integral sign means that we integrate in the radiation phase space of the current α_r, keeping the underlying Born variables $\bar{\Phi}_n^{\alpha_r}$ fixed and equal to $\mathbf{\Phi}_n$. The real contribution R has been properly regularized using the plus distributions (see eq. (2.88) and (2.89) in [2]), so that it is written with a hat, and has to be handled properly. The way we deal with the generation of the underlying Born kinematics in `POWHEG` is the following. For each singular region, we parametrize the radiation variables $\Phi_{\rm rad}$ as function of three variables (`xrad` in the code) in the unit cube,

$$X_{\rm rad} = \left\{ X_{\rm rad}^{(1)},\, X_{\rm rad}^{(2)},\, X_{\rm rad}^{(3)} \right\}, \tag{4.3}$$

and we also parametrize the z variable in eq. (4.2) as a function of $X_{\rm rad}^{(1)}$ in the $[0,1]$ interval. We then introduce the $\tilde{B}$ function, defined as

$$\tilde{B}(\mathbf{\Phi}_n, X_{\rm rad}) = \sum_{f_b} \tilde{B}^{f_b}(\mathbf{\Phi}_n, X_{\rm rad}), \tag{4.4}$$

with

$$\tilde{B}^{f_b}(\mathbf{\Phi}_n, X_{\rm rad}) = [B(\mathbf{\Phi}_n) + V(\mathbf{\Phi}_n)]_{f_b} + \sum_{\alpha_r \in \{\alpha_r | f_b\}} \left[\left| \frac{\partial \Phi_{\rm rad}}{\partial X_{\rm rad}} \right| \hat{R}(\mathbf{\Phi}_{n+1}) \right]_{\alpha_r}^{\bar{\Phi}_n^{\alpha_r} = \mathbf{\Phi}_n} + \sum_{\alpha_\oplus \in \{\alpha_\oplus | f_b\}} \frac{1}{z} \left| \frac{\partial z}{\partial X_{\rm rad}^{(1)}} \right| G_\oplus^{\alpha_\oplus}(\mathbf{\Phi}_{n,\oplus}) + \sum_{\alpha_\ominus \in \{\alpha_\ominus | f_b\}} \frac{1}{z} \left| \frac{\partial z}{\partial X_{\rm rad}^{(1)}} \right| G_\ominus^{\alpha_\ominus}(\mathbf{\Phi}_{n,\ominus}). \tag{4.5}$$

One now integrates $\tilde{B}(\mathbf{\Phi}_n, X_{\rm rad})$ in the full $(\mathbf{\Phi}_n, X_{\rm rad})$ phase space, using an integration routine that implements the possibility of generating the integrand with a uniform weight after a single integration. The routine that we use is `mint` [37], that was explicitly built for application in `POWHEG`. Since `mint` is designed to integrate in the unit hypercube, also the Born phase space has to be mapped in a hypercube, spanned by the variables $X_{\rm born}$, as described in section 2.2.

4.1 The `btilde` function

The `btilde(xx,www0,ifirst)` function implements the function $\tilde{B}$ in eq. (4.4). The first elements of the array `xx` correspond to `xborn` and the last 3 correspond to `xrad`. The

JHEP06(2010)043

weight `www0` is passed by the integration routine, and equals the weight factor (arising from integration volume and importance sampling) supplied by the integration routine. The flag `ifirst` has a rather involved use (described in detail in ref. [37]) that is needed in order to implement the folding of some integration variable. The subroutine `mint` may be in fact requested to use more points for some selected integration variable, keeping fixed all the others, for each single random contribution to the integral. In practice, one may require that more points for the variables $X_{\rm rad}$ are used for a single value of $X_{\rm born}$. Upon the first call with a new value of $X_{\rm born}$, the function `btilde` is called with `ifirst=0`. In all subsequent calls with the same $X_{\rm born}$, but different $X_{\rm rad}$, `btilde` is called with `ifirst=1`. After all calls with `ifirst=1`, a last call with `ifirst=2` is performed, where `btilde` accumulates all the values computed since the last `ifirst=0` call. The quantities that depend only upon $X_{\rm born}$ are computed only once when the call with `ifirst=0` is performed. Thus, the Born phase space is generated at this stage. The Born cross sections for all Born flavour structures are computed and made available in appropriate common blocks at this stage, by calling the subroutine `allborn`. The virtual contributions to `btilde` are also computed here. The subroutines `btildeborn(resborn,www)` and `btildevirt(resvirt,www)` fill the arrays `resborn` and `resvirt` with the contributions for each Born flavour structure. The contributions from the collinear remnants and from the real cross section (`btildecoll` and `btildereal`) are computed both for `ifirst=0` and `ifirst=1`. Notice that, in these cases, the index in the arrays `btildecoll` and `btildereal` refers to a given underlying Born. Thus, for each array entry in `btildereal`, for example, several contributions from different $\alpha_{\rm r}$ regions (sharing the same underlying Born) are summed up. All the contributions to `btilde` are accumulated in the array `results`. When called with `ifirst=2`, the behaviour of the program depends upon the setting of the flag `negflag`. If true, `btilde` should only compute the contribution of negative weights. Thus, the positive entries in `results` are zeroed, and the negative entries are replaced with their absolute value. If `negflag` is false (normal behaviour), the negative entries are zeroed. The array `results` is also stored in the `rad_btilde_arr` array, defined in the header file `pwhg_rad.h`. It is needed at the stage of event generation, where the underlying Born flavour structure will be chosen with a probability proportional to its entries.

4.2 The Born cross section

The Born contribution to the `btilde` function is evaluated as follows. When `btilde` is called with `ifirst=0`, the Born phase space is generated with a call to the `gen_born_phsp` subroutine, that, in turn, calls the user provided `born_phsp` subroutine. For cases when the Born cross section is itself infrared divergent (for example, in the Z + jet production case), we need either a generation cut for the underlying Born configuration, or a Born suppression factor. The `POWHEG BOX` provides a standard form for the latter. We do not further discuss this issue here; in the forthcoming reference [13] we will illustrate this problem in great detail. The factorization and renormalization scales are set with a call to `setscalesbtilde`, and then the routine `allborn` is called. The routine computes the Born contributions for all Born flavour structures, and stores them in the arrays `br_born` for the cross section, `br_bornjk` for the colour correlated Born cross section, and `br_bmunu`

JHEP06(2010)043

for the spin correlated one. Many subsequent calls make use of the Born cross sections, so it is mandatory that this call is performed first. In particular, soft and collinear remnants need the Born terms, and so do the singular limits of the real cross section.

In order to understand the code of the `allborn` routine, it is better to assume first that the flag `flg_smartsig` is set to false. The role of this flag is explained in detail in section 4.8.

The call to `btildeborn(resborn)`, in the `btilde` function, fills the array `resborn` with the Born cross section for each Born flavour configuration, including the corresponding parton distribution function (pdf) factors. In case the `flg_nlotest` is set, while computing the integral of the `btilde` function, an analysis routine is also called to perform a bare NLO calculation for several user-provided distributions. In the case of the Born result, this analysis routine is called within `btilde` at the end of a given folding sequence.

4.3 The soft-virtual cross section

The soft-virtual amplitude is described in detail in section 2.4.2 of ref. [2] in the case of massless coloured partons. We complete here the formulae given in [2] in order to include the case of massive partons. All these formulae are implemented in the `POWHEG BOX`, that can also deal with massive coloured partons. When massive coloured partons are present, formula (2.99) in [2] becomes

$$\mathcal{V} = \frac{\alpha_{\rm S}}{2\pi}\left(\mathcal{Q}\,\mathcal{B} + \sum_{i\neq j}\mathcal{I}_{ij}\,\mathcal{B}_{ij} - \mathcal{B}\sum_{i}\mathcal{I}_{i} + \mathcal{V}_{\rm fin}\right). \tag{4.6}$$

The quantity $\mathcal{I}_{ij}$ is non vanishing for i and j denoting any coloured initial and final-state partons. $\mathcal{I}_i$ is non zero for i denoting any massive coloured parton. They both arise from soft radiation, and are reported in appendix A in the same form that appears in the code. The $\mathcal{Q}$ term has the same expression given in ref. [2]

$$\begin{aligned}\mathcal{Q} = \sum_i \Bigg[&\gamma'_{f_i} - \log\frac{s\delta_0}{2Q^2}\left(\gamma_{f_i} - 2C_{f_i}\log\frac{2E_i}{\xi_c\sqrt{s}}\right)\\ &+2C_{f_i}\left(\log^2\frac{2E_i}{\sqrt{s}} - \log^2\xi_c\right) - 2\gamma_{f_i}\log\frac{2E_i}{\sqrt{s}}\Bigg]\\ &-\log\frac{\mu_{\rm F}^2}{Q^2}\left[\gamma_{f_\oplus} + 2C_{f_\oplus}\log\xi_c + \gamma_{f_\ominus} + 2C_{f_\ominus}\log\xi_c\right].\end{aligned} \tag{4.7}$$

E_i denotes the energy of parton i in the partonic CM frame, and f_i denotes the flavour, i.e. g for a gluon, q for a quark and $\bar{q}$ for an antiquark. In addition we have

$$C_g = C_{\rm A}\,, \qquad C_q = C_{\bar{q}} = C_{\rm F}\,, \tag{4.8}$$

$$\gamma_g = \frac{11C_{\rm A} - 4T_{\rm F}\,n_{\rm f}}{6}\,, \qquad \gamma_q = \gamma_{\bar{q}} = \frac{3}{2}C_{\rm F}\,, \tag{4.9}$$

$$\gamma'_g = \left(\frac{67}{9} - \frac{2\pi^2}{3}\right)C_{\rm A} - \frac{23}{9}T_{\rm F}\,n_{\rm f}\,, \qquad \gamma'_q = \gamma'_{\bar{q}} = \left(\frac{13}{2} - \frac{2\pi^2}{3}\right)C_{\rm F}\,. \tag{4.10}$$

We stress that now the index i in the sum of eq. (4.7) runs only over the massless coloured final-state partons. In fact, the contributions in square bracket arise from collinear (rather

JHEP06(2010)043

than soft) final-state singularities, and thus apply only to massless partons. The last line arises from initial-state collinear singularities.

The parameters δ_0 and ξ_c are arbitrary. In the framework of the POWHEG BOX, we have set $\delta_0 = 2$ and $\xi_c = 1$. An analogous parameter for initial-state collinear singularities, δ_I, that appears in the collinear remnants, is also set to 2. These parameters are hardwired in the code, since there is no reason to change them.

The soft-virtual contribution of eq. (4.6) is implemented as follows. The POWHEG BOX has access to the user-provided Born cross section, including colour correlations. It thus builds automatically the $\mathcal{Q}$, $\mathcal{I}_{ij}$ and $\mathcal{I}_i$ contributions of eq. (4.6) to the soft-virtual cross section. The corresponding code is found in the `btildevirt` subroutine, in the `sigsoftvirt.f` file. As already stated in section 2.4, the scale Q is chosen equal to the renormalization scale $\mu_{\rm R}$. The user-provided virtual cross section should then have $Q = \mu_{\rm R}$. The subroutine `btildevirt` fills the array `resvirt` with the contribution of the soft-virtual cross section for all possible underlying Born configurations.

4.4 The collinear remnants

The collinear remnants $G_{\oplus}^{\alpha_\oplus}$ and $G_{\ominus}^{\alpha_\ominus}$ of eq. (4.2) are the finite leftover from the subtraction of collinear singularities. Their general form, in the FKS framework, is given in eq. (2.102) of ref. [2]. They are implemented in the POWHEG BOX in the $\overline{\rm MS}$ scheme.[6] Their implementation in the `btilde` function has to properly handle the distributions in the z variable. This is done using the identities

$$\frac{1}{(1-z)_{\xi_c}} = \frac{1}{(1-z)_{1-x}} + \log\frac{1-x}{\xi_c}\,\delta(1-z)\,, \tag{4.11}$$

$$\left(\frac{\log(1-z)}{1-z}\right)_{\xi_c} = \left(\frac{\log(1-z)}{1-z}\right)_{1-x} + \frac{\log^2(1-x)-\log^2\xi_c}{2}\,\delta(1-z)\,, \tag{4.12}$$

where x stands for either $x_\oplus$ or $x_\ominus$. The z integration extends from x to 1, and thus it is performed using the rules

$$\int_x^1 dz\, f(z)\frac{1}{(1-z)_{\xi_c}} = \int_x^1 dz\,\frac{f(z)-f(1)}{1-z} + \log\frac{1-x}{\xi_c}\,f(1), \tag{4.13}$$

$$\int_x^1 dz\, f(z)\left(\frac{\log(1-z)}{1-z}\right)_{\xi_c} = \int_x^1 dz\,[f(z)-f(1)]\frac{\log(1-z)}{1-z} + \frac{\log^2(1-x)-\log^2\xi_c}{2}f(1), \tag{4.14}$$

where we have always set $\xi_c = 1$ in the code. In this case too, a loop over all underlying Born configurations is performed, and, for each of them, all singular remnant contributions appropriate to the flavours of the corresponding incoming legs are computed. The `btildecoll` function requires an extra integration variable, given as its first argument, to generate a value for z. Within the `btildecoll` function, if the `flg_nlotest` flag is set, a call to the analysis routines is performed to output the contribution of the collinear remnants to the user-defined kinematic plots.

[6] Since currently the DIS factorization schemes are no longer used, we do not implement them.

JHEP06(2010)043

4.5 The real contribution to `btilde`

The real contribution to the `btilde` function has a certain complexity, mostly due to the handling of the distributions in the ξ and y variables. The function `btilde` simply calls the function `btildereal` to get the contribution of the real cross section for each underlying Born configuration. The complex task of evaluating the real integrand is carried out in the subroutine `btildereal`. In this subroutine, there is a loop over all possible emitters, that envelopes its whole body. All contributions are then evaluated for a given emitter. With this choice we avoid repeating continuously complex phase-space evaluations. For each emitter, for given underlying Born and radiation variables, there is only one real kinematics to consider.

If the emitter is a final-state parton, the real phase space is generated by a call to the subroutine `gen_real_phsp_fsr`. For initial-state emission, a call to `gen_real_phsp_isr` is done. The program then calls the subroutine `sigreal_btl`, that fills its array argument with the contributions of all regions (i.e. `alr`) that have as emitter the current emitter `kn_emitter`. This subroutine returns an array whose elements are $R_\alpha(1-y^i)\xi^2$, with $i=1$ in the FSR case and $i=2$ in the ISR case, rather than R_α alone (R_α is defined in the notation of [2]). The quantity $R_\alpha(1-y^i)\xi^2$ has well defined soft, collinear and soft-collinear limits, that are obtained by a call to the subroutines `soft`, `collfsr` (for final-state radiation) or `collisrp` and `collisrm` (for initial-state $\oplus$ and $\ominus$ collinear regions), `softcollfsr` for soft-collinear limit in final-state radiation, and `softcollisrp` and `softcollisrm` for soft-collinear initial-state radiation. The handling of the ξ and y distributions require some care, and we describe it here in some detail.

We begin by looking at the final-state radiation case. The integral that we would like to perform has the form

$$\bar{B}_{\rm real} = \int d\mathbf{\Phi}_n \int_0^{2\pi} d\phi \int_{-1}^{1} dy \int_0^{X(y)} d\xi \, \frac{J(\xi,y,\phi)}{\xi} \left[(1-y)\,\xi^2 R_\alpha\right] \left(\frac{1}{\xi}\right)_+ \left(\frac{1}{1-y}\right)_+ , \quad (4.15)$$

the Jacobian being given by formula (5.40) in ref. [2]. We have indicated explicitly the dependence of the Jacobian upon the radiation variables, but one should keep in mind that it also depends upon the underlying Born variables. We have assumed, in all generality, that the upper limit in the ξ integration in eq. (4.15) may depend upon y, and we have denoted it as $X(y)$. This is in fact not the case in our choice of the final-state radiation kinematics, but it is the case for initial-state radiation (ISR). The `gen_real_phsp_fsr` subroutine returns the Jacobian for the integration of the radiation variables divided by ξ, in the variable `jac_over_csi`.

In eq. (4.15) we introduce a new rescaled variable $\tilde{\xi}$, whose upper bound does not depend upon y

$$\xi = X(y)\tilde{\xi}\,. \quad (4.16)$$

We can easily show that

$$\int_0^{X(y)} d\xi \left(\frac{1}{\xi}\right)_+ F(\xi) = \int_0^1 d\tilde{\xi} \left[\left(\frac{1}{\tilde{\xi}}\right)_+ + \log X(y)\delta(\tilde{\xi})\right] F(\xi)\,. \quad (4.17)$$

JHEP06(2010)043

We now rewrite eq. (4.15) as

$$\bar{B}_{\rm real} = \int d\mathbf{\Phi}_n \int_0^{2\pi} d\phi \int_{-1}^{1} dy \left(\frac{1}{1-y}\right)_+ \left[\int_0^1 d\tilde{\xi}\, \frac{J(\xi,y,\phi)}{\xi} \left[(1-y)\,\xi^2 R_\alpha\right] \left(\frac{1}{\tilde{\xi}}\right)_+ + \log X(y) \lim_{\xi\to 0} \left(\frac{J(\xi,y,\phi)}{\xi} \left[(1-y)\,\xi^2 R_\alpha\right]\right)\right], \tag{4.18}$$

where we should not forget that ξ is now also a function of y through eq. (4.16). Defining

$$f(\xi,y) = \frac{J(\xi,y,\phi)}{\xi} \left[(1-y)\,\xi^2 R_\alpha\right], \tag{4.19}$$

we get

$$\bar{B}_{\rm real} = \int d\mathbf{\Phi}_n \int_0^{2\pi} d\phi \int_{-1}^{1} \frac{dy}{1-y} \left\{ \int_0^1 d\tilde{\xi} \left[\frac{f(\tilde{\xi}X(y),y) - f(0,y)}{\tilde{\xi}} - \frac{f(\tilde{\xi}X(1),1) - f(0,1)}{\tilde{\xi}}\right] + \left[\log X(y)\, f(0,y) - \log X(1)\, f(0,1)\right]\right\}. \tag{4.20}$$

We now see that both $(1-y)\,\xi^2 R_\alpha$ and $J(\xi,y,\phi)/\xi$ in eq. (4.19) should be computed also with $\xi = 0$ (the soft limit), $y = 1$ at fixed $\tilde{\xi}$ (the collinear limit) and both $\xi = 0$ and $y = 1$ (the soft-collinear limit).

The case of initial-state radiation is handled similarly, except that now our starting formula is

$$\bar{B}_{\rm real} = \int d\mathbf{\Phi}_n \int_0^{2\pi} d\phi \int_{-1}^{1} dy \int_0^{X(y)} d\xi\, \frac{J(\xi,y,\phi)}{\xi} \left[(1-y^2)\,\xi^2 R_\alpha\right] \left(\frac{1}{\xi}\right)_+ \times \frac{1}{2}\left[\left(\frac{1}{1-y}\right)_+ + \left(\frac{1}{1+y}\right)_+\right], \tag{4.21}$$

and one proceeds as before, by treating the $y = 1$ and $y = -1$ regions independently.

4.6 The `btildereal` subroutine

We now give a more detailed description of the way `btildereal` is implemented. First of all, the generation of the radiation phase space is performed according to the description given in section 5.1.1 of ref. [2] for initial-state radiation, and in section 5.2.1 for final-state radiation. The subroutines `gen_real_phsp_fsr` and `gen_real_phsp_isr` generate the phase space as a function of the underlying Born kinematics, and of three real variables `xrad(3)`, that assume random values between zero and one. Since we use eq. (4.20), a common Jacobian factor `xjac` for the transformation `xrad(3)` into $\tilde{\xi}$, y and ϕ is also provided. The program also sets the variables `jac_over_csi`, `jac_over_csi_coll` and `jac_over_csi_soft` to $J(\xi,y,\phi)/\xi$, to its collinear limit and to its soft limit respectively, and multiplies them by `xjac`.

In `btildereal`, these limits are obtained through the calls to `soft`, `collfsr` and `softcollfsr`, and the limits for $J(\xi,y,\phi)/\xi$ are provided by the phase space subroutines `gen_real_phsp_fsr`, in the variables `jac_over_csi_coll` and `jac_over_csi_soft`. Notice

JHEP06(2010)043

also that, while the soft limit of $J(\xi, y, \phi)/\xi$ is y independent, this may not be the case for $(1-y)\,\xi^2 R_\alpha$. The computed values of the real contribution, the soft, collinear and soft-collinear counterterms are divided by $(1-y)\,\tilde{\xi}$ (as in eq. (4.20)) and accumulated with the appropriate sign in the array `resreal`, indexed by the underlying Born index of the current `alr`. In the case of final-state radiation, the upper limit for ξ does not depend upon y, being given by formula (5.49) of ref. [2], and is set by the phase space program `gen_real_phsp_fsr` in the common variable `kn_csimax`. Its value is taken from the array `kn_csimax_arr`, indexed by `kn_emitter`, that is filled by the routine `gen_born_phsp` when the underlying Born phase space is generated in `btilde`. In case of initial-state radiation, `kn_csimax` is y dependent, and is computed by an appropriate routine when the real phase space is generated. The last two terms in eq. (4.20), $\tilde{\xi}$-independent, are also accumulated in the `resreal` array.

After the appropriate calls to the routine that generates the real contributions and its various limits, the program loops through all real regions (i.e. `alr`), and accumulates the real contributions according to eq. (4.20) or its initial-state radiation version, in an array indexed by the index of the underlying Born of the current `alr`. This was set in the combinatoric routines, in the array `flst_alr2born`. The accumulated values include the underlying Born Jacobian, the real contribution or one of its various limits, `jac_over_csi` or one of its limits, and the remaining factor of $1/(\xi(1-y))$ for FSR, or $1/(2\xi(1\pm y))$ in the ISR case. The sign of each contribution can be read out from eq. (4.20). Besides filling the output array `resreal`, if the flag `flg_nlotest` is set, the result is also output to NLO analysis routines, that perform a parton-level NLO calculation of user-defined distributions. The analysis driver for the NLO output, i.e. the subroutine `analysis_driver`, described in section 2.6, is then called with the flag set to 1 for the true real contribution, and 0 for all remaining terms.

4.7 The subroutine `sigreal_btl`

The subroutine `sigreal_btl` fills its output array argument, indexed by the `alr`, with the real contributions that have as emitter `kn_emitter`. The real contribution should also be multiplied by $\xi^2(1-y)$ for FSR, or $\xi^2(1-y^2)$ for ISR, and, furthermore, should also be multiplied by the S^α functions, described in section 2.4 of ref. [2]. Two flags control the behaviour of this function: `flg_smartsig` and `flg_withdamp`. An explanation of the role of `flg_smartsig` is given in section 4.8. The `flg_withdamp` flag will instead be explained in section 5.

The code is better understood if one assumes, to begin with, that these flags are set to false. In this case, the program loops over all `alr`. For those that have emitter equal to the current emitter, it calls the subroutine `realgr`, passing as argument the list of flavours of the current configuration, and the real momenta. The function is supposed to return, in its last argument, the value of R, the real matrix element squared. Next, each contribution should be multiplied by its S^α factor. In the framework of the `POWHEG BOX`, we define the S^α factor in the following way. We consider the flavour structure of the α region under consideration, and call it f^α. We call R_{f^α} the corresponding real contribution. R_{f^α} can have several singular regions (see, for example, the list of pairs of indexes in the

JHEP06(2010)043

last column of table 5). Each such region is characterized by a pair of indexes in the legs of the real process, of the form (i, j). These can be the indexes of two final-state lines becoming collinear, or of an initial- and final-state line becoming collinear. According to the POWHEG conventions, one can also set $i = 0$, meaning that there are initial-state collinear singularities in both directions (gluon emission from initial-state partons), and they share the same underlying Born. We call $\mathcal{I}_\alpha$ the array of all singular regions, i.e. $\mathcal{I}_\alpha$ is an array of pairs of indexes. In particular, the emitter associated with the α region, together with the last parton (i.e. the `nlegreal` parton) form a pair that belongs to $\mathcal{I}_\alpha$. Let us call it the (k, n) pair. Then, S^α is given by

$$S^\alpha = \frac{1}{d_{kn}} \left(\sum_{(i,j)\in\mathcal{I}_\alpha} \frac{1}{d_{ij}} \right)^{-1} , \tag{4.22}$$

where d_{ij} are appropriate kinematic functions that vanish when lines i and j become collinear. The choice of the d_{ij} function implemented in the POWHEG BOX can be found in the `compdij` subroutine, in the `gen_real_phsp.f` file. When the phase space is generated, `compdij` is called, and the array `kn_dijterm` is filled. It is an ordered array, i.e. one always assumes $i < j$. For initial-state singularities it is given by the expressions

$$d_{0j} = \left[E_j^2 \left(1 - y_j^2\right)\right]^{p_1} , \tag{4.23}$$

$$d_{1j} = \left[2\, E_j^2 \left(1 - y_j\right)\right]^{p_1} , \tag{4.24}$$

$$d_{2j} = \left[2\, E_j^2 \left(1 + y_j\right)\right]^{p_1} , \tag{4.25}$$

where y_j is the cosine of the emission direction of parton j, in the real CM frame, relative to the positive collision direction. Notice that we assume, by convention, that $k = 0$ means that there are collinear singularities from both the positive and negative direction with the same underlying Born configuration, as is the case when a gluon is emitted. The positive and negative collinear directions need to be considered separately if the corresponding underlying Born differs. The parameter p_1 corresponds to the parameter `par_diexp`, defined in the `pwhg_kn.h` include file. Its default value is 1. For FSR regions we define

$$d_{ij} = \left[2\, (k_i \cdot k_j)\, \frac{E_i\, E_j}{(E_i + E_j)^2} \right]^{p_2} , \tag{4.26}$$

where p_2 corresponds to `par_dijexp`, and is set to 1 by default.

The `sigreal_btl` subroutine builds the S^α factor for each non vanishing `alr`. At the combinatoric stage, for each `alr`, a list of the singular regions associated with its flavour structure was built. This list was stored in the `flst_allreg` array, and is used to find the indexes ij for all the singular regions of the given `alr` (see table 5 for an example).

One extra factor is supplied for final-state singularities if both the emitter and radiated partons are gluons. One multiplies the result by

$$\frac{2E_{\rm em}}{E_{\rm em} + E_{\rm r}}, \tag{4.27}$$

JHEP06(2010)043

where $E_{\rm em}$ and $E_{\rm r}$ are the energy of the emitter and of the radiated parton, evaluated in the partonic CM. This does not change the cross section, because of the symmetry in the exchange of the two gluons, but guarantees that only when the radiated gluon becomes soft we can have a soft singularity. As a final step, the multiplicity of the current `alr` and the $\xi^2(1-y^2)$ (for ISR) or $\xi^2(1-y)$ (for FSR) factors are included.

4.8 The `flg_smartsig` flag

In several processes, organizing the program in the way described in the previous section would lead to several calls to the same matrix elements, with a consequent waste of computing time. In the process of W production, for example, the matrix element is the same whether it is a $u\bar{d}$ or a $c\bar{s}$ collision. Or it may differ only by a Cabibbo-Kobayashi-Maskawa matrix element factor. If the flag `flg_smartsig` is set to true, upon the first call to the `sigreal_btl` subroutine, the routine finds matrix elements that differ only by a constant factor, and builds appropriate array of pointers and proportionality constants. Upon subsequent entries in the program, multiple calls to proportional matrix elements will thus be avoided. All subprograms that invoke user routines for matrix element calculations are affected by the setting of `flg_smartsig`, and implement the same mechanism for avoiding useless calls to user routines. By closely examining the code, the reader can find out how this works. The output of the program, however, should be independent on the setting of this flag. Only speed will be affected. In fact, the random number generator, used to set up the random kinematics to check matrix elements for proportionality, is reset to its original value after the equivalent matrix elements are found.

In order to understand how the programs operate when the `flg_smartsig` flag is turned on (i.e. is set to true) it is better to examine the `allborn` routine. Upon the first call to `allborn`, the current random number is saved, and then the Born cross section for all flavour components is computed, for several value of randomly chosen external momenta. An integer array `equivto` is set up, its value being -1 by default. If the Born contribution for the $j^{\rm th}$ Born flavour configuration is found to be proportional to a previous $k^{\rm th}$ Born flavour configuration (with $k < j$), the value of `equivto(j)` is set equal to k, and the array of real numbers `equivcoeff(j)` is set to the proportionality constant. Upon subsequent calls to `allborn`, this information is used to avoid further calls to `setborn`, whenever possible. Notice the use of `randomsave` and `randomrestore`. By enclosing a set of instructions between a `randomsave` and a `randomrestore` call, we make sure that the random number sequence is not altered by the inserted instructions.

If the flag `flg_smartsig` is set to false, all calls to the matrix element routines are performed, but, thanks to the saving and restoring of the random number sequence, the output of the program should be independent of it. In other words, using `flg_smartsig=true` should only accelerate the program, without altering the output. This feature can be used to check that nothing weird has happened in the setup phase of the `equivto` and `equivcoef` arrays.

JHEP06(2010)043

`soft`	soft limit
`collfsr`	collinear limit for FSR
`softcollfsr`	soft-collinear limit for FSR
`collisrp`	collinear in the $\oplus$ direction
`softcollisrp`	soft-collinear limit in the $\oplus$ direction
`collisrm`	collinear in the $\ominus$ direction
`softcollisrm`	soft-collinear limit in the $\ominus$ direction

Table 7. Subroutines for the soft and collinear limits of the real contributions used in the computation of `btilde`.

4.9 The soft, collinear and soft-collinear limit functions

These limit functions could be obtained, in principle, by numerical methods, using the full real contribution. We have, however, preferred to compute them using the factorization formulae for collinear singularities, and the eikonal formulae for soft emission, to avoid numerical instabilities. Furthermore, the real contributions, and all the manipulations performed by the combinatoric package can be tested for consistency (see appendix F.1).

These soft, collinear and soft-collinear routines are collected in the file `sigcollsoft.f`. The subroutines relevant for the computation of the `btilde` function are reported in table 7. The basic formulae for the collinear limits are collected in appendix B. Here we illustrate the code of the `collfsr` routine. This routine in turns calls `collfsrnopdf`, and provides the luminosity factor to its output. Another ingredient that is necessary to build the collinear limit functions is the direction of the transverse momentum $\hat{k}_{\rm T}$ of the radiated parton with respect to the emitter in the collinear limit (see appendix B), defined in the partonic CM frame. This is a function of the emitter direction and of the azimuthal angle ϕ. The origin of the azimuth. i.e. the plane along which $\phi = 0$ (or π) should be consistent with what `gen_real_phsp_fsr` does in the collinear limit. A change of $\phi \to \phi + \pi$ is instead irrelevant. Our convention for the origin of ϕ is to take a plane containing $\bar{k}_{\rm em}$ (the momentum, in the CM frame, of the parton that will undergo the splitting in the underlying Born, see ref. [2]), and the third axis. The subroutine `buildkperp`, called from the `collfsrnopdf` routine, constructs the 4-vector `kperp(0:3)`. This vector is normalized arbitrarily, and has zero time component. Its only requirement is that it should be parallel to $\hat{k}_{\rm T}$. Its modulus squared is also returned by the `buildkperp` routine. For each `alr` sharing the current emitter, the subroutine `collfsralr` is called, with `kperp` also passed as an argument. The values of ξ and x, defined as

$$x = \frac{k^0}{\bar{k}^0_{\rm em}}, \qquad \xi = \frac{2k^0}{\sqrt{s}}, \tag{4.28}$$

and x/ξ are all passed to the subroutine, that is meant to work also if ξ and x vanish, with their ratio remaining finite. The subroutine `collfsralr` implements the formulae given in appendix B in a straightforward way, multiplying them by $\xi^2(1-y)$, and taking care to use the x/ξ variable when necessary, in such a way that one never divides by x or ξ. There

JHEP06(2010)043

is only one caveat to keep in mind. The collinear approximation is meant to reproduce an R^α contribution. In the collinear limit of the α region, this coincides with R, since the S^α factor becomes 1 in this limit. We should remember, however, that, in case the emitter and radiated partons are both gluons, we have also supplied a factor $2E_{\rm em}/(E_{\rm em}+E_{\rm r})$, which becomes $2(1-x)$ in the collinear limit (see eq. (4.27)). In addition, in this case, we supply a factor of $1/2$ to account for the two identical partons in the final state. Thus, an extra factor of $(1-x)$ is supplied.

The case of initial-state collinear singularities is handled similarly. In this case we simply have $x = 1-\xi$, and, as before, one should evaluate all contributions taking care of never dividing by $1-x$. The routine `collisralr` implements the formulae in appendix B. It carries an integer argument i that corresponds to 1 for the collinear $\oplus$ direction and 2 for the $\ominus$ direction.

In order to get the soft-collinear limits, the subroutines `softcollfsr`, `softcollisrp` and `softcollisrm` simply call the corresponding collinear subroutines setting temporarily `kn_csi` equal to zero.

The soft limit is obtained with the subroutine `soft`, that in turn calls `softalr` to get the soft contribution of a single `alr`. It implements formula (A.1) (for $\epsilon = 0$), multiplied by $\xi^2(1-y)$ for an FSR region, or $\xi^2(1-y^2)$ for an ISR region. In formula (A.1) there are two powers of the soft momentum k in the denominator. We then factorize k^0 in front of the four-vector k and define $k = k^0\hat{k}$, so that ($\xi = 2k^0/\sqrt{s}$)

$$\left(\frac{\xi}{k^0}\right)^2 = \frac{4}{s} \tag{4.29}$$

is finite. The vector $\hat{k}$ carries the information of the direction of the radiated parton. In practice, we thus replace $k \to \hat{k}$ in eq. (A.1), and supply the factor $4(1-y)/s$ or $4(1-y^2)/s$. The vector $\hat{k}$ should equal k/k^0 in the limit $\xi \to 0$, keeping y and ϕ fixed, for the given underlying Born kinematics. It is computed in the subroutine `gen_real_phsp_fsr` and `gen_real_phsp_isr`, with a call to the subroutine `setsoftvec`. Furthermore, we should remember that the functions S^α have non-trivial soft limits. They should be computed in the soft limit and multiplied by the result. For this purpose, the routine `compdijsoft`, in the `gen_real_phsp.f` file, computes the soft limit of the d_{ij} functions, and stores them in the array `kn_dijterm_soft`. This array has a single index, since the second one is the index of the soft parton. There is no need to consider the other d_{ij} terms (those not involving the soft parton) since in the soft limit they are finite, and do not contribute to S^α. Observe that `compdijsoft` assumes that the d_{ij} terms are homogeneous in k^0, which is the case if the two parameters `par_diexp` and `par_dijexp` are equal. If they differ, the fastest vanishing one (in the $k^0 \to 0$ limit) will dominate S^α. It is in principle possible to experiment with settings that have different `par_diexp` and `par_dijexp`, provided that the slowest vanishing ones are excluded in some way from the list. At the moment, this possibility has not been investigated. As a last point, special treatment was required for the FSR collinear limit of two outgoing gluons. Here, in fact, no action should be taken: one should provide a factor $(1-x)$, that equals one in the soft limit.

JHEP06(2010)043

5 Tuning the real cross section in POWHEG

In POWHEG it is possible to tune the contribution to the real cross section that is treated with the Monte Carlo shower technique. This was pointed out first in ref. [1], where the POWHEG method was formulated, and it was first implemented in ref. [10]. In POWHEG there is the possibility to separate the real cross section, in a given singular region α, as follows

$$R^\alpha = R^\alpha_s + R^\alpha_f, \tag{5.1}$$

where R^α_f has no singularities and only R^α_s is singular in the corresponding region. In practice, the separation may be achieved, for example, using a function of the transverse momentum of the radiation $0 \leqslant F\left(k_T^2\right) \leqslant 1$, that approaches 1 when its argument vanishes, and define

$$R^\alpha_s = R^\alpha F\left(k_T^2\right), \tag{5.2}$$

$$R^\alpha_f = R^\alpha \left[1 - F\left(k_T^2\right)\right]. \tag{5.3}$$

One carries out the whole POWHEG-style generation using R^α_s rather than R^α. The contribution R^α_f, being finite, is generated with standard techniques, and fed into a shower Monte Carlo as is.

More generally F can be chosen as a general function of the kinematic variables, provided it approaches 1 in the singular region. This turns out to be useful in all cases when the ratio R/B in the POWHEG Sudakov exponent becomes too large with respect to its corresponding collinear or soft approximation (see for example ref. [7]). In this case, radiation generation becomes highly inefficient. A general solution to this problem (which has already been implemented in refs. [14] and [13]) is to chose the function F in the following way: if the real squared amplitude (no parton distribution functions included), in a particular singular region, is greater than five times its soft and collinear approximation, then F is set to zero, otherwise is set to one. We also stress that this procedure remedies automatically to the Born zeros problem examined in ref. [7].

This feature is implemented in the POWHEG BOX. By setting the flag `flg_withdamp` to true, this behaviour is turned on. When computing the `btilde` function, the real contribution will always be multiplied by a damping factor, supplied by the routine `bornzerodamp`. The damping factor is not necessary in the soft and collinear counterterm contributions, since, in these cases, we will certainly have $F = 1$. The routine `bornzerodamp` takes as argument the α-region index (i.e. the `alr`), the value of $\mathcal{R}^\alpha$ (i.e. the real cross section *without* the parton distribution function factors) and the value of its collinear and soft limits (also without pdf factors). It returns the damping factor as its last argument. The presence of the collinear and soft limits of $\mathcal{R}^\alpha$ in the arguments of the subroutine, allows the user to set a damping factor that depends upon the distance of the real contribution from its collinear or soft approximation, as stated previously. This routine can be easily modified by the user: for example, the sharp theta function adopted in the POWHEG BOX can be replaced by a smoother function, the factor of five can be changed, and so on.

JHEP06(2010)043

One can see the effects of setting the `flg_withdamp` flag in the `sigreal_btl` subroutine. The soft and collinear limits of the real contribution are obtained with calls to the subroutines `collbtl` and `softbtl` (the `btl` ending standing for `btilde`), that make use of the subroutines already described previously for the calculation of the soft and collinear limits.

If a damping factor is used, the leftover term R_f^α of eq. (5.3) needs to be handled independently. The subroutine `sigremnants` deals with this term together with the real terms that do not have any associated singular region, if there are any. It has the same calling sequence of `btilde`. It is meant to be integrated using the `mint` integration program, that allows for the possibility of generating phase space kinematics distributed with a probability proportional to the integrand, after a single integration. Within `sigremnant`, the contribution from the regular real graphs can be integrated with an arbitrary phase-space parametrization, that we choose to be the initial-state radiation parametrization, i.e. the `gen_real_phsp_isr` subroutine. Both the underlying Born configuration and the real phase space are generated within `sigremnant`. The regular contributions to the real cross section are returned by the subroutine `sigreal_reg`. Within `sigreal_reg`, by making use of the list of regular real contributions (that is setup when the combinatorics is carried out), the regular contributions to the cross section are computed. The contribution of the R_f^α terms is more delicate. This is computed with a loop through all possible emitter values using the global variable `kn_emitter`. The real phase space is set according to it. Then the subroutine `sigreal_damp_rem` (where `damp_rem` stands for damp remnants) is invoked. This subroutine is very similar to the `sigreal_btl` subroutine. For all `alr` that share the current emitter, the corresponding R^α is computed, the damping factor `dampfac` is computed, and the real result is multiplied by `(1-dampfac)` (in `sigreal_btl` the result was instead multiplied by `dampfac`).

Notice that `sigreal_damp_rem` and `sigreal_btl` carry out very similar tasks, the only difference being the presence of the factor $(1 - F)$ in the first, and F in the second. This fact is exploited in the `POWHEG BOX` by implementing both of them via a call to a single subroutine `sigreal_btl0`, that carries an extra integer argument. When the extra argument is zero, the multiplication factor is set equal to F, and when it is 1, it is set to $(1 - F)$.

6 The initialization phase

The preparation of the grids for the generation of the events is carried out in the subroutine `bbinit`. Its most important task is to execute the integration of the `btilde` function, determine the fraction of negative weights, compute the total cross section, and, if required, plot the NLO distributions. At the first step, the subroutine `mint` [37] is invoked with `imode` set to 0. In this mode, `mint` integrates the absolute value of `btilde`, and sets up the importance-sampling grid. Next, `mint` is invoked with `imode` set to 1, and the flag `negflag` set to true. In this mode, `mint` computes the negative contribution to the `btilde` function. No histograms for the NLO results are generated up to now. At this stage, `negflag` is set to false, the `flg_nlotest` is set to true, and `mint` is invoked again on `btilde` to compute the positive contribution to the integral. At this stage, the NLO histograms are filled. We stress that also negative weights, if present, will end up in the histograms, so that the

JHEP06(2010)043

NLO histograms should exactly correspond to a standard NLO calculation. The positive weight total cross section computed by `mint` is combined with the negative weight part, and stored in a variable `rad_sigbtl`, defined in the header file `pwhg_rad.h`. After this, the contribution to the cross section from the real remnants is also computed. These are terms that arise either because there are real contributions with no associated singular regions, or because `flg_withdamp` is set to true (see section 5). The remnant cross section calculation is performed with an independent set of grids. Also the remnant contributions will end up in the NLO histograms. The remnant cross section is stored in the variable `rad_sigrm`, and the total in `rad_sigtot=rad_sigrm+rad_sigbtl`.

When `mint` is called with `imode` equal to 1, the upper bounding envelope of the integrated function is also computed, and stored in an array. This upper bounding envelope will be used later for the generation of unweighted events. The arrays `xgrid`, `ymax`, `xmmm` are all necessary for the generation of the events, and they can be saved in a file, so that the time consuming initialization phase does not need to be repeated if one wishes to generate more events in the same context.

The final important task of the `bbinit` routine is the call to the `do_maxrat` subroutine, that sets up the normalization of the upper bounding function for radiation, thus preparing the system for the generation of full events. This will be described in section 7.1.1. In `bbinit` an initialization call to the function `gen`, that generates the underlying Born configuration, is also performed.

7 The generation of radiation

There are two components that contribute to the generation of radiation: one arises from the $\bar{B}$ term and the other from the remnant. The total cross section for the two contributions is stored in the global variables `rad_sigbtl` and `rad_sigrm`. When radiation is generated, one begins by picking one of the two cases with a probability proportional to the respective cross section. In the `POWHEG BOX`, the generation of radiation is carried out in the subroutine `pwhgevent`, that begins precisely by performing this random choice. We describe, in turn, the two components.

7.1 Radiation from the $\bar{B}$ component

This begins with the generation of an underlying Born configuration distributed according to the $\bar{B}$ function. Radiation is generated using the `POWHEG` Sudakov form factor (see eq. (4.21) of ref. [2])

$$\Delta^{f_b}(\mathbf{\Phi}_n, p_{\mathrm{T}}) = \prod_{\alpha_{\mathrm{r}} \in \{\alpha_{\mathrm{r}} | f_b\}} \Delta^{f_b}_{\alpha_{\mathrm{r}}}(\mathbf{\Phi}_n, p_{\mathrm{T}}), \tag{7.1}$$

where

$$\Delta^{f_b}_{\alpha_{\mathrm{r}}}(\mathbf{\Phi}_n, p_{\mathrm{T}}) = \exp\left\{ -\left[\int d\Phi_{\mathrm{rad}} \, \frac{R(\mathbf{\Phi}_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)} \, \theta(k_{\mathrm{T}}(\mathbf{\Phi}_{n+1}) - p_{\mathrm{T}}) \right]_{\alpha_{\mathrm{r}}}^{\bar{\mathbf{\Phi}}_n^{\alpha_{\mathrm{r}}} = \mathbf{\Phi}_n} \right\}. \tag{7.2}$$

If R has been separated into a regular and singular part, according to eq. (5.1), only the singular part will appear in the Sudakov form factor. According to the notation of ref. [2],

JHEP06(2010)043

the square bracket with the α_r suffix indicates that all quantities inside the bracket should be taken relative to the α_r region. So, the $n+1$ phase space in eq. (7.2) is given as a function of the underlying Born phase space $\bar{\mathbf{\Phi}}_n^{\alpha_r}$, taken at the point $\mathbf{\Phi}_n$, and of the radiation variables $\Phi_{\rm rad}$, according to the phase-space mapping defined for the α_r region. In POWHEG, the individual Sudakov form factors for each α_r are further assembled into a product of form factors sharing the same underlying Born and the same radiation region. As we have seen, each singular region is characterized by an emitter and an emitted parton. Within POWHEG, the emitted parton is always the last one, while the emitter can be any light coloured parton in the initial or final state. There is one single initial-state radiation kinematics, independent of which incoming parton is emitting. The final-state radiation kinematics depends instead upon the index of the emitter. We introduce here the label rr to specify the radiation region kinematics: rr = 1, if `kn_emitter`=0, 1 or 2, and rr = `kn_emitter - flst_lightparton + 2`, if `kn_emitter` $\geq$ `flst_lightparton`. In fact, within the POWHEG BOX framework, the radiation kinematics is the same for `kn_emitter`=0, 1 or 2. We write

$$\Delta^{f_b}(\mathbf{\Phi}_n, p_{\rm T}) = \prod_{{\rm rr}\in\{{\rm rr}\,|f_b\}} \Delta^{f_b}_{\rm rr}(\mathbf{\Phi}_n, p_{\rm T}), \tag{7.3}$$

where

$$\Delta^{f_b}_{\rm rr}(\mathbf{\Phi}_n, p_{\rm T}) = \exp\left\{ - \sum_{\alpha_r\in\{\alpha_r|f_b,{\rm rr}\}} \left[\int d\Phi_{\rm rad}\, \frac{R(\mathbf{\Phi}_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)}\, \theta(k_{\rm T}(\mathbf{\Phi}_{n+1}) - p_{\rm T}) \right]_{\alpha_r}^{\bar{\mathbf{\Phi}}_n^{\alpha_r}=\mathbf{\Phi}_n} \right\}, \tag{7.4}$$

and the notation $\{\alpha_r|f_b, {\rm rr}\}$ indicates the ensemble of all α_r that share the same underlying Born f_b and the same radiation region kinematics rr. It makes then sense to define

$$R^{\rm rr}(\mathbf{\Phi}_{n+1}) = \sum_{\alpha_r\in\{\alpha_r|f_b,{\rm rr}\}} R^{\alpha_r}(\mathbf{\Phi}_{n+1}), \tag{7.5}$$

since the phase space only depends upon the radiation region kinematics rr, and not on the specific α_r. With this definition we have

$$\Delta^{f_b}_{\rm rr}(\mathbf{\Phi}_n, p_{\rm T}) = \exp\left\{ - \left[\int d\Phi_{\rm rad}\, \frac{R^{\rm rr}(\mathbf{\Phi}_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)}\, \theta(k_{\rm T}(\mathbf{\Phi}_{n+1}) - p_{\rm T}) \right]_{\alpha_r}^{\bar{\mathbf{\Phi}}_n^{\alpha_r}=\mathbf{\Phi}_n} \right\}. \tag{7.6}$$

In order to generate the radiation, the POWHEG BOX uses the highest-bid algorithm. For each rr, it generates a $p_{\rm T}$ value with a probability distribution equal to

$$P^{f_b}_{\rm rr}(p_{\rm T}) = \frac{\partial}{\partial p_{\rm T}} \Delta^{f_b}_{\rm rr}(\mathbf{\Phi}_n, p_{\rm T}). \tag{7.7}$$

The program then selects the highest $p_{\rm T}$ value, and thus fixes the corresponding rr region. The α_r value is picked in the ensemble $\{\alpha_r|f_b, {\rm rr}\}$, with a probability proportional to the corresponding R_{α_r}.

The individual $p_{\rm T}$ values for each $\Delta^{f_b}_{\rm rr}$ are generated with the veto method. We define

$$d\Phi_{\rm rad} = J^{\rm rr}\, d\xi\, dy\, d\phi, \tag{7.8}$$

JHEP06(2010)043

where $J^{\rm rr}$ is the Jacobian of the $\Phi_{\rm rad}$ phase space, when written as function the three radiation variables: ξ, y and ϕ. We then introduce a suitable upper bounding function $U^{\rm rr}(\xi, y)$, and determine its normalization $N^{\rm rr}_{f_b}$ by requiring

$$\frac{J^{\rm rr}\, R^{\rm rr}(\mathbf{\Phi}_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)} \leqslant N^{\rm rr}_{f_b}\, U^{\rm rr}(\xi, y)\,, \tag{7.9}$$

for all $\mathbf{\Phi}_n$ and $\Phi_{\rm rad}$. In order to generate the radiation, one first generates a $p_{\rm T}$ value according to the probability distribution

$$P^U_{\rm rr}(p_{\rm T}) = \frac{\partial}{\partial p_{\rm T}} \Delta^U_{\rm rr}(p_{\rm T}), \tag{7.10}$$

where

$$\Delta^U_{\rm rr}(p_{\rm T}) = \exp\left[-N^{\rm rr}_{f_b} \int d\xi\, dy\, d\phi\, U^{\rm rr}(\xi, y)\, \theta(k_{\rm T}(\mathbf{\Phi}_{n+1}) - p_{\rm T})\right], \tag{7.11}$$

and then generates the corresponding values for the radiation variables ξ, y and ϕ, distributed with a probability proportional to $U^{\rm rr}$. At this point one accepts the event with a probability

$$\frac{1}{N^{\rm rr}_{f_b}\, U^{\rm rr}(\xi, y)} \frac{J^{\rm rr}\, R^{\rm rr}(\mathbf{\Phi}_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)}\,. \tag{7.12}$$

If the event is rejected, one goes back to the beginning, and generates a new $p'_{\rm T}$ value, smaller than the current one, using the probability

$$P^U_{\rm rr}(p'_{\rm T}, p_{\rm T}) = \frac{\partial}{\partial p'_{\rm T}} \frac{\Delta^U_{\rm rr}(p'_{\rm T})}{\Delta^U_{\rm rr}(p_{\rm T})}\,, \qquad p'_{\rm T} \leqslant p_{\rm T}\,. \tag{7.13}$$

Ideally, the function $U^{\rm rr}$ should be chosen in such a way that the equation $r = \Delta^U_{\rm rr}(p_{\rm T})$ can be easily solved for $p_{\rm T}$, and a set of radiation variables, distributed according to $U^{\rm rr}$, are easily generated. In practice we only require this second feature, and solve for the equation $r = \Delta^U_{\rm rr}(p_{\rm T})$ numerically (in this way, if r is a uniform random number between 0 and 1, the corresponding $p_{\rm T}$ is distributed according to eq. (7.10)). Several choices are possible. In appendixes C and D we describe the functions used in the POWHEG BOX.

In the POWHEG BOX, a straightforward variant of the veto method is used several times. Suppose that we know a function $F(\Phi_{\rm rad})$ such that

$$\frac{J^{\rm rr}\, R^{\rm rr}(\mathbf{\Phi}_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)} \leqslant F(\Phi_{\rm rad}) \leqslant N^{\rm rr}_{f_b}\, U^{\rm rr}(\Phi_{\rm rad})\,. \tag{7.14}$$

Then we can first use the veto method accepting events with a probability

$$\frac{F(\Phi_{\rm rad})}{N^{\rm rr}_{f_b}\, U^{\rm rr}(\Phi_{\rm rad})}. \tag{7.15}$$

If the event is accepted, we go through a second veto, accepting the event with a probability

$$\frac{1}{F(\Phi_{\rm rad})} \frac{J^{\rm rr} R^{\rm rr}(\Phi_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)}. \tag{7.16}$$

JHEP06(2010)043

This is obviously the same as accepting according to the probability of eq. (7.12), but it has the advantage that, in many instances, when a veto is imposed, one only evaluates the function F, without the need to compute the real and Born contributions. This method is also applied in the POWHEG BOX by replacing eq. (7.9) with

$$\frac{J^{\rm rr}\, R^{\rm rr}(\mathbf{\Phi}_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)} \leqslant N^{\rm rr}_{f_b}(\xi, y)\, U^{\rm rr}(\Phi_{\rm rad}) \leqslant N^{\rm rr}_{f_b}\, U^{\rm rr}(\Phi_{\rm rad})\,, \tag{7.17}$$

where $N^{\rm rr}_{f_b}(\xi, y)$ is a stepwise function in the ξ and y radiation variables, and where

$$N^{\rm rr}_{f_b} = \max_{\xi, y} N^{\rm rr}_{f_b}(\xi, y)\,. \tag{7.18}$$

One determines the step function $N^{\rm rr}_{f_b}(\xi, y)$ so to have the smallest values that satisfy the first bound of eq. (7.17). Then, the method described above is used, where, according to eq. (7.14),

$$F(\Phi_{\rm rad}) = N^{\rm rr}_{f_b}(\xi, y)\, U^{\rm rr}(\Phi_{\rm rad})\,, \tag{7.19}$$

so that events are first accepted with a probability $N^{\rm rr}_{f_b}(\xi, y)/N^{\rm rr}_{f_b}$.

Within the POWHEG BOX, the generation of the underlying Born kinematics is performed by the routines `gen_btilde`, that invokes `gen` with the appropriate arguments. After that, the subroutine `gen_uborn_idx` is called and it generates the underlying Born flavour configuration. The purpose of this call is to pick a random f_b configuration, with a probability proportional to its contribution to the $\tilde{B}$ value at the given kinematic point. By inspecting the `gen_uborn_idx` subroutine, we see how this task is performed. We recall that when `gen` returns, the last call to the `btilde` function has been performed at the generated Born kinematics configuration. The contribution of each flavour component of the $\widetilde{B}$ cross section is stored in the array `rad_btilde_arr`. In `gen_uborn_idx` a generic utility subroutine `pick_random` is invoked with arguments `flst_nborn`, `rad_btilde_arr` and `rad_ubornidx`. The `pick_random` subroutine returns the value `rad_ubornidx` with a probability proportional to `rad_btilde_arr(rad_ubornidx)`. The variable `rad_ubornidx` represents the index of the currently generated underlying Born configuration. There are several other `rad_` prefixed global variables that need to be set, in order to perform the generation of radiation from the current underlying Born. First of all, a list of all `alr`, that share the current underlying Born structure, should be constructed. This is done by filling the array `rad_alr_list`, of length `rad_alr_nlist`, using `flst_born2alr`, that was constructed at the combinatoric stage of the program.

The variable denoting the singular region rr is represented by the global variable `rad_kinreg` in the POWHEG BOX. As already stated, it takes the value 1, for initial-state radiation, and the value `rad_kinreg = kn_emitter - flst_lightparton + 2` for final-state radiation. Not all values of `rad_kinreg` may be associated with an active radiation region for the given underlying Born. A logical array `rad_kinreg_on` is set up, with its entries indexed by the `rad_kinreg` values. The entries set to true correspond to active radiation regions. The array `rad_kinreg_on` is set in the subroutine `gen_uborn_idx`. In table 8 we summarize the global variables relevant to the generation of radiation. We do need all these variables, because we typically need to consider the `alr` that share the current underlying Born

JHEP06(2010)043

rad_ubornidx	index in the current underlying Born flavour structure
rad_alr_list	list of alr's that share the current underlying Born
rad_kinreg_on	marks the active singular regions for the current underlying Born
rad_kinreg	current singular region

Table 8. Global variables that characterize the generation of radiation for the given underlying Born configuration.

and the current kinematic region. This is done by going through the rad_alr_list, and considering only the alr's whose emitter is compatible with the current rad_kinreg value.

7.1.1 Normalization of the upper bounding function

Before the radiation is generated, the normalization of the upper bounding functions should be computed. This task is carried out by the subroutine do_maxrat, which, in turn, is invoked in the bbinit subroutine. The normalizations $N^{\rm rr}_{f_b}(\xi, y)$ and $N^{\rm rr}_{f_b}$ are stored in the arrays

rad_csiynorms(rad_ncsinorms, rad_nynorms, rad_nkinreg, flst_nborn),
rad_norms(rad_nkinreg, flst_nborn).

By inspecting the do_maxrat routine, one can see that there is a mechanism (that is better understood by studying the code) to store and retrieve previously computed values for these arrays. The core of the do_maxrat routine is a loop repeated nubound times, where nubound is a parameter set in the POWHEG BOX data file. Within this loop, gen_btilde and gen_uborn_idx are called in sequence. After that, radiation kinematic variables are set up randomly. The program then loops over all valid radiation regions (i.e. rad_kinreg values). For the ISR radiation region, the initial-state radiation phase space is generated, with a call to gen_real_phsp_isr_rad0, and for the final state a call to gen_real_phsp_fsr_rad0 is performed. The task of effectively increasing the norms is performed in the routine inc_norms. The phase space generation routines are slight variants of the phase space routines previously encountered. They perform a similar task, but are dependent upon the rad_kinreg setting (rather than the kn_emitter value) and furthermore they compute the kinematics starting from the values of kn_csitilde, kn_y and kn_azi (details are found in the gen_real_phsp.f code). The inc_norms subroutine first sets factorization and renormalization scales for radiation (the set_rad_scales call), then computes the Born and real cross section. The real cross section is multiplied by the Jacobian $J^{\rm rr}$. The upper bounding function $U^{\rm rr}$ is returned by the function pwhg_upperb_rad(), and the ratio

$$\frac{J^{\rm rr}\, R^{\rm rr}(\mathbf{\Phi}_{n+1})}{B^{f_b}(\mathbf{\Phi}_n)\, U^{\rm rr}(\xi, y)} \tag{7.20}$$

is formed. Its maximum gives the $N^{\rm rr}_{f_b}(\xi, y)$ normalization. Notice that the subroutines sigborn_rad and sigreal_rad return respectively the value of the Born cross section for the current underlying Born (i.e. for the underlying Born indexed by rad_ubornidx), and the $R^{\rm rr}$ real cross section. Thus, the subroutine sigreal_rad is similar to sigreal_btl,

JHEP06(2010)043

but it only computes the cross section contributions that share the underlying Born stored in `rad_ubornidx` and the singular region stored in `rad_kinreg`.

The first two indexes, ξ_i and y_i, in the array `rad_csiynorms` represent the step number of the stepwise function $N^{\rm rr}_{f_b}(\xi, y)$ (i.e. they are integer functions of ξ and y respectively). Their form can be found in the code, but may be subject to future modifications.

For the purpose of tuning the choice of the upper bounding function, each evaluation of formula (7.20) is histogrammed, and the histogram is printed in `TOPDRAWER` format at the end of the upper-bound evaluation, in the file `pwghistnorms.top`. The efficiency in the generation of radiation will depend upon the shape of this histogram. It is roughly estimated by the ratio of the average value of formula (7.20) over its maximum. Highest efficiencies are achieved if the histogram goes sharply to zero near the maximum value of the abscissa. Lowest efficiencies are characterized by histograms with long tiny tails.

7.1.2 The `gen_radiation` routine

This routine is invoked from `pwhgevent`, after the call to `gen_btilde` and `gen_uborn_idx`. It loops through the valid radiation regions (i.e. the allowed `rad_kinreg` values) and it calls either the `gen_rad_isr` or the `gen_rad_fsr` routines, that generate and store in the global variables `kn_csi`, `kn_y` and `kn_azi` a set of kinematics radiation variables. It also returns, in its argument, the value of the radiation transverse momentum squared, `t`, that is defined in the function `pwhg_pt2`. If `t` is the largest generated so far, the kinematics radiation variables and the `rad_kinreg` value are saved in local variables, because they are the candidate for the highest bid method (discussed in appendix B of ref. [2]). At the end of the loop, if no call has generated any radiation, the routine exits, after setting `kn_csi` to zero, which signals the generation of a Born-like event. If radiation was generated, the saved values of the radiation variables are restored in global variables, and the appropriate phase-space generation routine is invoked. The `sigreal_rad` routine is invoked again, followed by a `gen_real_idx` call. Besides returning $R^{\rm rr}$, `sig_real_rad` also stores each cross section contribution, so that, after the radiation kinematics is generated, the corresponding flavour structure can also be generated with a probability proportional to each cross section contribution. This is what the `gen_real_idx` call does. The index (i.e. the `alr`) of the corresponding real flavour structure is stored in the variable `rad_realalr`.

The subroutine `sigreal_rad`, as stated earlier, is similar to `sigreal_btl`, but it only computes the cross section contributions that share the underlying Born stored in `rad_ubornidx` and the singular region stored in `rad_kinreg`. It also takes care to avoid generating gluon splittings into heavy quark pairs below threshold. More precisely, if the radiation $k_{\rm T}^2$, returned by the function `pwhg_pt2()`, is below `rad_charmthr2` for $g \to c\bar{c}$ or below `rad_bottomthr2` for $g \to b\bar{b}$, the corresponding result is set to zero.

7.1.3 The `gen_rad_isr` and `gen_rad_fsr` routines

These routines generate a $p_{\rm T}$ value according to the Sudakov form factors in eq. (7.6). They make essential use of the function `pt2solve(pt2,i)`, that represents the function $\log \tilde{\Delta}(p_{\rm T}^{\rm old})/\tilde{\Delta}(p_{\rm T})$. The expression of $\log \tilde{\Delta}(p_{\rm T})$ is as given in eq. (D.18), in the case of initial-state radiation, or as given in eqs. (C.6) or (C.10) (in this last case, depending upon

the form of the upper-bounding function used, controlled by the value of the integer variable `iupperfsr`). The value of $p_{\rm T}^{\rm old}$ corresponds to the last vetoed $p_{\rm T}$. The value $\log\tilde{\Delta}(p_{\rm T}^{\rm old})$ is represented by the variable `xlr`. Thus, solving `pt2solve` for zeros represents the first step of each iteration of the veto procedure.

We examine now in detail the case of final-state radiation, with `iupperfsr=1`, that also illustrates how the other cases work. Looking at the function `pwhg_upperb_rad`, we see that, for `iupperfsr=1`, the upper bounding function (up to the normalization factor) has the form

$$\frac{U^{\rm rr}}{N^{\rm rr}_{f_b}(\xi,y)} = \frac{\alpha_{\rm S}^{\rm PW}}{\xi(1-y)}\,, \tag{7.21}$$

where $\alpha_{\rm S}^{\rm PW}$ is the variable-flavour two loop expression for the strong coupling constant used for radiation generation throughout the `POWHEG BOX` program. It is set by a call to `set_rad_scales` (the choice of scales are discussed in detail in appendix E). In appendix C, it is shown how to deal with this form of the upper bounding function for the case of one loop, constant flavour $\alpha_{\rm S}$, and for constant $N^{\rm rr}$ (i.e. not dependent upon ξ and y). We thus begin by considering the upper bounding function

$$\tilde{U}^{\rm rr} = N^{\rm rr}_{f_b}\,\frac{\alpha_{\rm S}^{\rm rad}}{\xi(1-y)}\,, \tag{7.22}$$

with $\alpha_{\rm S}^{\rm rad}$ given by the one-loop expression (see eq. (C.3))

$$\alpha_{\rm S}^{\rm rad}(\mu) = \frac{1}{b_0^{\rm rad}\log\frac{\mu^2}{\Lambda^2_{\rm rad}}}\,. \tag{7.23}$$

We must choose $b_0^{\rm rad}$ and $\Lambda_{\rm rad}$ in such a way that

$$\alpha_{\rm S}^{\rm rad}(\mu) \geqslant \alpha_{\rm S}^{\rm PW}(\mu)\,, \tag{7.24}$$

in all the range $\mu > p_{\rm T}^{\rm min}$, where $p_{\rm T}^{\rm min}$ is the minimum allowed $p_{\rm T}$ for radiation. If this inequality is fulfilled, we will have $\tilde{U}^{\rm rr} \geqslant U^{\rm rr}$ in all the relevant range. The value of $b_0^{\rm rad}$ is taken equal to

$$b_0^{\rm rad} = \frac{33-2\times 5}{12\pi}\,, \tag{7.25}$$

while $\Lambda_{\rm rad}$ is computed and stored in the global variable `rad_lamll` by a call to the subroutine `init_rad_lambda` at initialization stage, from the routine `init_phys`.

The routine `gen_rad_isr` proceeds by initializing the variable `unorm` to the value of $N^{\rm rr}_{f_b}$, stored in the `rad_norms` array. The variable `unorm` is also made available, via a common block, to the function `pt2solve`. In the same way, the value of `kt2max`, appropriate to the current kinematics and radiation region, is computed and made available to the `pt2solve` routine, together with the value of $\Lambda_{\rm rad}$ and the number of flavour (i.e. 5) to be used in $b_0^{\rm rad}$. At this stage the function `pt2solve` is in the condition to operate properly. Its return value, for `iupperfsr=1`, corresponds to formula (C.6). The veto loop is started, with the variable `xlr` set to the log of a uniform random number $0 < r < 1$. The zero of the

`pt2solve` function is found (using the `dzero` `CERNLIB` routine), which thus corresponds to a $p_{\rm T}$ value that solves the equation

$$\log(r) - \log \Delta^{(\tilde{U}^{\rm rr})}(p_{\rm T}) = 0. \tag{7.26}$$

If $p_{\rm T}^2$ (denoted by `t` in the `POWHEG BOX`) is below the allowed minimum value, a negative `t` is returned to signal the generation of an event with no radiation (i.e. with Born-like kinematics). Otherwise a sequence of vetoes is applied. First, the event is accepted with a probability

$$\frac{\alpha_{\rm S}^{\rm PW}(p_{\rm T}^2)}{\alpha_{\rm S}^{\rm rad}(p_{\rm T}^2)}, \tag{7.27}$$

and vetoed otherwise. After this veto is passed, the distribution of eq. (7.22) has been corrected for the use of $\alpha_{\rm S}^{\rm rad}$, instead of the correct one, $\alpha_{\rm S}^{\rm PW}$. At this stage, ξ is generated (at fixed $p_{\rm T}$): its probability distribution is uniform in its logarithm, as can be evinced from eq. (C.6). The value of y is obtained by solving for y the $k_{\rm T}^2 = p_{\rm T}^2$ definition for FSR, i.e. eq. (C.2). At this stage we can compute a further veto, accepting the event with a probability

$$\frac{N_{f_b}^{\rm rr}(\xi, y)}{N_{f_b}^{\rm rr}}, \tag{7.28}$$

which is the number returned by the subroutine `uboundfct`. After this veto, ξ and y have been generated with probability

$$\exp\left[-\int U^{\rm rr}\,\theta(k_{\rm T} - p_{\rm T})\,d\xi\,dy\,d\phi\right] 2\pi\; U^{\rm rr}\,d\xi\,dy\,. \tag{7.29}$$

At this stage, the Born cross section is computed with a call to `sigborn_rad`, a uniform azimuth for radiation is also generated and also `sigreal_rad` is called to compute the real cross section. One now vetoes again accepting the event with a probability

$$\frac{J^{\rm rr} R^{\rm rr}}{B_{f_b}} \times \frac{1}{U^{\rm rr}}, \tag{7.30}$$

and, after this, the ξ, y and ϕ variables have been generated according to the probability

$$\exp\left[-\int \frac{R^{\rm rr}}{B_{f_b}}\,\theta(k_{\rm T} - p_{\rm T})\,d\Phi_{\rm rad}\right] \frac{R^{\rm rr}}{B_{f_b}}\,d\Phi_{\rm rad}\,, \tag{7.31}$$

which is the desired result.

7.2 Remnant radiation

Within the subroutine `pwhgevent`, the generation of an event with the remnant component of the cross section is carried out as follows. First the subroutine `gen_sigremnant` is invoked. This subroutine uses the routine `gen` to generate a point in the full phase space, distributed with a probability proportional to the `sigremnant` cross section, using the grids previously prepared, as described in section 6. The phase space point remains stored in the kinematics global variables. After that, the `gen_remnant` subroutine is invoked.

JHEP06(2010)043

This subroutine generates the flavour structure of the current event with the appropriate probability. This is possible because the subroutine `gen_remnant` stores in the global arrays `rad_damp_rem_arr` and `rad_reg_arr` (defined in `pwhg_rad.h`) each contribution to the cross section for the last kinematics point computed. The `gen_remnant` subroutine picks a contribution with a probability proportional to the values stored in these arrays. If the contribution is a remnant (described in section 5), its index is stored in `rad_realalr`, and the corresponding underlying Born and emitter is found. The radiation phase space is thus generated again with this value of the emitter, and the same values for the three parameters used to parametrize the radiation phase space in `sigremnant`. These parameters are stored by the `sigremnant` subroutine in the global array `rad_xradremn`. The recalculation of the radiation phase space is necessary, since only in the case when `gen_remnant` picks the last contribution computed, the phase space would already have the appropriate settings. The `gen_remnant` subroutine also returns in its integer argument the value 1 for remnant contributions or the value 2 for regular contributions. If the contribution is from a regular part, its index is retrieved and stored in `rad_realreg`, and the ISR phase space is used, since this is the one we have chosen to use for all regular contributions.

7.3 Completion of the event

For simplicity, in the `POWHEG BOX`, one always assumes that there is an azimuthal symmetry, so that, in the generation of the Born phase space, one can always require that some reference particle in the final state lies on the xz (or yz) plane, where z is the direction of the beam axis. At the end of the event generation, a random azimuthal rotation of the whole event is performed. This is done within the `pwhgevent` routine, through a call to the subroutine `add_azimuth`.

Besides setting up the kinematics and the flavour structure, in order to pass the event to the Les Houches Interface for User Processes [38] (LHIUP from now on), we must also decide up to which scale the subsequent (SMC generated) shower should start. In case of a `btilde` generated event, this scale should coincide with the radiation transverse momentum. In case of remnant or regular contribution, this choice is to some extent ambiguous. In order to maintain some continuity of the remnant events with the `btilde` events, we also set this scale to the radiation transverse momentum. For regular contributions, this value is better decided on the basis of the specific process, and an appropriate function `pt2max_regular` should be provided by the user, in the file `pt2maxreg.f`. The global variable `rad_pt2max` is set to the maximum p_{T} for the subsequent shower. It will be used in the LHIUP interface to set the variable `SCALUP`.

8 The Les Houches interface for user processes

At last, the generated event is put on the LHIUP interface. The scale for subsequent radiation is setup, and colours are assigned to the incoming and outgoing partons. For $\bar{B}$ generated and remnant events, this task is carried out by the subroutine `gen_leshouches`. For regular remnants, a special routine, `gen_leshouches_reg`, does the job and should be provided by the user in the file `LesHouchesreg.f`. The different treatment in the two cases

JHEP06(2010)043

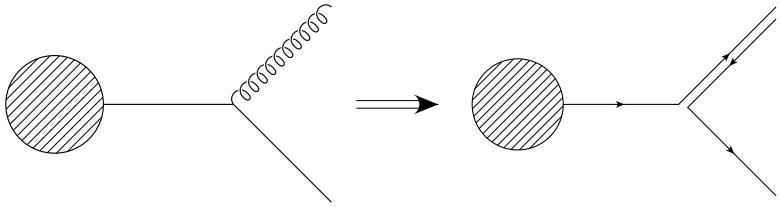

Figure 3. Colour assignment for a singular region corresponding to a quark radiating a collinear gluon in the final state.

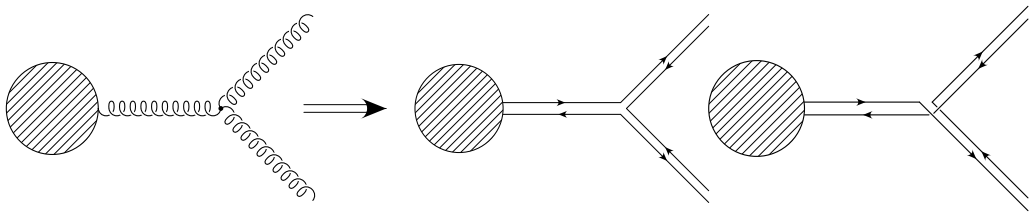

Figure 4. The two alternative colour assignments for a singular region corresponding to a gluon radiating a collinear gluon in the final state.

is due to the fact that, in the case of $\bar{B}$ and remnant events, we have a standard method to assign colour, that is correct in the singular region. For regular contributions, instead, other methods should be used, like resorting, for example, to the planar limit of the cross section formulae.

In general, there is much room for improvement in the technique used for colour assignment [2]. We do not consider this a crucial problem at the present stage. However, if the need of a better colour treatment will emerge, it is clear that the user should provide more colour information. We thus limit ourselves, in the present case, to an approximate colour implementation that is general enough to be process independent. What we do is to assign colour on the basis of the underlying Born configuration first. Then, depending upon the region we are considering, we assign the colour of the real emitter and radiated parton as if a collinear splitting process had really taken place. In the planar limit, this yields a unique colour prescription for the emitter and the radiated parton, except for the case of a gluon splitting into two gluons, that yields two possible colour assignments with equal probabilities. In figures 3 and 4 two particular cases are illustrated.

The routine `gen_leshouches` begins with a call to `born_lh`, that sets up a few event-specific LHIUP parameters, like the flavour, status and mothers of the underlying Born incoming and outgoing particles. In order to understand this part of the code, the reader should refer to the specifications of the LHIUP [38]. The `born_lh` subroutine sets up also the colours of the underlying Born process, by calling the `borncolour_lh` subroutine. This subroutine is process dependent, and must be provided by the user. It should return, in the LHIUP, a planar colour connection, with a probability proportional to its Born contribution in the planar limit. For the most simple processes, like for example Z production, there

JHEP06(2010)043

is one single planar colour connection, i.e. the incoming quark and antiquark should have complementary colours. For more complicated processes, more colour structures can arise. In the planar limit, different colour structures do not interfere, so it is possible to generate a single colour structure with a probability equal to the corresponding contribution to the cross section. In some cases, it may be useful to include also some colour-suppressed contributions. This is the case, for example, in heavy flavour production, where the leading colour configuration always leads to a heavy-flavour pair in an octet colour state. Singlet production may be more suited to the direct production of bound states, and so, it may be better to include it. This can be done, as long as different colour configurations do not interfere with each other.

In the case when $\xi = 0$, a Born event is produced, with a very low value of the `SCALUP` variable, so that no further radiation is generated by the SMC. Within the `gen_leshouches` subroutine, this is achieved by calling `born_lh`, by copying the `kn_pborn` momenta on the LHIUP (a task carried out by the subroutine `momenta_lh`) and by setting `SCALUP` to the minimum radiation transverse momentum. If $\xi \neq 0$, radiation has taken place. The routine `born_lh` is called first, and one more parton is added with the flavour of the radiated parton; the leg corresponding to the emitter in the real graph is assigned the correct flavour. The colours are assigned on the basis of the Born colour already stored in the LHIUP. The subroutine `setcolour_rad` is used to perform this task for both initial- and final-state radiation.

A final task of the LHIUP subroutine is to put on the interface the intermediate resonances. The LHIUP specifies how resonances should be put in the interface. This information should be made available to the shower program, since resonance masses must be preserved by the shower. This is achieved by calling the user routine `resonances_lh`, which calls the `add_resonance` routine for each particle id of intermediate resonances, specifying also its decay products.

9 Conclusions

In this work, we have introduced and documented the `POWHEG BOX`, a computer framework for the construction of a `POWHEG` implementation of any given NLO process. The `POWHEG BOX` code is available at the http://mobydick.mib.infn.it/~nason/POWHEG/.

In the `POWHEG BOX` package, the user can found three directories: `W`, `Z` and `VBF_H`. They contain the code for W, Z and Higgs boson production in vector boson fusion, and can serve as template for any further process that a user may want to implement.

We would like to emphasize that the `POWHEG BOX` is a tool to develop programs. It is not something ready to run out of the box. Thus, even in order to compile the examples, the user should examine the `Makefile`, and make sure that the pdf and jet libraries are available in the system and are linked with the correct path. The copy of the code present in the repository is the SVN current version, marked with its version number. From time to time, we will make new SVN versions available as we implement new processes ourselves, or following important code improvements.

JHEP06(2010)043

A byproduct of our work is the implementation of the NLO corrections for an arbitrary hadronic collision process, within the Frixione, Kunszt and Signer (FKS) subtraction scheme. The authors of ref. [32] have also proposed a general FKS implementation for e^+e^- processes, and they are currently extending it to hadronic collisions [39]. We stress that in our approach, however, the role of the NLO calculation is only meant to test the consistency of the implementation, and no effort was made to improve its efficiency.

Acknowledgments

The work of S.A. has been supported in part by the Deutsche Forschungsgemeinschaft in SFB/TR 9.

A Soft contributions

In this appendix we document the calculation of the soft contribution in the FKS subtraction framework. The real cross section in the soft approximation is given by

$$\mathcal{R} = 4\pi\alpha_{\mathrm{S}}\mu_{\mathrm{R}}^{2\epsilon}\left[\sum_{i\neq j}\mathcal{B}_{ij}\frac{k_i\cdot k_j}{(k_i\cdot k)(k_j\cdot k)} - \mathcal{B}\sum_i \frac{k_i^2}{(k_i\cdot k)^2}\,C_i\right] + \mathcal{R}^f, \tag{A.1}$$

where $\mathcal{B}_{ij}$ is the colour correlated Born cross section of eq. (2.6), and C_i is the Casimir invariant for the i^{th} leg. $\mathcal{R}^f$ has no singularities as the momentum of the radiated parton, k, goes to zero.

A.1 Soft phase space

The phase space in the soft limit always factorizes as

$$d\Phi^{n+1} = d\Phi^n \frac{d^{d-1}k}{2k_0(2\pi)^{d-1}}. \tag{A.2}$$

We write now

$$d^{d-1}k = dk_1\, dk_2\, d^{d-3}k_\perp = dk_1\, dk_2\, dk_\perp k_\perp^{-2\epsilon}\,\Omega^{1-2\epsilon}, \tag{A.3}$$

where we have set $d = 4-2\epsilon$, and Ω^α is the solid angle in α dimension

$$\Omega^\alpha = \frac{\alpha\,\pi^{\alpha/2}}{\Gamma(1+\alpha/2)} = \frac{\pi^{\alpha/2}2^\alpha\Gamma\left(\frac{\alpha+1}{2}\right)}{\sqrt{\pi}\Gamma(\alpha)} \quad\Longrightarrow\quad \Omega^{1-2\epsilon} = 2\frac{(4\pi)^{-\epsilon}\Gamma\left(1-\epsilon\right)}{\Gamma(1-2\epsilon)}. \tag{A.4}$$

Turning eq. (A.3) into polar coordinates we get

$$\frac{d^{d-1}k}{2k^0(2\pi)^{d-1}} = \frac{\pi^\epsilon\,\Gamma\left(1-\epsilon\right)}{\Gamma(1-2\epsilon)}\frac{1}{(2\pi)^3}k_0^{1-2\epsilon}(\sin\theta\sin\phi)^{-2\epsilon}dk_0\,d\cos\theta\,d\phi, \tag{A.5}$$

where we have defined

$$k_1 = k_0\cos\theta, \qquad k_2 = k_0\sin\theta\cos\phi, \qquad k_\perp = k_0\sin\theta\sin\phi. \tag{A.6}$$

JHEP06(2010)043

Since $k_\perp \geq 0$, this means that $0 \leq \phi \leq \pi$, and that only even quantities can be integrated in this way. In other words, $k_\perp$ should not be confused with k_3 (k_3 is no longer available at this stage). Inserting

$$k_0 = \xi \frac{\sqrt{s}}{2}, \tag{A.7}$$

this becomes

$$\frac{d^{d-1}k}{2k_0(2\pi)^{d-1}} = \left[\frac{(4\pi)^\epsilon \Gamma\left(1-\epsilon\right)}{\Gamma(1-2\epsilon)}\right] s^{-\epsilon} \frac{1}{(2\pi)^3} \frac{s}{4} \xi^{1-2\epsilon} (\sin\theta \sin\phi)^{-2\epsilon} d\xi \, d\cos\theta \, d\phi \,. \tag{A.8}$$

By writing the $\mathcal{R}$ term as $\xi^{-2}(\xi^2\mathcal{R})$, we notice that $(\xi^2\mathcal{R})$ has a finite limit as $\xi \to 0$. The ξ integration is performed by separating first

$$\xi^{-1-2\epsilon} = -\frac{\xi_c^{-2\epsilon}}{2\epsilon}\delta(\xi) + \left(\frac{1}{\xi}\right)_{\xi_c} - 2\epsilon \left(\frac{\log\xi}{\xi}\right)_{\xi_c}, \tag{A.9}$$

where the $\delta(\xi)$ term yields the soft contribution. We thus have that the integral of the soft-divergent part of $\mathcal{R}$ is given by

$$\begin{aligned}\mathcal{R}^s = & -\frac{1}{2\epsilon}\left[\frac{(4\pi)^\epsilon \Gamma\left(1-\epsilon\right)}{\Gamma(1-2\epsilon)}\right] s^{-\epsilon} \xi_c^{-2\epsilon} \frac{1}{(2\pi)^3} \int d\cos\theta \, d\phi (\sin\theta\sin\phi)^{-2\epsilon} \\ & \times \frac{s\xi^2}{4} 4\pi\alpha_{\rm S}\mu_{\rm R}^{2\epsilon} \left[\sum_{i\neq j} \mathcal{B}_{ij} \frac{k_i\cdot k_j}{(k_i\cdot k)(k_j\cdot k)} - \mathcal{B}\sum_i \frac{k_i^2}{(k_i\cdot k)^2} C_i\right],\end{aligned} \tag{A.10}$$

where $\mathcal{R}^s$ is now independent upon ξ (the dependence on ξ of k is canceled by the ξ^2 term in the numerator). Collecting the normalization factor of eq. (2.12) in front, we get

$$\begin{aligned}\mathcal{R}^s = & \mathcal{N}\left[1-\frac{\pi^2}{6}\epsilon^2\right]\left(\frac{Q^2}{s\xi_c^2}\right)^\epsilon \left(\frac{-1}{2\epsilon}\right) \frac{\alpha_{\rm S}}{2\pi} \int d\cos\theta \, \frac{d\phi}{\pi} (\sin\theta\sin\phi)^{-2\epsilon} \\ & \times \frac{s\xi^2}{4} \left[\sum_{i\neq j} \mathcal{B}_{ij} \frac{k_i\cdot k_j}{(k_i\cdot k)(k_j\cdot k)} - \mathcal{B}\sum_i \frac{k_i^2}{(k_i\cdot k)^2} C_i\right].\end{aligned} \tag{A.11}$$

We now define

$$\mathcal{I}_{ij} = \left[1-\frac{\pi^2}{6}\epsilon^2\right]\left(\frac{Q^2}{s\xi_c^2}\right)^\epsilon \left(-\frac{1}{2\epsilon}\right) \int d\cos\theta \, \frac{d\phi}{\pi} (\sin\theta\sin\phi)^{-2\epsilon} \frac{s\xi^2}{4} \frac{k_i\cdot k_j}{(k_i\cdot k)(k_j\cdot k)}, \tag{A.12}$$

$$\mathcal{I}_{i} = \left[1-\frac{\pi^2}{6}\epsilon^2\right]\left(\frac{Q^2}{s\xi_c^2}\right)^\epsilon \left(-\frac{1}{2\epsilon}\right) \int d\cos\theta \, \frac{d\phi}{\pi} (\sin\theta\sin\phi)^{-2\epsilon} \frac{s\xi^2}{4} \frac{C_i k_i^2}{(k_i\cdot k)^2}, \tag{A.13}$$

so that

$$R^s = \mathcal{N}\frac{\alpha_{\rm S}}{2\pi}\left[\sum_{i\neq j} \mathcal{I}_{ij}\mathcal{B}_{ij} - \mathcal{B}\sum_i \mathcal{I}_i\right]. \tag{A.14}$$

We introduce then our basic integral

$$I(p,q) = \int d\cos\theta \, \frac{d\phi}{\pi} (\sin\theta\sin\phi)^{-2\epsilon} \left[\frac{s\xi^2}{4}\frac{p\cdot q}{p\cdot k \, q\cdot k}\right] = \frac{1}{\epsilon} I_d(p,q) + I_0(p,q) + \epsilon I_\epsilon(p,q). \tag{A.15}$$

The expression of $I(p,q)$ will be substantially different for the case when both p and q are massless, when one is massive and one massless, and when both are massive.

A.2 One massless and one massive particle

Consider two momenta p and m, with $p^2 = 0$ and $m^2 \neq 0$. We want to evaluate $I(p, m)$. We separate out the collinear divergent component from the eikonal factor

$$\frac{p \cdot m}{p \cdot k \; m \cdot k} = \left[\frac{p \cdot m}{p \cdot k \; m \cdot k} - \frac{n \cdot p}{p \cdot k \; n \cdot k}\right] + \frac{p \cdot n}{p \cdot k \; n \cdot k}, \tag{A.16}$$

where the square bracket term has no collinear singularities. Assuming n along the time direction, we have:

$$\frac{s\xi^2}{4} \frac{n \cdot p}{p \cdot k \; n \cdot k} = \frac{1}{1 - \cos\theta}, \tag{A.17}$$

and

$$\int d\cos\theta \, \frac{d\phi}{\pi} (\sin\theta \sin\phi)^{-2\epsilon} \, \frac{1}{1 - \cos\theta} = -\frac{1}{\epsilon}, \tag{A.18}$$

so that

$$I_d(p, m) = -1. \tag{A.19}$$

The remaining integral has no collinear singularities so that we can write

$$\int d\cos\theta \, \frac{d\phi}{\pi} (\sin\theta \sin\phi)^{-2\epsilon} \, \frac{s\xi^2}{4} \left[\frac{p \cdot m}{p \cdot k \; m \cdot k} - \frac{n \cdot p}{p \cdot k \; n \cdot k}\right] = I_0(p, m) + \epsilon I_\epsilon(p, m)\,. \tag{A.20}$$

Defining

$$\hat{p} = \frac{p}{p_0}, \qquad \hat{m} = \frac{m}{m_0}, \qquad \beta = \frac{|\vec{m}|}{m_0}, \tag{A.21}$$

we have

$$I_d(p, m) = -1, \tag{A.22}$$

$$I_0(p, m) = \log \frac{(\hat{p} \cdot \hat{m})^2}{\hat{m}^2}, \tag{A.23}$$

$$I_\epsilon(p, m) = -2 \left[\frac{1}{4} \log^2 \frac{1 - \beta}{1 + \beta} + \log \frac{\hat{p} \cdot \hat{m}}{1 + \beta} \log \frac{\hat{p} \cdot \hat{m}}{1 - \beta} + \mathrm{Li}_2\left(1 - \frac{\hat{p} \cdot \hat{m}}{1 + \beta}\right) + \mathrm{Li}_2\left(1 - \frac{\hat{p} \cdot \hat{m}}{1 - \beta}\right)\right]. \tag{A.24}$$

Thus, eq. (A.12), in the case where k_1 is massless and k_2 is not, is given by

$$\begin{aligned} \mathcal{I}_{12} &= \left[1 - \frac{\pi^2}{6}\epsilon^2\right] \left(\frac{Q^2}{s\xi_c^2}\right)^\epsilon \left(-\frac{1}{2\epsilon}\right) \int d\cos\theta \, \frac{d\phi}{\pi} (\sin\theta \sin\phi)^{-2\epsilon} \left[\frac{s\xi^2}{4} \frac{k_1 \cdot k_2}{k_1 \cdot k \, k_2 \cdot k}\right] \\ &= \left[1 + \epsilon \log \frac{Q^2}{s\xi_c^2} + \left(\frac{1}{2} \log^2 \frac{Q^2}{s\xi_c^2} - \frac{\pi^2}{6}\right) \epsilon^2\right] \left(-\frac{1}{2\epsilon}\right) I(k_1, k_2) \\ &= \frac{A}{\epsilon^2} + \frac{B}{\epsilon} + C\,, \end{aligned} \tag{A.25}$$

where

$$A = \frac{1}{2}\,, \tag{A.26}$$

$$B = \frac{1}{2} \log \frac{Q^2}{s\xi_c^2} - \frac{1}{2} I_0(k_1, k_2)\,, \tag{A.27}$$

$$C = \frac{1}{2} \left[\log^2 \frac{Q^2}{s\xi_c^2} - \frac{\pi^2}{6}\right] - \frac{1}{2} I_0(k_1, k_2) \log \frac{Q^2}{s\xi_c^2} - \frac{1}{2} I_\epsilon(k_1, k_2)\,. \tag{A.28}$$

JHEP06(2010)043

A.3 Two massless particles

Using the identity

$$\frac{k_1\cdot k_2}{k_1\cdot k\, k_2\cdot k}=\frac{k_1\cdot(k_1+k_2)}{k_1\cdot k\,(k_1+k_2)\cdot k}+\frac{k_2\cdot(k_1+k_2)}{k_2\cdot k\,(k_1+k_2)\cdot k}, \tag{A.29}$$

that holds if $k_1^2=0$ and $k_2^2=0$, we can immediately obtain the expression of $I(k_1,k_2)$ for two massless momenta

$$I(k_1,k_2)=I(k_1,k_1+k_2)+I(k_2,k_1+k_2), \tag{A.30}$$

and use the results of the previous subsection for the two terms on the right hand side. We can write

$$\begin{aligned}\mathcal{I}_{12}&=\left[1-\frac{\pi^2}{6}\epsilon^2\right]\left(\frac{Q^2}{s\xi_c^2}\right)^{\epsilon}\left(-\frac{1}{2\epsilon}\right)\int d\cos\theta\,\frac{d\phi}{\pi}(\sin\theta\sin\phi)^{-2\epsilon}\left[\frac{s\xi^2}{4}\frac{k_1\cdot k_2}{k_1\cdot k\,k_2\cdot k}\right]\\&=\left[1+\epsilon\log\frac{Q^2}{s\xi_c^2}+\left(\frac{1}{2}\log^2\frac{Q^2}{s\xi_c^2}-\frac{\pi^2}{6}\right)\epsilon^2\right]\left(-\frac{1}{2\epsilon}\right)\left[I(k_1,k_1+k_2)+I(k_2,k_1+k_2)\right]\\&=\frac{A}{\epsilon^2}+\frac{B}{\epsilon}+C\end{aligned} \tag{A.31}$$

with

$$A=1 \tag{A.32}$$

$$B=\log\frac{Q^2}{s\xi_c^2}-\frac{1}{2}\left[I_0(k_1,k_1+k_2)+I_0(k_2,k_1+k_2)\right] \tag{A.33}$$

$$\begin{aligned}C=&\left[\frac{1}{2}\log^2\frac{Q^2}{s\xi_c^2}-\frac{\pi^2}{6}\right]-\frac{1}{2}\left[I_0(k_1,k_1+k_2)+I_0(k_2,k_1+k_2)\right]\log\frac{Q^2}{s\xi_c^2}\\&-\frac{1}{2}\left[I_\epsilon(k_1,k_1+k_2)+I_\epsilon(k_2,k_1+k_2)\right].\end{aligned} \tag{A.34}$$

A.4 Two massive particles

In case both k_1 and k_2 are massive, we define

$$I(k_1,k_2)=I_0(k_1,k_2)+\epsilon I_\varepsilon(k_1,k_2), \tag{A.35}$$

$$I_0(k_1,k_2)=\int d\cos\theta\,\frac{d\phi}{\pi}\left[\frac{s\xi^2}{4}\frac{k_1\cdot k_2}{k_1\cdot k\;k_2\cdot k}\right], \tag{A.36}$$

$$I_\epsilon(k_1,k_2)=-2\int d\cos\theta\,\frac{d\phi}{\pi}\log\left(\sin\theta\sin\phi\right)\left[\frac{s\xi^2}{4}\frac{k_1\cdot k_2}{k_1\cdot k\;k_2\cdot k}\right], \tag{A.37}$$

and get (neglecting now ϵ^2 terms)

$$\begin{aligned}\mathcal{I}_{12}&=\left(\frac{Q^2}{s\xi_c^2}\right)^{\epsilon}\left(-\frac{1}{2\epsilon}\right)\int d\cos\theta\,\frac{d\phi}{\pi}(\sin\theta\sin\phi)^{-2\epsilon}\left[\frac{s\xi^2}{4}\frac{k_1\cdot k_2}{k_1\cdot k\;k_2\cdot k}\right]\\&=\left[1+\epsilon\log\frac{Q^2}{s\xi_c^2}\right]\frac{-I(k_1,k_2)}{2\epsilon}=\frac{B}{\epsilon}+C\,,\end{aligned} \tag{A.38}$$

JHEP06(2010)043

with

$$B = -\frac{1}{2} I_0(k_1, k_2)\,, \tag{A.39}$$

$$C = -\frac{1}{2} I_0(k_1, k_2) \log \frac{Q^2}{s\xi_c^2} - \frac{1}{2} I_\epsilon(k_1, k_2)\,, \tag{A.40}$$

and

$$I_0(k_1, k_2) = \frac{1}{\beta} \log \frac{1+\beta}{1-\beta}\,, \qquad \beta = \sqrt{1 - \frac{k_1^2 k_2^2}{(k_1 \cdot k_2)^2}}\,. \tag{A.41}$$

The expression for I_ϵ is defined by the equations below

$$a = \beta_1^2 + \beta_2^2 - 2\,\vec{\beta}_1 \cdot \vec{\beta}_2, \tag{A.42}$$

$$x_1 = \frac{\beta_1^2 - \vec{\beta}_1 \cdot \vec{\beta}_2}{a}, \tag{A.43}$$

$$x_2 = \frac{\beta_2^2 - \vec{\beta}_1 \cdot \vec{\beta}_2}{a} = 1 - x_1,$$

$$b = \frac{\beta_1^2 \beta_2^2 - (\vec{\beta}_1 \cdot \vec{\beta}_2)^2}{a},$$

$$c = \sqrt{\frac{b}{4a}}, \tag{A.44}$$

$$z_+ = \frac{1 + \sqrt{1-b}}{\sqrt{b}}, \tag{A.45}$$

$$z_- = \frac{1 - \sqrt{1-b}}{\sqrt{b}}, \tag{A.46}$$

$$z_1 = \frac{\sqrt{x_1^2 + 4c^2} - x_1}{2c}, \tag{A.47}$$

$$z_2 = \frac{\sqrt{x_2^2 + 4c^2} + x_2}{2c}, \tag{A.48}$$

$$\begin{aligned} K(z) = &-\frac{1}{2} \log^2 \frac{(z - z_-)(z_+ - z)}{(z_+ + z)(z_- + z)} - 2\,\mathrm{Li}_2\left(\frac{2z_-(z_+ - z)}{(z_+ - z_-)(z_- + z)}\right) \\ &-2\,\mathrm{Li}_2\left(-\frac{2z_+(z_- + z)}{(z_+ - z_-)(z_+ - z)}\right), \end{aligned} \tag{A.49}$$

$$I_\epsilon(k_1, k_2) = [K(z_2) - K(z_1)]\ \frac{1 - \vec{\beta}_1 \cdot \vec{\beta}_2}{\sqrt{a(1-b)}}, \tag{A.50}$$

where

$$\vec{\beta}_1 = \frac{\vec{k}_1}{(k_1)_0}\,, \qquad \vec{\beta}_2 = \frac{\vec{k}_2}{(k_2)_0}\,. \tag{A.51}$$

The calculation of this integral is long and cumbersome. However, its correctness can be easily checked numerically, once the analytic answer has been obtained. Codes for checking the soft integrals are included in the `Notes` subdirectory of the `POWHEG BOX`.

JHEP06(2010)043

A.4.1 Two massive particles with equal momenta

In the particular case when $k_1 = k_2 = p$, $p^2 \neq 0$, we have

$$I_0(p,p) = 2, \qquad I_\epsilon(p,p) = \frac{2}{\beta}\log\frac{1+\beta}{1-\beta}, \qquad \beta = \frac{|\vec{p}|}{p_0}, \tag{A.52}$$

so that

$$\mathcal{R}^s_{12} = \frac{B}{\epsilon} + C, \tag{A.53}$$

with

$$B = -1\,, \tag{A.54}$$

$$C = -\log\frac{Q^2}{s\xi_c^2} - \frac{1}{\beta}\log\frac{1+\beta}{1-\beta}\,. \tag{A.55}$$

B Collinear limits

We list here the relationship between real- and Born-level squared amplitude, summed over the final-state and averaged over initial-state colours and spins, and divided by the appropriate flux factor.

B.1 Initial-state radiation

We call p the momentum of the incoming parton and k the momentum of the parton that enters the underlying Born process. Thus, $(p-k)$ is the momentum of the on-shell radiated parton, and $k^2 < 0$. We define

$$k^\mu = z\,p^\mu - \eta^\mu\,\frac{\left|\vec{k}_{\rm T}\right|^2}{2p\cdot\eta\,(1-z)} + k^\mu_{\rm T}\,, \tag{B.1}$$

and

$$\hat{k}^\mu_{\rm T} = \frac{k^\mu_{\rm T}}{\left|\vec{k}_{\rm T}\right|}\,. \tag{B.2}$$

We have

$$\begin{aligned}
\mathcal{R}^{(g)}(p) \;:\; & \mathcal{B}^{(g)}_{\mu\nu}(zp)\frac{8\pi\alpha_{\rm S}C_{\rm A}}{-k^2}\left\{-2\left[\frac{z}{1-z}+\frac{1-z}{z}+z(1-z)\right]g^{\mu\nu}\right. \\
& \left. \qquad + \frac{4(1-z)}{z}\left[\hat{k}^\mu_{\rm T}\hat{k}^\nu_{\rm T}+\frac{g^{\mu\nu}}{2}\right]\right\} \\
= \; & \mathcal{B}^{(g)}_{\mu\nu}(zp)\frac{8\pi\alpha_{\rm S}C_{\rm A}}{-k^2}\left\{-2\left[\frac{z}{1-z}+z(1-z)\right]g^{\mu\nu}+\frac{4(1-z)}{z}\hat{k}^\mu_{\rm T}\hat{k}^\nu_{\rm T}\right\}\,, \qquad \text{(B.3)}\\
\mathcal{R}^{(q)}(p) \;:\; & \mathcal{B}^{(q)}(zp)\frac{8\pi\alpha_{\rm S}C_{\rm F}}{-k^2}\frac{1+z^2}{1-z}\,, \qquad \text{(B.4)}\\
\mathcal{R}^{(q)}(p) \;:\; & \mathcal{B}^{(g)}_{\mu\nu}(zp)\frac{8\pi\alpha_{\rm S}C_{\rm F}}{-k^2}\left\{-g^{\mu\nu}\frac{1+(1-z)^2}{z}+\frac{4(1-z)}{z}\left[\hat{k}^\mu_{\rm T}\hat{k}^\nu_{\rm T}+\frac{g^{\mu\nu}}{2}\right]\right\} \\
= \; & \mathcal{B}^{(g)}_{\mu\nu}(zp)\frac{8\pi\alpha_{\rm S}C_{\rm F}}{-k^2}\left\{-g^{\mu\nu}z+\frac{4(1-z)}{z}\hat{k}^\mu_{\rm T}\hat{k}^\nu_{\rm T}\right\}\,, \qquad \text{(B.5)}\\
\mathcal{R}^{(g)}(p) \;:\; & \mathcal{B}^{(q)}(zp)\frac{8\pi\alpha_{\rm S}T_{\rm F}}{-k^2}\left(z^2+(1-z)^2\right)\,. \qquad \text{(B.6)}
\end{aligned}$$

JHEP06(2010)043

B.2 Final-state singularities

We call k the momentum of the splitting parton, and write

$$k^\mu = \frac{p^\mu}{z} + \eta^\mu \frac{z\left|\vec{k}_\mathrm{T}\right|^2}{2p\cdot\eta\,(1-z)} + k_\mathrm{T}^\mu\,. \tag{B.7}$$

We have

$$\begin{aligned}
\mathcal{R}^{(gg)}(p) \;:\; & \mathcal{B}^{(g)}_{\mu\nu}\left(\frac{p}{z}\right)\frac{8\pi\alpha_\mathrm{S}C_\mathrm{A}}{k^2}\left\{-2\left[\frac{z}{1-z}+\frac{1-z}{z}+z(1-z)\right]g^{\mu\nu}\right. \\
& \qquad \left. +4z(1-z)\left[\hat{k}_\mathrm{T}^\mu\hat{k}_\mathrm{T}^\nu+\frac{g^{\mu\nu}}{2}\right]\right\} \\
= \; & \mathcal{B}^{(g)}_{\mu\nu}\left(\frac{p}{z}\right)\frac{8\pi\alpha_\mathrm{S}C_\mathrm{A}}{k^2}\left\{-2\left[\frac{z}{1-z}+\frac{1-z}{z}\right]g^{\mu\nu}+4z(1-z)\hat{k}_\mathrm{T}^\mu\hat{k}_\mathrm{T}^\nu\right\}\,, \qquad \text{(B.8)}\\
\mathcal{R}^{(qg)}(p) \;:\; & \mathcal{B}^{(q)}\left(\frac{p}{z}\right)\frac{8\pi\alpha_\mathrm{S}C_\mathrm{F}}{k^2}\frac{1+(1-z)^2}{z}\,, \qquad \text{(B.9)}\\
\mathcal{R}^{(q\bar{q})}(p) \;:\; & \mathcal{B}^{(g)}_{\mu\nu}\left(\frac{p}{z}\right)\frac{8\pi\alpha_\mathrm{S}T_\mathrm{F}}{k^2}\left\{-g^{\mu\nu}\left(z^2+(1-z)^2\right)-4z(1-z)\left[\hat{k}_\mathrm{T}^\mu\hat{k}_\mathrm{T}^\nu+\frac{g^{\mu\nu}}{2}\right]\right\} \\
= \; & \mathcal{B}^{(g)}_{\mu\nu}\left(\frac{p}{z}\right)\frac{8\pi\alpha_\mathrm{S}T_\mathrm{F}}{k^2}\left\{-g^{\mu\nu}-4z(1-z)\hat{k}_\mathrm{T}^\mu\hat{k}_\mathrm{T}^\nu\right\}\,. \qquad \text{(B.10)}
\end{aligned}$$

In eq. (B.9), it is assumed that p is the momentum of the gluon.

C Upper bounding functions for FSR

We use an upper bounding function of the following form (for ease of notation we write α_S instead of $\alpha_\mathrm{S}^\mathrm{rad}$)

$$U(\xi,y) = N\frac{\alpha_\mathrm{S}(k_\mathrm{T}^2)}{\xi(1-y)}\,, \tag{C.1}$$

with

$$k_\mathrm{T}^2 = \frac{s}{2}\,\xi^2(1-y)\,, \tag{C.2}$$

s being the partonic CM energy squared, and

$$\alpha_\mathrm{S}\left(k_\mathrm{T}^2\right) = \frac{1}{b_0\log\left(k_\mathrm{T}^2/\Lambda^2\right)}\,. \tag{C.3}$$

The ranges of ξ, y and ϕ are given by

$$0\le\xi\le\xi_\mathrm{max}\equiv\frac{s-M_\mathrm{rec}^2}{s}, \qquad -1\le y\le 1, \qquad 0\le\phi<2\pi\,, \tag{C.4}$$

where M_rec is the mass of the system recoiling against the emitter and emitted partons (see eq. (5.49) or ref. [2]). We want to generate p_T uniformly in

$$\Delta^{(U)}(p_\mathrm{T}) = \exp\left[-\int U(\xi,y)\,\theta(k_\mathrm{T}-p_\mathrm{T})\;d\xi\,dy\,d\phi\right]. \tag{C.5}$$

JHEP06(2010)043

Trading y for k_{T}^2 (see eq. (C.2)), we get

$$
\begin{aligned}
-\log\Delta^{(U)}(p_{\mathrm{T}}) &= 2\pi N\int_0^{\xi_{\max}}\frac{d\xi}{\xi}\int_{p_{\mathrm{T}}^2}^{\xi^2 s}\frac{dk_{\mathrm{T}}^2}{k_{\mathrm{T}}^2}\,\alpha_{\mathrm{S}}(k_{\mathrm{T}}^2)\\
&= \frac{\pi N}{b_0}\,\theta\left(\xi_{\max}^2-\frac{p_{\mathrm{T}}^2}{s}\right)\left[\log\frac{\xi_{\max}^2 s}{\Lambda^2}\log\frac{\log\left(\xi_{\max}^2 s/\Lambda^2\right)}{\log\left(p_{\mathrm{T}}^2/\Lambda^2\right)}-\log\frac{\xi_{\max}^2 s}{p_{\mathrm{T}}^2}\right]. \qquad \text{(C.6)}
\end{aligned}
$$

The equation $r=\Delta^{(U)}(p_{\mathrm{T}})$ is translated into $\log r=\log\Delta^{(U)}(p_{\mathrm{T}})$, and solved numerically for p_{T}. Once p_{T} is generated, we generate ξ uniformly in $\log\xi$ (see second member of eq. (C.6)) within the limits

$$
\frac{p_{\mathrm{T}}^2}{s}\leqslant\xi\leqslant\xi_{\max}\,, \qquad \text{(C.7)}
$$

and then use

$$
p_{\mathrm{T}}^2=\frac{s}{2}\,\xi^2(1-y) \qquad \text{(C.8)}
$$

to obtain y, while ϕ is generated uniformly between 0 and 2π.

More options are offered in the POWHEG BOX. They are both related to the use of the upper bounding function

$$
U(\xi,y)=N\frac{\alpha_{\mathrm{S}}(k_{\mathrm{T}}^2)}{\xi^2(1-y)\left(1-\frac{\xi}{2}(1-y)\right)^2}\,, \qquad \text{(C.9)}
$$

which yields

$$
\begin{aligned}
-\log\Delta^{(U)}(p_{\mathrm{T}}) &= 2\pi N\int_{p_{\mathrm{T}}^2}^{s\xi_{\max}^2}\frac{dk_{\mathrm{T}}^2}{k_{\mathrm{T}}^2}\int_{\sqrt{k_{\mathrm{T}}^2/s}}^{\xi_{\max}}\frac{d\xi}{(\xi-k_{\mathrm{T}}^2/s)^2}\\
&= 4\pi N\left[\frac{1}{2\xi_{\max}}\left(\log\frac{1-\xi_{\max}}{\xi_{\max}}(1-2\xi_{\max})-2\right)\right.\\
&\quad\left.-\frac{1}{2p\,\xi_{\max}}\left(p\log\left(\xi_{\max}-p^2\right)-2\left(p\log\frac{1-p}{p}+1\right)\xi_{\max}-2p\log p\right)\right], \qquad \text{(C.10)}
\end{aligned}
$$

where $p=\sqrt{p_{\mathrm{T}}^2/s}$. The second option implemented in the POWHEG BOX makes use of the upper bounding function in eq. (C.9) multiplied by a factor of ξ. One then uses the same expression (C.10) for the generation of p_{T}, and uses a further veto, accepting the event if a given random number is less than ξ, to implement the extra factor of ξ. The third option is similar to the second, but with an extra factor of $1-\xi(1-y)/2$.

D Upper bounding functions for ISR

The upper bounding function is (where $x=1-\xi$, so that the singular limit is reached when $x\to 1$)

$$
U(x,y)=N\frac{\alpha_{\mathrm{S}}(k_{\mathrm{T}}^2)}{(1-x)(1-y^2)}\,, \qquad \text{(D.1)}
$$

with

$$
k_{\mathrm{T}}^2=\frac{s_b}{4x}\,(1-x)^2(1-y^2)\,, \qquad \text{(D.2)}
$$

JHEP06(2010)043

s_b being the underlying Born CM energy squared. The range of $U(x,y)$ must cover the range of the radiation variables for the given underlying Born configuration. A practical restriction for the range of $U(x,y)$ is

$$\rho \leqslant x \leqslant 1\,, \qquad k_{\mathrm{T}}^2 \leqslant k_{\mathrm{T}\,\mathrm{max}}^2\,, \tag{D.3}$$

where

$$\rho = \frac{s_b}{S}\,, \qquad k_{\mathrm{T}\,\mathrm{max}}^2 = s_b \frac{\left(1-\bar{x}_\oplus^2\right)\left(1-\bar{x}_\ominus^2\right)}{(\bar{x}_\oplus + \bar{x}_\ominus)^2}\,. \tag{D.4}$$

We want to generate p_{T} uniformly in

$$\Delta^{(U)}(p_{\mathrm{T}}) = \exp\left[-\int U(x,y)\,\theta(k_{\mathrm{T}}-p_{\mathrm{T}})\,dx\,dy\,d\phi\right], \tag{D.5}$$

in the given range. We assume $0 \leqslant \phi \leqslant 2\pi$. Trading y for k_{T}^2 we find

$$|y| = \sqrt{1 - \frac{4x}{(1-x)^2}\frac{k_{\mathrm{T}}^2}{s_b}}\,, \tag{D.6}$$

and

$$\int dx \int dy \int_0^{2\pi} d\phi\, U(x,y)\,\theta(k_{\mathrm{T}}-p_{\mathrm{T}}) = 2\pi N \int_\rho^{x_-} dx \int_{p_{\mathrm{T}}^2}^{k_{T\,\mathrm{max}}^2} \frac{dk_{\mathrm{T}}^2}{k_{\mathrm{T}}^2} \frac{\alpha_{\mathrm{S}}\left(k_{\mathrm{T}}^2\right)}{\sqrt{(x_+-x)(x_--x)}}\,, \tag{D.7}$$

where

$$x_\pm = \left(\sqrt{1+\frac{k_{\mathrm{T}}^2}{s_b}} \pm \frac{k_{\mathrm{T}}}{s_b}\right)^2. \tag{D.8}$$

The x integration can be performed to yield

$$\int dx \int dy \int_0^{2\pi} d\phi\, U(x,y)\,\theta(k_{\mathrm{T}}-p_{\mathrm{T}}) = \int_{p_{\mathrm{T}}^2}^{k_{T\,\mathrm{max}}^2} \frac{dk_{\mathrm{T}}^2}{k_{\mathrm{T}}^2}\, V(k_{\mathrm{T}}^2)\,, \tag{D.9}$$

where

$$V(k_{\mathrm{T}}^2) = 2\pi N \alpha_{\mathrm{S}}(k_{\mathrm{T}}^2) \log \frac{\sqrt{x_+-\rho}+\sqrt{x_--\rho}}{\sqrt{x_+-\rho}-\sqrt{x_--\rho}}\,. \tag{D.10}$$

We observe that

$$\log \frac{\sqrt{x_+-\rho}+\sqrt{x_--\rho}}{\sqrt{x_+-\rho}-\sqrt{x_--\rho}} \leqslant \log \frac{\sqrt{x_+}+\sqrt{x_-}}{\sqrt{x_+}-\sqrt{x_-}} = \frac{1}{2}\log\frac{k_{\mathrm{T}}^2+s_b}{k_{\mathrm{T}}^2}\,. \tag{D.11}$$

In ref. [3], it is suggested to use the bound

$$\frac{1}{2}\log\frac{k_{\mathrm{T}}^2+s_b}{k_{\mathrm{T}}^2} \leq \frac{1}{2}\log\frac{q^2}{k_{\mathrm{T}}^2}\,, \qquad \text{with } \; q^2 = k_{\mathrm{T}\,\mathrm{max}}^2 + s_b\,, \tag{D.12}$$

and to define

$$\tilde{V}(k_{\mathrm{T}}^2) = 2\pi N \alpha_{\mathrm{S}}(k_{\mathrm{T}}^2)\frac{1}{2}\log\frac{q^2}{k_{\mathrm{T}}^2} \geqslant V(k_{\mathrm{T}}^2)\,. \tag{D.13}$$

JHEP06(2010)043

The $dk_{\rm T}^2$ integral of $\tilde{V}$ can be performed analytically, yielding

$$\int_{p_{\rm T}^2}^{k_{T\,\max}^2} \frac{dk_{\rm T}^2}{k_{\rm T}^2}\,\tilde{V}(k_{\rm T}^2) = \frac{\pi N}{b_0}\left[\log\frac{q^2}{\Lambda^2}\log\frac{\log\left(k_{\rm T\,max}^2/\Lambda^2\right)}{\log\left(p_{\rm T}^2/\Lambda^2\right)} - \log\frac{k_{\rm T\,max}^2}{p_{\rm T}^2}\right]. \tag{D.14}$$

One generates $p_{\rm T}$ uniformly in

$$\tilde{\Delta}(p_{\rm T}) = \exp\left[-\int_{p_{\rm T}^2}^{k_{T\,\max}^2}\frac{dk_{\rm T}^2}{k_{\rm T}^2}\,\tilde{V}(k_{\rm T}^2)\right], \tag{D.15}$$

and then use the veto method to get the $p_{\rm T}$ distributed according to $\Delta^{(U)}(p_{\rm T})$.

The following variant of this procedure has been introduced in ref. [4]. We have used the bound

$$\log\frac{k_{\rm T}^2+s_b}{k_{\rm T}^2} \leqslant \begin{cases} \log\frac{2s_b}{k_{\rm T}^2} & \text{for } k_{\rm T}^2 < s_b \\ \log 2 & \text{for } k_{\rm T}^2 > s_b \end{cases} \tag{D.16}$$

so

$$\tilde{V}(k_{\rm T}^2) = \pi N\alpha_{\rm S}(k_{\rm T}^2)\left[\theta(s_b-k_{\rm T}^2)\log\frac{2s_b}{k_{\rm T}^2} + \theta(k_{\rm T}^2-s_b)\log 2\right]. \tag{D.17}$$

We then get

$$\begin{aligned}\log\tilde{\Delta}(p_{\rm T}) \;=\; & \theta\big(s_b-p_{\rm T}^2\big)\,\frac{\pi N}{b_0}\left\{\theta\left(k_{\rm T\,max}^2-s_b\right)\left[\log\frac{2s_b}{\Lambda^2}\log\frac{\log\left(s_b/\Lambda^2\right)}{\log\left(p_{\rm T}^2/\Lambda^2\right)} - \log\frac{s_b}{p_{\rm T}^2}\right.\right. \\ & \left.\left. +\;\log(2)\,\log\frac{\log\left(k_{\rm T\,max}^2/\Lambda^2\right)}{\log\left(s_b/\Lambda^2\right)}\right] \right. \\ & \left. +\,\theta\big(s_b-k_{\rm T\,max}^2\big)\left[\log\frac{2s_b}{\Lambda^2}\;\log\frac{\log\left(k_{\rm T\,max}^2/\Lambda^2\right)}{\log\left(p_{\rm T}^2/\Lambda^2\right)} - \log\frac{k_{T\,\max}^2}{p_{\rm T}^2}\right]\right\} \\ & +\theta\big(p_{\rm T}^2-s_b\big)\,\frac{\pi N}{b_0}\log(2)\,\log\frac{\log\left(k_{\rm T\,max}^2/\Lambda^2\right)}{\log\left(p_{\rm T}^2/\Lambda^2\right)}\,.\end{aligned} \tag{D.18}$$

To improve the behaviour for small x effects it may be convenient to use instead

$$U(x,y) = N\frac{\alpha_{\rm S}(k_{\rm T}^2)}{x(1-x)(1-y^2)} \tag{D.19}$$

as upper bounding function. As derived in ref. [3]

$$\int U(x,y)\,\theta(k_{\rm T}-p_{\rm T})\;d\Phi_r = \int_{p_{\rm T}^2}^{k_{\rm T\,max}^2}\frac{dk_{\rm T}^2}{k_{\rm T}^2}\,V\left(k_{\rm T}^2\right), \tag{D.20}$$

where

$$V\left(k_{\rm T}^2\right) = \pi N\alpha_{\rm S}(k_{\rm T}^2)\log\frac{\sqrt{x_+-\rho}+\sqrt{x_--\rho}}{\sqrt{x_+-\rho}-\sqrt{x_--\rho}}. \tag{D.21}$$

With the new upper bounding function one gets instead

$$V(k_{\rm T}^2) = \pi N\alpha_{\rm S}(k_{\rm T}^2)\left[\log\frac{2}{\rho}+\log\frac{\sqrt{(x_+-\rho)(x_--\rho)}+1-\frac{\rho}{2}(x_++x_-)}{x_+-x_-}\right]. \tag{D.22}$$

JHEP06(2010)043

In this case too $V(k_{\mathrm{T}}^2)$ satisfies a simple upper bound

$$V(k_{\mathrm{T}}^2) < \pi N \alpha_{\mathrm{S}}(k_{\mathrm{T}}^2) \log \frac{S}{k_{\mathrm{T}}^2}\,, \tag{D.23}$$

that can be used for fast generation by vetoing.

E Choice of scales

E.1 Scales and couplings for the inclusive cross section

In the evaluation of $\tilde{B}$ and of the remnant function, a user provided, process dependent subroutine `set_fac_ren_scales(muf,mur)` sets the factorization and renormalization scales. It should only depend upon the underlying Born kinematics (thus, for example, it cannot depend upon the radiation transverse momentum). It is called by the `POWHEG BOX` subroutine `setscalesbtilde` (called during the evaluation of the $\tilde{B}$ function and of the remnant cross section) that stores the square of the factorization scale, the square of the renormalization scale and the strong coupling constant in three global variables of the `st_` common block: `st_muren2`, `st_mufact2` and `st_alpha`. By inspecting the subroutine we see that the factorization and renormalization scales are set equal to the square of the scales returned by the `set_fac_ren_scales` subroutine multiplied by renormalization and factorization scale factors `st_facfact` and `st_renfact`. These factors are in turn read from the variables `facscfact` and `renscfact` in the `POWHEG` data file, normally during the execution of the user initialization routine `init_phys`. If they are not present in the data file, they are set by default to 1. The function `pwhg_alphas(mu2,Lambda5,n)` returns the strong coupling constant with `n` light flavours, as a function of the square of the scale and of $\Lambda^{(5)}_{\overline{\mathrm{MS}}}$. If called with `n` < 0 it uses, as number of flavours, the number of quarks with mass less than the input scale. In the evaluation of the $\tilde{B}$ function, the number of flavours `st_nlight` is used. This is set by the user in the `init_couplings` routine.

E.2 Scales and couplings for radiation

The choice of scales and couplings in the generation of radiation requires particular attention, since the shape of the Sudakov peak, in the radiation transverse momentum, is deeply affected by it. It is discussed in detail in ref. [2]. Here we report how this is actually done in the `POWHEG BOX`. The relevant code is in the subroutine `set_rad_scales(ptsq)`. It is typically invoked with the radiation transverse momentum as argument. With this choice one can achieve, in some cases, complete next-to-leading logarithmic (NLL) accuracy in the `POWHEG` Sudakov form factor [2, 3]. By inspecting the code, we can see that it sets `st_mufact2` and `st_muren2` to `ptsq`. The factorization and renormalization scale factors, `st_facfact` and `st_renfact`, are not used in this context. The program also takes care that the factorization scale never goes below the minimum allowed value in the pdf's. The strong coupling is then evaluated, the number of flavours is taken as the number of quarks with mass below the `ptsq` scale. Furthermore, the strong coupling constant is multiplied by the factor given in formula (4.32) in ref. [2]. This factor improves the NLL accuracy of the Sudakov form factor.

JHEP06(2010)043

During the generation of radiation, we need a simplified one loop expression for the running coupling that is an upper bound of the running coupling that we use. We choose

$$\alpha_{\rm S}^{\rm rad} = \frac{1}{b_0^{(5)} \log \frac{\mu^2}{\Lambda_{\rm ll}^2}}, \qquad b_0^{(5)} = \frac{33 - 2 \times 5}{12\pi} , \tag{E.1}$$

and we fix $\Lambda_{\rm ll}$ by requiring

$$\alpha_{\rm S}^{\rm rad}(\mu^0) = \alpha_{\rm S}^{\rm PW}(\mu^0) , \tag{E.2}$$

where the scale μ_0 is the minimum allowed value for the renormalization scale, and is taken equal to $2\Lambda^{(5)}_{\overline{\rm MS}}$. The value of $\Lambda_{\rm ll}$ is stored in the global variable `rad_lamll`.

F Miscellaneous features of the code

F.1 Checking the soft, collinear and soft-collinear limits

In the `POWHEG BOX`, the routine `checklims`, through a call to the subroutines `checksoft` and `checkcoll`, allows the user to check the limiting behaviours of the real squared amplitudes, against their soft and collinear approximations. This routine has been used as a debug feature in the developing stages of the implementation of specific processes. If activated by the flags `dbg_softtest` and `dbg_colltest` in the `init_phys.f` file, it can be used now to check if the Born, the colour-correlated and spin-correlated Born amplitudes (used to build the limiting expressions of the real amplitudes) are consistent with the real contributions, computed in the kinematic configurations where a gluon becomes soft or when it becomes collinear to another parton or when two quarks of opposite flavours become collinear.

The double soft-collinear and collinear-soft limits are also tested. They do not depend upon the real squared amplitudes, but only upon the Born amplitudes. They have been used initially to check the consistency of the `POWHEG BOX` code, but they can also be used to perform some checks of the colour correlated Born amplitudes during the development of new processes.

F.2 Names

In the present work, we have made little references to the names of the fortran files where the various subroutines are stored. We assume that the reader can find them out by using standard command-line tools. However, while we have not spent much energy in deciding how to organize fortran files, the include files are rather well organized. For example, the include file `pwhg_flst.h` declares common-block variables that refer to flavour structures. All these variables start with a prefix `flst_`. Other important global variables are the kinematics ones (`kn_`), those involving the scales and the strong coupling (`st_`), those involving the generation of radiation (`rad_`), and so on. In this way, when finding such variables in the code, by inspecting the include files, the reader can better follow what is their purpose.

JHEP06(2010)043

F.3 Input variables

The POWHEG BOX gets its input data from the routine powheginput. Upon its first invocation, this routine looks for a file named powheg.input. If it does not find it, asks interactively to enter a prefix, and then looks for the file "prefix"-powheg.input. It reads all the input file at once. Then, if invoked in the form powheginput("string") it returns the (real) value associated to "string" in the input file. If no matching string is found, it prints a message and aborts the program. If invoked in the form "#string", in case no matching string is found, it returns a very unlikely value -10^6. This last mechanism is used in the POWHEG BOX to set default values.

F.4 User files

The files that contain the user routines are organized in the same way in all the examples provided with the code of the POWHEG BOX. They are:

- nlegborn.h contains the number of legs of the Born process, nlegborn;
- init_processes.f contains the subroutine init_processes, whose major task is to fill the list flst_born and flst_real;
- init_couplings.f contains the subroutine init_couplings, that initializes process-dependent couplings;
- in PhysPars.h, there is a collection of physical variables that are in common with many subroutines (masses, electroweak couplings, widths...);
- Born.f contains the setborn subroutine;
- in Born_phsp.f the user can found the born_phsp subroutine;
- in real.f, the setreal routine;
- and finally, in virtual.f, the setvirtual routine.

A template for the analysis subroutine can be found in pwhg_analysis.f.

References

[1] P. Nason, *A new method for combining NLO QCD with shower Monte Carlo algorithms*, *JHEP* **11** (2004) 040 [hep-ph/0409146] [SPIRES].

[2] S. Frixione, P. Nason and C. Oleari, *Matching NLO QCD computations with Parton Shower simulations: the POWHEG method*, *JHEP* **11** (2007) 070 [arXiv:0709.2092] [SPIRES].

[3] P. Nason and G. Ridolfi, *A positive-weight next-to-leading-order Monte Carlo for Z pair hadroproduction*, *JHEP* **08** (2006) 077 [hep-ph/0606275] [SPIRES].

JHEP06(2010)043

[4] S. Frixione, P. Nason and G. Ridolfi, *A Positive-Weight Next-to-Leading-Order Monte Carlo for Heavy Flavour Hadroproduction*, *JHEP* **09** (2007) 126 [arXiv:0707.3088] [SPIRES].

[5] O. Latunde-Dada, S. Gieseke and B. Webber, *A positive-weight next-to-leading-order Monte Carlo for e^+e^- annihilation to hadrons*, *JHEP* **02** (2007) 051 [hep-ph/0612281] [SPIRES].

[6] O. Latunde-Dada, *Applying the POWHEG method to top pair production and decays at the ILC*, *Eur. Phys. J.* **C 58** (2008) 543 [arXiv:0806.4560] [SPIRES].

[7] S. Alioli, P. Nason, C. Oleari and E. Re, *NLO vector-boson production matched with shower in POWHEG*, *JHEP* **07** (2008) 060 [arXiv:0805.4802] [SPIRES].

[8] K. Hamilton, P. Richardson and J. Tully, *A Positive-Weight Next-to-Leading Order Monte Carlo Simulation of Drell-Yan Vector Boson Production*, *JHEP* **10** (2008) 015 [arXiv:0806.0290] [SPIRES].

[9] A. Papaefstathiou and O. Latunde-Dada, *NLO production of W' bosons at hadron colliders using the MC@NLO and POWHEG methods*, *JHEP* **07** (2009) 044 [arXiv:0901.3685] [SPIRES].

[10] S. Alioli, P. Nason, C. Oleari and E. Re, *NLO Higgs boson production via gluon fusion matched with shower in POWHEG*, *JHEP* **04** (2009) 002 [arXiv:0812.0578] [SPIRES].

[11] K. Hamilton, P. Richardson and J. Tully, *A Positive-Weight Next-to-Leading Order Monte Carlo Simulation for Higgs Boson Production*, *JHEP* **04** (2009) 116 [arXiv:0903.4345] [SPIRES].

[12] S. Alioli, P. Nason, C. Oleari and E. Re, *NLO single-top production matched with shower in POWHEG: s- and t-channel contributions*, *JHEP* **09** (2009) 111 [arXiv:0907.4076] [SPIRES].

[13] S. Alioli, P. Nason, C. Oleari and E. Re, *NLO $Z+1$ jet production matched with shower in POWHEG*, to appear soon.

[14] P. Nason and C. Oleari, *NLO Higgs boson production via vector-boson fusion matched with shower in POWHEG*, *JHEP* **02** (2010) 037 [arXiv:0911.5299] [SPIRES].

[15] S. Frixione and B.R. Webber, *Matching NLO QCD computations and parton shower simulations*, *JHEP* **06** (2002) 029 [hep-ph/0204244] [SPIRES].

[16] G. Corcella et al., *HERWIG 6.5: an event generator for Hadron Emission Reactions With Interfering Gluons (including supersymmetric processes)*, *JHEP* **01** (2001) 010 [hep-ph/0011363] [SPIRES].

[17] G. Corcella et al., *HERWIG 6.5 release note*, hep-ph/0210213 [SPIRES].

[18] T. Sjöstrand, S. Mrenna and P.Z. Skands, *PYTHIA 6.4 Physics and Manual*, *JHEP* **05** (2006) 026 [hep-ph/0603175] [SPIRES].

[19] R. Frederix, T. Gehrmann and N. Greiner, *Automation of the Dipole Subtraction Method in MadGraph/MadEvent*, *JHEP* **09** (2008) 122 [arXiv:0808.2128] [SPIRES].

[20] K. Hasegawa, S. Moch and P. Uwer, *AutoDipole – Automated generation of dipole subtraction terms*, arXiv:0911.4371 [SPIRES].

[21] T. Binoth et al., *A proposal for a standard interface between Monte Carlo tools and one-loop programs*, arXiv:1001.1307 [SPIRES].

JHEP06(2010)043

[22] T. Hahn and M. Pérez-Victoria, *Automatized one-loop calculations in four and D dimensions*, *Comput. Phys. Commun.* **118** (1999) 153 [hep-ph/9807565] [SPIRES].

[23] Y. Kurihara et al., *QCD event generators with next-to-leading order matrix-elements and parton showers*, *Nucl. Phys.* **B 654** (2003) 301 [hep-ph/0212216] [SPIRES].

[24] G. Bélanger et al., *Automatic calculations in high energy physics and Grace at one-loop*, *Phys. Rept.* **430** (2006) 117 [hep-ph/0308080] [SPIRES].

[25] R.K. Ellis, W.T. Giele and Z. Kunszt, *A Numerical Unitarity Formalism for Evaluating One-Loop Amplitudes*, *JHEP* **03** (2008) 003 [arXiv:0708.2398] [SPIRES].

[26] G. Ossola, C.G. Papadopoulos and R. Pittau, *CutTools: a program implementing the OPP reduction method to compute one-loop amplitudes*, *JHEP* **03** (2008) 042 [arXiv:0711.3596] [SPIRES].

[27] T. Binoth, J.P. Guillet, G. Heinrich, E. Pilon and T. Reiter, *Golem95: a numerical program to calculate one-loop tensor integrals with up to six external legs*, *Comput. Phys. Commun.* **180** (2009) 2317 [arXiv:0810.0992] [SPIRES].

[28] C.F. Berger et al., *An Automated Implementation of On-Shell Methods for One- Loop Amplitudes*, *Phys. Rev.* **D 78** (2008) 036003 [arXiv:0803.4180] [SPIRES].

[29] A. Lazopoulos, *Multi-gluon one-loop amplitudes numerically*, arXiv:0812.2998 [SPIRES].

[30] J.-C. Winter and W.T. Giele, *Calculating gluon one-loop amplitudes numerically*, arXiv:0902.0094 [SPIRES].

[31] Particle Data Group collaboration, C. Amsler et al., *Review of particle physics*, *Phys. Lett.* **B 667** (2008) 1 [SPIRES].

[32] R. Frederix, S. Frixione, F. Maltoni and T. Stelzer, *Automation of next-to-leading order computations in QCD: the FKS subtraction*, *JHEP* **10** (2009) 003 [arXiv:0908.4272] [SPIRES].

[33] Z. Kunszt, A. Signer and Z. Trócsányi, *One loop helicity amplitudes for all $2 \to 2$ processes in QCD and $N = 1$ supersymmetric Yang-Mills theory*, *Nucl. Phys.* **B 411** (1994) 397 [hep-ph/9305239] [SPIRES].

[34] S. Catani and M.H. Seymour, *A general algorithm for calculating jet cross sections in NLO QCD*, *Nucl. Phys.* **B 485** (1997) 291 [hep-ph/9605323] [SPIRES].

[35] S. Frixione, Z. Kunszt and A. Signer, *Three jet cross-sections to next-to-leading order*, *Nucl. Phys.* **B 467** (1996) 399 [hep-ph/9512328] [SPIRES].

[36] S. Frixione, *A General approach to jet cross-sections in QCD*, *Nucl. Phys.* **B 507** (1997) 295 [hep-ph/9706545] [SPIRES].

[37] P. Nason, *MINT: a Computer Program for Adaptive Monte Carlo Integration and Generation of Unweighted Distributions*, arXiv:0709.2085 [SPIRES].

[38] E. Boos et al., *Generic user process interface for event generators*, hep-ph/0109068 [SPIRES].

[39] R. Frederix, private communication.

JHEP06(2010)043