Prepared for submission to JHEP

Changes to the POWHEG Box

Daniel Quill

E-mail: daniel.quill.14@ucl.ac.uk

Abstract: Here is a list of the changes that I have had to make to the POWHEG box to get it running, and to implement the FNO NLL resummation

1 Getting the POWHEG box up and running with both PYTHIA 6 and 8

This guide assumes we have just downloaded a fresh copy of the POWHEG box code, and the *hvq* process directory. In *hvq/testrun-tdec-lhc* run

⇒ ln -s ../../pdfdata/cteq6m.tbl cteq6m

to give the program a PDF file.

1.1 Changes to the Makefile

1. In *hvq/Makefile*, change the PDF to native (I never managed to get LHAPDF installed, although I didn't try very hard)
2. Change the analysis variable in line 9 from *dummy* to *new* - I have been using an analysis file called *pwhg_analysis-new.f*
3. Add in the lines

   ⇒ ifeq ("$(ANALYSIS)","new")

   ⇒ FASTJET_CONFIG=$(shell which fastjet-config)

   ⇒ LIBSFASTJET += $(shell $(FASTJET_CONFIG) --libs --plugins --rpath=no) -lstdc++ -lc++

   ⇒ FJCXXFLAGS+= $(shell $(FASTJET_CONFIG) --cxxflags)

   ⇒ PWHGANAL=pwhg_analysis-new.o fastjetfortran.o

   to the analysis IF statement (around line 86 in the Makefile), most of it is already in the comments for the B or D analysis, except the --rpath=no and -lc++
4. Change all instances of *pwhg_bookhist.o* to *pwhg_bookhist-multi.o* - this is the newer way of doing histograms
5. At line 16, replace line

   ⇒ F77= gfortran -fno-automatic

   with

   ⇒ F77= gfortran -fno-automatic -ffixed-line-length-132

   This allows us to have longer lines in the code (which was needed because I pulled the analysis file together from different sources, some of which had long lines)

This is all we need to change in the Makefile for now.

1.2 Running the main program

Now, if we take a starter analysis file (I've taken the one from *ttb_NLO_dec/pwhg_analysis-top.f* and copied it to *hvq/pwhg_analysis-new.f*), we can compile and run the *pwhg_main* program (it needs an analysis file called *new* to run), so long as we have changed the *powheg.input* file to look for the native PDFs instead of LHAPDF (lines 12-15). Next we need to look at the analysis file.

1.3 Changes to *pwhg_analysis-new.f*

I have pulled together the analysis file from a combination of different sources, but the base is the file *ttb_NLO_dec/pwhg_analysis-top.f*

1. First I deleted all of the histograms initialised in *init_hist* at the beginning of *pwhg_analysis-new.f* and initialised my own ones, as there was a lot of nonsense there I didn't care about
2. Now on to changing the analysis. In the subroutine *analysis(dsig0)*, there is a flag called *IsForClustering(jhep)* introduced around line 201. This should be be "true" for final state particles excluding fermions, however in the existing code, this picks up anything that is final state, and has a particle code < 11 or > 16. The problem with this is that the code < 11 condition also picks up *any* final state antiparticles (as antiparticles have the same code as their corresponding particle, with a minus sign, and so will all be less than 0). Also, the code>16 condition misses out all the "correct" antiparticles (*i.e.* final state anti-quarks). To fix this, I have changed the condition to be

   ⇒ $\big($abs(particle code) $< 11\big)$ .or. $\big($abs(particle code) $> 16\big)$

3. Next, I deleted all of the code filling the histograms that I had deleted in step 1, and wrote code to fill these new ones that I had added in their place. To do this, I had to declare the following variables at the beginning of the subroutine:

   ⇒ Reals - y_t, y_tbar, azi, deltaphi

   ⇒ Integers - i_jets, ixx, jzz, qxx, jet_index,jet_position(maxjets)

   ⇒ External - azi, deltaphi

4. In order to study the non-b jets, I added some code starting around line 347 which picks out the position of (at most) the 3 hardest non-b jets, and stores them as *jet_position(1)*, *jet_position(2)*, and *jet_position(3)* respectively.

5. The *azi* function in the code was calculating the azimuth $\varphi = \text{atan}\big(x/E\big)$ rather than $\text{atan}\big(y/x\big)$ so this also needed to be changed.

6. I added in a function $deltaphi(azi1, azi2)$ which calculates the angular separation between two different azimuthal angles

7. In the subroutine *buildjets*, I initialised all of the arrays as 0. Before, only some of them were 0 to start, and so we were getting incorrect results

1.4 PYTHIA 6

To shower the les Houches file with Pythia 6, we need to update the file *setup-PYTHIA-lhef.f*.

1. First, we need to turn off Hadronisation. As mentioned above, leaving Hadronisation on ruins our analysis, as b hadrons are not tagged correctly, so we don't always tag b jets. If needs be I can go back and fix this, but for now I am just leaving it

2. Next we need to update the subroutine PYAEND. We use the definition of PYAEND from *setup-PYTHIA-lhef.f* in HZ rather than *hvq*, as this uses the new way to book histograms

1.5 PYTHIA 8

Getting the program up and running with PYTHIA 8 took a bit more work.

1. Copy the files *pythia8F77.cc* and *main-PYTHIA8.f* from *ttb_NLO_dec* to *hvq*

2. Edit the Makefile to make another executable *main-PYTHIA8-lhef*. For this, add the corresponding lines from the *ttb_NLO_dec/ Makefile*, and also include any *.o files that are needed to create main-PYTHIA-lhef. Also, after then analysis section of the Makefile (line 98) I added in the location PYTHIA 8 was installed in. This was done for PYTHIA 8.185, and I could only get it working on this version (PYTHIA 8.2 had changed where some files were installed so POWHEG couldn't find them, so I am just sticking with 8.185)

   ⇒ # PYTHIA 8

   ⇒ PYTHIA8LOCATION=/Users/Dan/code/pythia8185

   ⇒ FJCXXFLAGS+=-I$(PYTHIA8LOCATION)/include -I$(PYTHIA8LOCATION)/include/Pythia8

   ⇒ LIBPYTHIA8=-L$(PYTHIA8LOCATION)/lib/archive -lpythia8 -lstdc++

3. For the PYTHIA 8 showering to work, I had to enable shared libraries (whatever that means). I had to go to my PYTHIA folder, and run

   ⇒ ./configure --enable-shared

4. Next I needed to make a symlink to this shared library */Users/Dan/code/pythia8185/lib/libpythia8.dylib* and place it in my *testrun-tdec-lhc* folder, otherwise the PYTHIA8 executable just crashes as it says that the library is not loaded

5. In my notes it says that I was having some problem with the loop in *main-PYTHIA8.f* at line 143, apparently it was looping over something indefinitely. However, I don't seem to be having that problem this time, but it is something to watch out for.

6. I turned off hadronisation in *pythia8F77.cc* so that we could compare the output with our PYTHIA 6 result

2 NLL Upgrades

Assuming that we have got everything running properly, we can now look at making the changes suggested in FNO for NLL resummation in the planar limit. I'll now give a brief overview of what changes we need to make, and then will describe how we implement them.

Differences between Sudakovs

UCL

Old

$$\mathrm{d}\sigma = \sum_{f_b} \bar{B}^{f_b}(\mathbf{\Phi}_n)\mathrm{d}\mathbf{\Phi}_n \times \left\{ \Delta^{f_b}(p_\perp^{min}) \quad + \sum_{\alpha_r \in \{\alpha_r | f_b\}} \mathrm{d}\Phi_{rad} \frac{R^{\alpha_r}(\mathbf{\Phi}_n, \Phi_{rad})}{B^{f_b}(\mathbf{\Phi}_n)} \Delta^{f_b}(k_\perp)\Theta(k_\perp - p_\perp^{min}) \right\}$$

$$\Delta^{f_b}(p_\perp) = \exp\left\{ - \sum_{\alpha_r \in \{\alpha_r | f_b\}} \int \mathrm{d}\Phi_{rad} \frac{R^{\alpha_r}}{B^{f_b}} \Theta(k_\perp - p_\perp) \right\}$$

New

$$\mathrm{d}\sigma_{NLL} = \sum_{f_b,\rho} \bar{B}^{f_b,\rho}(\mathbf{\Phi}_n)\mathrm{d}\mathbf{\Phi}_n \times \left\{ \Delta^{f_b,\rho}(p_\perp^{min}) + \sum_{\alpha_r,\rho_r \in \{\alpha_r,\rho_r | f_b,\rho\}} \mathrm{d}\Phi_{rad} \frac{R^{\alpha_r,\rho_r}}{B^{f_b,\rho}} \Delta^{f_b,\rho}(k_\perp)\Theta(k_\perp - p_\perp^{min}) \right\}$$

$$\Delta^{f_b,\rho}(p_\perp) = \exp\left\{ - \sum_{\alpha_r,\rho_r \in \{\alpha_r,\rho_r | f_b,\rho\}} \int \mathrm{d}\Phi_{rad} \frac{R^{\alpha_r,\rho_r}}{B^{f_b,\rho}} \Theta(k_\perp - p_\perp) \right\}$$

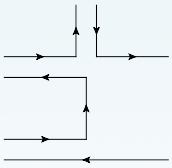
Here we have the differences between the old and new Sudakov form factors (and the differences in cross section too). Whereas the old cross section is split into contributions from different Born flavour structures f_b (and therefore, the numerator of the Sudakov form factor is composed of all Real singular regions α_r that can arise from a given f_b), the new cross section is further divided into Born planar flows ρ. This is only relevant for the Born process $gg \to t\bar{t}$, as $q\bar{q} \to t\bar{t}$ has only one Born colour flow.

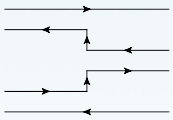
Differences between Sudakovs

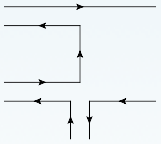
$$\Delta^{f_b}(p_\perp) = \exp\left\{ - \sum_{\alpha_r \in \{\alpha_r | f_b\}} \int \mathrm{d}\Phi_{rad} \frac{R^{\alpha_r}}{B^{f_b}} \Theta(k_\perp - p_\perp) \right\}$$

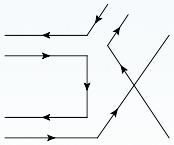
$$R^{f_b} = \sum_{\alpha_r \in \{\alpha_r | f_b\}} R^{\alpha_r} \qquad \{\alpha_r | f_b = q\bar{q} \to t\bar{t}\} = \begin{cases} q\bar{q} \to t\bar{t} + g \\ qg \to t\bar{t} + q \\ g\bar{q} \to t\bar{t} + \bar{q} \end{cases} \qquad \{\alpha_r | f_b = gg \to t\bar{t}\} = \begin{cases} gg \to t\bar{t} + g \\ qg \to t\bar{t} + q \\ gq \to t\bar{t} + q \\ g\bar{q} \to t\bar{t} + \bar{q} \\ \bar{q}g \to t\bar{t} + \bar{q} \end{cases}$$

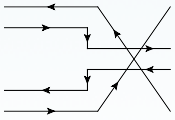
Can split $R^{\alpha_r = gg}$ up further $\qquad R^{\alpha_r, \rho_r} = R^{\alpha_r} \dfrac{R^{\alpha_r, \rho_r}_{\mathrm{pl}}}{\sum_{\rho_r} R^{\alpha_r, \rho_r}_{\mathrm{pl}}}$

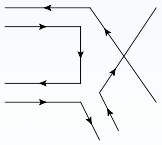
$R^{gg,1}_{pl}$ = 

$R^{gg,2}_{pl}$ = 

$R^{gg,3}_{pl}$ = 

$R^{gg,4}_{pl}$ = 

$R^{gg,5}_{pl}$ = 

$R^{gg,6}_{pl}$ = 

In this slide, we can see the 6 Real planar flows that we can have for $gg \to t\bar{t} + g$. Obviously, flows 1 to 3 can only arise from $\rho = 1$ (the "uncrossed" Born planar flow for $gg \to t\bar{t}$), and conversely, flows 4-6 come when we have $\rho = 2$. Using the expressions for these Real planar flows (which we have calculated using Mathematica, and discussed in the report), we can calculate the weightings

$$\frac{R^{\alpha_r, \rho_r}_{pl}}{\sum_{\rho_r} R^{\alpha_r, \rho_r}_{pl}} \tag{2.0.1}$$

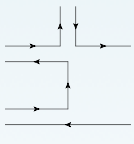
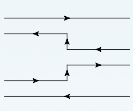
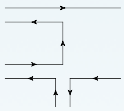
which, as we will see on the next slide, are all we need to calculate the modified Real cross section for our new Sudakov form factor. These weightings are simply functions of real phase space, so once we have that they are trivial to compute.

Differences between Sudakovs

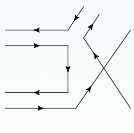
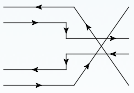
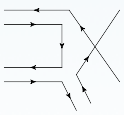
$$\Delta^{f_b}(p_\perp) = \exp\left\{ - \sum_{\alpha_r \in \{\alpha_r | f_b\}} \int \mathrm{d}\Phi_{rad} \frac{R^{\alpha_r}}{B^{f_b}} \Theta(k_\perp - p_\perp) \right\}$$

$$\Delta^{f_b,\rho}(p_\perp) = \exp\left\{ - \sum_{\alpha_r,\rho_r \in \{\alpha_r,\rho_r | f_b,\rho\}} \int \mathrm{d}\Phi_{rad} \frac{R^{\alpha_r,\rho_r}}{B^{f_b,\rho}} \Theta(k_\perp - p_\perp) \right\}$$

$$\sum_{\rho_r \in \{\rho=1\}} R^{gg,\rho_r} = R^{gg} \times \frac{1}{\sum_{\rho_r=1}^{6} R^{gg,\rho_r}_{\text{pl}}} \times \left(\quad + \quad + \quad \right)$$

$$\sum_{\rho_r \in \{\rho=2\}} R^{gg,\rho_r} = R^{gg} \times \frac{1}{\sum_{\rho_r=1}^{6} R^{gg,\rho_r}_{\text{pl}}} \times \left(\quad + \quad + \quad \right)$$

In the new Sudakov form factor $\Delta^{f_b,\rho}$, all the terms in the numerator of the exponent are the same as the in the old one, except for the $\alpha_r = gg$ terms. As discussed before, we only sum over the contributions associated with the underlying Born planar flow, and these are related to the full Real cross section for $gg \to t\bar{t} + g$ by the simple weighting shown above (for $\rho = 1, 2$ respectively). Obviously, summing these two terms together will give us R^{gg} back, so we haven't done anything strange here.

So this means we have two main things to do to implement our new Sudakov

1. If $f_b > 1$ (i.e. our Born process is $q\bar{q} \to t\bar{t}$), do nothing
2. If $f_b = 1$ (i.e. our Born process is $gg \to t\bar{t}$), choose a value of ρ according to the probabilities

$$\frac{B_{pl}^{gg,\rho=1}}{B_{pl}^{gg,\rho=1} + B_{pl}^{gg,\rho=2}}, \frac{B_{pl}^{gg,\rho=2}}{B_{pl}^{gg,\rho=1} + B_{pl}^{gg,\rho=2}} \tag{2.0.2}$$

3. Reweight our Born cross section by the appropriate factor (**??**) to get $B^{gg,\rho}$
4. Reweight our Real cross section for $gg \to t\bar{t}+g$ by one of the weightings in the slide above, depending on which value of ρ we chose in step 2

3 Implementation

In this section I will explain how I implemented the aforementioned upgrades.

3.1 Initialisation

The initialisation phase in POWHEG is started by a call to *pwhginit*. The two (main) things this subroutine does is

- Generate the grid used for generating events. This stage is unchanged, as this has nothing to do with Sudakov form factors.
- Normalise the upper bounding function U^{rr} used in the veto algorithm. This corresponds to calculating a value of N^{rr} such that the condition

$$\frac{J^{rr}R^{rr}}{B^{f_b}} \leq N^{rr}_{f_b}U^{rr} \tag{3.1.1}$$

  in all of phase space. However, when we do our veto algorithm later, we will be vetoing with a modified condition

$$\frac{J^{rr}\overbrace{R^{rr,\rho}}^{\leq R^{rr}}}{\underbrace{B^{f_b,\rho}}_{\leq B^{f_b}}} \leq N^{rr}_{f_b}U^{rr} \tag{3.1.2}$$

  For some values of $B^{f_b,\rho}$ (especially where $B^{f_b,\rho}/B^{f_b}$ is very small), this condition may not be satisfied, however, as we shall see later, we always know the numerical value of $B^{f_b,\rho}/B^{f_b}$ for each event, so we know how much (on an event by event basis) we need to raise the normalisation $N^{rr}_{f_b}$ by so that (**??**) is always satisfied

So we don't actually change anything in the initialisation stage, but we now know that we will need to change the normalisation $N^{rr}_{f_b}$ for each event later on.

3.2 Generation of events

The next stage in the POWHEG program is the event generation stage, and this is where the major differences (between the current POWHEG method, and our "upgraded" POWHEG method) arise. Before we start, I will make a "newsuda" flag, so that we can specify in the powheg.input file whether to use the old Sudakov or the new Sudakov.

1. Add the logical variable $flg\_newsuda$ to the header *pwhg_flg.h*
2. Add a line to the beginning of the *powheg.input* file which gives a value for the *newsuda* (i.e. 1 for yes or 0 for no)
3. In *pwhg_main.f*, at the beginning, add the lines

   ⇒ if(powheginput('#newsuda').eq.1d0) then

   ⇒ flg_newsuda=.true.

   ⇒ else

   ⇒ flg_newsuda=.false.

   ⇒ endif
4. We then need to include the header $pwhg\_flg.h$ in any subroutines that will be changed for the new Sudakov.
5. Any changes we make we should put inside an IF(flg_newsuda) condition, so that we don't have to change anything in the main code (only the value of *newsuda* in *powheg.input*) to switch between the old and new methods.

The first part of the event generation (the generation of a Born event) is unchanged. We generate values for $(f_b, \mathbf{\Phi}_n)$ with probabilities $\bar{B}(\mathbf{\Phi}_n) = \sum_{f_b} \bar{B}^{f_b}(\mathbf{\Phi}_n)$, but now we need to generate a value for ρ i.e. we

need to choose a Born colour flow for this process. In order to calculate ρ, we need first to calculate the weighting

$$\frac{B_{pl}^{gg,\rho=1}(\mathbf{\Phi}_n)}{B_{pl}^{gg,\rho=1}(\mathbf{\Phi}_n) + B_{pl}^{gg,\rho=2}(\mathbf{\Phi}_n)}, \tag{3.2.1}$$

where the Born planar flows $B^{gg,\rho=1,2}$ are given as a function of $\mathbf{\Phi}_n$ by

$$\begin{aligned} B_{pl}^{gg,\rho=1} &= 2g_s^4 \left(\frac{u}{s^2 t}(t^2+u^2) + 4\frac{m^2}{s}\frac{u}{t} - 4\frac{m^4}{t^2} \right) \\ B_{pl}^{gg,\rho=2} &= 2g_s^4 \left(\frac{t}{s^2 u}(t^2+u^2) + 4\frac{m^2}{s}\frac{t}{u} - 4\frac{m^4}{u^2} \right) \end{aligned} \tag{3.2.2}$$

where the Mandelstam variables are given by

$$t = (p_1 - p_3)^2 - m^2, u = (p_1 - p_4)^2 - m^2 \tag{3.2.3}$$

3.3 Generating a value for ρ

1. Declare the INTEGER *rho_idx* and the REAL *rhoweight* in the header *pwhg_rad.h*
2. Include *pwhg_flg.h* and *pwhg_rad.h* in *setborn* in *Born.f*
3. Declare and define the Mandlestam variables $t = -2p_{13}$ and $u = -2p_{14} = -2p_{23}$, and the REALS *ggbornplanar*1 and *ggbornplanar*2 in *setborn*. These will correspond to the functions in (**??**)
4. Inside an IF(flg_newsuda) condition, calculate rhoweight $= B^{\rho=1}/(B^{\rho=1} + B^{\rho=2})$, ignoring the factors of $2g_s^4$ as they cancel out
5. rhoweight is now stored in the header *pwhg_rad.h*, and we can access this from *gen_index.f* to calculate a value for ρ
6. Include *pwhg_flg.h* in the subroutine *gen_uborn_idx* in *gen_index.f*
7. Declare the REAL random and an EXTERNAL random
8. This subroutine chooses a value of f_b for our event (inside another IF(flg_newsuda) condition). In the POWHEG code $f_b = 1$ corresponds to the Born process $gg \to t\bar{t}$, and $f_b = 2...11$ correspond to the different $q\bar{q} \to t\bar{t}$ Born processes. We choose the ρ in the following way:

   $\Rightarrow$ If $f_b = 1$ and $random() \leq rhoweight$ then $\rho = 1$

   $\Rightarrow$ If $f_b = 1$ and $random() > rhoweight$ then $\rho = 2$

   $\Rightarrow$ If $f_b > 1$ then $\rho = f_b + 1$

   For some reason, not selecting a random seed makes the program run much more slowly, so I have just set iseed in powheg.input - it was called randomseed for some reason, but pwhg_main reads the value iseed.
9. We can now test that this is giving sensible values for ρ, if we get *pwhg_main* to output the value of ρ to a text file each time we have an event, then by counting the occurrences of the different values of ρ (for example, using Mathematica) we see that they come in the proportions

$$\begin{aligned} \rho &= 1 \ (\sim 45\%) \\ \rho &= 2 \ (\sim 45\%) \\ \rho &> 2 \ (\sim 10\%) \end{aligned} \tag{3.3.1}$$

   which is what we would expect at 14 TeV centre of mass energies.

3.4 Modifying the Real cross section

We will need to modify the calculation of the Real cross section for $gg \to t\bar{t} + g$, which happens in *sigread_rad(sig)* in *sigreal.f*, which returns

$$sig = \sum_{\alpha_r \in \{\alpha_r | f_b\}} R^{\alpha_r} \tag{3.4.1}$$

for the underlying Born flavour structure f_b through

$$sigreal\_rad(sig) \to realgr \to real\_ampsq \to setreal \tag{3.4.2}$$

This is actually used in the calculation of the upperbounding function normalisation in the previous section, and so we should not change this. Instead, in *sigreal.f*, I copied this subroutine to an identical subroutine called $sigreal\_rad2(sig)$, which we will need to change, but first we will need to calculate the weightings (**??**). The code also has a function *ggplanar* which takes in the momenta of the $n+1$ (i.e. Born+gluon) partons as input and gives out the 6 planar flows, we just need to double check that we call that with the correct signs etc on the momenta

1. The subroutine $sigreal\_rad2(sig)$ (and obviously $sigreal\_rad(sig)$) calculates the real cross section for a given α_r through a call to *setreal* in *real.f*.
2. First we will need to declare the array or REALs $rhorweight(6)$ in the header *pwhg_rad.h*
3. The function *ggplanar* takes Les Houches momenta[1] which have the form

   $$pup(i) = (p_x^{(i)}, p_y^{(i)}, p_z^{(i)}, E^{(i)}, m^{(i)}) \tag{3.4.3}$$

   as inputs, whereas the subroutine *setreal* has momenta in the centre of mass frame[2] of the form

   $$p(i) = (E^{(i)}, p_x^{(i)}, p_y^{(i)}, p_z^{(i)}) \tag{3.4.4}$$

   so we need to convert these into a form like the Les Houches momenta. First we declare $p\_pup(0:4, nlegreal)$ in REALS in *setreal*, and an integer *ixx* for a do loop *NB - When I first put this in the code I accidentally had *ixx* running from 0..4, however this should have been 1..5, I updated this later
4. Loop over the momenta p, rearranging them into the array *p_pup* in the order of (**??**)
5. Declare the REALS {T125,T152,T512,T521,T251,T215,TTOT}, these will be the planar amplitudes (and TTOT is their total, used to turn them into weights)
6. call ggplanar(p_pup(:,1), 1, p_pup(:,2), 1,p_pup(:,5), 1, p_pup(:,3), p_pup(:,4), xm2, t512, t152, t125, t521, t251, t215) TODO - double check the direction of all of these momenta...
7. Set TTOT=T125+T152+... and then set

   $$\begin{aligned} rhorweight(1) &= T512/TTOT \\ rhorweight(2) &= T152/TTOT \\ rhorweight(3) &= T125/TTOT \\ rhorweight(4) &= T521/TTOT \\ rhorweight(5) &= T251/TTOT \\ rhorweight(6) &= T215/TTOT \end{aligned} \tag{3.4.5}$$

8. Now, back to *sigreal.f*, in the subroutine $sigreal\_rad2(sig)$ (which I have copied and pasted from $sigreal\_rad(sig)$ directly below the original) at around line 908 we have the part of the code that calculates each R^{α_r} (called $rr(alr)$ in the code) that is "on" for our underlying Born flavour f_b, and sums them together (to give the numerator in the (old) Sudakov form factor). The only thing that

[1] These are in the lab frame of reference
[2] The function *ggplanar* is built from invariants, so is frame independant

we need to do here is to reweight $rr(alr)$ in the case that $alr = gg \to t\bar{t}$. Therefore we say that if both $flst\_alr(1, alr) = 0$ and $flst\_alr(2, alr) = 0$ (i.e. the first and second parton in the current flavour list are both gluons) then

- If $\rho = 1$, then multiply $rr(alr)$ by $\Big(rhorweight(1) + rhorweight(2) + rhorweight(3)\Big)$
- Elseif $\rho = 2$ then multiply $rr(alr)$ by $\Big(rhorweight(4) + rhorweight(5) + rhorweight(6)\Big)$
- Else - do nothing

This is simply doing what is laid out on the slide on page 5 above.

3.5 Modifying the veto algorithm

Next we will edit the part of the code that does the veto algorithm, and so we edit the Sudakov form factor that generates the radiation. This occurs in *pwhgevent→gen_ radiation→gen_ rad_ isr(t).*

1. Include *pwhg_ flg.h* in *gen_ rad_ isr(t)*
2. Declare the REAL *factor* - defined as
$$\frac{B^{f_b,\rho=1,2}}{B^{f_b}} = rhoweight, (1 - rhoweight) = factor \tag{3.5.1}$$
3. This is the amount that we need to raise the normalisation of the upper bounding function $N^{rr}_{f_b}$ as discussed in Subsection 2.1. Therefore, we make the substitution
$$unorm \to unorm/factor \tag{3.5.2}$$
4. When the code calls $sigborn\_rad(born)$ to get B^{f_b}, we multiply this by $factor$ to get $B^{f_b,\rho}$
5. Next, if using the new Sudakov, we simply call $sigreal\_rad2(sig)$ rather than $sigreal\_rad(sig)$, and with this our radiation should be generated according to the new NLL Sudakov form factor!