Manual for `PeTeR`

Version 2.1.0

Christian Lorentzen

December 15, 2016

1 Introduction

The flexible C++ program `PeTeR` [1] was developed for the purpose of performing the expensive numerical computation of the resummed cross section according to [2, 3, 4], i.e. the transverse momentum distribution of electroweak boson production at hadron colliders (γ, Z, $W^{\pm}$, H). It consists of two parts, the Unix-like command-line program `peter`, which provides a user-friendly interface, and `libpeter`, which contains all the computational ingredients. `libpeter` is a library, and as such it may easily be used to build other programs.

So far, `PeTeR` is able to calculate the NLL, N^{2}LL, N^{3}LL resummed cross sections as well as the corresponding singular ones, LO, NLO$_{\text{sing}}$ and NNLO$_{\text{sing}}$. In addition, the full NLO fixed-order results for vector boson and for Higgs production are implemented and were extensively cross-checked against `QT` [5, 6][1] and `HqT2.0` [7].

PeTeR includes the full two-loop hard functions for electroweak boson production computed in [4], but for simplicity the tiny two-loop contributions which arise when a neutral vector boson couples to an internal quark loop[2] are neglected (N_V^v and N_V^a, cf. Eqs.(5.2) and (5.3) of [4]).

2 Installation

The code can be downloaded from `http://peter.hepforge.org/`. The gzipped tar file unpacks into a directory `PeTeR-k.m.n`, where `k.m.n` denotes the version number. Change into this directory and type

```
./configure
make
```

A successful compilation creates `src/libpeter/libpeter.a` and `src/peter/peter`. It is possible to run the code directly inside `src/peter`. In order to install/copy `peter` and `libpeter` into `/usr/local` type

[1] I am grateful for discussions with Richard Gonsalves.
[2] The axial part at two-loop level is still unknown. It is, however, already very small at one loop.

```
make install
```

Super user privileges may be necessary (type `sudo make install`). The directory can be changed by the option `--prefix=directory` of the `configure` command.

If the library LHAPDF [8], version `5.4.0` or higher (but still version `5`, version `6` is not yet tested), is installed, its capabilities can be included by specifying the configure option `--with-LHAPDF`. If the LHAPDF library is installed in a non-standard directory, i.e. the executable `lhapdf-config` is not in a directory included in the environment variable `$PATH`, one has to specify the complete path to the executable `lhapdf-config`

```
./configure --with-LHAPDF \
LHAPDF_CONFIG=/path-to-lhapdf-config/lhapdf-config
```

How to choose a pdf from LHAPDF in the executable `peter` is explained in the next chapter. More details about the installation process are given in the file `INSTALL` and by invoking `./configure --help`.

3 Command Line `peter`

In order to facilitate the user interface of the command line program `peter`, the whole command line parsing is done by the `boost/program_options` library [9]. After storing all input parameters, `peter` calls the function `sigma_pT` of `libpeter`. Note that `peter` automatically takes advantage of parallelization on multi-core processors by the build-in functionality of the `CUBA` library [10]. An overview of library dependencies is depicted in Figure 1. A simple call `peter -h` produces the following detailed help message:

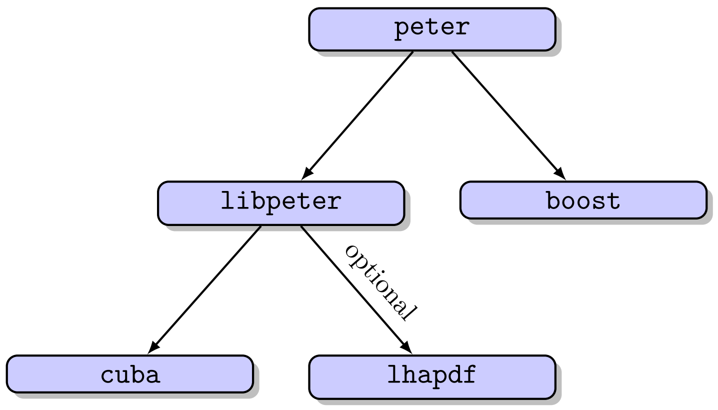

Figure 1: Dependencies of `peter`

```
Call: peter [options]

Description: Specify at least one option otherwise '-h' is assumed.
Peter calculates the transverse momentum distribution
of electroweak boson production at colliders such as the LHC:
dsigma/dpT[pbarn] of hadron + hadron -> V + X.
```

```
See http://peter.hepforge.org/ for more informations.

Allowed options:
  -h [ --help ]                       Print help message.
  --version                           Print version number.
  -v [ --verbosity ] arg (=1)         Set the level of verbosity:
                                         0=silent
                                         1=input values (reusable as
                                           input file by copy paste)
                                         2=add pdf information if available
                                           (not yet working for LHAPDF)
  -f [ --configfile ] arg             Set the name for a configuration file
                                      that will be parsed. Command line input
                                      overwrites config file input.
  --boson arg (=1)                    Set the boson being produced:
                                         0 = Photon
                                         1 = Z
                                         2 = W+
                                         3 = W-
                                         4 = Higgs
  --collider arg (=0)                 Set the collider:
                                         0 = proton-proton
                                         1 = proton-antiproton
                                         2 = antiproton-antiproton
  --cms-energy arg (=8000)            Set cms-energy sqrt(S) [GeV].
  -p [ --transverse-momentum ] arg (=100)
                                      Set pT start point [GeV].
  --pT-steps arg (=1)                 Set the number of pT points to be
                                      calculated.
  --pT-step-size arg (=10)            Set the step size between pT points
                                      [GeV].
  --log-scale [=arg(=1)] (=0)         If true, set logarithmic steps.
                                      i=0..n-1, n=pTsteps, d=pT-step-size
                                      linear: pTstep(i) = pT + i*d, i=0..n-1
                                      log: pTstep(i) = pT *
                                      ((pT+(n-1)*d)/pT)^(i/(n-1))
                                      So both cases start with pT and end
                                      with pT + (n-1)*d.
  --min-rapidity arg (=-1000000)      Set minimal rapidity.
  --max-rapidity arg (=1000000)       Set maximal rapidity.
  --hard-scale arg (=-1)              Set hard scale muH[GeV]. The general
                                      scale choices are
                                         >0 : mu = input [GeV]
                                         -1 : mu = (13 pT + 2 M)/12 -
                                                   pT^2/sqrt(S)
                                         -2 : mu = (7 pT + 2 M)/12 * (1 - 2
                                                   pT/sqrt(S))
                                         -3 : mu = mu_jet^2/mu_hard
                                         -4 : mu = sqrt(pT^2 + M^2)
                                      where M is the boson mass.
                                      Option -3 is forbidden for the hard
                                      scale.
--jet-scale arg (=-2)                 Set jet scale muJ [GeV]. Same options
                                      as hard scale. Option -3 is forbidden.
--soft-scale arg (=-3)                Set soft scale muS [GeV]. Same options
                                      as hard scale.
--factorization-scale arg (=-1)       Set factorization scale muF [GeV]. Same
                                      options as hard scale. If jet, soft and
                                      factorization scale are equal, eta
                                      equals 0. For each partonic channel,
                                      the singularity 1/Gamma[eta] is treated
                                      by expanding in eta for |eta| < 0.001.
--hard-scale-factor arg (=1)          Multiply the hard scale.
--jet-scale-factor arg (=1)           Multiply the jet scale.
--soft-scale-factor arg (=1)          Multiply the soft scale.
--factorization-scale-factor arg (=1) Multiply the factorization scale.
--flavors arg (=5)                    Set the number of active flavors.
--order-resum arg (=0)                Set the order of resummation:
                                         3 = N^3LL
                                         2 = NNLL
                                         1 = NLL
                                         0 = fixed-order LO (no resummation)
                                        -1 = fixed-order NLO (no resummation)
                                        -2 = matched NNLL+NLO
                                        -3 = matched N^3LL+NLO
                                      For resummed results: h*j*s, log(U),
                                      eta are expanded up to
                                      alphas^(order-1). If all scales are set
                                      equal to muF one gets the
                                      singular/threshold expanded cross
                                      section N^xLOsing with x=order-1.
                                      For fixed-order results: muR=hard
                                      scale, muF=fact. scale.
                                      For matched results:
                                      resummed(muH,muJ,muS,muF) + nlo(muF) -
                                      nlo_sing(muF).
                                      Please note that the matched results
                                      (-1,-2,-3) for photon production are
                                      incomplete (!) since they do not
                                      include the NLO photon fragmentation
                                      contribution. To obtain finite results
                                      without fragmentation, the photon mass
                                      parameter needs to be kept at a
                                      non-zero value and acts as a collinear
                                      cutoff.
--order-alphas arg (=0)               Set the order of the running of alphas:
                                         0 = 1-loop (beta0)
                                         1 = 2-loop (beta1)
                                         2 = 3-loop (beta2)
                                         3 = 4-loop (beta3).
--alphas-epsrel arg (=1e-06)          Set the relative precision for the
                                      computation of the running of alphas.
                                      Newton's method quits with a warning
                                      after 1000 iterations if accuracy is
                                      not reached.
--with-triangles [=arg(=1)] (=0)      Include triangle contributions for b
                                      and t quarks (only Z bosons). Quark
                                      masses >= 10^6 GeV are treated as
                                      infinitely heavy.
--with-two-loop-const [=arg(=1)] (=0) Include the two loop constant in the
                                      hard function. Without it, the alphas^2
                                      part of the hard function is normalized
                                      as h2(muH=pT)=0.
--with-ct2-evolution [=arg(=1)] (=1)  For Higgs only: If true, the top-quark
                                      matching scale mut=mass_t. If false,
                                      mut=muH. For resummed and fixed-order
                                      results, the Wilson coefficient Ct2 is
                                      evaluated at scale mut, the cross
                                      section is multiplied by the evolution
                                      factor U_Ct2(mut,muH). Note that
                                      U_Ct2(muH,muH)=1. For strict
                                      fixed-order results, i.e.
                                      order-resum=-1 (NLO), one has to set
                                      this option to false.
--no-hard [=arg(=1)] (=0)             Switch off the hard function, i.e. sets
                                      hard functions to 1.
--no-jet [=arg(=1)] (=0)              Switch off the jet function, i.e. sets
                                      jet functions to 1.
--no-soft [=arg(=1)] (=0)             Switch off the soft function, i.e. sets
                                      soft functions to 1.
--scale-variation arg (=0)            Compute theoretical errors from scale
                                      variation by a factor of 1/2 and 2:
                                         0: Don't compute scale variation.
                                         1: Add the errors in the additional
                                            columns scale+ and scale-.
                                         2: Additionally output the scale
                                            varied cross sections.
                                      For LO, NLO and singular: The scale
                                      muF=muR is varied up and down. Then, a
                                      parabola is fitted through the 3 points
                                      mu_var/mu=(1/2,1,2), and max and min
                                      values in this interval minus the
                                      central value (mu_var/mu=1) give the
                                      error estimates.
                                      Resummed and matched: Same procedure,
                                      but each scale muH, muJ, muS and muF is
                                      varied separately. The individual
                                      errors are added in quadrature.
                                      Note: The central value (mu_var/mu=1)
                                      might differ from the one computed
                                      without scale variation (within the
                                      given integration error), as the 3 and
                                      9 cross sections, respectively, are
                                      computed at once. As the scale
                                      variation is usually larger than a few
                                      percent, one might consider setting
                                      cuba-epsrel not too small.
--subtractions arg (=2)               Set the order of subtractions of the
                                      resummed integrand. 0-2 are
                                      implemented.
--MH arg (=126)                       Set Higgs boson mass [GeV].
--MP arg (=1)                         Set photon mass [GeV]. See option
                                      order-resum.
--MZ arg (=91.1874)                   Set Z boson mass [GeV].
--MW arg (=80.381)                    Set W boson mass [GeV].
--sin2theta arg (=0.22296)            Set sin^2(theta_w).
--alpha arg (=0.00781592)             Set electroweak coupling constant
                                      alpha.
--GF arg (=1.16638e-05)               Set Fermi constant GF. It is used for
                                      Higgs production only (instead of
                                      alpha,...).
--alphas arg (=0.11707)               Set strong coupling constant alphas at
                                      MZ.
--mu-alphas arg                       Set the default scale for alphas.
                                      Default is MZ.
--Vud arg (=0.97427)                  Set CKM matrix element Vud.
--Vus arg (=0.22534)                  Set CKM matrix element Vus.
--Vub arg (=0.00351)                  Set CKM matrix element Vub.
--Vcd arg (=0.2252)                   Set CKM matrix element Vcd.
--Vcs arg (=0.97344)                  Set CKM matrix element Vcs.
--Vcb arg (=0.0412)                   Set CKM matrix element Vcb.
--Vtd arg (=0.00867)                  Set CKM matrix element Vtd.
--Vts arg (=0.0404)                   Set CKM matrix element Vts.
--Vtb arg (=0.999146)                 Set CKM matrix element Vtb.
--mass-b arg (=0)                     Set the b-quark mass.
--mass-t arg (=173.5)                 Set the t-quark mass.
--pdf-type arg (=0)                   Set the pdf type:
                                        -1 = fake pdfs 'x*(1-x)'
                                         0 = MSTW2008nnlo
                                         1 = LHAPDF
--pdf-filename arg                    For LHAPDF, set the filename of the pdf
                                      including the full path.
--pdf-member arg (=0)                 For LHAPDF, set the member of the pdf
                                      (if it has different members).
--cuba-routine arg (=3)               Set the integration routine of cuba:
                                         0 = Vegas
                                        1 = Suave
                                        2 = Divonne
                                        3 = Cuhre
                                     For more informations see the
                                     documentation of the CUBA library.
--cuba-epsrel arg (=0.001)           Set the maximal relative error of
                                     integration.
--cuba-epsabs arg (=1e-20)           Set the maximal absolute error of
                                     integration.
--cuba-verbosity arg (=0)            Set the verbosity level of the cuba
                                     integration.
--cuba-seed arg (=0)                 Set the seed for the pseudo-number
                                     generator and chooses the generator.
--cuba-level arg (=0)                Choses the random number generator:
                                     seed | level | generator
                                        0 |  any  | Sobol q
                                      !=0 |   0   | Mersenne-Twister p
                                      !=0 | !=0   | Ranlux p
                                     q=quasi-random, p=pseudo-random
--cuba-mineval arg (=0)              Set minimum number of integrand
                                     evaluations required.
--cuba-maxeval arg (=100000)         Set the (approximate) maximum number of
                                     integrand evaluations allowed.
--cuba-last-sampling arg (=0)        It true, only the last (largest) set of
                                     samples is used in the final result.
--cuba-smoothing arg (=1)            (Vegas and Suave only) If true, apply
                                     additional smoothing to the importance
                                     function, this moderately improves
                                     convergence for many integrands.
--cuba-nnew arg (=2000)              (Suave only) Set the number of new
                                     integrand evaluations in each
                                     subdivision (approximate).
--cuba-flatness arg (=25)            (Suave only) Set a flatness parameter.
                                     The flater the higher the value might
                                     be.
--cuba-key0 arg (=9)                 (Cuhre only) Choses the basic
                                     integration rule. key0=7,9,11,13
                                     selects the cubature rule of degree
                                     key0. Note that the degree-11 rule is
                                     available only in 3 dimensions, the
                                     degree-13 rule only in 2 dimensions.
                                     For other values, the default rule is
                                     taken: degree-13 in 2 dimensions,
                                     degree-11 in in 3, degree-9 otherwise.
                                     The pT-spectrum of on-shell bosons has
                                     a 3 dimensional integration (y,x,mx2).
--cuba-key1 arg (=47)                (Divonne only) Determine sampling in
                                     the partition phase.
--cuba-key2 arg (=1)                 (Divonne only) Determine sampling in
```

```
                                        the final integration phase.
  --cuba-key3 arg (=1)                  (Divonne only) Set the strategy for the
                                        refined phase.
  --cuba-maxpass arg (=5)               (Divonne only) Control the thoroughness
                                        of the partitioning phase integration
                                        phase.
  --cuba-border arg (=1e-10)            (Divonne only) Set the width of the
                                        border of the integration region. Use a
                                        non-zero border if the integrand
                                        subroutine cannot produce values
                                        directly on the integration boundary.
  --cuba-maxchisq arg (=10)             (Divonne only) Set the maximum chi^2
                                        value a single subregion is allowed to
                                        have in the final integration phase.
  --cuba-mindeviation arg (=0.25)       (Divonne only) Set a bound, given as
                                        the fraction of the requested error of
                                        the entire integral, which determines
                                        whether it is worthwhile further
                                        examining a region that failed the
                                        chi^2 test. Only if the two sampling
                                        averages obtained for the region differ
                                        by more than this bound is the region
                                        further treated.
```

Options can be specified on the command line and by providing a configuration file via `--configfile filename` (or `-f filename`). Note that the leading '`--`' of the option names must be omitted in configuration files and that command line input overwrites configuration file input. If you call `peter` with `--verbosity 1` (or `-v1`), it prints the options in a format suitable for copy & paste into a configuration file. In addition, the file `sampleInput.txt` in `src/peter` is a good starting point for your own configuration file. The call

```
./peter -v0 -f sampleInput.txt --boson 4 --cms-energy 8000 \
--order-resum 2 -p 50 --pT-steps 4 --pT-step-size 50 --cuba-routine 2
```

produces an output that looks like

```
pT [GeV] dsigma/dpT [pbarn/GeV] error [pbarn/GeV] prob        fail
50       0.0505339              5.06951e-05       4.10783e-06 0
100      0.0100421              1.00097e-05       1.67229e-08 0
150      0.0030726              3.1214e-06        1.75918e-05 0
200      0.0011569              1.13585e-06       7.11653e-14 0
```

The `error` is the presumed absolute error of integration, `prob` is the χ^2-probability that `error` is not a reliable error estimate, and `fail` evaluates to 0 if the desired accuracy was reached; for more details see [10].

If you compiled `PeTeR` with LHAPDF, you can specify a pdf by the options `--pdf -type 1 --pdf-filename /path-to-share/lhapdf/PDFsets/xxx.LHgrid`, where `xxx` is the pdf name of your choice. As you see, the option `pdf-filename` requires the full path of the pdf grid file.

As a last point, note that you can redirect the output of `peter` in the usual way, e.g. `peter 1> output.dat 2> error.log` redirects the standard output into the file `output.dat` and all warnings and error massages into `error.log`.

4 Library `libpeter`

Everything in the library `libpeter` is declared in the namespace `peter`. The main function in the file `sigma_pT.hpp`

```
std::vector<double> sigma_pT(const Cuba_Parameters& cp, const
    Process_Data& pd, const SM_Parameters& sm, const PDF& pdf)
```

provided by the library `libpeter` integrates the cross section via the `CUBA` library and returns the result as a vector in the form (p_T, $\frac{d\sigma}{dp_T}$, `error`, `prob`, `fail`). The input is given by the self explaining classes as shown above. The structure of the library can be seen in Figure 2. In the following, we give short summaries of the most important header files; the actual implementation might nevertheless be in the corresponding source file (`.cpp` instead of `.hpp`).

`alphas.hpp` The running of α_s up to 4-loop (β_3) is implemented by solving the equation

$$\ln\frac{\mu^2}{\mu_0^2} = \int_{\alpha_s(\mu_0)}^{\alpha_s(\mu)} d\alpha \left(-\alpha \sum_{n=0}^{\text{order}} \left(\frac{\alpha}{4\pi}\right)^{n+1} \beta_n\right)^{-1} \tag{1}$$

for $\alpha_s(\mu)$ with Newton's method. The integral is calculated analytically for the corresponding value of order (option `order-alphas`).

`mathfunctions.hpp` Several special functions are implemented at `double` precision, among these

- `pow10`: 10^x
- `dilog` and `cdilog`: dilogarithm $\mathrm{Li}_2(x)$ for real and complex arguments
- `gamma`: Euler gamma function $\Gamma(x)$
- `polygamma`: polygamma functions $\Psi^{(n)}(x)$ for $n \in \{0, 1, 2, 3, 4, 5, 6\}$ (Higher n are implemented but with less accuracy since they are not required.)

All of the above functions have been extensively checked against their *Mathematica* implementations.

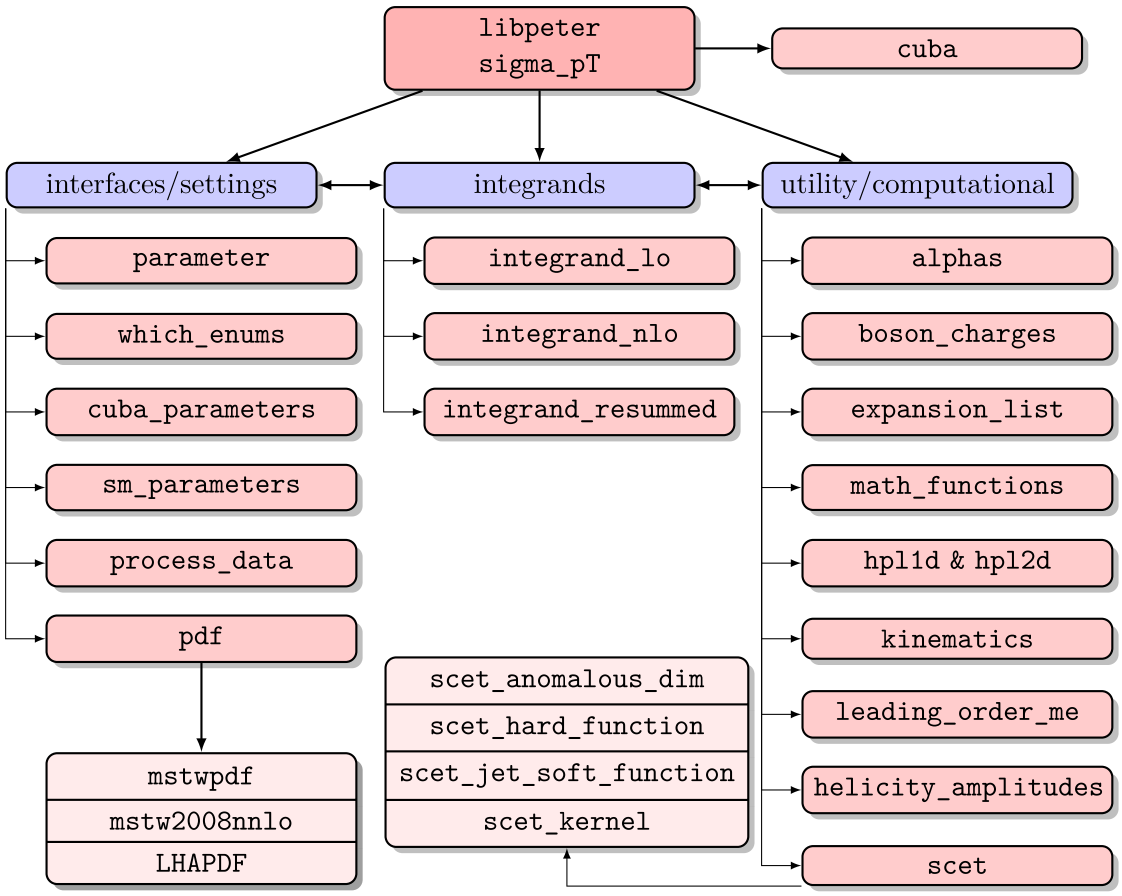

Figure 2: Schematic structure of `libpeter`. Blue boxes represent logical units, reddish boxes represent files (without indication for vector bosons `_V` and higgs boson `_H`, without file extensions `.hpp` and `.cpp`) or libraries.

`hpl1d.hpp` and `hpl2d.hpp` For the numerical computation of one- and two-dimensional harmonic polylogarithms, as required for the two-loop hard functions, these files provide full C++ versions of [11] and [12], which were originally written in `Fortran`.

The conversion has been achieved by modifying the `Fortran` code to comply to the C++ syntax. Apart from other differences between C++ and `Fortran`, the way of addressing elements of arrays was trivialized by template classes `Fortranarray1d`, `Fortranarray2d` and so forth. Schematically they look like

```
template <class T, const int n1, const int n2>
class FortranArray1d
{
private:
    const static int n_ = n2-n1+1;
    T value_[n_];
public:
    FortranArray1d() {
        for (int n=0; n<n_; ++n) value_[n] = T();
    };

    FortranArray1d(const T t[n_]) {
        for (int n=0; n<n_; ++n) value_[n] = t[n];
    };

    T operator() (int n) const {
        if (n<n1 || n2<n ) {
            /*error handling*/
        }
        return value_[n-n1];
    };

    T& operator() (int n) {
        if (n<n1 || n2<n ) {
            /*error handling*/
        }
        return value_[n-n1];
    };

    int size1() {return n_;};
};
```

Thus, no statement of accessing array elements on which the **Fortran** code heavily relies had to be changed. The evaluation of harmonic polylogarithms seems to be the bottleneck for $\mathrm{N^3LL}$ and $\mathrm{NNLO_{sing}}$ computations with the option `with-two-loop-const` enabled.

`helicity_amplitudes.hpp` The very long expressions for the helicity amplitudes necessary for NNLO hard functions are implemented by automatic insertion of the preprocessed expressions. It would be nice to implement the coefficients α, β, γ and δ of [13] as classes with a common interface. The easiest and most standard way of achieving this is to derive all of them from a common (base) class, which allows the storing of real and imaginary parts of coefficients of different orders in α_s, multiplying and complex conjugating them. This approach, however, has one drawback: it calls virtual functions, which are considered slow for high performance computations. Therefore, the *magic* of the Curiously Recursive Template Pattern (CRTP) [14] is invoked. There is a template class `template <class Derived> class Coefficient`, which provides the requested common interface. Every coefficient is then derived from a template of its own type, e.g. `class Alpha_2a : public Coefficient<Alpha_2a>`.

`expansion_list.hpp` In order to achieve the expansion of hard times jet times soft functions in α_s, these functions return the type `ExpList`, which is a class storing the order. Actually, `ExpList` is just an alias for `ExpansionList<double>`, while `ExpansionList<T>` is a template class for any type `T`. The template function

```
template <class T>
ExpansionList<T> operator*(const ExpansionList<T>& left, const
   ExpansionList<T>& right)
{
    unsigned int order = left.order();
    if (right.order() < order) order = right.order();
    ExpansionList<T> result(order);
    result[0] = left[0] * right[0];
    if (order >= 1)
    {
        result[1] = left[0] * right[1] + left[1] * right[0];
    }
    if (order >= 2)
    {
        result[2] = left[0] * right[2] + left[2] * right[0] + left
           [1] * right[1];

    }
    return result;
};
```

provides the multiplication of two `ExpansionList<T>`s to the right order.

`scet.hpp` The header file `scet.hpp` simply includes `scet_anomalous_dim.hpp`, `scet_hard_function.hpp`, `scet_jet_soft_function.hpp` and `scet_kernel_function.hpp`. These implement hard, jet and soft functions, evolution factors, the kernel and its derivatives according to [2, 3, 4].

`pdf.hpp` There are two implementations for pdfs. One is an interface to the LHAPDF library [15, 8]. Another is a modified version of MSTW [16, 17], `mstwpdf.hpp`. The pdf mstw2008nnlo is hard coded in the C++ file `mstw2008nnlo.cpp`; neither an external data file nor an IO-operation is required anymore. The tradeoff in memory space is negligible for modern computers. An additional feature required for second subtractions is the possibility to compute the first derivative in x of pdfs.

References

[1] Ch. Lorentzen, http://peter.hepforge.org.

[2] T. Becher, C. Lorentzen and M. D. Schwartz, Phys. Rev. Lett. **108** (2012) 012001 [arXiv:1106.4310 [hep-ph]].

[3] T. Becher, C. Lorentzen and M. D. Schwartz, Phys. Rev. D **86** (2012) 054026 [arXiv:1206.6115 [hep-ph]].

[4] T. Becher, G. Bell, C. Lorentzen and S. Marti, JHEP **1402** (2014) 004 [arXiv:1309.3245 [hep-ph]].

[5] R. Gonsalves, `http://www.physics.buffalo.edu/gonsalves/ewbqt/`.

[6] R. J. Gonsalves, J. Pawlowski and C. -F. Wai, Phys. Rev. D **40**, 2245 (1989).

[7] G. Bozzi, S. Catani, D. de Florian and M. Grazzini, Nucl. Phys. B **737** (2006) 73 [hep-ph/0508068], `http://theory.fi.infn.it/grazzini/codes.html`.

[8] M.R. Whalley and A. Buckley, LHAPDF, `http://lhapdf.hepforge.org/`

[9] V. Prus, boost/program_options, `http://www.boost.org`

[10] T. Hahn, Comput. Phys. Commun. **168** (2005) 78 [hep-ph/0404043], `http://www.feynarts.de/cuba/`.

[11] T. Gehrmann and E. Remiddi, Comput. Phys. Commun. **141** (2001) 296 [hep-ph/0107173].

[12] T. Gehrmann and E. Remiddi, Comput. Phys. Commun. **144** (2002) 200 [hep-ph/0111255].

[13] L. W. Garland, T. Gehrmann, E. W. N. Glover, A. Koukoutsakis and E. Remiddi, Nucl. Phys. B **642** (2002) 227 [hep-ph/0206067].

[14] J.O. Coplien, C++ Rep. Volume 6 Issue 2 (1995), `http://dl.acm.org/citation.cfm?id=229227.229229`

[15] M. R. Whalley, D. Bourilkov and R. C. Group, hep-ph/0508110.

[16] A. D. Martin, W. J. Stirling, R. S. Thorne and G. Watt, Eur. Phys. J. C **63** (2009) 189 [arXiv:0901.0002 [hep-ph]].

[17] A.D. Martin, W.J. Stirling, R.S. Thorne and G. Watt, `http://mstwpdf.hepforge.org/`