Manual for

Version 1.0

Guido Bell, Rudi Rahn, and Jim Talbert

April 21, 2020

Quick reference guide for `SoftSERVE` 1.0

Step 0	Install using `./configure` from program root directory (use flag `--MPFR` for multiprecision)
Step 1	Bring measurement function into exponential form (see 4.1)
Step 2	Derive parameters n, m, and functions F_X and G_{Xy} (4.2,4.3,4.4)
Step 3	Descend into `SCET1` ($n \neq 0$) or `SCET2` ($n = 0$) folder and edit input files (4.5,4.6)
Step 4	Call `make` to compile executables (call `make list` for available individual targets) (4.7)
Step 5	Run binaries from the `Executables` subfolder manually or using `./execsftsrv` script (4.8)
Step 5a	Recompile and run with different `border` setting to check for rounding errors (4.9)
Step 5b	Re-build and run executables using multiprecision variables, if needed (4.10)
Step 6	Combine output of all executables manually or using `./sftsrvres` script (4.11)
Step 7	(Only for Fourier space): Call `./fourierconvert` script to account for phases (4.12)
Step 8	Renormalise manually or using `./laprenorm` or `./momrenorm` (4.13)

Syntax for the scripts

Script	effect
`./configure`	Install and set up
`./configure --MPFR`	Install with GMP/MPFR support
`make list`	Write list of all possible make targets
`make target`	Build binaries as specified by “target”
`make target BOOST=1`	Build binaries using multiprecision support
`make target RRG=1`	Build binaries using web regularisation
`./execsftsrv -o observable -d subdir`	Run all binaries and tallies results
`./sftsrvres -o observable -d subdir`	Tally results from individual result files
`./fourierconvert -o observable -d subdir`	Creates Fourier-adjusted results file
`./laprenorm -o observable -n nvalue -d subdir`	Renormalise in Laplace space, SCET-1
`./momrenorm -o observable -n nvalue -d subdir`	Renormalise in momentum space, SCET-1
`./laprenorm -o observable -d subdir`	Renormalise in Laplace space, SCET-2
`./momrenorm -o observable -d subdir`	Renormalise in momentum space, SCET-2
`./[any script] -h`	Explain the script

In all cases, the `-d subdir` and `-n nvalue` options (and `-h` flag, of course) are optional.

`target` must be replaced with the colour structure target that is supposed to be built, see `make list`.

`observable`, `nvalue` and `subdir` must be replaced with their observable-dependent values as specified in the input files (`observable` and `n`), or as nested in the directory tree (`subdir`).

Instead of `-o observable`, `-n nvalue`, `-d subdir` and `-h`, the user can use `--obs=observable`, `--n=nvalue`, `--dir=subdir` or `--help`.

Scripts with `-o` flag exist also in `NAE`-suffixed form assuming non-Abelian exponentiation. These scripts bypass the uncorrelated emission binaries and their results, and use the square of the one-loop result instead.

Renormalisation, tallying and Fourier conversion scripts exist in `RRG`-suffixed form to accommodate the rapidity renormalisation group. Doubly suffixed scripts exist, as well.

Contents

Chapter 1

Overview

This document serves as the manual for users of `SoftSERVE` wishing to compute the NNLO bare soft functions for generic global observables in settings involving two light-like back-to-back Wilson lines, with subsequent renormalisation. `SoftSERVE` in turn can be seen as the analytic framework laid out in [1, 2] crystallised in C++ code. Therefore [1, 2] should be consulted for any detail on the analytic background in general, and on the criteria deciding whether observables are amenable to computation using `SoftSERVE`, in particular.

This manual is structured to make it accessible mainly for first-time users, with section indices and a dedicated section for known and typical problems to assist troubleshooters and repeat users. To this purpose we will lay out the primary requirements and initial setup for the program suite in chapter 2. We then present a brief step-by-step list of the program flow for a typical computation of a hypothetical observable's soft function in chapter 3. Each step on this checklist corresponds and links to a subsection of chapter 4, which explains in detail what needs to be done and why, at that particular step along the way. We also include a list of known problems and difficulties often encountered, and show how to solve or circumvent these in chapter 5. We will add problems encountered by users to this list, it will be updated ad-hoc, as problems are reported to the developers.

Finally, where necessary or helpful we refer to explanatory appendices that lay out the analytic background or computational circumstances that make a given procedure necessary, if these arise due to computational considerations. For features originating in the analytic framework, we refer to the first main papers [1, 2]. We also provide explicit template examples in Appendix F for multiple observables that we have treated with `SoftSERVE`, including the relevant program inputs, which should serve as a guide for users who wish to use `SoftSERVE` to make novel predictions.

Chapter 2

Initial installation and setup

A user seeking to use `SoftSERVE` needs at base level two things:

- Rudimentary understanding of C++ code, slightly above `HelloWorld.cpp` level and familiarity with the cmath syntax, and
- a POSIX standard machine equipped with a working C++ compile environment.

The former is required to edit the template C++ files that encode the input for a given observable. For the latter, the program has been tested on several Linux distributions,various MacOS versions, and Cygwin[1] using `gcc`/`g++`, `clang`/`clang++` and `icc`/`icpc`.

All dependencies are included in the `SoftSERVE` package. For a vanilla run of `SoftSERVE` this is essentially the `Cuba`[3] library of numerical integrators.

To setup `SoftSERVE` simply expand the contents of the tarball downloaded from `HEPForge` and run

```
./configure
```

This sets up the compile commands used later on. Running `make` does not happen at this stage, we use it to `make` the observable dependent executables later on.

The configuration script contains a short explanation of its features, simply call

```
./configure --help
```

Attention MacOS users:

If you are only running the `Xcode` package on your machine and do not have a dedicated, self-contained, `gcc` installation, you need to run

```
./configure CC=clang CXX=clang++
```

instead. If you do not, the configuration script may mistake `clang` for `gcc`, and subsequent compilations may fail.

[1]Using Cygwin is explicitly not recommended, and using MinGW will fail, both due to their implementation of `fork` or lack thereof.

In some cases (see 4.10) the program needs to internally use multiprecision data types, if the typical C++ `double` type variables do not provide enough digits to differentiate between two close-by numbers. For this case we include the header-only `boost`[4] library's implementation of multiprecision variables. This works out of the box and without modification of the installation of `SoftSERVE`.

However, the implementation of multiprecision floating point calculations via `boost` is slow, it is in fact roughly a factor 2 slower than the multiprecision backbone provided by the GMP[5] and MPFR[6] libraries[7]. We therefore ship the GMP and MPFR libraries as well, in case a user really needs the factor of 2.

If so, the GMP and MPFR libraries must be compiled during the `./configure` call, as well. The script should then be called via

```
./configure --MPFR
```

to enforce this.

As GMP is especially prone to miscompilation we do not recommend compiling those libraries unless you find that you really need both the multiprecision capability and the slight increase in performance.

It is possible to reconfigure with `./configure --MPFR` having already called `./configure` alone at some point before, so we highly recommend not including the `--MPFR` flag as the default option.

Actually using GMP or MPFR, rather than Boost, requires a small modification of the source code, see section 4.10 for details.

Chapter 3

Program flow

The actual step-by-step flow through the program is as follows:

Section 4.1	Bring the soft function into the correct, exponential form.
Section 4.2	Apply the parametrisation for the correlated and uncorrelated emission cases separately.
Section 4.3	Extract the input required for `SoftSERVE` from the correctly parametrised exponentials.
Section 4.4	Optional: Check again for compatibility with `SoftSERVE`.
Section 4.5	Write the observable details to the input files.
Section 4.6	Specify the details for the numerical integration.
Section 4.7	Compile the executables.
Section 4.8	Run the executables.
Section 4.9	Check the output for consistency.
Section 4.10	If necessary, recompile using multiprecision and run.
Section 4.11	Optional: Tidy up and get full result.
Section 4.12	Fourier space only: Adjust for phases.
Section 4.13	Optional: Renormalise either by hand or via script.

The reason the approach still requires unscripted action by the user is that maintaining full generality is difficult if the input for the script isn't restricted: We could provide scripts or Mathematica notebooks that handle the parametrisations, but these would likely often fail for intricate observables[1]. We therefore force the user to do the analytic part themselves, which e.g. also allows you to use shortcuts where possible, or to use non-trivial analytic properties to simplify expressions.

The next chapter contains all the details about the individual steps phrased as generically as possible. Then, in order to connect to the real world we provide a number of template observables, for both SCET-1 and SCET-2, in Appendix F. These templates describe what the flow of ideas and actions are that leads to the correct input files for each observable treated.

[1]Looking at you, Transverse Thrust...

Chapter 4

Detailed program steps

4.1 Exponential form

The form of the soft function we want to compute, complete to NNLO — two-loop — order, is derived in [1] as

$$\begin{aligned} S(\tau) = 1 &+ \left(\frac{Z_\alpha \alpha_s}{4\pi}\right) \left(\mu^2 \bar{\tau}^2\right)^\epsilon (\nu\bar{\tau})^\alpha \, S_R(\epsilon, \alpha) \\ &+ \left(\frac{Z_\alpha \alpha_s}{4\pi}\right)^2 \left(\mu^2 \bar{\tau}^2\right)^{2\epsilon} \Big((\nu\bar{\tau})^\alpha \; S_{RV}(\epsilon,\alpha) + (\nu\bar{\tau})^{2\alpha} \; S_{RR}(\epsilon,\alpha) \Big) + \mathcal{O}(\alpha_s^3), \end{aligned} \tag{4.1}$$

where the S_i are the Laurent series calculated by `SoftSERVE`. These are of the schematic form

$$S_i \sim \int \underbrace{\left|\mathcal{A}^{(i)}(\{k\})\right|^2}_{\text{Matrix element}} \overbrace{\exp(-\tau\omega(\{k\}))}^{\text{Measurement function}} \underbrace{\prod_{\substack{R,RV:p=k\\ RR:p=k,l}} 2\pi\,\delta^+(p^2) \; \overbrace{(p_+ + p_-)^{-\alpha}}^{\text{Analytic regulator}} \; \frac{\mathrm{d}^d p}{(2\pi)^d}}_{\text{Analytically regularised on-shell phase space}} \,. \tag{4.2}$$

The matrix elements $\mathcal{A}$ are fixed and the same for all possible dijet soft functions (merely different between loop and real emission numbers) and the analytic regulator is fixed by convention, leaving only the measurement function to be specified.

The constraints on the measurement function are, to recap [1], as follows:

- It is of the given exponential form.
- The variable τ has a unit of inverse energy, demanding that ω, which encodes the information about the observable, has mass dimension one.
- ω observes infrared and collinear safety, meaning in particular that the relations $\omega^{(2)}(k, \alpha k) = \omega^{(1)}([1+\alpha]k)$ and $\omega^{(2)}(k, 0) = \omega^{(1)}(k)$ between its 1- and two-particle instances hold.
- ω does not depend on the regulators ϵ and α. This can in some cases restrict the transformation to exponential form, if auxiliary Fourier or other transformations are involved.

- ω is real and non-negative in all of phase space (see appendix B of the first main paper[1] for purely imaginary cases). Note that this requirement for the numerical treatment is stricter than the assumption for the analytic formulae.
- ω vanishes or diverges only on hypersurfaces of at least codimension one of the full phase space. In other words, ω can vanish or diverge, but the measure of all those points must be zero. It cannot vanish on extended regions of the phase space for the emissions.
- It depends at most on one angle per emission, measured between the transverse space component of the emission momentum and a common reference vector in transverse space.

How this form, in particular the exponential, is achieved, is irrelevant. For many observables, such as event shapes like Thrust or C-Parameter, a Laplace transform will do it.

The Template Guide section at the end of the manual contains examples of several of these "standard" observables (Thrust, Threshold Drell-Yan,...), as well as one purely imaginary Fourier space observable (p_T resummation).

4.2 Parametrisations

Now we apply the two parametrisations for the correlated emission case (i.e. the terms in the matrix element appearing with $C_F C_A$ or $C_F T_F n_f$ colour factors) and the uncorrelated emission case (i.e. the C_F^2 contribution) to $\omega^{(2)}(k,l)$. These differ only in the way the light cone components of k and l are parametrised. The angular parametrisation is the same for both.

4.2.1 Correlated emissions

For the $C_F C_A$ and $C_F T_F n_f$ colour structures we parametrise the light-cone components of the two emission momenta as:

$$\begin{aligned} k_+ &= \frac{b}{a+b}\, p_T \sqrt{y} \qquad & l_+ &= \frac{a}{a+b}\, p_T \sqrt{y} \\ k_- &= \frac{ab}{1+ab}\, p_T \frac{1}{\sqrt{y}} \qquad & l_- &= \frac{1}{1+ab}\, p_T \frac{1}{\sqrt{y}}. \end{aligned} \tag{4.3}$$

We cover the full phase space for k_+, k_-, l_+, and l_- if p_T, a, b, $y \in [0, \infty[$.

The angular parametrisation in the transverse space, assuming the common reference vector is represented by a unit vector $\hat{v}_\perp$, is given by

$$\begin{aligned} \vec{k}_T \cdot \vec{l}_T &= \sqrt{k_+ k_- l_+ l_-}\cos\theta_{kl} =: \sqrt{k_+ k_- l_+ l_-}(1-2t_{kl}) \\ \vec{k}_T \cdot \hat{v}_\perp &= \sqrt{k_+ k_-}\, c_k =: \sqrt{k_+ k_-}(1-2t_k) \\ \vec{l}_T \cdot \hat{v}_\perp &= \sqrt{l_+ l_-}\, c_l =: \sqrt{l_+ l_-}(1-2t_l), \end{aligned} \tag{4.4}$$

where the magnitude of the transverse space vectors is set by the on-shell condition, and the coloured variables are coded in the C++ program and can be used as variables in the measurement functions. Whether c_k or t_k, or whether c_l or t_l are used is left to the preference of the user (t_{kl} cannot be represented by c_{kl}).

4.2.2 Uncorrelated emissions

For the C_F^2 colour structure we parametrise

$$
\begin{aligned}
k_+ &= \frac{b}{1+b}\, q_T \sqrt{y_k} \left(\frac{\sqrt{y_l}}{1+y_l}\right)^n & \qquad l_+ &= \frac{1}{1+b}\, q_T \sqrt{y_l} \left(\frac{\sqrt{y_k}}{1+y_k}\right)^n \\
k_- &= \frac{b}{1+b}\, q_T \frac{1}{\sqrt{y_k}} \left(\frac{\sqrt{y_l}}{1+y_l}\right)^n & \qquad l_- &= \frac{1}{1+b}\, q_T \frac{1}{\sqrt{y_l}} \left(\frac{\sqrt{y_k}}{1+y_k}\right)^n ,
\end{aligned}
\tag{4.5}
$$

where the value for n is observable dependent and can be derived from the correlated emission case, as will be shown in subsection 4.3.1. The full phase space for k_+, k_-, l_+, and l_- is covered if q_T, b, y_k, $y_l \in [0, \infty[$.

The angular integrations are parametrised as in the correlated emission case.

4.3 Extracting input

The input we need are the functions F_A, F_B, G_{Aa}, G_{Ab}, and G_B, and the the parameters n and m. The one-loop function f is recovered by the program directly via infrared safety.

Note: The notation for functions G_{Ai} differs from [2]: Here the index $i \in \{a, b\}$, while in the paper we used $i \in \{1, 2\}$. This change is purely cosmetic, of course (and a relic of different people being in charge of analytics and code).

Which input can be extracted depends on the parametrisation in use, and both cases are needed. The uncorrelated case depends on the value of the parameter n extracted from the correlated parametrisation case, it therefore is a good idea to start with the latter.

4.3.1 Correlated emission parametrisation

Focusing only on the measurement exponential for the two-loop contribution to the soft function, we find that it now can be written in the form

$$
\exp\left(-\tau\omega\left(k, l\right)\right) = \exp\left(-p_T y^{\frac{n}{2}} F(a, b, y, t, c_k, c_l)\right) . \tag{4.6}
$$

The p_T dependence factorises because p_T is the only surviving dimensionful quantity, and we factor out $y^{\frac{n}{2}}$ in such a way that F is finite in the limit $y \to 0$[1].

We can now already read off the value for the parameter n, and proceed to the functions F_X.

A quick recap to summarise [1]: The integration runs over domains $[0, \infty[$ for the variables a, b, y, which is bad for numerical integration, which needs finite integral boundaries. We therefore split each integration domain $[1, \infty[$ into two subdomains, $[0, 1]$ and $[1, \infty[$. Combining all possibilities for the different variables we find eight distinct 3-dimensional integration regions with different integration boundaries. One of these, $[0, 1] \times [0, 1] \times [0, 1]$, we identify as region A, and we identify the function F in equation (4.3.1) as the measurement function F_A.

[1] "Finite", as in "generically finite". There may still be combined limits involving y which vanish/diverge. The usual candidate is $y \to 0$, $c_k, c_l \to 0$. As long as $y \to 0$ alone is not sufficient for a zero/divergence, that's fine.

In the first main paper[1] we lay out how symmetry considerations reduce the other regions to region A and a second region, B, which is related to A by one of three possible mappings (angular variables suppressed):

- Replacing a with $\frac{1}{a}$
- Replacing b with $\frac{1}{b}$
- Replacing y with $\frac{1}{y}$

The measurement function F is in general not the same in the two regions A and B. To get F_B we therefore choose one of the three replacements[2], and apply it to the measurement exponential[3]. Then we bring the exponential into the same form as (4.3.1), and read off the function F, which we now label F_B.

Alternatively, we can apply the mappings directly to F_A, in which case we need to be careful to take the $y^{\frac{n}{2}}$ we factored into account, yielding as an alternative derivation scheme (angular variables suppressed):

- $F_B(a, b, y) = F_A(1/a, b, y)$
- $F_B(a, b, y) = F_A(a, 1/b, y)$
- $F_B(a, b, y) = y^{-n} F_A(a, b, 1/y)$

Finally, we pick any of the functions F_X and expand it in the limit of small y, which is of the form $F_X \approx c_0 + y^m c_1 + o(y^m)$, i.e. we extract the parameter m from the first non-constant term in the expansion of F. For most observables we will find $F_X \approx c_0 + y c_1 + \mathcal{O}(y^2)$ (i.e. $m = 1$), but there are also observables for which e.g. $F_X \approx c_0 + \sqrt{y}\, c_1 + \mathcal{O}(y)$, ($m = 0.5$), and there may even be other values allowed. If no such value can be extracted, or if the extracted value is zero, try again in the uncorrelated case.

4.3.2 Uncorrelated emission parametrisation

After applying the uncorrelated emission parametrisation to the two-loop measurement observable we recover the form[4]

$$\exp\left(-\tau\omega\left(k, l\right)\right) = \exp\left(-\tau\, q_T\, y_k^{\frac{n}{2}}\, y_l^{\frac{n}{2}}\, G(b, y_k, y_l, c_k, c_l, t)\right). \tag{4.7}$$

Splitting the integration domains and using slightly different symmetry considerations (see the second main paper[2]), we can again read off the function G_A immediately after applying the uncorrelated emission parametrisation, and find the functional form of G_B after applying one of

- y_k replaced by $\frac{1}{y_k}$
- y_l replaced by $\frac{1}{y_l}$,

to the exponential, again bringing it to the form of eq. (4.7), and then reading off G and assigning it the label G_B. We can alternatively directly use (irrelevant variables suppressed) one of

- $G_B(y_k, y_l) = y_k^{-n} G_A(1/y_k, y_l)$
- $G_B(y_k, y_l) = y_l^{-n} G_A(y_k, 1/y_l)$.

[2]The choice is yours, and which one is the best is observable-dependent. If your observable depends trivially on a and intricately on y, maybe invert a, rather than y.

[3]The functions F_B resulting from the three possible mapping need not be the same. All three will, however, yield the same results upon integration.

[4]The parameter n is already known from the correlated emission case, so factoring out $y_i^{\frac{n}{2}}$ is a bit easier now

To arrive at the functions G_{Aa} and G_{Ab}, one more step is required[5]: For subregions a and b we apply the substitutions

$$\begin{array}{llll} a: & y_k \to ry & y_l \to y \\ b: & y_k \to y & y_l \to ry \end{array} \tag{4.8}$$

to G_A and carry out any simplifications that are possible .

No such substitution is required in region B[6].

A final note: In case the value for the parameter m couldn't be derived in the correlated emission case, we expand G_B for small y_k (or y_l) and extract it from the series $G_B \approx c_0 + y_k^m c_1 + \mathcal{O}(y_k)$. If even after this we either can't find a value for m or we find $m \leq 0$, just use $m = 1$.

4.4 Optional: Checking compatibility

If you're at all uncertain if you've made a mistake, or if your programs threw up error messages that lead you to believe you might have made a mistake in the input extraction: This section lists the constraints as they manifest themselves on the input. You can skip this section if your functions F and G, and parameters m and n look reasonable to you.

If you're unsure, here are the relevant checks:

- We should now have for the correlated emission input:
  - The value for n
  - The two functions[7] $F_X(a, b, y, t_{kl}, c_k, c_l)$ with $X \in \{A, B\}$
  - The value for m, except for rare cases, where we look at the uncorrelated emission parametrisation.
- For the uncorrelated emission input:
  - The two functions[7] $G_{Ai}(b, r, y, c_k, c_l, t_{kl})$ with $i \in \{a, b\}$
  - The function[7] $G_B(b, y_k, y_l, c_k, c_l, t_{kl})$
  - The value for m, if it wasn't possible to extract it from the correlated emission case.
- All functions F and G must be non-negative in all, and finite and non-vanishing in most of phase space, meaning the regions b, a, y, t_{kl}, y_k, y_l, $r \in [0, 1]$, and c_k, $c_l \in [-1, 1]$ or t_k, $t_l \in [0, 1]$. Here "most" of phase space means that they vanish or diverge only on surfaces of codimension ≥ 1. As you will need it in the next step, it is advisable to make a list of all appearing zeroes and divergences here already.
- Any combination of one or more of the limits $y \to 0$, $b \to 0$, $y_k \to 0$, $y_l \to 0$, $r \to 0$, as well as the combined limit $(a, t_{kl}) \to (1, 0)$ must be finite. Only when other variables take on special values

[5]This step disentangles the joint "jet-collinear" limit $y_k, y_l \to 0$ into the joint limit of jet-collinearity at fixed rapidity difference ($r = \mathit{const}, y \to 0$) and successive jet-collinearity ($r \to 0, y \to 0$).

[6]The physical reason is that B represents emissions into different hemispheres, and A is the same-hemisphere case. The jointly-collinear limit is only relevant for emissions into the same hemisphere.

[7]c_k and c_l may be resolved in terms of t_k and t_l.

alongside them may the functions F and G vanish or diverge. If this constraint is violated you either made a mistake in the derivation of the functions F and G, or your observable is not infrared and collinear safe.

- The limits $b \to 0$ and $(a, t_{kl}, c_k) \to (1, 0, c_l)$ of the functions F must reduce to the same function[8], up to exchange of indices k and l.
- Similarly, the limits $b \to 0$ for all three functions G_{Xi}, and the limits $(r, t_{kl}, t_k) \to (1, 0, t_l)$ of the two functions G_{Aa} and G_{Ab}, as well as the limit $(y_k, t_{kl}, t_k) \to (y_l, 0, t_l)$ of the function G_B must reduce to the same function[9], up to relabelling.

4.5 Writing input files

Armed with the functions F and G and the parameters n and m we can now adjust the input files as needed.

First, we choose the branch of the program to be used based on the value for n. If $n \neq 0$, the correct branch is the SCET-1 branch. If $n = 0$, we must use the SCET-2 branch. We descend into the appropriate subdirectory of the main `SoftSERVE` package, and everything that follows is applicable to both cases identically.

4.5.1 `Input_Common.cpp` and `Input_Parameters.h`

We start with the input that is common for all colour structures, in `Input_Common.cpp`.

Here, we give the observable project a name by assigning it to the string called `observable`. The default is set to “Observable”.

We then turn to parametric dependence in the observable. Many observables depend on parameters, e.g. the Angularities event shape with its parameter A, or Transverse Thrust, which depends on a non-dynamical angle between jet- and beam-axes. These parameters must be properly declared and defined in the program, as they can’t be left parametric for a numerical evaluation.

To that end we need to add a line of the form

```
datatype name=value;
```

to the `Input_Common.cpp` file for a “datatype”-type parameter with name “name” and value “value”, and to make it visible to the rest of the program we add the line

```
extern datatype name;
```

[8] As a further consistency check you can check if this function matches the function f extracted from the one-particle measurement exponential if you parametrise the emission momentum l as $l_+ = p_T\sqrt{y}$, $l_- = p_T\frac{1}{\sqrt{y}}$, $\vec{l}_T \cdot \vec{n}_j = p_T c_l$, and write the exponential as $\exp(-\tau\omega(k)) = \exp(-\tau p_T y^{\frac{n}{2}} f(y, c_l))$

[9] Expressed in terms of the one-loop measurement function f from the preceding bullet point this function is $\frac{f(y_l,t_l)}{(1+y_l)^n}$

to the `Input_Parameters.h` file. Both locations are marked with appropriate comments in the respective files, and the default template does declare and define a double precision parameter called `placeholder`, and sets its value to 0.5.

The usual datatype for parameters will be `double`[10], and the name can be freely chosen, with the exception of a few names which are already taken by internal functions and variables of the program. A list of prohibited names are listed in table 4.1. This list is not complete, so if your code fails to compile, try giving the parameters a different name.

a	A	B	beta	c	ck	Ck	Ck1	Ck2	cl	Cl	decimal			
f	FA	FB	FA1	FA2	FB1	FB2	GAa	GAb	GA1a	GA1b	GA2a			
GA2b	GB	GB1	GB2											
n	m	M	mexp	mm	q	r	s	s5	s6	t	T	t5	tk	Tk1
Tk2	tkl	t6	u	v	w	x	xk	xl	y	yk	yl	z		

Table 4.1: List of prohibited parameter names. A large number of longer names of functions has been omitted, as we judge the chance of your parameters being called `maxpass` or `PreRVEM3` rather low.

Note: This list only applies to names of parameters you define for your observable. In the `Input` files you should of course use `a`, `y`, and all other variables directly — you can't use them as parameter names precisely because they're already taken by the integration variables.

Finally for `Input_Common.cpp` we input the value for n (SCET-1 case only) and m extracted before at the appropriate places[11].

4.5.2 `Input_Measurement_Correlated.cpp`

Here we insert the definitions of the functions F_X we derived using the correlated emission parametrisation below the giant comment stating "Measurement functions Correlated". There are two predefined C++ functions, one for each of the two functions, of the form (here for A)

```
double FA(v,t6,y,tk,u,b) {
((variable definitions))

   ((comment))
   FA=[[assignment]];
   [[catching zeroes or divergences]]

   ((fail-safe and return))
}
```

where double brackets denote a code fragment by its purpose in this write-up, and round vs. square brackets denote pre- vs. user-defined input. The F_X functions must be entered at `[[assignment]]` in C++ `cmath` syntax, and all zeroes or divergences must be set to a value other than 0 or ∞ in `[[catching zeroes or divergences]]`. This also applies to non-trivial limits of e.g. the form $\frac{a}{a+b}$,

[10]Though `integer` and `float` are also possible.

[11]Reminder: $m \overset{!}{>} 0$, and if that doesn't come out naturally, use $m = 1$.

which is analytically finite for all relevant variable values, but can look to a straightforward evaluation like the ill-defined $\frac{0}{0}$. It is advisable to treat this as an exercise in evading Murphy's Law: If there is any possible way for your function to be evaluated that results in superficially ambiguous expressions, C++ will almost certainly find it. Here we try to idiot-proof the input accordingly[12], via if-clauses that set the function to its correct value in these cases. There are a few template observables that show this problem.

The template code assumes a measurement function $F_X = \sqrt{a}(1 + \text{placeholder} * c_k)^2$, which is zero at $a = 0$. Accordingly, this zero is defused using an `if`-clause in `[[catching zeroes or divergences]]`. We suggest setting the function to 1 in such cases.

For the definition of angle dependent functions you can use either the predefined `ck` and `cl` as stand-ins for $\cos\theta_k$ and $\cos\theta_l$, or `tk` or `tl` for the t_i defined by $\cos\theta_i = 1 - 2t_i$, or you can mix and match.

If you feel the need to define internal variables to store intermediate results inside one of the functions F_X, please use `decimal` type variables wherever you'd usually use `double` or `float` type variables. This is related to multiprecision issues and is explained in appendix C.

For an argument about why setting arbitrary function values at zeroes is okay, as well as more information about the parameter m, which is related to this issue, see appendix A.

4.5.3 `Input_Measurement_Uncorrelated.cpp`

Editing this file proceeds analogously to `Input_Measurement_Correlated.cpp`, just that now we have to fill in G_{Aa}, G_{Ab}, and G_B, as derived using the parametrisation for uncorrelated emission.

The warning boxes of the preceding subsections apply here, as well.

4.6 (Kind of) optional: `Input_Integrator.cpp`

Now we proceed to the settings for the integrator, where we here list all settings and how they can be adjusted.

This is not strictly a required part of the calculation, you can just use one of the two predefined integrator settings templates. We (strongly) recommend using the `Standard` template settings, which should work well with most observables, and yield reasonable results in reasonable time. For exceptional cases in which higher precision is needed, a second template exists, as well.

First, note that some setting variables come with integer suffixes, like `epsrel2` or `key12`. This originates from a splitting in the number of integration variables, with the reasoning that lower dimensional integrals are easier to solve and will converge quicker to more precise values. So we can increase the precision of the overall result at fixed running time by computing the lower dimensional integrals really precisely, but allowing the higher dimensional ones to be a bit less precise, and having them all finish in reasonable time.

We find that we need between 2 and 5 integration dimensions, so we have variables like `epsrelI` with `I`$\in \{2, 3, 4, 5\}$.

The relevant settings now are:

[12]In the words of The Doctor: "The trouble with computers, of course, is that they're very sophisticated idiots."

`epsrelI` is the required relative accuracy, i.e. we demand that the numerical evaluation for the `I`-dimensional integral terminates only, once for a given integral estimate E and error estimate Δ the relation $\Delta \leq |\texttt{epsrelI} \cdot E|$ is fulfilled. Due to technical reasons the accuracy we achieve is typically of one to two orders of magnitude higher than the what is set here. Diminishing returns set in rather quickly — varying the default settings by more than two orders of magnitude will usually only increase computing time, not the precision.

`keyJI` Although the Cuba library[3] provides several numerical integrators, we only use one: the choice of *Divonne* is hard-coded. The three settings `key1I`, `key2I`, and `key3I` set the precise integration strategy *Divonne* uses for the `I`-dimensional integration, i.e. whether to use random number sampling or cubature rules, the choice of random number generator, etc.

For details we refer to the Cuba manual [3]. The default settings amount to a first partitioning stage using deterministic cubature rules, followed by a sampling stage using Korobov random numbers, followed by a refinement strategy.

We strongly suggest not changing these settings. Trial and error has shown that this combination of partitioning and sampling yields the best results - it is the origin of the two order of magnitude improvement we mention above; Korobov sampling improves on cubature. Other settings don't show this feature, or may even increase the error estimate in the sampling step.

`maxpassI` *Divonne*'s partition stage as a crude first approximation may miss important feature of the integrand. We therefore can force it to add an additional `maxpass` partitioning steps once it has concluded that it is ready to proceed to the sampling stage. This is the easiest dial to crank up precision, and the only difference between the standard and precision template settings is here.

This value is essentially arbitrary, but 25 yields good results for the Standard settings, and for the precision template we use values of several 100.

`maxchisq` and the next setting,

`mindev,` set the criteria for the refinement stage. *Divonne* partitions the integration region, and then samples. If the tentative result for a partition region from the partitioning stage, when compared to the final result from sampling, fails a χ^2-test by exceeding `maxchisq`, the subregion is sent to refinement if the error estimate of its contribution to the total error estimate is at least a fraction of `mindev` of the total error estimate. In other words, *Divonne* only refines if there is both something to refine and doing so is worth it.

The default values work well enough here. Play around with them at your leisure.

`border` This setting is the reason we hard-coded *Divonne*. It delineates an exclusion zone of width `border` at the boundaries of the integration region. Points inside this region are not sampled directly, but extrapolated from two points outside the exclusion zone, i.e. inside the bulk of the integration region. We use this feature because the integrand exhibits suppressed logarithmic divergences of the form $x \ln x$ at the integration boundaries. Using the `border` feature suppresses numerical instabilities. The full explanation can be found in appendix A.

We suggest varying the default value by two orders of magnitude at best — if it is too small we encounter numerical instabilities, and if it is too large we introduce systematic uncertainties. It is also related to the "`BOOST`" feature, which we encounter in section 4.10.

`maxevalI` This feature introduces a hard cutoff on the number of integrand function evaluations. We want in general to be guided by the error estimates, hence we choose a large value for these variables. Nevertheless, in some cases it can be nice to have a setting that just kills the integration if it takes too long, so we leave this here.

`seed` In case the integration strategy is changed (different `key` settings), `seed` governs the choice of random number generator. For details see the Cuba manual[3].

`flags` This variable sets the verbosity of the integrator. `flags=3` floods you with log files, whereas `flags=0` is silent. Choose intermediate integers freely.

`epsabsI` Purely maintained for compatibility. Where `epsrelI` sets a termination constraint on the relative error, `epsabsI` sets one on the absolute error. Whichever is fulfilled first, wins. For the numerical integration we separate constant prefactors from integrand structures, so `epsabsI` would set constraints on meaningless intermediate results, so we don't use it. It is listed purely in case somebody really needs it, for whatever reason.

4.7 Compiling executables

Having edited the input files we are now ready to compile and then run the executables for a given observable's soft function.

To that end we call `make` from the folder containing the input files and the makefile.

The makefile will compile the different object files containing numerator functions, prefactors and the input data, and link everything together to form five files in the `Executables` folder. The filenames are generated from the value of the `observable` string in `Input_Common.cpp`, by adding one of five suffixes:

`1P` This executable will calculate the regulator pole coefficients for the one-loop soft function and the two-loop real-virtual interference contribution, to the orders required for renormalisation[13]

`CFA` This yields one contribution to the C_F^2 colour structure, here the phase space is restricted to having both emissions in the same hemisphere[14]

`CFB` Computes the different-hemisphere contribution to the C_F^2 colour structure. The total C_F^2 structure is simply the sum of `CFA` and `CFB`, coefficient by coefficient.

`CA` denotes the executable for the $C_F C_A$ colour structure, absent the real-virtual interference one-particle cut contribution, which is computed by the `1P` executable

`NF` The $C_F T_F n_F$ contribution to the bare soft function is computed by the executable with this suffix.

`ADLap` This executable calculates the Laplace space soft anomalous dimension, using semi-analytic formulae.

`ADMom` This executable calculates the soft anomalous dimension following an inverse Laplace transform, using semi-analytic formulae.

The strategy using `make` is sensitive to recycling of object files. After the first run the input-independent object files do not have to be recompiled, and only files modified between `make` calls will be recompiled.

[13]i.e. there are positive regulator power coefficients computed in the one-loop result.

[14]The hemispheres are defined as separated by the plane perpendicular to the Wilson lines.

To single out individual colour structures for compilations and to provide auxiliary functions, the following targets have been defined:

`all`	Default, same as running `make` without a target, compiles all binaries
`list`	Prints a short card listing all targets and their function
`clean`	Removes the intermediate object files
`purge`	Removes the intermediate object files and all executables in the `Executables` folder adhering to the standard naming scheme (i.e. ending in one of the seven suffixes). Subfolders of `Executables` are not affected, and neither are log and result files
`1P`	Only compiles the `1P` executable
`CA`	Only compiles the `CA` executable
`NF`	Only compiles the `NF` executable
`CFA`	Only compiles the `CFA` executable
`CFB`	Only compiles the `CFB` executable
`correlated`	Only compiles the `1P`, `CA` and `NF` executables
`uncorrelated`	Only compiles the `CFA` and `CFB` executables
`ADLap`	Only compiles the `ADLap` executable
`ADMom`	Only compiles the `ADMom` executable

To use the `boost` library's multiprecision variables, which is described in detail in section 4.10, we set[15] `BOOST=1`.

Similarly, the `RRG` variable determines the regularisation strategy. Emission regularisation is the default, setting a non-zero `RRG` value switches to web regularisation.

So to compile the $C_F C_A$ colour structure executables using multiprecision variables we'd call

```
make CA BOOST=1
```

Without multiprecision variables, we just use

```
make CA
```

The `RRG` and `BOOST` settings are compatible, so to get the correlated emission executables using multiprecision variables and web regularisation, we call

```
make correlated BOOST=1 RRG=1
```

4.8 Running the executables

To run the executable simply either descend into the `Executables` folder and run them directly and manually, or run the `execsftsrv` script from the directory containing the input files and makefile. Its syntax is

[15]In principle any value different from 0 is fine, even strings. `BOOST=thecakeisalie` would work as well.

```
./execsftsrv -o observable -d subdir
```

where `observable` must match the name of the binaries without suffix[16], and `subdir` is an optional argument specifying a subdirectory in the `Executables` folder, in which the binaries may reside.

The `./execsftsrv` script runs the five executables in the order 1P $\rightarrow$ CFA $\rightarrow$ CFB $\rightarrow$ NF $\rightarrow$ CA, which corresponds for typical observables to the ordering "fastest-to-slowest", meaning that `1P` usually terminates quicker than `CFA` and `CFB`, which are faster than `NF`, with `CA` taking the longest. It then combines the five results into one final result, writes that to the file "`Result observable Full.txt`", and moves the executables to a subfolder for storage.

A variation of this script, `./execsftsrvNAE` (designed for non-Abelian exponentiating observables), performs the same job, except that it neither requires the uncorrelated emission binaries (`CFA` and `CFB`), nor does it call them. The final output will be of the same form as that of `./execsftsrv`, except that the C_F^2 results in the full result file are constructed from the square of the one-loop result.

The `./execsftsrv` and `./execsftsrvNAE` do *not* come in an `RRG` postfixed version, so for both emission- and web-based regularisation we call the same scripts[17].

The executables will run sequential numerical integrations over 2D (first) to 5D (last) domains. This means that for a given colour structure the leading, most divergent, regulator poles are the first results to be available. This strategy was chosen because mistakes in the input can then in some cases be spotted from the leading poles, which are available quickly.

During the executables' run the status and results are printed to the console, the results are also written to a result file, and the output of the numerical integration is funnelled into log files, with result and log files both in the resident folder of the executable that's running.

The `ADLap` and `ADMom` executables provide shortcuts to the usual chain of scripts and binaries, and they must be called manually from the folder in which they reside, as there are no scripts to call them from the main SCET-1/2 folders.

4.9 Consistency checks

So you've run the executables, your integrations have terminated and the results are in. Now we need to check for the single biggest problem facing our program: Rounding errors.

The details of why this problem appears can be found in appendix C, so here we only mention how to check for its appearance. Put succinctly the problem is related to the fact that `double` type variables in C++ only store about ~ 15 digits of a number, which can lead to problems if two almost identical numbers are subtracted. In most cases the appearance of the problem will manifest itself by the numerical evaluation returning `NaN`, but in principle it can lead to simply a wrong result which is not obviously wrong at first glance.

[16] i.e. if `observable` is "`abcd`", the script expects to find five binaries called "`abcd 1P`", "`abcd CA`", "`abcd NF`", "`abcd CFA`", and "`abcd CFB`" .

[17] The reason is that for the executables there is no difference between the two. Only some of the output results will be slightly different, and therefore the result files will constitute the only difference.

Fortunately for us the problem only appears in its critical form at the boundaries of the integration domain, which is subject to special treatment thanks to the `border`[18] variable.

To check for rounding errors we therefore suggest that you run the program at least twice, the second time with a settings for `border` which differs from the first by two orders of magnitude. 10^{-6} and 10^{-8} are good values, for example. If you have read appendix C and are confident that your observable is not affected, you can skip this step, at your own peril.

If you do run twice and do get different results between the two runs, or if one of the two runs returns `NaN`, follow the strategy in the next section, and see if that solves the problem. If the result is the same with different values for `border`, feel free to proceed immediately to the renormalisation in section 4.11.

4.10 Multiprecision

So your program either `NaN`ed or returns `border`-dependent values. The solution for this problem is to use internal variable types which resolve more digits than the 15 present in `double`. The simplest way of doing that is to only change the command used to compile the executables in section 4.7 and add `BOOST=1` to the `make` command. The files you then generate use multiprecision variables and should not be affected anymore. However, they will be significantly slower than "standard" files, so it is advisable to lower the requested accuracy in the `epsrel` and `maxpass` variables.

In some cases the programs are unacceptably slow, or the accuracy cannot be lowered significantly. In this case there is not much that can be done, but we can tease an additional factor of ~ 2 in run time out of the code, by using a different implementation of the multiprecision variables.

The standard implementation is provided by the `boost` library, which is a header-only library and therefore does not need to be compiled. The alternative implementations are provided by the GMP[5] and MPFR[6] libraries, which do need to be compiled. To do that, we ascend to the `SoftSERVE` main directory, and call the configurations script with the appropriate flag:

```
./configure --MPFR
```

This compiles the GMP and MPFR libraries, and writes the output of their configuration and compilation steps to the `GMPcompilation.log` and `MPFRcompilation.log` files. Once the configuration script has terminated, check these for potential errors.

Assuming GMP and MPFR have compiled correctly, you now need to choose the multiprecision provider you want to use. The standard is still the `boost` implementation, even if GMP and MPFR are now compiled. To change this open the problematic observable's `Input_Parameters.h`[19], and look for the line

```
#define BOOST_H
```

As per the comment preceding it we know what we want to do, namely we change it to

```
#define GMP_H
```

if we want to use the GMP library, or

[18]See section 4.6.

[19]Note that this means the setting is tied to one observable. This is not a global change on the level of the full framework.

```
#define MPFR_H
```

for the MPFR implementation.

If you now call the makefile with the `BOOST=1` setting, the executables are compiled using `boost-`, GMP- or MPFR-provided multiprecision variables, according to what you `#define`d

The executables compiled using `make` *without* setting `BOOST=1` are unaffected by all of this.

Now compute the bare soft functions again using the newly compiled executables and perform the consistency check again. If it still fails, something else is going on, so please check the troubleshooting chapter 5 in this case.

4.11 Tidying up and getting the full result

Once you are convinced that the result is correct and the integrations are finished, you find the results in the `Result observable XX.txt` files, where `observable` is replaced with the value of the `observable` string in the input files, and `XX` is the executable in question, i.e. `1P`, `CFA`, `CFB`, `CA` and `NF`.

To save you the trouble of having to perform the sum of `CFA` and `CFB` results for the full C_F^2 colour structure, and the sum of `CA` and the real-virtual correction from `1P` for the full $C_F C_A$ structure — and in particular to save you the trouble of having to combine the error estimates —, we provide a script to do that for you. Once all result files for a given observable are in, you can call

```
./sftsrvres -o observable -d subdir
```

where `observable` is the value of the `observable` string in the input files, surrounded by quotes if it contains blanks, and `subdir` is the name of the subdirectory of the `Executables` folder that contains the five individual results files. The latter argument is optional and can be omitted, if the files are in `Executables` directly. `./sftsrvres -h`. or `./sftsrvres --help` provides help and an example.

The `./execsftsrv` script calls this script automatically, and the final result is written to the file `Result observable Full.txt`, again with `observable` specified in the input.

As for `./execsftsrv`, there is a variant for non-Abelian exponentiating observables, `./sftsrvresNAE`. This script still combines the separate $C_F C_A$ results, and writes a nice comprehensive result file, but it does again not use any explicit C_F^2 input, and the result for that colour structure written into `Result observable Full.txt` is constructed from the square of the one-loop result.

Finally, two more scripts exist carrying the `RRG` suffix, designed for web regularised calculations in vanilla (`./sftsrvresRRG`) and NAE-expecting (`./sftsrvresNAERRG`) flavours.

4.12 Fourier-adjusting the results (Fourier space functions only)

If you're wondering if you have to follow this section's instructions, you very likely don't. In this case, feel free to skip ahead to the renormalisation section.

If you have read the relevant appendix in the first main paper[1] and are calculating soft functions involving imaginary measurement functions, this is for you.

In an appendix of the first main paper[1] we outline how we can compute the real part of Fourier space soft functions using `SoftSERVE`, by running it on the absolute value of the measurement function, and subsequently multiplying the result with $\cos\frac{(2\epsilon+\alpha)\pi}{2}$, $\cos\frac{(4\epsilon+\alpha)\pi}{2}$, or $\cos(2\epsilon+\alpha)\pi$, for one-loop, two-loop interference, and two-loop two-particle results, respectively[20].

To spare the user the effort of having to do this multiplication manually we provide a script to do it, instead. The `./fourierconvert` script requires the presence of the results files for the `SoftSERVE` run mentioned above (either combined or a full set of individual results files), and is called via `./fourierconvert -o observable -d subdir`, with `observable` the name of the observable, as provided to the `SoftSERVE` input (and as appearing in the names of the results files, mainly), and `subdir` the optional name of the subdirectory of the `Executables` folder, in which the results files reside.

The script performs the required linear combinations of existing regulator orders with the relevant powers of π coefficients (the cosine expanded), and calculates the modified error bars via the sum of squares. It then writes the adjusted results and error bars to a file mirroring the structure of `SoftSERVE` results files for an observable called `Fourier_observable`, where `observable` was the name of the original input.

So for an observable called `whatevs` renormalising multiplicatively in Fourier space, with (imaginary) measurement function iF, the `SoftSERVE` sequence would be

1. Write input files using the measurement function $|F|$
2. Compile the binaries
3. Call `./execsftsrv -o whatevs` to run the integrations
4. Call `./fourierconvert -o whatevs` to adjust the results and account for the cosine factors
5. Call `./laprenorm -o Fourier_whatevs` to renormalise (see below)

As with the other scripts, the non-Abelian exponentiation compatible version `./fourierconvertNAE` bypasses explicit results from uncorrelated emission binaries and generates any C_F^2 output from the one-loop square, and the `./fourierconvertRRG` script converts web regularised results. `./fourierconvertNAERRG` combines both functionalities.

4.13 Renormalising

Having derived the full bare soft function we now want to renormalise. In principle we should now undo the transformations we performed in section 4.1 to arrive at the soft function in its original space, and then renormalise there. We leave this task for complicated transformations to the user.

However, there are two types of observables which appear relatively often, for which we have scripted the renormalisation. These two observable types are

`type 1` observables whose soft function renormalise multiplicatively in the space (usually Laplace space) in which the calculation was performed[21].

[20] With $\alpha = 0$ in the SCET-1 case, of course, and slightly different factors for the web regularisation case.

[21] Soft functions renormalising multiplicatively in Fourier space treated as outlined in appendix B of the first main paper may also fall in this category, so don't forget to use the `./fourierconvert` script correctly. Everything else in this section applies as it does to observables of type 1 in Laplace space — just mentally replace "Laplace" with "Fourier" wherever you encounter it.

`type 2` observables whose soft functions renormalise multiplicatively following an inverse Laplace transform.

With "multiplicatively" we mean that the bare and renormalised soft functions are related by multiplication with a Z-factor, rather than a convolution with it.

Assuming we have a full set of result files for the individual executables in section 4.8, or a combined result file generated via script 4.11, we can now call either

- the `./laprenorm` script for type 1 observables, or
- the `./momrenorm` script for type 2 observables.

The scripts use the syntax `./laprenorm -o observable -d subdirectory -n nvalue` for the SCET-1 case, or `./laprenorm -o observable -d subdirectory` for the SCET-2 case, and extract the anomalous dimensions and matching corrections (SCET-1), or anomaly exponents (SCET-2) following the conventions used in [1] and [2], especially the "Renormalisation" and "Numerical implementation" sections.

`observable` must be replaced with the string in the `observable` variable of the input files, and the `subdirectory` and `nvalue` options are optional, specifying a subfolder of the `Executables` folder which contains the result files on which the script operates, and the value of the parameter n, respectively. The `-n` option only exists in case the script cannot read the value from the results files, in which case you'll be asked to provide it manually via this option. If either `observable` or `subdirectory` contain blank spaces, they must be enclosed by single or double quotes, or escaped using backslashes.

In both cases the renormalisation script requires either a set of all five individual result files, or one combined result file as output by the programs and scripts described in the sections above.

All scripts come with the flags `--help` and `-h`, which list their features again.

Examples for type 1 observables renormalising in calculation space are (Laplace space) event shapes like C-Parameter or Angularities, as well as observables like Drell-Yan production at threshold, or weak boson production at large transverse momentum, and (Fourier space) transverse momentum resummation. An example for type 2 observables are jet vetoes.

As before we have non-Abelian exponentiation compatible scripts `./laprenormNAE` and `./momrenormNAE` which bypass explicit results from uncorrelated emission binaries. Any C_F^2 results derived here are from the one-loop square. And `./laprenormRRG` and `./laprenormNAERRG` implement web regularisation and the extraction using the rapidity renormalisation group.

Chapter 5

Troubleshooting and tips and tricks

This list will be periodically updated as more ‘trouble to be shot’ is reported to the developers.

5.1 General grievances

- GMP and MPFR will fail to compile if the absolute `SoftSERVE` directory path contains spaces (programs using autotools generally have this problem).
- Functions exhibiting zeroes or divergences in the bulk of the integration region (e.g. p_T resummation) will yield results at reduced accuracy. This is because the zero/divergence generates a logarithmic divergence in the numerical integrand, which introduces instabilities. Such observables can also produce `NaN` as an integration result if the accuracy goal is chosen too ambitiously. This is essentially the numerical integrator giving up.
- Results for the higher regulator orders tend to appear with an insecure error estimate (“Probability of incorrect error estimate: 1”). This warning tends to be overly paranoid, mainly because these results consist of multiple terms originating from multiple integrations that are added up for the final result. `SoftSERVE` estimates the probability that the final result has incorrect error bars as the probability that at least one contributing intermediate result has incorrect error bars, which is the most conservative way to estimate this. Increasing the accuracy can get rid of this warning, but in many cases it can be ignored, if there are no other warning signs (like extensive refining required in the main integration part of the numerical integration, as seen by the appearance of “Split” in the log files, which indicates a complicated structure of the integrand not resolved by the partitioning stage), as the error bars of lower dimensional integrations tend to be very conservative (For known literature results the error bars tend to overestimate the actual deviation by up to two orders of magnitude) and partially make up for the higher dimensional ones.
- Scripts prefer full results files over individual results files. So if you run a full set of binaries via `./execsftsrv`, and then detect a mistake in one of the binaries, and only correct that one and run it again (yielding one corrected individual results file), renormalisation scripts will still use the old, incorrect, combined results file. So try to call binaries using `./execsftsrv` wherever possible, or use `./sftsrvres` liberally to keep the combined bare results up to date.

- Renormalising a Fourier space observable using the `./laprenorm` script still mentions "Laplace space renormalised [. . .]". This can be ignored, the script was written for Laplace space, it just can be used for Fourier space by sheer accident.

5.2 Cryptic compilation and run time error messages and their meaning

- Cuba 4.2 throws up compiler warnings (can be found in the logfile) indicating that vanilla C does not like floating point variables in some places, that some sources are deprecated, and that some libraries are built in deterministic mode. These are not a problem, the libraries compile just fine.

5.3 Tips and tricks

- Executables for machines other than the one used to compile them can be produced by statically linking against the libraries. To do so set `CXXFLAGS=-static` during the `make` call.
- The number of cores the Cuba library uses is dynamically determined by the number of idle cores. If you want to force Cuba to use a fixed number of cores, set the environment variable `CUBACORES` to however many cores you want to use.

Chapter 6

License and contact details

This program uses parts of the `boost`, and GMP and MPFR libraries, which are released under the Boost Software License, dual Lesser GNU Public Licence v3 and GPLv2, and LGPLv3, respectively.

We therefore release `SoftSERVE` under the terms of the GPLv3. Its full text can be found in the `COPYING` file in the `SoftSERVE` main directory.

The developers of `SoftSERVE` are Guido Bell, Rudi Rahn and Jim Talbert. You can contact us at softserve@projects.hepforge.org.

Appendix A

Endpoint suppression

Here we look at some transformations that can improve the convergence rate of the numerical integration, and simplify the behaviour at the boundaries of the integration region.

All of these transformations and substitutions are handled internally by the program and are mostly invisible to the user. However, in some cases knowledge of internal mechanisms is important and influences how the user should operate. We therefore explain these transformations here.

A.1 Non-critical boundaries

The problem we try to solve appears due to the presence of structures like $[t(1-t)]^{-\frac{1}{2}-\epsilon}$ from the angular parametrisation and $b^{-2\epsilon}$ from the regularised phase space measure[1]. We have to expand to subleading orders in ϵ, which means that we find square root and logarithmic divergences in the integrand. These are integrable divergences, but they potentially pose a problem for a numerical integration, which samples points in the integration region and therefore should ideally not have to sample the vicinity of an unbounded peak.

Moreover, the presence of square root divergences is incompatible with numerical integration routines that rely on variance reduction techniques, as *Divonne* and the Cuba library in general do. This is due to the fact a numerical integrator essentially computes an estimate for the mean of the integrand function probabilistically, and uses that the variance of this estimate (which serves as a good error estimate) is determined by the variance of the integrand function and the sample size. The variance of the integrand function is usually not known, but can be estimated from the sample variance.

The problem is now that for square-root-divergent integrand functions the variance of the integrand is proportional to $\int \mathrm{d}x\ f(x)^2 \sim \int \mathrm{d}x\ \frac{1}{x}$, which is infinite. The sample variance on the other hand is always finite. So an adaptive integration technique which seeks to reduce the variance to reduce the error estimates and to improve the integration precision will fail, because it operates under the wrong assumption that the variance of the integrand function is finite[2].

[1]The latter appears only in the $C_F T_F n_f$ structure, the former appears in all cases.
[2]As a side note: The *Vegas* algorithm is less susceptible to this problem, but *Divonne* is very sensitive to it.

Fortunately we can solve this problem by creatively substituting — We substitute for purely logarithmic divergences at 0:

$$\int_0^1 \mathrm{d}b\, b^{-2\epsilon} X(b) = \int_0^1 \mathrm{d}c^2\; c^{-4\epsilon} X(c^2) = 2\int_0^1 \mathrm{d}c\; c^{1-4\epsilon} X(c^2) \approx 2\int_0^1 \mathrm{d}c\; X(c^2)(c - 4\epsilon\, c \ln c + \mathcal{O}(\epsilon^2)), \quad \text{(A.1)}$$

where we take X to be a finite function containing all other factors in our problem, and have made explicit that the logarithmic divergences that appear in the regulator expansion are now suppressed and of the form $c \ln^n c$.

We can extend this to square root divergences (here at v=0) by substituting to higher powers:

$$\int_0^1 \mathrm{d}v\; v^{-\frac{1}{2}-\epsilon} X(v) = \int_0^1 \mathrm{d}w^4\; w^{-2-4\epsilon} X(w^4) = 4\int_0^1 \mathrm{d}w\; w^{1-4\epsilon} X(w^4) \approx 4\int_0^1 \mathrm{d}w\; X(w^4)(w - 4\epsilon\, w \ln w + \dots \quad \text{(A.2)}$$

Finally, we can even deal with divergences at 0 and 1, by substituting

$$t = 1 - (1 - s^i)^j \quad \text{(A.3)}$$

This scales as js^i for small s, and $i(1-s)^j$ for small $1-s$, and can therefore suppress appropriate divergences at both ends. As an example, if there is a logarithmic divergence at 0 and a square root divergence at 1, we'd choose $i = 2$ and $j = 4$, whereas for square-root divergences at both ends we'd use $i = j = 4$.

The integrand functions we use are full of such divergences, we therefore suppress most limits that are not associated with a divergence.

A.2 Consequences for the measurement function

The endpoint suppression has interesting consequences for the measurement function.

As can be found in [1], the measurement functions F only appear as $F^{4\epsilon}$ in the integrand, which means that at higher orders in ϵ they contribute at best integrable logarithmic divergences. If now such a zero/divergence for F coincides with a suppressed limit, we never have to worry about the divergence, and we never have to worry about $\ln F = \ln 0$ not being defined: The logarithm is suppressed and of the form $0 \ln 0$. In all those cases we can therefore include in the input C++ files an `if`-clause which sets the measurement function to any positive number other than zero.

This is the reasoning behind many of the clauses in the template observables — If the observable vanishes at the combination $a = 0$, $b = \sqrt{e}^{\pi}$, we do not have to worry about the value for b in the combined limit, as $a = 0$ is always suppressed.

In table A.1 we list all the limits which are suppressed, and some which are not suppressed. For the typical user, the measurement function will vanish or diverge in some limits. If the limit in which it is zero/divergent is one of the suppressed limits above, the measurement function at the zero/divergence[3] can be reset to a harmless value without changing the integral at all.

Critical limits on their own can never lead to a vanishing or divergent measurement function, based on infrared and collinear safety, but they have their own way of creating trouble. We therefore list them and employ a special suppression technique for them as well, as will be explained in section A.3.

[3]which `SoftSERVE` would complain about by throwing up error messages.

Limit	**divergent**	**suppressed**
$b \to 0$	logarithm ($C_F T_F n_f$) critical ($C_F C_A$)	yes yes
$b \to 1$	no, finite	no
$a \to 0$	logarithm	yes
$a \to 1$	critical	yes
$y \to 0$	critical	yes
$y \to 1$	no, finite	no
$t_{kl} \to 0$	square root	yes
$t_{kl} \to 1$	critical	yes
$c_l \to -1$	square root	yes
$c_l \to 1$	square root	yes
$c_k \to -1$	square root ($t_5 \neq 0$) critical ($t_5 = 0$)	yes yes
$c_k \to 1$	square root ($t_5 \neq 0$) critical ($t_5 = 0$)	yes yes

Limit	**divergent**	**suppressed**
$b \to 0$	critical	yes
$b \to 1$	no, finite	no
$y_{k,l} \to 0$	critical	yes
$y_{k,l} \to 1$	no, finite	no
$t_{kl} \to 0$	square root	yes
$t_{kl} \to 1$	square root	yes
$c_l \to -1$	square root	yes
$c_l \to 1$	square root	yes
$c_k \to -1$	square root ($t_5 \neq 0$) critical ($t_5 = 0$)	yes yes
$c_k \to 1$	square root ($t_5 \neq 0$) critical ($t_5 = 0$)	yes yes

Table A.1: In this table we lay out the endpoint suppression as implemented in the correlated (left) and uncorrelated (right) emission case. The mechanism for the suppression of logarithmic and root divergences was explained in the preceding paragraphs. The suppression mechanism for critical limits will be explained in the next section.

So what about zeroes/divergences which are *not* suppressed, because they are e.g. inside the bulk on the integration region? In these cases the measurement function can be set to a harmless value based on the reasoning that for a logarithmic divergence at x_0 the contribution of the interval $[x_0 - \delta, x_0 + \delta]$ to the full integral vanishes as $\delta \to 0$. We can therefore argue that even in these cases, setting the measurement function to a non-zero value at the critical point should not change the integral estimate, as it is determined by the entire peak, and not just the immediate neighbourhood of the divergence. In practice, the mere existence of the peak changes the way the numerical integrator works, and yields results that are less precise. The integrator essentially tries to sample a divergence using finitely many points. This leads to a reduced numerical accuracy and instabilities if the desired accuracy is set too ambitiously.

A.3 Critical limits

One reason we suppressed the endpoints was so that the integrand function, including all numerator, Jacobian and other functions, tends to zero at the integration boundaries, to avoid potential boundary divergences from contributing and screwing up the numerical integration via infinite variance or other divergence effects.

Unfortunately, the presence of plus-distributions complicates things, so let's recap what a plus-distribution does:

In our calculation the plus distributions appear (here for a fictitious variable x) in the combination

$$[x^{-1+n\epsilon}]_+ R(x) \approx \frac{R(x) - R(0)}{x} + n\epsilon \frac{\ln x}{x}[R(x) - R(0)] + \mathcal{O}(\epsilon^2), \tag{A.4}$$

where $R(x)$ contains, in our case, numerator and measurement functions, Jacobians, etc., and is in general ϵ-dependent, which we shall ignore in this discussion.

Near $x = 0$, we can see the problem if we expand

$$\begin{aligned}
[x^{-1+n\epsilon}]_+ R(x) &\approx \frac{R(0) + xR'(0) + \mathcal{O}(x^2) - R(0)}{x} + n\epsilon \frac{\ln x}{x}[R(0) + xR'(0) + \mathcal{O}(x^2) - R(0)] + \mathcal{O}(\epsilon^2) \\
&= R'(0) + n\epsilon \ln(x) R'(0) + \mathcal{O}(x, \epsilon^2).
\end{aligned} \tag{A.5}$$

We see that the plus-distribution gets rid of the x^{-1} divergence by throwing away the constant term in the function that multiplies it, enabling a cancellation. The integrand function then has a surviving logarithmic divergence at $x = 0$, which is integrable and a nuisance, from our point of view. Worse, if $R(x)$ doesn't have such a nice expansion, but expands as e.g. $R(x) \approx r_0 + r_1\sqrt{x} + \mathcal{O}(x)$, we'd find

$$\begin{aligned}
[x^{-1+n\epsilon}]_+ R(x) &\approx \frac{r_0 + r_1\sqrt{x} + \mathcal{O}(x) - r_0}{x} + n\epsilon \frac{\ln x}{x}[r_0 + r_1\sqrt{x} + \mathcal{O}(x) - r_0] + \mathcal{O}(\epsilon^2) \\
&= \frac{r_1}{\sqrt{x}} + n\epsilon r_1 \frac{\ln x}{\sqrt{x}} + \mathcal{O}(x^0, \epsilon^2),
\end{aligned} \tag{A.6}$$

which has a square root divergence, our nemesis when it comes to numerical integration.

Fortunately, the substitutions we used to suppress the integrand at the endpoints also work here. If the function R expands as $R(x) \approx r_0 + r_1 x^m + o(x^m)$, we can substitute $x = y^{\frac{2}{m}}$, and find

$$\begin{aligned}
\int_0^1 \mathrm{d}x\, [x^{-1+n\epsilon}]_+ R(x) &= \int_0^1 \mathrm{d}y^{\frac{2}{m}}\, [y^{-\frac{2}{m}+n\frac{2}{m}\epsilon}]_+ R(y^{\frac{2}{m}}) = \frac{2}{m}\int_0^1 \mathrm{d}y\, [y^{-1+n\frac{2}{m}\epsilon}]_+ R(y^{\frac{2}{m}}) \\
&= \frac{2}{m}\int_0^1 \mathrm{d}y\, \frac{R(y^{\frac{2}{m}}) - R(0)}{y} + \frac{2n\epsilon}{m}\frac{\ln y}{y}[R(y^{\frac{2}{m}}) - R(0)] \\
&= \frac{2}{m}\int_0^1 \mathrm{d}y\, \frac{r_0 + r_1(y^{\frac{2}{m}})^m + o((y^{\frac{2}{m}})^m) - r_0}{y} + \frac{2n\epsilon}{m}\frac{\ln y}{y}[R(y^{\frac{2}{m}}) - R(0)] \\
&= \frac{2}{m}\int_0^1 \mathrm{d}y\, r_1 y + o(y) + \frac{2n\epsilon}{m}\frac{\ln y}{y}[r_1 y^2 + o(y^2)] + \mathcal{O}(\epsilon^2) \\
&= \frac{2}{m}\int_0^1 \mathrm{d}y\ \, r_1\, y + \frac{2n\epsilon}{m}\, r_1\, y \ln y + o(y, \epsilon),
\end{aligned} \tag{A.7}$$

which is suppressed at $y = 0$.

A careful analysis of the integrand functions shows that the leading variable dependence in the function R is set — in the actual real world case of our integrand function — by the measurement functions F_X and G_X , and that we therefore need to make the parameter m observable dependent. Infrared and collinear safety allow us to determine how the observable scales in most of the critical variables, with the rapidity type variables being the exception, i.e. y in the correlated emission case, and y_k and y_l in the uncorrelated case.

For small y_k, y_l, the correlated emission variable y scales as y_l or y_k, respectively. We therefore extract m from the correlated emission case and reuse it in the uncorrelated emission case. This explains why we first try to extract m from the correlated emission case. If this fails we try the uncorrelated case, and if that still fails we use $m = 1$, which is the default value that suppresses the endpoints if R has a nice and standard Taylor expansion.

Appendix B

Computing parametrisation

In this chapter we list — for completeness' sake — the master formulae in the actual parametrisation used by the program. This form arises after all relevant endpoints are suppressed, as detailed in the previous chapter (following a reparametrisation getting rid of an overlapping divergence in the correlated emission case).

B.1 Correlated emission case

In the correlated emission case we first reparametrise to get rid of the overlapping collinear divergence at $(a,t) = (1,0)$. To do this we introduce new variables u and v via

$$a = 1 - u(1-v) \qquad t_{kl} = \frac{u^2 v}{1-u(1-v)}. \tag{B.1}$$

This leads to a critical divergence at $u = 0$, the new form of the collinear divergence, and square root divergences at $u = 1$ and $v = 0$. Having solved the overlapping divergence we reparametrise to suppress the square root and logarithmic divergences for the relevant variables listed in the previous chapter. We therefore introduce:

$$\begin{aligned} u &= 1-(1-z^2)^4 & v &= w^4 \\ b &= c^2 & t_l &= 1-(1-s_l^4)^4 \\ y &= x^M & t_5 &= s_5^2 \end{aligned} \tag{B.2}$$

The exponent M is related to the parameter m and adjusted to suppress the relevant critical limit. It is set to $M = \frac{2}{m}$ if $m \in]0,1]$, and $M = 2$ otherwise.

Using these reparametrisations, and splitting the $C_F C_A$ contribution into a part that matches $C_F T_F n_f$'s divergence structure — called Pseudo-n_f or PNF — and the rest — called Rest-C_A or RCA — we find the master formulae for `SoftSERVE` below. For `SoftSERVE` the monomials marked in blue contribute regulator poles, which are made explicit by introducing plus-distributions. Following this subtraction the complete expressions are expanded in the regulators and can be used for numerical integration. For brevity the argument dependence of the measurement functions F_X has been suppressed, and the labels

$i = 1, 2$ on F_{Xi} correspond to the two distinct substitutions for the angular variable t_k in terms of the angular variables t, t_l and t_5 introduced in [1] (or their resubstituted versions z, w, s_l and s_5 as introduced above). The expressions are then given by:

$$
\begin{aligned}
S_{n_f} = \int_0^1 & \mathrm{d}z\,\mathrm{d}c\,\mathrm{d}w\,\mathrm{d}x\,\mathrm{d}s_5\,\mathrm{d}s_l\,\frac{2^{17-4\epsilon}M\epsilon\Gamma[-2\alpha-4\epsilon]}{e^{2(\epsilon+\alpha)\gamma_e}\pi^{\frac{3}{2}}\Gamma[\frac{1}{2}-\epsilon]\Gamma[1-\epsilon]} z^{-1-4\epsilon}\,x^{-1+M\alpha+2nM\epsilon}\,s_5^{-1-2\epsilon} \\
&\cdot \frac{1}{(1+w^4)^3} c^{1-2\alpha-4\epsilon} w^{1-4\epsilon} s_l^{1-4\epsilon}\left(F_{A1}^{4\epsilon+2\alpha}+F_{A2}^{4\epsilon+2\alpha}+F_{B1}^{4\epsilon+2\alpha}+F_{B2}^{4\epsilon+2\alpha}\right)(2-s_5^2)^{-1-\epsilon} \\
&\cdot (1-z^2)^{1-4\epsilon}(1-s_l^4)^{1-4\epsilon}\left((2-s_l^4)(2-2s_l^4+s_l^8)\right)^{-\frac{1}{2}-\epsilon}\left((2-z^2)(2-2z^2+z^4)\right)^{-1-2\epsilon} \\
&\cdot \left(1+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{-\frac{1}{2}-\epsilon}\left((1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{1-\alpha} \\
&\cdot \left(1+c^2\left((1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)\right)^{-2+2\alpha+2\epsilon} \\
&\cdot \left(c^2+(1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{-2+2\epsilon}\Bigg[-4w^4\Bigg(c^2+w^4z^2\Big(2-z^2\Big) \\
&\cdot \Big(z^4-2z^2+2\Big)+\Big(1-z^2\Big)^4\Bigg)\left(1+c^2\Big(w^4z^2\big(2-z^2\big)\big(z^4-2z^2+2\big)+\big(1-z^2\big)^4\Big)\right) \\
&- \big(1-c^2\big)^2\big(1-w^4\big)^2\Big(w^4z^2\big(2-z^2\big)\big(z^4-2z^2+2\big)+\big(1-z^2\big)^4\Big)\Bigg],
\end{aligned}
$$

$$
\begin{aligned}
S_{PNF} = \int_0^1 & \mathrm{d}z\,\mathrm{d}c\,\mathrm{d}w\,\mathrm{d}x\,\mathrm{d}s_5\,\mathrm{d}s_l\,\frac{(\epsilon-1)2^{4(4-\epsilon)}M\epsilon\Gamma[-2\alpha-4\epsilon]}{e^{2(\epsilon+\alpha)\gamma_e}\pi^{\frac{3}{2}}\Gamma[\frac{1}{2}-\epsilon]\Gamma[1-\epsilon]} z^{-1-4\epsilon}\,x^{-1+M\alpha+2nM\epsilon}\,s_5^{-1-2\epsilon} \\
&\cdot \frac{1}{(1+w^4)^3} c^{3-2\alpha-4\epsilon} w^{1-4\epsilon} s_l^{1-4\epsilon}\left(F_{A1}^{4\epsilon+2\alpha}+F_{A2}^{4\epsilon+2\alpha}+F_{B1}^{4\epsilon+2\alpha}+F_{B2}^{4\epsilon+2\alpha}\right)(2-s_5^2)^{-1-\epsilon} \\
&\cdot (1-z^2)^{1-4\epsilon}(1-s_l^4)^{1-4\epsilon}\left((2-s_l^4)(2-2s_l^4+s_l^8)\right)^{-\frac{1}{2}-\epsilon}\left((2-z^2)(2-2z^2+z^4)\right)^{-1-2\epsilon} \\
&\cdot \left(1+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{-\frac{1}{2}-\epsilon}\left((1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{1-\alpha} \\
&\cdot \left(1+c^2\left((1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)\right)^{-2+2\alpha+2\epsilon} \\
&\cdot \left(c^2+(1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{-2+2\epsilon} \\
&\cdot \Big[2+w^4z^2\left(2-z^2\right)\left(2-2z^2+z^4\right)-z^2\left(2-z^2\right)\left(2-2z^2+z^4\right)\Big]^2,
\end{aligned}
$$

$$S_{RCA} = \int_0^1 \mathrm{d}z\,\mathrm{d}c\,\mathrm{d}w\,\mathrm{d}x\,\mathrm{d}s_5\,\mathrm{d}s_l\,\frac{2^{4(4-\epsilon)}M\epsilon\Gamma[-2\alpha-4\epsilon]}{e^{2(\epsilon+\alpha)\gamma_e}\pi^{\frac{3}{2}}\Gamma[\frac{1}{2}-\epsilon]\Gamma[1-\epsilon]} z^{-1-4\epsilon}\, x^{-1+M\alpha+2nM\epsilon}\, s_5^{-1-2\epsilon}$$
$$\cdot c^{-1-2\alpha-4\epsilon}\frac{1}{(1+w^4)}w^{1-4\epsilon}s_l^{1-4\epsilon}\left(F_{A1}^{4\epsilon+2\alpha}+F_{A2}^{4\epsilon+2\alpha}+F_{B1}^{4\epsilon+2\alpha}+F_{B2}^{4\epsilon+2\alpha}\right)(2-s_5^2)^{-1-\epsilon}$$
$$\cdot(1-z^2)^{1-4\epsilon}(1-s_l^4)^{1-4\epsilon}\left((2-s_l^4)(2-2s_l^4+s_l^8)\right)^{-\frac{1}{2}-\epsilon}\left((2-z^2)(2-2z^2+z^4)\right)^{-1-2\epsilon}$$
$$\cdot\left(1+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{-\frac{1}{2}-\epsilon}\left((1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{-\alpha}$$
$$\cdot\left(1+c^2\left((1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)\right)^{-1+2\alpha+2\epsilon}$$
$$\cdot\left(c^2+(1-z^2)^4+w^4z^2(2-z^2)(2-2z^2+z^4)\right)^{-1+2\epsilon}$$
$$\cdot\Bigg[2c^2\Big(w^4z^2\big(2-z^2\big)\big(z^4-2z^2+2\big)+\big(1-z^2\big)^4\Big)$$
$$-\left(1-\frac{2w^4z^4(2-z^2)^2(2-2z^2+z^4)^2}{w^4z^2(2-z^2)(2-2z^2+z^4)+(1-z^2)^4}\right)\Big(c^2$$
$$+2\big(1+c^4\big)\Big(w^4z^2\big(2-z^2\big)\big(z^4-2z^2+2\big)+\big(1-z^2\big)^4\Big)$$
$$+c^2\Big(w^4z^2\big(2-z^2\big)\big(2-2z^2+z^4\big)+\big(1-z^2\big)^4\Big)^2+c^2\Big)\Bigg],.$$

B.2 Uncorrelated emission case

In the uncorrelated emission case, we do not need to resolve any overlapping divergences, we merely remove endpoint divergences. To that end, we introduce the substitutions

$$\begin{aligned} y &= x^M & r &= q^M \\ y_k &= x_k^M & y_l &= x_l^M \\ t_l &= 1-(1-s_l^4)^4 & t_l &= 1-(1-s_l^4)^4 \\ b &= c^2 & t_5 &= s_5^2, \end{aligned} \tag{B.3}$$

where the first and second line only affect region A or B, respectively. The exponent M is again related to the parameter m and adjusted to suppress the relevant critical limit. It is set to $M=\frac{2}{m}$ if $m\in]0,1]$, and $M=2$ otherwise.

With this we find for the master formulae in regions A and B:

$$S_{S-A} = \int_0^1 \mathrm{d}q\,\mathrm{d}x\,\mathrm{d}c\,\mathrm{d}s_5\,\mathrm{d}s\,\mathrm{d}s_6\,\frac{2^{15-4\epsilon}M^2\Gamma(-2\alpha-4\epsilon)}{e^{2(\epsilon+\alpha)\gamma_e}\pi^{3/2}\Gamma(1/2-\epsilon)\Gamma(-\epsilon)} q^{-1+Mn\epsilon+M\alpha(1+n)/2}\, x^{-1+2Mn\epsilon+M\alpha(1+n)}$$
$$\cdot c^{-1-2\alpha-4\epsilon}\, s_5^{-1-2\epsilon}\left(1+c^2\right)^{2\alpha+4\epsilon}\left(2-s_5^2\right)^{-1-\epsilon}$$
$$\cdot\left(F_{A1a}^{4\epsilon+2\alpha}+F_{A2a}^{4\epsilon+2\alpha}+F_{A1b}^{4\epsilon+2\alpha}+F_{A2b}^{4\epsilon+2\alpha}\right)\left[\left(1+x^M\right)\left(1+r^Mx^M\right)\right]^{2\epsilon n+\alpha(n-1)}$$
$$\cdot s^{1-4\epsilon}\left(1-s\right)^{1-4\epsilon}\left(1+s\right)^{1-4\epsilon}\left(1+s^2\right)^{1-4\epsilon}\left(2-s^4\right)^{-1/2-\epsilon}\left(2-2s^4+s^8\right)^{-1/2-\epsilon}$$
$$\cdot s_l^{1-4\epsilon}\left(1-s_l\right)^{1-4\epsilon}\left(1+s_l\right)^{1-4\epsilon}\left(1+s_l^2\right)^{1-4\epsilon}\left(2-s_l^4\right)^{-1/2-\epsilon}\left(2-2s_l^4+s_l^8\right)^{-1/2-\epsilon}$$

$$
\begin{aligned}
S_{S-B} = \int_0^1 \mathrm{d}x_k\,\mathrm{d}x_l\,\mathrm{d}c\,\mathrm{d}s_5\,\mathrm{d}s\,\mathrm{d}s_6\, & \frac{2^{15-4\epsilon} M^2 \Gamma\left(-2\alpha-4\epsilon\right)}{e^{2(\epsilon+\alpha)\gamma_e}\pi^{3/2}\Gamma\left(1/2-\epsilon\right)\Gamma\left(-\epsilon\right)} \\
& \cdot x_k^{-1+Mn\epsilon+M\alpha(1+n)/2}\, x_y^{-1+Mn\epsilon+M\alpha(1+n)/2}\, c^{-1-2\alpha-4\epsilon}\, s_5^{-1-2\epsilon}\left(1+c^2\right)^{2\alpha+4\epsilon}\left(2-s_5^2\right)^{-1-\epsilon} \\
& \cdot \left(F_{B1}^{4\epsilon+2\alpha}+F_{B2}^{4\epsilon+2\alpha}\right)\left[\left(1+x_k^M\right)\left(1+x_l^M\right)\right]^{2\epsilon n+\alpha(n-1)} \\
& \cdot s^{1-4\epsilon}\left(1-s\right)^{1-4\epsilon}\left(1+s\right)^{1-4\epsilon}\left(1+s^2\right)^{1-4\epsilon}\left(2-s^4\right)^{-1/2-\epsilon}\left(2-2s^4+s^8\right)^{-1/2-\epsilon} \\
& \cdot s_l^{1-4\epsilon}\left(1-s_l\right)^{1-4\epsilon}\left(1+s_l\right)^{1-4\epsilon}\left(1+s_l^2\right)^{1-4\epsilon}\left(2-s_l^4\right)^{-1/2-\epsilon}\left(2-2s_l^4+s_l^8\right)^{-1/2-\epsilon}
\end{aligned}
$$

Appendix C

Multiprecision

C.1 The problem, explained

To understand the necessity for multiprecision variables, it is best to study a concrete example. Take a measurement function which contains a structure like

$$F(x) = \sqrt{\frac{1}{x} + \frac{2\Delta}{\sqrt{x}}} - \sqrt{\frac{1}{x}} \tag{C.1}$$

where x represents any of the variables that come associated with a plus-distribution, and Δ is some x-independent quantity that may well depend on other variables. For positive Δ this function is finite and positive in the region $x \in [0, 1]$, and has a nontrivial limit as $x \to 0$: $F(0) = \Delta$.

The problem now arises because for small but non-zero x, the two square roots in the difference can become arbitrarily large, but their difference will hover around Δ. In other words, the smaller we choose x, the further down in decimal places is the first digit at which the roots' values will differ. In the above example, for $\Delta = 1$, $x = 10^{-10}$, the result is

$$F(10^{-10}) \approx (1.0000000001 - 1) \times 10^{10} \approx 1 \tag{C.2}$$

This is at odds with the way floating point calculations are performed in any computer program, because for a computer there is a direct relation between the number of digits that are stored for a variable, and the memory that is required to store them. C++ `double` type variables store typically $\approx 15 - 16$ digits[1], and are blind to all digits further down the decimal form. In the above example, if $x \approx 10^{-15}$ or smaller, a computer program would calculate the value of the two square roots and not see that they are different, because the difference occurs only at a digit that is rounded away in `double` type variables. The program would therefore conclude that $F(10^{-15}) = 0$, rather than $F(10^{-15}) \approx \Delta$.

The reason this is a large problem for us, rather than just a small rounding error, becomes clear when plus-distributions come into the fray.

[1]The ambiguity of 15 vs. 16 arises because a computer stores binary numbers, not decimal numbers.

If we apply a $\left[\frac{1}{x}\right]_+$ plus-distribution to F as it appears in our code, we find a structure like

$$\left[\frac{1}{x}\right]_+ \ln F(x) = \frac{\ln \frac{F(x)}{F(0)}}{x} \tag{C.3}$$

in the integrand, because F appears only as $F^{4\epsilon}$, which expands to logarithms. $F(0)$ is a non-trivial limit. If we follow the `SoftSERVE` procedures we will have explicitly included an `if`-clause in the input files that makes sure that $F(0)$ is evaluated correctly. But $F(x \neq 0)$ is evaluated using the internal C++ floating point math libraries, and can therefore be subject to the rounding errors we showed above.

This is now a two-faceted problem. First, for an F as demonstrated above the program sees that the integrand can contain the schematic expression

$$\left[\frac{1}{x}\right]_+ \ln F(x)\bigg|_{x=10^{-15}} = 10^{15} \cdot \ln \frac{F(10^{-15})}{F(0)} = 10^{15} \cdot \ln \frac{0}{\Delta} \to \texttt{NaN}, \tag{C.4}$$

where the logarithm cannot be evaluated, because the program suffered from rounding problems in $F(10^{-15})$. If this occurs, the numerical integrator would return `NaN` as the result of the integration.

But second, and more dangerous, is if we have a measurement function $F'(x) = F(x)+c$, for some constant c, then a computer would return a *wrong* (rather than a *nonsensical*) result, because the computer would calculate

$$\left[\frac{1}{x}\right]_+ \ln F'(x)\bigg|_{10^{-15}} = 10^{15} \cdot \ln \frac{0+c}{\Delta + c} = 10^{15} \ln \frac{c}{c+\Delta}, \tag{C.5}$$

rather than the correct result of

$$\left[\frac{1}{x}\right]_+ \ln F'(x)\bigg|_{10^{-15}} = 10^{15} \cdot \ln \frac{\Delta + \delta + c}{\Delta + c} \approx 10^{15} \ln 1 \approx 0, \tag{C.6}$$

where δ stands in for the small difference between $F(0)$ and $F(10^{-15})$.

This problem can occur for any measurement function that relies on large cancellations between numbers, and it is not just a theoretical problem; Transverse Thrust is an observable that shows exactly this behaviour.

C.2 The diagnosis

So what can we do to solve the problem?

First we need to have a method of diagnosing the critical feature, to see if a given observable potentially suffers from it. For this we recall that *Divonne* delineates an exclusion zone at the integration boundaries via the `border` setting, in which it does not evaluate the integrand directly. This helps us, because it means that below a certain threshold no variable values will be plugged into any functions. Still, we do not know at which point the rounding errors appear, as this is observable dependent. For some observables $y \sim 10^{-5}$ may trigger them already, for others they may only appear below $y \sim 10^{-20}$. Also recall that the rounding errors do not just cause the integrand to exhibit `NaN` occurrences as the rounding artefacts, but they can also just change its value. Both these artefacts will upset the continuity of the integrand function, and that means that changing the `border` variable will cause a change in the numerical integration result, because the extrapolation into the boundary region relies on the continuity of the integrand function.

To check for the presence of rounding errors, we therefore perform the calculation with different values for the `border` variable, and if the final result changes we very likely are sensitive to rounding errors, and have to use the multiprecision solution outlined below.

C.3 The solution

The solution to rounding errors due to not resolving enough digits is to resolve more digits.

We can do this by using non-standard data types which store more than the C++ standard `double` type's 15 digits. To keep things simple we define a new datatype `decimal`, which is typedef'd to `double` if the usual precision suffices, but which is internally set to a larger datatype, if needed.

To provide the backbone for multiprecision variables, we implement three options:

- The `boost` library[4] provides the `cpp_dec_float_100` datatype, which is implemented via header-only libraries.
- The `GMP` library[5] library provides the `mpf_float_100` datatype, which requires the GMP library to be compiled before use.
- The `MPFR` library[6] library provides the `static_mpfr_float_100` datatype, which requires the GMP and MPFR libraries to be compiled before use.

All of these provide 100 decimal digit precision, and are ordered roughly in performance: `boost` is slowest but easiest to implement, while `GMP` and `MPFR` are fastest but typically increase memory use (`MPFR` in particular).

Because `boost` does not need to be compiled, we use it as the default — how to change the multiprecision backbone to `GMP` or `MPFR` is explained in section 4.10.

Depending on which backbone is chosen, the `decimal` datatype is typedef'd as an alias of the relevant one of the three choices itemised above.

Appendix D

Program structure

Having explained the methods and strategies used for the *calculation*, this chapter serves as the documentation of the *implementation*, i.e. the layout of the code.

This is to a large part dictated by the structure of the calculation itself. The most important of the features of the calculation is that there is a correlation between the dimensionality of the numerical integration and the regulator orders for the first few orders. In detail:

Any master equation in [1] or appendix B (taking SCET-1 without loss of generalisation) can be split schematically into three structures, viz.

$$S = \underbrace{P(\epsilon)}_{\substack{\text{constant}\\\text{prefactors}}} \left(\Pi_i \int \mathrm{d}x_i\right) \underbrace{X(\{x_i\};\epsilon)}_{\substack{\text{finite}\\\text{functions}}} \underbrace{\phi\left(F(\{x_i\}), G(\{x_i\}), x_i;\epsilon\right)}_{\substack{\text{measurement function}\\\text{and divergences}}} \tag{D.1}$$

We have split off constant, i.e. integration variable independent, prefactors like π^ϵ or $\Gamma(1-\epsilon)$ from the function to be integrated, and we've collected non-divergent, but regulator-dependent functions into the structure X. These mainly are relics of the matrix element numerators, Jacobians and the likes. Finally, ϕ contains the measurement functions F or G , as well as all divergent monomials (like $y^{-1+n\epsilon}$) before the subtraction.

Two features are now important:

- First, only ϕ contains regulator poles: P and X are finite as $\epsilon \to 0$ (SCET-1) or $\alpha \to 0$, $\epsilon \to 0$ (SCET-2). In the SCET-2 case there may be terms like $\frac{\alpha}{\epsilon}$, but these only exchange the type of the regulator poles. They do not add a net divergence.
- The poles in ϕ only arise from the δ-part in the subtraction $x^{-1+n\epsilon} = \frac{\delta(x)}{n\epsilon} + \left[\frac{1}{x}\right]_+ + n\epsilon\left[\frac{\ln x}{x}\right]_+ + \dots$

We can draw several lessons from this.

D.1 Prefactors

First, note that a given regulator order of the integrand structure $\int X\phi$ contributes to multiple regulator orders of the final result for S, because the prefactor can add any positive power of the regulators. As an example, the lowest regulator order of $\int X\phi$ in the C_A colour structure for an SCET-1 observable has regulator power ϵ^{-4}. This expression, multiplied with the lowest (ϵ^0) order of the prefactor P, appears in the ϵ^{-4} order of the final result soft function. It also appears in the ϵ^{-3} coefficient of the final result when multiplied with the ϵ^1 order of the prefactor, in the ϵ^{-2} coefficient when multiplied with the ϵ^2 order of the prefactor, and so forth.

In other words, a naive expansion in regulator powers for the full soft function master formula (including prefactors) yields expressions for the higher order terms that contain all the lower order expressions, just with different constants multiplying individual functions.

To reduce the number of individual function evaluations the computer has to perform, we therefore split the prefactors from the variable dependent functions, expand the latter and numerically integrate them alone, and then plug the different regulator orders of the numerical results back together with the correct prefactor orders to get the final result coefficients.

D.2 Poles and Deltas

The second point of order concerns the origin of the poles. As each pole reliably comes with a Dirac delta, this means that leading poles arise from integrations with reduced dimensionality of the integration.

Consider as an example again the ϵ^{-4} order of the C_A colour structure for a SCET-1 observable, which is the leading order. The four inverse regulator powers arise in the form $\Pi_i \left(\frac{\delta(x_i)}{\eta_i \epsilon}\right)$ from the subtraction, where the x_i are the relevant integration variables and η_i are constants that differ between the x_i.

To get the result, we therefore do not need to perform the full six-dimensional integral, it suffices to set the four variables that come with a delta to zero analytically, and only perform the surviving two-dimensional integral numerically.

At higher orders this induces a hierarchy. Take the first subleading order of the same colour structure. To get to the ϵ^{-3} pole we can either start with all four deltas from the subtraction, and cancel one of them with an ϵ^1 factor from X or the measurement function F (which appears as $F^{4\epsilon}$ in ϕ), or we can take only three deltas and use the lowest order (ϵ^0) for all other remaining factors from X and F.

In the former case, we only need to perform a 2-dimensional integration, in the latter case we need to perform only 3-dimensional integrations, albeit with terms depending on different sets of three integration variables.[1]

We therefore conclude that for the leading regulator orders the number of integrations depends on the regulator power. In particular, we find that the number of integration dimensions is given by table D.1 The $C_F T_F n_f$ structure only has three divergence monomials, which account for the difference.

One last piece of fortune reduces the number of integration dimensions further. Note that the first finite order (ϵ^0) could arise from the terms in which no delta is present, and all other factors contribute at

[1]There are four different sets of three integration variables, as there are four possibilities of choosing one out of four subtractions. For that one subtraction we *don't* use the delta, but the plus-distribution.

$C_F C_A$ and C_F^2

regulator powers	integration dimension
-4 (leading)	2
-3	3
-2	4
-1	5
0	6

$C_F T_F n_f$

regulator powers	integration dimension
-3 (leading)	3
-2	4
-1	5
0	6

Table D.1: *Apparent* number of integration dimensions for $C_F C_A$ and C_F^2 colour structures (left) and $C_F T_F n_f$ structure (right).

$C_F C_A$ and C_F^2

regulator powers	integration dimension
-4 (leading)	2
-3	3
-2	4
-1	5
0	5

$C_F T_F n_f$

regulator powers	integration dimension
-3 (leading)	3
-2	4
-1	5
0	5

Table D.2: *Actual* number of integration dimensions for $C_F C_A$ and C_F^2 colour structures (left) and $C_F T_F n_f$ structure (right).

lowest order. In this case it turns out that the relevant expressions from X are independent of one of the integration variables — y in the correlated emission case, b for uncorrelated — which only appears in the measurement function. However, the measurement function appears only as $F^{4\epsilon}$ or $G^{4\epsilon}$, whose leading order is 1, which obviously is also independent of y and b. We therefore find that the 6-dimensional integral evaluates to zero, because it involves the integral over a plus-distribution $\left[\frac{1}{y}\right]_+$, multiplied with only y-independent functions (b in place of y for uncorrelated emissions).

The number of dimensions of the integrals *actually* required is therefore listed in table D.2.

The case is virtually identical the SCET-2 case, with one minor change:

For SCET-2 observables the appearance of the second regulator, and in particular of terms like $\frac{\alpha}{\epsilon}$, may cause trouble. In practice, careful analysis shows that apart from inflating the number of appearing terms in various expressions, this changes nothing. The dimensionality of the integration is set by the sum of the powers of the two regulators.

D.3 Source files

With what we've learned in the preceding section, and as a primer for the next, we can now list the source files and why they appear where and how they do.

First, note that most source files have an associated header file. The only exceptions are `Master.cpp`, and the input source files. The `Master.cpp` file only contains the `main{}` function, while the input files are headed by `Auxiliaries.h` in the `Sources` subfolder.[2]

All branches of the program have a certain set of source files, namely

`Master.cpp`	contains the `main()` function. It is found in the `Sources` subfolder, and starts sequential child processes that perform the numerical integration, one for each integration domain dimensionality. Once an integrator run has finished, these results are combined with the constant prefactors to calculate the soft function Laurent series coefficients.
`Prefactors.cpp`	contains the prefactors, which multiply the results of the numerical integration in the Laurent coefficients of the final result soft function. It is also hidden out of view, in the `Sources` folder.
`XFactor.cpp`	resides in `Sources` as well, and contains the functions we grouped as X in the discussion above, in their expansion in regulator orders as required by the numerical integration. These files act as libraries for the next set of files.
"`Int`" files	in the `Sources` folder adhere to a naming scheme. As an example, the `3IntAM1.cpp` file contains the $\epsilon^i\alpha^{-1}$ orders of the functions that appear in the 3-dimensional Integration, for all relevant ϵ orders i. The α orders are specified by `AM1` and `AM2`, standing in for *Alpha to the Minus 1* or *Alpha to the Minus 2*. `AM0` is omitted wherever it appears. With the exception of the `2Int` files (or `3Int` for the $T_F n_f$ structure), all of these files contain multiple functions at a given regulator order, corresponding to the different possible subsets of integration variables that can be formed at a given number of integration dimensions. So the `3IntCA` files contain four functions for each regulator order, corresponding to the four possible sets of integration variables (t_l, v, b), (t_l, v, y), (t_l, v, t_5), (t_l, v, u) that span the integration domain, for example. The Real-Virtual (`RV`) and one-loop (`OL`) computations only have 2-dimensional integrals, so the prefix number is therefore omitted in these cases and similarly for the 4-dimensional integrals in the shortcut anomalous dimension (`AD`) integrals.
`Auxiliaries.cpp`	contains auxiliary function definitions, variable transformations, and the structures that collect the various functions that need to be integrated. The latter takes the different functions defined in the `Int` files and homogenises their integration domain.
"`Input`" files	contain all definitions that are observable specific, and integrator parameters. We've already encountered those in the bulk of the manual before.

The `Sources/SCETX` folders are split into five main subfolders, `S-A` for "C_F Squared, subregion A", and analogous for `S-B`, and `1AN`, short for "1-particle, C_A and n_f", and each contains the files listed above. `1ANR` contains sources for web regularised executables. The subfolder `AD` contains the relevant sources for the shortcut anomalous dimension calculation, i.e. for the `ADLap` and `AdMom` binaries.

The `1AN` folder includes (almost) multiple sets of these files, marked with `CA` (one part of the $C_F C_A$ structure), `PNF` ("Pseudo-n_f", the second part of the $C_F C_A$ structure which behaves like the $C_F T_F n_f$ structure), `NF` ($C_F T_F n_f$ structure), `RV` ("Real-Virtual", i.e. one-particle NNLO), `OL` ("One Loop", i.e. NLO) and `1P` ("one-particle" in general).

All these files share one subfolder as there are shared source files: the `Input` and `Auxiliaries.cpp` files.

[2]With the exception of the content of `Input_Parameters.h`, of course.

To get more familiar with the content of these files it is instructive to see them in action, so we run along a typical program run, and explain along the way what happens.

D.4 Program flow

The schematic flow is described in figure D.1, where the $C_F C_A$ or C_F^2 structures serve as the template. The other colour structures are similar with minor modifications.

Having defined the relevant input in the `Input` files and compiled the program, we call — in a typical run — the binary for a given colour structure.

The `main` function then spawns a child process[3] for the lowest dimensional integration: For all colour structures except $C_F T_F n_f$ this is the 2-dimensional integration. $C_F T_F n_f$ starts at the 3-dimensional integration

This child process sets up variables for the integration results, and calls the `llDivonne` function as defined in the Cuba library to integrate all regulator orders required for any Soft function Laurent series coefficient in parallel, to fill these variables.

To do this, `Divonne` uses the functions defined in the "`Int`" files, which in turn depend on the measurement functions F or G and parameters n and m defined in the `Input` files and the X functions in `XFactor.cpp`. The functions from the "`Int`" files are funnelled through `Auxiliaries.cpp` where the different integration variable sets are aligned.[4]

The integration then proceeds according to the parameters specified in the `Input` files.

Once the first integration is finished, by the arguments explained in the previous chapter, enough information is available to compute the leading pole coefficient from the integration result and the prefactors in `Prefactors.cpp`.

The one-particle calculation terminates here, as it only needs 2-dimensional integrals. For all other colour structures we proceed to the second integration, which has one integration variable more[5]. There, the same procedure happens. Once this integration finishes, enough information is available for the first subleading pole.

The whole process repeats until the 5-dimensional integration is finished. When this happens the program tidies up, calculates the finite and linear regulator divergence coefficients, and terminates.

The procedures for SCET-2 are identical, except that the appearance of terms is set by the sum of the ϵ and α orders. So e.g. for C_F^2 the $\epsilon^{-4}\alpha^0$, $\epsilon^{-3}\alpha^{-1}$ and $\epsilon^{-2}\alpha^{-2}$ pole coefficients all count as leading and are available once the 2-dimensional integration has terminated.

[3]The reason for using child processes is technical and laid out in the comments in the `Master.cpp` files.

[4]If e.g. multiple functions contribute to (for example) the 3-dimensional integration, which depend on different sets of three integration variables, e.g. (t_l, v, b), (t_l, v, y), etc., `Auxiliaries.cpp` provides one function depending on variables (x_1, x_2, x_3) which is the sum of all 3-variable functions in the "`Int`" file.

[5]i.e. 4-dim for $C_F T_F n_f$, 3-dim for everybody else.

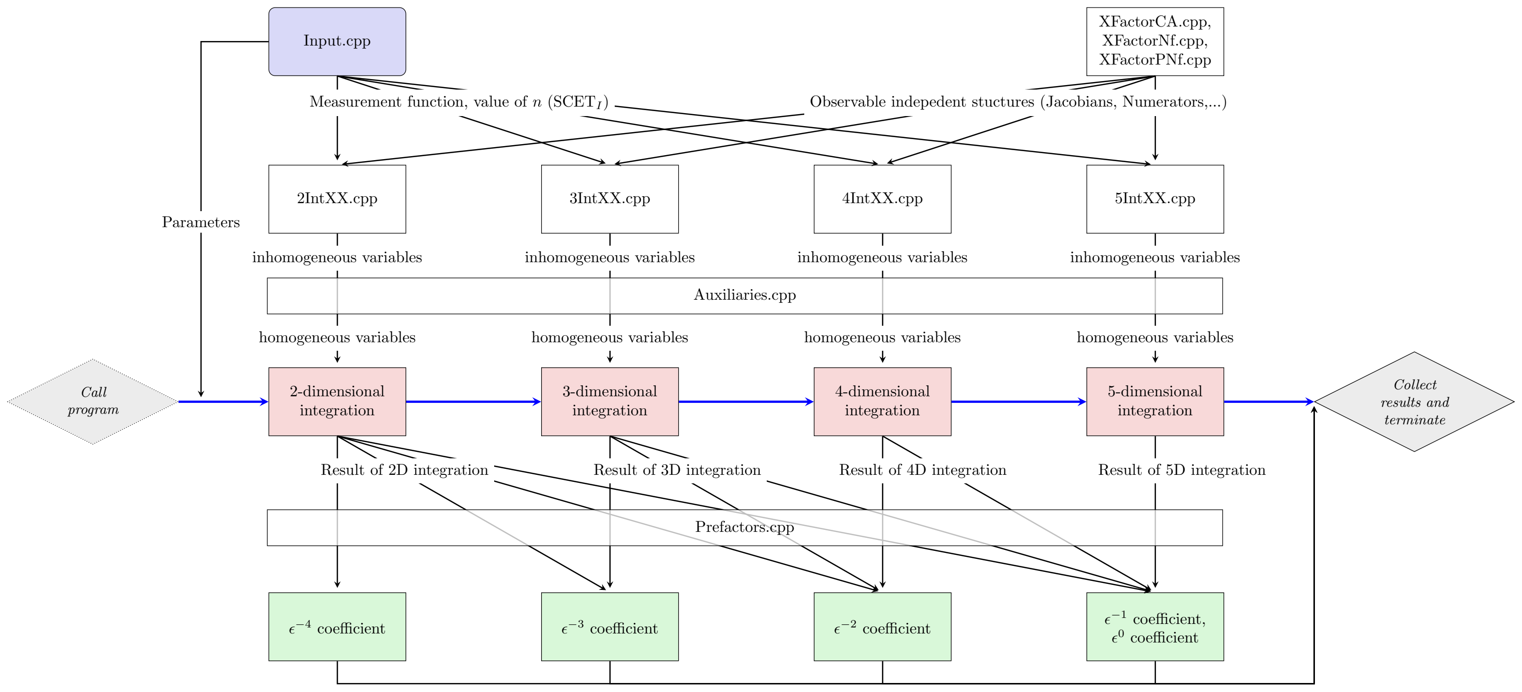
Input.cpp
XFactorCA.cpp, XFactorNf.cpp, XFactorPNf.cpp
Measurement function, value of n (SCET$_I$)
Observable indepedent stuctures (Jacobians, Numerators,...)
Parameters
2IntXX.cpp
3IntXX.cpp
4IntXX.cpp
5IntXX.cpp
inhomogeneous variables
Auxiliaries.cpp
homogeneous variables
Call program
2-dimensional integration
3-dimensional integration
4-dimensional integration
5-dimensional integration
Collect results and terminate
Result of 2D integration
Result of 3D integration
Result of 4D integration
Result of 5D integration
Prefactors.cpp
ϵ^{-4} coefficient
ϵ^{-3} coefficient
ϵ^{-2} coefficient
ϵ^{-1} coefficient, ϵ^{0} coefficient

Figure D.1: Program flow through the program for a SCET-1 observable. The blue box marks the only files requiring user input, red boxes represent program steps, white boxes mark the source files. Green boxes represent results. The blue arrows trace the program flow, black arrows of the form $A \to B$ represent “Variables in A passed to B” or “Function defined in A called by B”. The chart should be read left-to-right and top-down, and external libraries are not included.

D.5 Changelog

D.5.1 Changes in 1.0.0

- Fixed a bug in the emission regularised SCET-2 α^0 terms
- Added support for uncorrelated emissions:
  - Updated makefiles
  - New scripts
  - Additional templates
- Added support for renormalisation following inverse Laplace transform
- Added rapidity renormalisation group support
- Fixed usage of parameter m in SCET2
- Fixed wrong coefficients in Fourier conversion scripts SCET1
- Fixed order of calculations in Fourier conversion scripts SCET1 and SCET2
- Fixed typoes in the manual (t was used instead of the correct t_{kl}, C_f^2 instead of C_F^2,...)
- Added `ADLap` and `ADMom` targets to implement the semianalytic formulae in [8]
- Added checks for `NaN` in renormalisation scripts

Appendix E

Pole cancellation checks

Here we list the analytic form of the pole cancellation checks implemented in `SoftSERVE`, for both SCET-1 and SCET-2 observables, and for both Laplace space and cumulant soft functions. For the cumulant soft functions' checks, the constraints affect the same coefficients as in Laplace space, but only subleading pole cancellations are modified compared to Laplace space observables. We therefore only list the modified expressions, unmodified check formulae can be taken from the corresponding Laplace space sections.

E.1 SCET-1

The coefficients x_i are NLO coefficients as found in the `SoftSERVE` output, the y_i and z_i are their NNLO Double-Real and Real-Virtual counterparts, respectively. Analytically speaking they are the Laurent coefficients of the master formulae (post-integration) as outlined in [1] and [2]. The x_i carry colour factors C_F (generically), y_i carry C_F^2, $C_F C_A$, and $C_F T_F n_f$, and z_i carry $C_F C_A$.

E.1.1 Laplace space

$$\begin{aligned}
x_2 &= -\frac{\Gamma_0}{n} \\
y_4 + z_4 &= \frac{\Gamma_0^2}{2n^2} \\
y_3 + z_3 &= -\frac{\Gamma_0 \gamma_0^S}{n^2} + \frac{\beta_0 \Gamma_0}{4n} \\
y_2 + z_2 &= \frac{(\gamma_0^S)^2}{2n^2} - \frac{\beta_0 \gamma_0^S}{2n} - \frac{c_1^S \Gamma_0}{n} - \frac{\Gamma_1}{4n}
\end{aligned} \tag{E.1}$$

Note that different constraints for separate colour structures are hiding in the combinations $y_i + z_i$. We have e.g. $y_4 + z_4|_{C_A} = 0$ and $y_4 + z_4|_{C_F^2} = \Gamma_0^2/2n$.

E.1.2 Cumulant soft functions

$$y_2 + z_2 = \frac{(\gamma_0^S)^2}{2n^2} - \frac{\beta_0\gamma_0^S}{2n} - \frac{c_1^S\Gamma_0}{n} - \frac{\Gamma_1}{4n} + \frac{\Gamma_0^2\pi^2}{3n^2} \tag{E.2}$$

E.2 SCET-2 using emission-based regularisation

The coefficients x_i^j are NLO coefficients as found in the `SoftSERVE` output, the y_i^j and z_i^j are their NNLO Double-Real and Real-Virtual counterparts, respectively. Analytically speaking they are the Laurent coefficients of the master formulae (post-integration) as outlined in [1] and [2]. The x_i^j carry colour factors C_F (generically), y_i^j carry C_F^2, C_FC_A, and $C_FT_Fn_f$, and z_i^j carry C_FC_A.

E.2.1 Laplace space

$$\begin{aligned}
&x_1^1 = -2\Gamma_0 \\
&y_2^2 = 2\Gamma_0^2 \qquad\qquad y_1^2 = 4\Gamma_0\, d_1 \qquad\qquad y_0^2 = 2(d_1)^2 - 2\Gamma_0 x_{-1}^1 \\
&y_3^1 + \frac{z_3^1}{2} = -2\Gamma_0 x_2^0 \qquad y_2^1 + \frac{z_2^1}{2} = -\frac{\beta_0\Gamma_0}{2} - 2\Gamma_0 x_1^0 - 2d_1 x_2^0 \\
&y_1^1 + \frac{z_1^1}{2} = -\beta_0 d_1 - \frac{\Gamma_1}{2} - 2\Gamma_0 x_0^0 - 2d_1 x_1^0 + x_2^0 x_{-1}^1
\end{aligned} \tag{E.3}$$

E.2.2 Cumulant soft functions

$$\begin{aligned}
y_0^2 &= 2(d_1)^2 - 2\Gamma_0 x_{-1}^1 + \frac{4}{3}\Gamma_0^2\pi^2 \\
y_1^1 + \frac{z_1^1}{2} &= -\beta_0 d_1 - \frac{\Gamma_1}{2} - 2\Gamma_0 x_0^0 - 2d_1 x_1^0 + x_2^0 x_{-1}^1 - \frac{4}{3}\Gamma_0(\Gamma_0 + x_2^0)\pi^2
\end{aligned} \tag{E.4}$$

E.3 SCET-2 using web-based regularisation (rapidity RGE)

The coefficients x_i^j are NLO coefficients as found in the `SoftSERVE` output, while the y_i^j are their NNLO Uncorrelated-Double-Real counterparts, and the z_i^j correspond to the sum of Real-Virtual and Correlated-Double-Real emission coefficients. Analytically speaking they are the Laurent coefficients of the master formulae (post-integration) as outlined in [1] and [2] (The sum of such coefficients, in the case of z_i^j) with modified rapidity regulator. The x_i^j carry colour factors C_F (generically), y_i^j carry C_F^2, and z_i^j carry C_FC_A and $C_FT_Fn_f$. In other words, the x_i^j correspond to the NLO single-web case, y_i^j to the NNLO two-web case, and the z_i^j to the NNLO single-web case. Note explicitly that the assignment of y_i^j and z_i^j is different to the preceding sections.

E.3.1 Laplace space

$$
\begin{aligned}
&x_1^1 = -2\Gamma_0 \\
&x_2^0 = \Gamma_0 && x_1^0 = -\gamma^S_{\mu,0} \\
&z_3^1 = 0 && z_2^1 = -\beta_0\Gamma_0 && z_1^1 = -\Gamma_1 - 2\beta_0\gamma^S_{\nu,0} \\
&z_4^0 = 0 && z_3^0 = \frac{\beta_0\Gamma_0}{4} && z_2^0 = \frac{\Gamma_1}{4} - \frac{\beta_0\gamma^S_{\mu,0}}{2} \\
&z_1^0 = \beta_0 x_0^0 - \frac{\gamma^S_{\mu,1}|_C}{2} \\
&y_2^2 = 2\Gamma_0^2 && y_1^2 = 4\Gamma_0\gamma^S_{\nu,0} && y_0^2 = -2\Gamma_0 x^1_{-1} + 2(\gamma^S_{\nu,0})^2 \\
&y_3^1 = -2\Gamma_0^2 && y_2^1 = 2\Gamma_0(\gamma^S_{\mu,0} - \gamma^S_{\nu,0}) && y_1^1 = 2\gamma^S_{\mu,0}\gamma^S_{\nu,0} - 2x_0^0\Gamma_0 + x^1_{-1}\Gamma_0 \\
&y_4^0 = \frac{\Gamma_0^2}{2} - 2\Gamma_0 x_3^{-1} && y_3^0 = -\Gamma_0\gamma^S_{\mu,0} - 2\gamma^S_{\nu,0}x_3^{-1} - 2\Gamma_0 x_2^{-1} \\
&y_2^0 = \frac{(\gamma^S_{\mu,0})^2}{2} + \Gamma_0 x_0^0 - 2\Gamma_0 x_1^{-1} - 2\gamma^S_{\nu,0}x_2^{-1} + x^1_{-1}x_3^{-1} \\
&y_1^0 = \Gamma_0(x^0_{-1} - 2x_0^{-1}) - \gamma^S_{\mu,0}x_0^0 - 2\gamma^S_{\nu,0}x_1^{-1} + x^1_{-1}x_2^{-1} + x^1_{-2}x_3^{-1} - \frac{\gamma^S_{\mu,1}|_U}{2}
\end{aligned}
\tag{E.5}
$$

where $\gamma^S_{\mu,0} = \gamma^S_{\mu,1}|_U = 0$ and

$$
\gamma^S_{\mu,1}|_C = C_F C_A\left(-\frac{808}{27} + \frac{11\pi^2}{9} + 28\zeta_3\right) + C_F T_F n_f\left(\frac{224}{27} - \frac{4\pi^2}{9}\right) \tag{E.6}
$$

are known analytically.

E.3.2 Cumulant soft functions

$$
\begin{aligned}
y_0^2 &= -2\Gamma_0 x^1_{-1} + 2(\gamma^S_{\nu,0})^2 + \frac{4\Gamma_0^2\pi^2}{3} \\
y_2^0 &= \frac{(\gamma^S_{\mu,0})^2}{2} + \Gamma_0 x_0^0 - 2\Gamma_0 x_1^{-1} - 2\gamma^S_{\nu,0}x_2^{-1} + x^1_{-1}x_3^{-1} - \frac{2\Gamma_0^2\pi^2}{3} - \frac{4\Gamma_0 x_3^{-1}\pi^2}{3} \\
y_1^0 &= \Gamma_0(x^0_{-1} - 2x_0^{-1}) - \gamma^S_{\mu,0}x_0^0 - 2\gamma^S_{\nu,0}x_1^{-1} + x^1_{-1}x_2^{-1} + x^1_{-2}x_3^{-1} - \frac{\gamma^S_{\mu,1}|_U}{2} \\
&\quad + \frac{2\pi^2}{3}\left(\Gamma_0\left(\gamma^S_{\mu,0} - \gamma^S_{\nu,0} - 2x_2^{-1}\right) - 2\gamma^S_{\nu,0}x_3^{-1}\right) - 16\Gamma_0\left(\Gamma_0 + 2x_3^{-1}\right)\zeta_3
\end{aligned}
\tag{E.7}
$$

and where again $\gamma^S_{\mu,0} = \gamma^S_{\mu,1}|_U = 0$.

Appendix F

Template Observables

F.1 Preliminaries

Below we provide the derivation of the measurement functions for the various SCET-1 and SCET-2 observables included as pedagogical template examples in the `SoftSERVE` package. While on occasion we will present abbreviated code fragments for brevity and clarity of presentation, we remind the reader that the complete templates are available in the `SoftSERVE` package download, and should work 'out of the box.'

The ultimate aim is to derive the functions F_X, and G_{Xi}, with placeholder indices X and i, to be used in the `SoftSERVE` program. We will use the term "measurement function" interchangeably for the F and G functions as well as the full measurement exponential. The meaning is in general clear from the context.

To recap the main manual and articles:

- The initial measurement function — we call it F_A or G_A — can be derived from the measurement exponential using the appropriate parametrisations outlined in section 4.2.
- The function F_B and G_B arises from the measurement exponential the same way as the function indexed A, after inverting one suitable variable, according to the symmetry discussions in [1] and [2].
- The F_X functions' behaviour around $y = 0$ determines the parameter m. If that doesn't work, we use G_B near $y_k = 0$.
- For G_A, a second substitution is required, which directly yields G_{Aa} and G_{Ab}.

We run through the examples below, but it is advisable to list briefly which issues in general require attention:

- Non-trivial limits (e.g. of the form $\frac{0}{0}$) must be caught by `if`-clauses.
- Limits in which the measurement function vanishes and/or diverges must be explicitly "defused" by setting the measurement function equal to a constant[1] at these points.

[1]Setting them to 1 is computationally sensible, some terms drop out of the integrand then.

- Heaviside step functions are best implemented using `if`-clauses. This has the advantage that any considerations of `NaN` appearances or zeroes/divergences can be confined to the regions where the `if`-clause triggers, i.e. where the step functions have support.

F.2 First appearance of critical features

Many observables share intricate features. To keep this manual short, we generally do not explain in detail how a certain issue is handled, if another observable covered in the sections before also exhibits it. Instead, here is a table of interesting features that lists for which observable we've covered them in detail.

Feature	Example observable	Section
Basic vanilla observable	Threshold Drell-Yan	F.3
Different regions A, B	C-Parameter	F.4
Vanishing/diverging F_X/G_X	C-Parameter	F.4
Parameter dependence	Angularities	F.5
Non-trivial variable m	Angularities, $W/Z/H$ at large p_T	F.5 F.10
Step functions	Angularities	F.5
Angular dependence	Gauge boson at large p_T	F.10
Non-standard RG equation	Transverse Thrust	F.12
Non-trivial limit	Transverse Thrust	F.12
Rounding errors	Transverse Thrust	F.12
Imaginary F	p_T resummation	F.17

Table F.1: Observables illustrating the first appearances of non-trivial `SoftSERVE` features.

Warning!

The symmetry discussion in the two main papers shows explicitly that there are four distinct integration regions covered by the two regions A and B, each. The symmetry enforces that these give the same result *upon integration*, allowing us to get away with only calculating the results in A and B, and accounting for the others by multiplying these results by 4.

The symmetry does ***not*** enforce equality of the *integrands* in these mirror regions. Depending on how you derive F_B from F_A or the measurement exponential, you may get different functional forms for F_B. Symmetry then states that these must give the same final result. The same holds for G_A and G_B.

So do not be alarmed if there are different F_B/G_B functions you can derive. Chances are they yield the same final result.

F.3 Threshold Drell-Yan

The soft function for Threshold Drell-Yan involves the measurement exponential

$$\exp(-\tau\omega^{DY}(k,l)) = \exp(-\tau(k_+ + k_- + l_+ + l_-)), \tag{F.1}$$

as derived in the initial sections of the first main paper.

F.3.1 Correlated emission

We can now apply the parametrisations, and read off the measurement function, as well as — in the correlated emission case — the parameter n:

$$\exp(-\tau\omega^{DY}(k,l)) = \exp(-\tau\, p_T\, \frac{1+y}{\sqrt{y}}) \tag{F.2}$$

To derive F_B we need to invert one of a, b, y, which has no effect.

We conclude that for the correlated emission:

$$n = -1 \qquad F_A = F_B = 1+y \tag{F.3}$$

The leading non-constant power in y can be easily read off: $F_{A/B} = const. + \mathcal{O}(y)$, so following the manual we have $m = 1$.

F.3.2 Uncorrelated emission

The uncorrelated emission parametrisation depends on the parameter n, which we determined in the previous subsection to be $n = -1$. Using this we find

$$\begin{aligned}\mathcal{M}_{DY} &= \exp(-\tau(k_+ + k_- + l_+ + l_-)) &\text{(F.4)}\\ &= \exp(-\tau\, q_T\, \frac{1+y_k}{\sqrt{y_k}}\, \frac{1+y_l}{\sqrt{y_l}}) &\text{(F.5)}\end{aligned}$$

To derive G_B we need to invert one of y_k, y_l, which has no effect.

We also need to reparametrise in region A to arrive at its subregions a and b, leading to

$$\mathcal{M}_{DY} = \exp(-\tau\, q_T \frac{(1+y)(1+ry)}{\sqrt{r}\, y}) \tag{F.6}$$

in both regions Aa and Ab. We conclude that for the uncorrelated emission:

$$\begin{aligned}G_{Aa} = G_{Ab} &= (1+y)(1+ry) &\text{(F.7)}\\ G_B &= (1+y_k)(1+y_l) &\text{(F.8)}\end{aligned}$$

Problematic limits

There are none, this function is well behaved in the entire integration space.

F.4 C-Parameter

The C-Parameter soft function is given by

$$\mathcal{S}(\omega) = \sum_{k,l} \langle 0| S^\dagger S |k,l\rangle \langle k,l| S S^\dagger |0\rangle \; \delta(\omega - \frac{k_+ k_-}{k_+ + k_-} - \frac{l_+ l_-}{l_+ + l_-}). \tag{F.9}$$

F.4.1 Correlated emission

Following a Laplace transform we find for the correlated emission case the measurement exponential

$$\mathcal{M}_{C,A} = \exp(-\tau\, p_T \,\sqrt{y}\,(\frac{a}{a+b+y(1+ab)} + \frac{ab}{a(a+b)+y(1+ab)})). \tag{F.10}$$

From this we can read off

$$n = 1 \qquad F_A = \frac{a}{a+b+y(1+ab)} + \frac{ab}{a(a+b)+y(1+ab)} \tag{F.11}$$

The function in region B is different, now, and to get it we can either invert y, a, or b[2]. This yields

$$\mathcal{M}_{C,B} = \exp(-\tau\, p_T \,\sqrt{y}\,(\frac{a}{a(1+ab)+y(a+b)} + \frac{ab}{1+ab+ay(a+b)})). \tag{F.12}$$

and so

$$F_B = \frac{a}{a(1+ab)+y(a+b)} + \frac{ab}{1+ab+ay(a+b)} \tag{F.13}$$

Expanding F_X around $y = 0$ yields the leading non-constant power in y, again we have $F_X = const. + \mathcal{O}(y)$, so again we find $m = 1$.

F.4.2 Uncorrelated emission

The value for n needed here is $n = 1$, as just derived. Using this we find

$$\mathcal{M}_{C,A} = \exp(-\tau\, q_T \,\frac{\sqrt{y_k}}{1+y_k}\,\frac{\sqrt{y_l}}{1+y_l}). \tag{F.14}$$

To derive G_B we need to invert one of y_k, y_l, which has no effect. Finally, following the rescaling in subregion A we conclude that for the uncorrelated emission:

$$G_{Aa} = G_{Ab} = \frac{1}{(1+y)(1+ry)} \tag{F.15}$$

$$G_B = \frac{1}{(1+y_k)(1+y_l)} \tag{F.16}$$

[2]Either will do. Choose the one that causes you the least amount of work.

Problematic limits

The measurement functions in the correlated emission case vanish if $a = 0$. This is a lower dimensional hypersurface and not a limit associated with a divergence/plus-distribution, and therefore acceptable. Nevertheless the program will throw up error messages if this is not handled. We therefore include a segment

```
if(a<=0.) {
  FA=1.;
}
```

for F_A and similar for F_B in their sections of the `Input_Measurement_Correlated.cpp` file.

For the reasons why changing the value of the function is acceptable, please consult appendix A.

Additionally, there are overlapping non-trivial limits of the form $\frac{0}{0}$, e.g. in F_A's first term, at $(a, b, y) \to (0, 0, 0)$. If C++ tried to evaluate the function there, it would encounter the expression $\frac{0}{0+0+0} + \ldots$, which it can't resolve on its own. However, all of these require $a \to 0$, and therefore are already covered by the if clause above.

This exhausts the list. The functions are well-behaved everywhere else.

F.5 Angularities

Angularities is again of a form requiring a Laplace transform, yielding

$$\mathcal{M}_A = \exp(-\tau\,\omega^A(k,l)), \tag{F.17}$$

where $\omega(k,l)$ is the definition of the angularities observable in terms of emission momenta.

This definition reads:

$$\omega^A(k,l) = \Theta(k_+ - k_-)\,k_-^{1-\frac{A}{2}}\,k_+^{\frac{A}{2}} + \Theta(k_- - k_+)\,k_+^{1-\frac{A}{2}}\,k_-^{\frac{A}{2}} \tag{F.18}$$

$$+\,\Theta(l_+ - l_-)\,l_-^{1-\frac{A}{2}}\,l_+^{\frac{A}{2}} \quad + \Theta(l_- - l_+)\,l_+^{1-\frac{A}{2}}\,l_-^{\frac{A}{2}}. \tag{F.19}$$

As we can see, the angularities are parameter dependent, which must be properly declared to the program, and assigned a value, as `SoftSERVE` cannot handle parametric expressions. We therefore have to add a line to `Input_Common.cpp` and `Input_Parameters.h` each:

```
extern double AA;
```

in `Input_Parameters.h` to declare the parameter and

```
double AA=0.5;
```

in `Input_Common.cpp` to define it. We chose `AA` instead of `A` because the letter A is already in use — it is used by one of the internal functions. For a list of parameter names to avoid, see Table 4.1.

F.5.1 Correlated emissions

Plugging in the parametrisation for correlated emission, we find for region A

$$\omega_A^A(k,l) = p_T y^{\frac{1-A}{2}} \Bigg(\Theta(\frac{a(a+b)}{1+ab} - y) \quad \Big[a^{-\frac{A}{2}}(a+b)^{-1+\frac{A}{2}}(1+ab)^{-\frac{A}{2}}(a+a^A b)\Big] + \Theta(y - \frac{a(a+b)}{1+ab})\Big[a^{1-\frac{A}{2}}\Big((a+b)^{-1+\frac{A}{2}}(1+ab)^{-\frac{A}{2}} + b(a+b)^{-\frac{A}{2}}(1+ab)^{-1+\frac{A}{2}}y^{-1+A}\Big)\Big]\Bigg),$$

where we already used that some of the step functions will always evaluate to 0 or 1 if the variables a, b and y are in $[0,1]$. We also use that $\Theta(-x) = 1 - \Theta(x)$.

For region B we again invert one of y, a, or b in the full exponential, and find

$$\omega_B^A(k,l) = p_T y^{\frac{1-A}{2}} \Bigg(\Theta\left(\frac{a(1+ab)}{a+b} - y\right) \quad \Big[a^{-\frac{A}{2}}(a+b)^{-\frac{A}{2}}(1+ab)^{-1+\frac{A}{2}}(a^A+ab)\Big] + \Theta\left(y - \frac{a(1+ab)}{a+b}\right)\Big[a^{1-\frac{A}{2}}\Big(b(a+b)^{-\frac{A}{2}}(1+ab)^{-1+\frac{A}{2}} + (a+b)^{-1+\frac{A}{2}}(1+ab)^{-\frac{A}{2}}y^{-1+A}\Big)\Big]\Bigg),$$

where now different step functions always evaluate to 0 or 1.

From this we can read off that $n = 1 - A$, and we can easily identify the functions F_A and F_B . We also brought the entire expression in a form where only one of the step functions is ever non-vanishing, which matches an `if-else` combination.

Our `Input_Measurement_Correlated.cpp` file will therefore contain the code fragment

```
if (y <= a*(a+b)/(1+a*b) ) {
   FA=pow(a+b,-1+AA/2.)*(a+pow(a,AA)*b)*(pow(a,-AA/2.)
      *pow(1+a*b,AA/2.));
} else {
   FA=pow(a,1-AA/2.)*(pow(a+b,-1+AA/2.)*pow(1+a*b,-AA/2.)
      +b*pow(1+a*b,-1+AA/2.)*pow(y,-1+AA)*pow(a+b,-AA/2.));
}
```

for F_A, and similar for F_B.

The behaviour for small y is flat, thanks to the step functions, except if $a = 0$, which is suppressed. This means that expanding $F_X - F_X|_{y=0}$ in the region near $y = 0$ doesn't give us a well-defined value for m.

However, the value for m is also used for the uncorrelated emission functions in the vanishing y_k or y_l limits. We merely ranked the needs of the correlated emissions' functions higher and didn't consider the uncorrelated case, so far. Expanding these functions (see below) in the corresponding limits suggests we should use $m = 1$, so we'll do that.

F.5.2 Uncorrelated emission

The uncorrelated emission functions turn out much simpler. Plugging in the correct parametrisation with $n = 1 - A$ we find

$$\omega^A(k,l) = q_T\, y_k^{\frac{1-A}{2}}\, y_l^{\frac{1-A}{2}}\, \frac{b(1+y_l)^{A-1} + (1+y_k)^{A-1}}{1+b}, \tag{F.20}$$

even after inverting either y_k or y_l. We can therefore immediately read off that

$$G_{Aa} = \frac{b(1+y)^{A-1} + (1+ry)^{A-1}}{1+b} \tag{F.21}$$

$$G_{Aa} = \frac{b(1+ry)^{A-1} + (1+y)^{A-1}}{1+b} \tag{F.22}$$

$$G_B = \frac{b(1+y_l)^{A-1} + (1+y_k)^{A-1}}{1+b} \tag{F.23}$$

Problematic limits

First we note that $A = 1$ corresponds to a SCET-2 type observable, and so the SCET-2 branch of `SoftSERVE` must be used for that parameter value, accordingly. For all other values we can simply use the SCET-1 branch and adjust the parameter for different runs.

Furthermore, there is again a zero/divergence in the correlated emission functions at $a = 0$, which we catch the same way as for the C-parameter. This renders the functions well-behaved — any other zero/divergence is overlapping and needs $a = 0$, or is killed by the `if`-clause (e.g. $y \to 0$).

The uncorrelated emission functions are well-behaved everywhere.

Finally, there is a numerical instability arising from terms like y^{-1+A} in the correlated emission case (the uncorrelated emission case does not show this problem). This possible divergence is formally cut off by the step function, but for small values of a we can get close to $y = 0$ and still fulfill the step function's condition. This problem can be slightly ameliorated by choosing $m < 1$. For e.g. $m = 0.5$ this effectively turns the step function from $\theta(y - \text{something})$ to $\theta(y^2 - \text{something}) \leftrightarrow \theta(y - \sqrt{\text{something}})$. As something < 1, this has the effect of raising the value of the cutoff. 'Something' is of course a placeholder here, and the condition on its range arises because otherwise the step function would be 0 or 1 identically.

F.6 Broadening-axis Angularities

The broadening axis angularities are very similar to the angularities above, and share all relevant features with them.

F.7 Thrust

Thrust is merely a special case of the angularities with $A = 0$ (or broadening-axis angularities at $B = 2$), any and all relevant input can be easily derived.

F.8 Recoil-free Broadening

Like Thrust, Broadening with recoil neglected is merely a special case of the angularities with $A = 1$, and any and all relevant input can be easily derived. The only added difficulty involves a change of directory to the SCET-2 branch of `SoftSERVE`.

F.9 Hemisphere soft function

As a generalisation of Thrust, the hemisphere soft function is closely related to the angularities on a technical level. While it has different functions F and G, all relevant technical difficulties are the same as for the angularities — with the sole exception of the symmetrisation, which does, however, not affect the `SoftSERVE` input.

F.10 Gauge boson production at large p_T

Weak gauge boson production at large transverse momentum is one of the easiest observables exhibiting angular dependence. The origin of this dependence is the presence of a third jet against which the boson recoils. It turns out that although this third jet does not give additional diagrammatic contributions, its presence breaks rotational invariance around the beam axis, which introduces the angular dependence.

As before, the Laplace space soft function has the correct form for `SoftSERVE`, with

$$\omega^W(k,l) = n_j \cdot (k+l) = k_+ + k_- - 2\sqrt{k_+k_-}\cos\theta_k + l_+ + l_- - 2\sqrt{l_+l_-}\cos\theta_l \tag{F.24}$$

F.10.1 Correlated emission

Using the correlated emission parametrisation, we find

$$\omega^W(k,l) = p_T y^{-\frac{1}{2}}\left(1 + y - 2\sqrt{\frac{ay}{(a+b)(1+ab)}}\,(b\,c_k + c_l)\right). \tag{F.25}$$

Inverting any of y, a or b has no effect up to relabelling c_k and c_l, so

$$n = -1 \qquad F_A = F_B = 1 + y - 2\sqrt{\frac{ay}{(a+b)(1+ab)}}\,(b\,c_k + c_l) \tag{F.26}$$

$$= 1 + y - 2\sqrt{\frac{ay}{(a+b)(1+ab)}}\,(b\,(1-2t_k) + 1 - 2t_l) \tag{F.27}$$

where the angular dependence can be written in terms of both c_i or t_i variables, however you prefer.

Finally, it is readily apparent that $F_X = c_0 + \sqrt{y}c_1 + \ldots$, and so $m = 0.5$.

F.10.2 Uncorrelated emission

Using the correct parametrisation with $n = -1$ we find

$$\omega^W(k,l) = q_T\, y_k^{-\frac{1}{2}} y_l^{-\frac{1}{2}}\,\frac{b(1+y_l)(1+y_k-2c_k\sqrt{y_k}) + (1+y_k)(1+y_l-2c_l\sqrt{y_l})}{1+b}, \tag{F.28}$$

and again, inverting y_l or y_k has no effect. Therefore,

$$\begin{aligned}
G_{Aa} &= \frac{b(1+y)(1+ry-2c_k\sqrt{ry}) + (1+ry)(1+y-2c_l\sqrt{y})}{1+b} \\
G_{Ab} &= \frac{b(1+ry)(1+y-2c_k\sqrt{y}) + (1+y)(1+ry-2c_l\sqrt{ry})}{1+b} \\
G_B &= \frac{b(1+y_l)(1+y_k-2c_k\sqrt{y_k}) + (1+y_k)(1+y_l-2c_l\sqrt{y_l})}{1+b},
\end{aligned} \tag{F.29}$$

or the same functions with $1-2t_i$ instead of c_i, if you prefer that version.

Problematic limits

There's a zero hiding at $y = a = c_l = 1$, $b = 0$ and a potentially difficult limit at $a = b = 0$ in the correlated emission, which can be dealt with by setting the function to a constant if either $a = 0$ or $c_l = 1$.

Only capturing a and c_l is enough, as the integrand vanishes as $a \to 0$, or $c_l \to \pm 1$, as outlined in the appendices. We therefore do not have to specify that the problematic points only affect submanifolds of the $c_l = 1$ and $a = 0$ hyperplanes. Both entire hyperplanes do not contribute. We can skip the second if-clause nailing down the $y = a = 1$ or $b = 0$ constraints, which would in principle be required to fully localise the zeroes, as well.

The uncorrelated emission case has a similar zero, which can also be defused by removing the $c_l = 1$ case.

F.11 Exclusive Drell-Yan

The exclusive soft function in Drell-Yan production matches the case of gauge boson production at large transverse momentum almost perfectly, with one change of a parameter multiplying the square root part sneaked in. So apart from parameter dependence it matches all features present in the preceding section.

F.12 Transverse Thrust

Transverse Thrust is a hadron collider dijet observable, and therefore in principle needs four Wilson lines (2 jets, 2 beams) in the Soft function. It is, however, possible to reconstruct the anomalous dimension from related observables: leptonic dijet ($0 \to 2$ jets) and hadronic 0-jet ($2 \to 0$ jets) Transverse Thrust. The former is a SCET-1 observable while the latter is of SCET-2 type. Here we consider the SCET-1 component. The SCET-2 component is trivial.

Non-standard RG equation

Transverse Thrust is one of the most computationally complicated observables we've treated so far, with its functional dependence on the emission momenta reading

$$\omega^{TT}(k,l) = \sum_{p=k,l} \frac{1}{2|s|}\sqrt{\left((p_- - p_+)s + 2c\,c_p\sqrt{p_+p_-}\right)^2 + 4(1-c_p^2)p_+p_-} \tag{F.30}$$

$$- \frac{1}{2|s|}\left|(p_- - p_+)s + 2c\,c_p\sqrt{p_+p_-}\right|, \tag{F.31}$$

where s and c are the sine and cosine of the angle between beam- and jet- axes, and are used as parameters which must be declared and defined as usual[3].

Naively we would now apply parametrisations, read off parameters and measurement functions, and run through the whole sequence of steps to get the bare soft function. And then we'd encounter a problem:

So far, all of our observables had measurement exponentials of the form

$$\mathcal{M} = \exp(-\tau\omega(k,l)), \tag{F.32}$$

and they fulfilled Laplace space RG equations of the form

$$\frac{dS(\tau)}{\ln\mu} = \frac{1}{n}\left[\gamma_s + \Gamma_{Cusp}\ln\mu\tau e^{\gamma_e}\right]S(\tau). \tag{F.33}$$

For these types of observables we have a script (`./laprenorm`), which performs the extraction of the 2-loop contribution to γ_s much quicker than we could ever do manually.

Transverse Thrust in [9], however, fulfills a Laplace space RGE of the form[4]

$$\frac{dS_{TT}(\tau)}{\ln\mu} = \frac{1}{n}\left[\gamma_s + \Gamma_{Cusp}\ln\frac{\mu\tau e^{\gamma_e}}{4s^2}\right]S_{TT}(\tau). \tag{F.34}$$

If we use the scripts provided, this would amount to shifting a term $-\Gamma_{Cusp}\ln 4s^2$ to the non-cusp part of the anomalous dimension. We'd have to manually account for that.

Alternatively we can define $\tilde{\tau} = \frac{\tau}{4s^2}$, and use the fact that the τ that appears in the RGE also appears in the measurement exponential. We can therefore account for multiplicative factors by shifting them between τ and the functions F and G which multiply it.

We therefore perform such a shift, and use the measurement exponential

$$\mathcal{M}_{TT} = \exp(-\tilde{\tau}\tilde{\omega}^{TT}(k,l)), \tag{F.35}$$

with

$$\tilde{\omega}^{TT}(k,l) = \sum_{p=k,l} 2s\sqrt{\left((p_- - p_+)s + 2c\,c_p\sqrt{p_+p_-}\right)^2 + 4(1-c_p^2)p_+p_-} \tag{F.36}$$

$$- 2s\left|(p_- - p_+)s + 2c\,c_p\sqrt{p_+p_-}\right|, \tag{F.37}$$

[3]See F.5.

[4]See its eq. 4.5, where κ maps onto our Laplace space $\bar{\tau}^{-1}$.

F.12.1 Correlated emission

Using the correlated emission parametrisation we find

$$\begin{aligned}\tilde{\omega}^{TT}(k,l) = \frac{4s\,p_T}{\sqrt{y}}\Bigg[& b\sqrt{\left(\frac{as}{2(1+ab)} + \frac{c\,c_k\sqrt{ay}}{\sqrt{(a+b)(1+ab)}} - \frac{sy}{2(a+b)}\right)^2 + \frac{ay(1-c_k^2)}{(a+b)(1+ab)}} \\ & - b\left|\frac{as}{2(1+ab)} + \frac{c\,c_k\sqrt{ay}}{\sqrt{(a+b)(1+ab)}} - \frac{sy}{2(a+b)}\right| \\ & + \sqrt{\left(\frac{s}{2(1+ab)} + \frac{c\,c_l\sqrt{ay}}{\sqrt{(a+b)(1+ab)}} - \frac{asy}{2(a+b)}\right)^2 + \frac{ay(1-c_l^2)}{(a+b)(1+ab)}} \\ & - \left|\frac{s}{2(1+ab)} + \frac{c\,c_l\sqrt{ay}}{\sqrt{(a+b)(1+ab)}} - \frac{asy}{2(a+b)}\right|\Bigg].\end{aligned} \tag{F.38}$$

Naively we would now conclude that $n = -1$ due to the leading factor in the string of functions. However, an expansion around $y = 0$ shows that $\omega \sim \sqrt{y}$ to leading order. We must therefore factor out a factor of $\sqrt{y}$, i.e. $n = 1$. Inverting y, a or b to derive F_B, we find and read off

$$\begin{aligned}F_A = 4s\Bigg[& b\sqrt{\left(\frac{as}{2(1+ab)y} + \frac{c\,c_k\sqrt{a}}{\sqrt{(a+b)(1+ab)y}} - \frac{s}{2(a+b)}\right)^2 + \frac{a(1-c_k^2)}{(a+b)(1+ab)y}} \\ & - b\left|\frac{as}{2(1+ab)y} + \frac{c\,c_k\sqrt{a}}{\sqrt{(a+b)(1+ab)y}} - \frac{s}{2(a+b)}\right| \\ & + \sqrt{\left(\frac{s}{2(1+ab)y} + \frac{c\,c_l\sqrt{a}}{\sqrt{(a+b)(1+ab)y}} - \frac{as}{2(a+b)}\right)^2 + \frac{a(1-c_l^2)}{(a+b)(1+ab)y}} \\ & - \left|\frac{s}{2(1+ab)y} + \frac{c\,c_l\sqrt{a}}{\sqrt{(a+b)(1+ab)y}} - \frac{as}{2(a+b)}\right|\Bigg] \\ F_B = 4s\Bigg[& b\sqrt{\left(\frac{as}{2(1+ab)} + \frac{c\,c_k\sqrt{a}}{\sqrt{(a+b)(1+ab)y}} - \frac{s}{2(a+b)y}\right)^2 + \frac{a(1-c_k^2)}{(a+b)(1+ab)y}} \\ & - b\left|\frac{as}{2(1+ab)} + \frac{c\,c_k\sqrt{a}}{\sqrt{(a+b)(1+ab)y}} - \frac{s}{2(a+b)y}\right| \\ & + \sqrt{\left(\frac{s}{2(1+ab)} + \frac{c\,c_l\sqrt{a}}{\sqrt{(a+b)(1+ab)y}} - \frac{as}{2(a+b)y}\right)^2 + \frac{a(1-c_l^2)}{(a+b)(1+ab)y}} \\ & - \left|\frac{s}{2(1+a)b} + \frac{c\,c_l\sqrt{a}}{\sqrt{(a+b)(1+ab)y}} - \frac{as}{2(a+b)y}\right|\Bigg],\end{aligned} \tag{F.39}$$

When we use these formulae in `Input_Measurement_Correlated.cpp` we now must explicitly insert an `if`-clause to resolve the limit $y \to 0$, because cmath can of course not simply evaluate the function there.

The limit of $y \to 0$ is for e.g. F_A:

$$F_A|_{y\to 0} = \frac{4a(1-c_l^2) + 4b(1-c_k^2)}{(a+b)} \tag{F.40}$$

Accordingly the function definition in `Input_Measurement_Correlated.cpp` takes the schematic form (double brackets are short for entire code segments)

```
double FA(decimal v,decimal t6,decimal y,decimal t5,decimal u,decimal b) {
[[variable definitions]]

   if ( y<=0. ) {
      FA= [[analytic limit for y=0]];
   } else {
      FA= [[Full expression]];
   }

   if ( a<=0. || cl>=1. || cl<=1. ) {
      FA=1.;
   }

[[predefined error corrections and return]]
}
```

where we also again introduced a clause capturing the zeroes that are present when either $a \to 0$ or $(c_l, c_k) \to (\pm1, \pm1)$, or $(c_l, b) \to (\pm1, 0)$. As for gauge boson production in section F.10, capturing a and c_l is enough, as the integrand vanishes as $a \to 0$, or $c_l \to \pm1$.

To top off the list of difficulties, expanding the functions F around $y = 0$ shows that we have $m = 0.5$, as for section F.10.

F.12.2 Uncorrelated emission

These functions are again simpler:

$$
\begin{aligned}
G_{Aa} &= \frac{b\sqrt{(s(1-ry)+2c\,c_k\sqrt{ry})^2+4(1-c_k^2)ry}-b\left|s(1-ry)+2c\,c_k\sqrt{ry}\right|}{2s(1+b)(1+y)ry} \\
&+\frac{\sqrt{(s(1-y)+2c\,c_l\sqrt{y})^2+4(1-c_l^2)y}-\left|s(1-y)+2c\,c_l\sqrt{y}\right|}{2s(1+b)(1+ry)y} \\
G_{Ab} &= \frac{b\sqrt{(s(1-y)+2c\,c_k\sqrt{y})^2+4(1-c_k^2)y}-b\left|s(1-y)+2c\,c_k\sqrt{y}\right|}{2s(1+b)(1+ry)y} \\
&+\frac{\sqrt{(s(1-ry)+2c\,c_l\sqrt{ry})^2+4(1-c_l^2)ry}-\left|s(1-ry)+2c\,c_l\sqrt{ry}\right|}{2s(1+b)(1+y)ry} \\
G_{B} &= \frac{b\sqrt{(s(1-y_k)+2c\,c_k\sqrt{y_k})^2+4(1-c_k^2)y_k}-b\left|s(1-y_k)+2c\,c_k\sqrt{y_k}\right|}{2s(1+b)(1+y_l)y_k} \\
&+\frac{\sqrt{(s(1-y_l)+2c\,c_l\sqrt{y_l})^2+4(1-c_l^2)y_l}-\left|s(1-y_l)+2c\,c_l\sqrt{y_l}\right|}{2s(1+b)(1+y_k)y_l}
\end{aligned}
\tag{F.41}
$$

The limits of $y_i \to 0$ need to be handled via `if`-clauses, just like for the correlated emissions.

Problematic limits

As already mentioned, the non-trivial and non-zero limits of $y \to 0$ (correlated) and $y_k \to 0$, $y_l \to 0$ (uncorrelated) must be handled properly.

Also there are zeroes at $a = 0$, or $|c_l| = 1$ together with certain values for b and c_k (correlated, already mentioned), and at $|c_l| = 1$ together with appropriate b and c_k values (uncorrelated), which must be captured in the code, as done above.

Rounding errors - multiprecision

Finally, the potentially large cancellation between root and absolute value is a problem for computer-based methods, because the usual data types (`double`, mainly) only resolve a certain finite number of digits. For Transverse Thrust this is an actual problem, and the "vanilla" flavour of `SoftSERVE` is not able to calculate this bare soft function. Using more digits solves this problem, and we therefore need to call `make [target] BOOST=1` to compile the executable for whatever `[target]` colour structure we need.

The default multiprecision backbone is provided by the header-only *boost* library. Two additional flavours of GMP are implemented via the *boost* library, but these need to be compiled before use, and are therefore not set as the default. More information can be found in the main bulk of the manual above, as well as in the appendices.

F.13 Rapidity-dependent Jet Veto - Thrust-like

The rapidity dependent jet vetoes are our prototypical non-Abelian exponentiation violating observables, and they deviate slightly from the Laplace transform recipe we've used so far.

The soft function here is

$$\begin{aligned}\mathcal{S}(\mathcal{T}^{Veto})_{R_0} = \sum_{k,l} \langle 0| S^\dagger S |k,l\rangle \langle k,l| SS^\dagger |0\rangle \times \\ \Theta\Big(\mathcal{T}^{Veto} - \Theta(R-R_0)\max\big(\min(k_+,k_-),\min(l_+,l_-)\big) \\ - \Theta(R_0-R)\min\big((k+l)_+,(k+l)_-\big)\Big),\end{aligned} \tag{F.42}$$

where the "distance" of two emissions R is given by

$$R = \sqrt{\frac{1}{4}\ln^2\frac{k_+l_-}{k_-l_+} + \theta_{kl}^2}\,. \tag{F.43}$$

This can again be Laplace transformed in $\mathcal{T}^{Veto}$, and the only change the step function introduces — compared to the Dirac-delta we had before — is a global factor $\frac{1}{\tau}$, where τ is the Laplace space conjugate to $\mathcal{T}^{Veto}$. This factor is a constant as seen by the integration, so for the numerical evaluation nothing changes. The only added complexity arises when we try to renormalise because we need to transform the bare function back to momentum space before doing so, and this is where we have to worry about τ^{-1}.

F.13.1 Correlated emission

Applying the correlated-emission parametrisation is straightforward, and we find

$$\begin{aligned}& n=1 \\ & F_A = \Theta\big(R(k,l)-R_0\big)\max\left(\frac{a}{a+b},\min\left(\frac{b}{a+b},\frac{ab}{(1+ab)y}\right)\right) + \Theta\big(R_0-R(k,l)\big)1 \\ & F_B = \Theta\big(R(k,l)-R_0\big)\max\left(\frac{ab}{1+ab},\min\left(\frac{1}{1+ab},\frac{a}{(a+b)y}\right)\right) + \Theta\big(R_0-R(k,l)\big)1\end{aligned} \tag{F.44}$$

with

$$R(k,l) = \sqrt{\arccos^2(1-2t_{kl}) + \ln^2 a}\,, \tag{F.45}$$

F.13.2 Uncorrelated emission

This observable is one of the very few for which the uncorrelated-emission functions are more complicated than the correlated-emission ones. We find

$$\begin{aligned}G_{Aa} &= \Theta\big(R(k,l)-R_0\big)\frac{1}{1+b}\frac{1}{1+ry} + \Theta\big(R_0-R(k,l)\big)\frac{1+b+y+bry}{(1+b)(1+y)(1+ry)} \\ G_{Ab} &= \Theta\big(R(k,l)-R_0\big)\frac{1}{1+b}\max\left(\frac{1}{1+y},\frac{b}{1+ry}\right) + \Theta\big(R_0-R(k,l)\big)\frac{1+b+ry+by}{(1+b)(1+y)(1+ry)}\end{aligned} \tag{F.46}$$

with $R(k,l) = \sqrt{\arccos^2(1-2t_{kl}) + \frac{1}{4}\ln^2 r}$ in region A, and

$$G_B = \Theta\big(R(k,l) - R_0\big)\frac{1}{1+b}\max\left(\frac{1}{1+y_k}, \frac{b}{1+y_l}\right) + \Theta\big(R_0 - R(k,l)\big)\frac{1+b+\min\left(\frac{1}{y_l}+by_k, \frac{b}{y_k}+y_l\right)}{(1+y_k)(1+y_l)(1+b)} \tag{F.47}$$

with $R(k,l) = \sqrt{\arccos^2(1-2t_{kl}) + \frac{1}{4}\ln^2 y_k y_l}$ in region B.

Note the appearance of $\ln r$ in region A, which arises from a $\ln \frac{y_k}{y_l}$ via the reparametrisation. This is a good place to demonstrate why this rescaling was necessary: The limit $y_k, y_l \to 0$ is ambiguous unless we specify if the two variables run to zero at fixed ratio or independently, necessitating the introduction of the variables r and y in this region.

Problematic limits

There is one problematic limit we need to defuse:

- $a \to 0$ in the correlated-emission case

All other apparent zeroes or poles are killed by the step functions, max, or min, before they can be realised.

F.14 Rapidity-dependent Jet Veto - C-Par.-like

The C-Parameter inspired version of the rapidity-dependent jet vetoes is structurally identical to the Thrust like version.

F.14.1 Correlated emission

Applying the correlated-emission parametrisation is straightforward, and we find

$$\begin{aligned}
&n = 1\\
&F_A = \Theta\big(R(k,l) - R_0\big)\max\left(\frac{a}{a+b+ay(1+ab)}, \frac{ab}{a(a+b)+y(1+ab)}\right) + \Theta\big(R_0 - R(k,l)\big)\frac{1}{1+y}\\
&F_B = \Theta\big(R(k,l) - R_0\big)\frac{a}{y(a+b)+a(1+ab)} + \Theta\big(R_0 - R(k,l)\big)\frac{1}{1+y}
\end{aligned} \tag{F.48}$$

with the same appropriate R-measure from the Thrust-like jet veto.

F.14.2 Uncorrelated emission

In the uncorrelated emission case we find

$$\begin{aligned}
G_{Aa} &= \Theta\big(R(k,l)-R_0\big)\frac{1}{(1+b)(1+ry)(1+y)} \\
&\quad + \Theta\big(R_0-R(k,l)\big)\frac{([1+y]+b[1+ry])(r[1+y]+b[1+ry])}{(1+b)(1+ry)(1+y)(b[1+ry]^2+r[1+y]^2)} \\
G_{Ab} &= \Theta\big(R(k,l)-R_0\big)\frac{1}{(1+b)(1+y)(1+ry)} \\
&\quad + \Theta\big(R_0-R(k,l)\big)\frac{(b[1+y]+[1+ry])(br[1+y]+[1+ry])}{(1+b)(1+ry)(1+y)([1+ry]^2+br[1+y]^2)} \\
G_B &= \Theta\big(R(k,l)-R_0\big)\frac{1}{(1+b)(1+y_k)(1+y_l)} \\
&\quad + \Theta\big(R_0-R(k,l)\big)\frac{(b[1+y_k]+y_k[1+y_l])([1+y_l]+by_l[1+y_k])}{(1+b)(1+y_k)(1+y_l)(by_l[1+y_k]^2+y_k[1+y_l]^2)}
\end{aligned} \tag{F.49}$$

again with the same appropriate, region-dependent $R(k,l)$ as for the Thrust-like jet veto.

Problematic limits

The observable vanishes in the correlated emission limit $a \to 0$, which requires the appropriate `if`-clause to defuse it.

F.15 Transverse momentum Jet Veto

This observable puts a veto on the rapidity-independent transverse momentum of the clustered or unclustered emissions, and therefore is an SCET-2 observable. Its soft function's definition reads

$$\begin{aligned}
\mathcal{S}(\mathcal{T}^{Veto})_{R_0} = \sum_{k,l} \langle 0|\, S^\dagger S\, |k,l\rangle \langle k,l|\, SS^\dagger\, |0\rangle \times \\
\Theta\Big(\mathcal{T}^{Veto} - \Theta(R-R_0)\max\big(k_T, l_T\big) - \Theta(R_0-R)(k+l)_T\Big),
\end{aligned} \tag{F.50}$$

with the same distance measure as for the rapidity-dependent jet vetoes above.

The only slight difficulty here arises from $(k+l)_T$, which is properly defined as

$$(k+l)_T = \sqrt{(\vec{k}_T+\vec{l}_T)^2} = \sqrt{k_+k_- + l_+l_- + 2\sqrt{k_+k_-l_+l_-}\,(1-2t_{kl})}\,, \tag{F.51}$$

and hence introduces angular dependence.

F.16 Soft drop jet mass

The soft-drop jet-grooming mass is an SCET-1 observable with parameter dependence and is straightforward up to one added complication: It contains nested `if`-clauses due to products of step functions, and therefore represents a good example of the possible complexity in the input that we may encounter.

To highlight the extent to which we can still deal with such complexity — and to be able to point to a specific example when we're invariably asked to provide automated tools to derive `SoftSERVE` input and need to show why that's just not possible — it is instructive to present this calculation in more detail.

The initial insights can be derived at one-loop order: The form of the global soft function is given in [10] by equation (C.2). This involves Heaviside step functions, rather than Dirac deltas, but a Laplace transform will work just as well, very much like for the jet vetoes above. The natural choice for the momentum space "observable" is the veto scale $z_{cut}\frac{Q}{2}$, as it is dimensionful. We would then Laplace transform with respect to this quantity and proceed as usual. However, we note in equation (C.5) that the logarithms to be resummed are $\ln\frac{2^\beta z_{cut}Q}{\mu}$. We therefore should use as the momentum-space observable $\Lambda = 2^\beta\, z_{cut}Q$, and Laplace transform with respect to that quantity. Otherwise we would find unaccounted factors and powers of 2 floating around. Doing so, we find at the two-loop level the measurement exponential's exponent to be the usual $-\tau\omega^{SD}(k,l)$ with

$$\begin{aligned}
\omega^{SD}(k,l) = \theta_1 \cdot \Big(&\theta_2 \cdot \Big[(k_- + k_+)^{1+\beta/2} k_-^{-\beta/2}\Big] \\
&+ \bar{\theta}_2 \cdot \Big[(l_- + l_+)^{1+\beta/2} l_+^{-\beta/2}\Big]\Big) \\
+ \theta_5 \cdot \Big\{&\theta_3 \cdot \Big[(k_+ + k_- + l_+ + l_-)^{1+\beta/2}(k_- + l_-)^{-\beta/2}\Big] \\
&+ \bar{\theta}_3 \cdot \Big(\theta_4 \cdot \Big[(k_+ + k_-)^{1+\beta/2} k_-^{-\beta/2}\Big] \\
&\qquad + \bar{\theta}_4 \cdot \Big[(l_+ + l_-)^{1+\beta/2} l_-^{-\beta/2}\Big]\Big)\Big\} \\
+ (n \leftrightarrow \bar{n})&
\end{aligned} \tag{F.52}$$

with

$$\begin{aligned}
\theta_1 &= \theta(k_+ - k_-)\theta(l_- - l_+) \\
\theta_2 &= \theta\Big((k_+ + k_-)^{1+\beta/2} k_-^{-\beta/2} - (l_+ + l_-)^{1+\beta/2} l_+^{-\beta/2}\Big) \\
\theta_3 &= \theta\left(k_- - \frac{2\,k\cdot l}{l_- + l_+}\right)\theta\left(l_- - \frac{2\,k\cdot l}{k_- + k_+}\right) \\
\theta_4 &= \theta\Big((k_+ + k_-)^{1+\beta/2} k_-^{-\beta/2} - (l_+ + l_-)^{1+\beta/2} l_-^{-\beta/2}\Big) \\
\theta_5 &= \theta(k_+ - k_-)\theta(l_+ - l_-),
\end{aligned} \tag{F.53}$$

and $\bar{\theta}_i$ denoting the complement to the thetas above, i.e. $\bar{\theta}_i = 1 - \theta_i$.

Note that the contents of the square brackets, i.e. the possible measurement values, are generically positive, and that — once the $(n \leftrightarrow \bar{n})$ is taken into account — there is a complement for every step function that appears.

F.16.1 Correlated emission case

The observable is as we can see rather complicated, and applying the parametrisations is not easy. Fortunately, there is a silver lining: The restriction to the unit cube for all variables means that some step functions will never have support in one (or sometimes both[5]) integration regions A and B. The rest is still complicated enough, though.

In the template source code we assign auxiliary variables `thIu` and `thIb` (for various values for ”`I`”) to the arguments of the relevant step functions that appear, with every θ_i then represented with an `if`, and every $\bar{\theta}_i$ corresponing to the accompanying `else`, and `resIu` and `resIb` to the possible functional values of the observable (i.e. the contents of the square brackets). In this scheme, `u` and `b` stand for the two halves of the observable as written down above: `u` (“unbarred”) stands for the half that is written down, and `b` (“barred”) stands for the $n \leftrightarrow \bar{n}$ exchanged part indicated in the last line. `I` is a number index labelling the different possible step function arguments according to the list above, or observable definitions in the order in which they appear from top downwards. Some step functions vanish entirely in one or both regions A or B, which explains why we find e.g. `if(th1u) {...}`, rather than `if(th1u && foo) {...}`. In the region under consideration, one of the two step functions in θ_1 is identically equal to 1.

Examining the behaviour of the step functions in the $y \to 0$ limit we find that

$$n = -1 - \beta \qquad \text{and} \qquad m = 1 \tag{F.54}$$

The template source file now contains therefore structures such as[6]

```
double FA(decimal v,decimal t6,decimal y,decimal t5,decimal u,decimal b) {
((predefined variable definitions))

th1u = [[some function of variables]]
[[other definitions of thIu and thIb]]

res1u = [[some function of variables]]
[[other definitions of resIu and resIb]]

   if ( th1u >= 0. ) {
      if ( th2u >= 0. ) {
         FA= res1u;
      } else {
         FA= res2u;
      }
   } else if ( th3b1 >= 0. && th3b2 >= 0. ) {
         FA= res3b;
      } else if ( th4b >= 0.) {
            FA= res4b;
         } else {
            FA= res5b;
         }

((predefined error corrections and return))
}
```

[5]At first glance it may be worrying that step functions should have no support in any of the input files. This is, however, a relic of the symmetry reduction; these step functions would have support in regions which are related by symmetry to A and B.

[6]We changed notation away from `boolean` variables and towards values for this schematic representation, to make it more readable.

where each `if` statement corresponds to one θ_i and the sequence of step functions diverts us to the result for a given momentum configuration.

To make more explicit how the form above arises, a description in words for region A (region B is similar):

- θ_1, θ_5, and their equivalents arising from $(n \leftrightarrow \bar{n})$ form a network of four complementary combinations of step functions.
- In region A we find that $\theta(l_- - l_+) \equiv 1$ identically, and so there are only two possibilites, rather than four: $\theta(k_+ - k_-)$ from θ_1 corresponding to `if (th1u >=0.)`, and $\theta(k_- - k_+)$ from the $(n \leftrightarrow \bar{n})$-exchanged version of θ_5, which provides the complementary `else`.
- Staying in the `if`-branch we have again a choice: either θ_2 or its complement $\bar{\theta}_2$ has support[7], yielding one of two possible results: `res1u` or `res2u`
- The `else` complementing to θ_1/`th1u` arises from the $(n \leftrightarrow \bar{n})$ part in the definition above, so all variables appearing in it carry the b suffix.
- Technically this `else` should also involve `th5b`, but as we can see analytically that it is the complement to `th1u`, we can get away with omitting it.
- The `if ( th3b1 >= 0.  && th3b2 >= 0.  )` is sourced by the $n/\bar{n}$-exchanged θ_3, which does not reduce to a single step function in region A (or B, for that matter), and therefore requires the `AND` operation `&&`.
- If this θ_3 is true, we find as the result `res3b`, the largest square bracket ($n/\bar{n}$-exchanged) in the definition above, and
- if it isn't, we need to check whether $n/\bar{n}$-exchanged θ_4 is true, yielding either the fourth or fifth line ($n/\bar{n}$-exchanged of course) of the definition above.

The actual values of the various auxiliary variables are easily but tediously derived from the initial observable definition, and hence we will not list them here. They can of course be found in the `SoftSERVE` distribution.

F.16.2 Uncorrelated emission case

The uncorrelated emission case proceeds identically to the correlated case, the expressions are as usual a bit simpler — we in particular find that several additional step functions are true or false identically in the various regions, reducing the number of nested `if-else`s — but nothing out of the ordinary happens.

Problematic limits

As usual, the correlated emissions' $a = 0$ causes trouble, but fortunately — saving us even more `if` clauses — that's the only problematic limit.

So *that*'s why

As should be readily obvious by now it is basically impossible to automatise the effort that goes into making an observable such as the soft drop jet mass ready for being `SoftSERVE`'d. And that's why it has to be done manually by the user.

[7]Note that here $\bar{\theta}_2 = 1 - \theta_2$. This bar is not associated with the $n \leftrightarrow \bar{n}$ exchange.

F.17 p_T resummation

Apart from being the first SCET-2 case in this list, the soft function relevant for transverse momentum resummation is schematically not terribly difficult, compared to others, with one exception: as it naturally arises in Fourier space, its measurement function (shown here for the correlated emission case) includes an imaginary unit:

$$\mathcal{M}_{p_T}(k,l) = \exp(-i\tau\omega^{p_T}(k,l)) = \exp(-2i\tau p_T\sqrt{\frac{a}{(a+b)(1+ab)}}(c_l + bc_k)) \tag{F.55}$$

It therefore falls under the special case outlined in appendix A of [1], and accordingly we provide the source files for the required modified input, i.e. that involving the measurement function $|F|$, with $n = 0$ (i.e. SCET-2), and

$$F_A = F_B = 2\sqrt{\frac{a}{(a+b)(1+ab)}}\,|c_l + bc_k| = 2\sqrt{\frac{a}{(a+b)(1+ab)}}\,|1 - 2t_l + b(1-2t_k)| \tag{F.56}$$

as well as

$$G_{Aa} = G_{Ab} = G_B = 2\frac{|c_l + bc_k|}{1+b} = 2\frac{|1-2t_l + b(1-2t_k)|}{1+b} \tag{F.57}$$

We merely have to remember that after getting the results for this observable, we have to run the `./fourierconvert` script before renormalising.

Non-trivial limits

As so often, there is a zero at $a = 0$, which we handle as usual. There is also an overlapping zero at $c_l = \pm 0$ with appropriate values for b or c_k. This latter zero resides in the bulk of the integration region, but only constitutes a lower dimensional hypersurface, which has measure zero. We therefore simply set the measurement function to 1 for relevant points. Additionally, there is a non-trivial limit at $(a, b) = (0, 0)$, which is already covered by the $a = 0$ zero.

Bibliography

[1] Guido Bell, Rudi Rahn, and Jim Talbert. "Generic dijet soft functions at two-loop order: correlated emissions". In: *JHEP* 07 (2019), p. 101. arXiv: 1812.08690 [hep-ph].

[2] Guido Bell, Rudi Rahn, and Jim Talbert. *Generic dijet soft functions at NNLO: uncorrelated emissions.* arXiv: 2004.08396 [hep-ph].

[3] Thomas Hahn. "CUBA: A Library for multidimensional numerical integration". In: *Comput. Phys. Commun.* 168 (2005). Korobov numbers set to 16033 numbers file, pp. 78–95. arXiv: hep-ph/0404043 [hep-ph]. URL: http://www.feynarts.de/cuba/.

[4] The boost library developer team. *boost C++ libraries.* 1.71.0. Version 1.71.0, retrieved 16/10/2019. 2019. URL: http://www.boost.org/.

[5] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library.* 6.1.2. 2016. URL: http://gmplib.org/.

[6] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. "MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding". In: *ACM Trans. Math. Softw.* 33.2 (June 2007). Version 4.0.2, retrieved and patched on 16/10/2019. URL: http://www.mpfr.org/.

[7] The boost library developer team. *boost library documentation, Performance comparison.* 2019. URL: http://www.boost.org/doc/libs/1_71_0/libs/multiprecision/doc/html/boost_multiprecision/perf/realworld.html.

[8] Guido Bell, Rudi Rahn, and Jim Talbert. "Two-loop anomalous dimensions of generic dijet soft functions". In: *Nucl. Phys.* B936 (2018), pp. 520–541. arXiv: 1805.12414 [hep-ph].

[9] Thomas Becher and Xavier Garcia i Tormo. "Factorization and resummation for transverse thrust". In: *JHEP* 06 (2015), p. 071. arXiv: 1502.04136 [hep-ph].

[10] Christopher Frye, Andrew J. Larkoski, Matthew D. Schwartz, and Kai Yan. "Factorization for groomed jet substructure beyond the next-to-leading logarithm". In: *JHEP* 07 (2016), p. 064. arXiv: 1603.09338 [hep-ph].