Generation of counterterms in MadNkLO

Simone Lionetti

April 3, 2019

This document describes the contents of the file `subtraction.py` (located in `madgraph/core`).

1 Classes

1.1 `SubtractionLeg`

For the purpose of the subtraction, it is impractical to carry around the whole information that is contained in an object of the type `base_objects.Leg`. A simpler object `SubtractionLeg` is therefore defined with the following three attributes:

- `n`: an integer that indicates the leg number in the process,
- `pdg`: the PDG identifier which specifies the type of particle,
- `state`: a flag to specify if the leg is in the initial or in the final state.

For convenience, a class `SubtractionLegSet` which represents a set of `SubtractionLeg`'s is also implemented. Internally, this is just a sorted `tuple` since it is assumed that order is irrelevant. However, this object also provides some additional useful methods.

1.2 `SingularStructure`

The `SingularStructure` class is designed to identify unresolved limits in phase space, and by extension counterterms. It is a recursive structure which represents a tree of `SubtractionLeg`'s in a process. At each level, the leaves are gathered into a `SubtractionLegSet` attribute called '`legs`' and the `SingularStructure` that specify sub-trees are grouped into a `list` called '`substructures`'. There are currently three classes that inherit from `SingularStructure` and represent different unresolved limits:

- `SoftStructure` indicates that all of its sub-legs are soft,
- `CollStructure` indicates that all of its sub-legs are collinear,
- `BeamStructure` specifies that a leg is taken from a hadron beam after some splitting.

The generic parent class `SingularStructure` can be instantiated to group together several structures and specify an additional set of legs; this feature is exploited in the implementation of momentum mappings to identify which particles to recoil against.

For the sake of concreteness, a non-trivial example of `SingularStructure` is illustrated in fig. 1. In the conversion to a string, `SingularStructure`'s are converted to a single character, and the PDG codes and the state labels of `SubtractionLeg`'s are suppressed,[1] such that the object of fig. 1 prints as

$$\texttt{C(C(C(5,13),C(7,10,11,16)),S(4,6,8),1,9).} \tag{1}$$

The conversion of objects to characters is carried out through the method `name()`, according to the rules

$$\begin{aligned} &\texttt{SingularStructure} \to \text{`}\,\text{'}, \quad &&\texttt{CollStructure} \to \text{`}\texttt{C}\text{'}, \\ &\texttt{SoftStructure} \to \text{`}\texttt{S}\text{'}, &&\texttt{BeamStructure} \to \text{`}\texttt{F}\text{'}. \end{aligned} \tag{2}$$

[1] This is the default behaviour, the printout can be tuned by calling the function `__str__()` explicitly and passing the keyword arguments `print_n`, `print_pdg` and `print_state`.

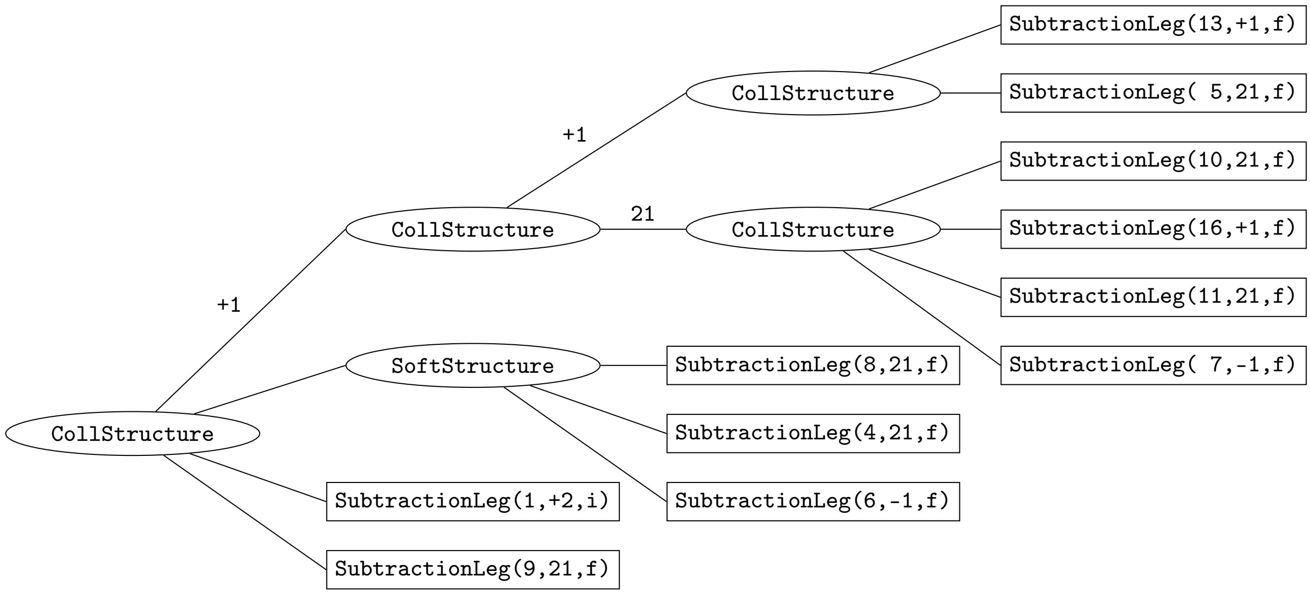

Figure 1: Example scheme of a `SingularStructure` object. The nodes in bubbles belong to the list of `substructure` of the structure they are linked to on the left, while the ones in boxes belong to its `legs`. Within `SubtractionLeg` objects, initial and final states are here abbreviated with the letters `i` and `f` respectively.

Whenever the species of particles and their state are not relevant or clear from the context, we will identify `SingularStructure`'s using their printout for brevity.

In order to understand how limits are specified using these objects, let us consider some simpler examples. While the `SingularStructure C(3,5,7,8)` indicates the limit where legs number 3, 5, 7 and 8 are simultaneously taken to be collinear, `(C(3,5),C(7,8))` denotes the limit where 3 goes collinear to 5, and 7 to 8.[2] These situations are still distinct from the nested limit `C(C(3,5),7,8)`, which approaches the configuration where all four particles are collinear by first letting legs 3 and 5 go to the same direction, and subsequently sending the angles among their common parent, leg 7 and 8 to zero. Similarly, `S(C(4,7))` indicates the limit of legs 4 and 7 going collinear to each other, and their parent leg subsequently going soft.

1.3 `Current`

1.4 `Counterterm`

1.5 `IRSubtraction`

2 Generation of local counterterms

2.1 Listing elementary structures

Given a process, the first step towards constructing a local infrared subtraction is to enumerate all of the different unresolved limits which potentially need to be regulated. In order to achieve this, we start with the identification of all possible *elementary* limits of a single set of particles simultaneously approaching a singular configuration. In practice, this corresponds to a set of particles which are either all soft or all collinear to each other, and may be represented by a `SingularStructure` of the appropriate type with no substructure.

After an `IRSubtraction` module has been initialised with the appropriate model and couplings specifications, a list of all relevant elementary structures can be obtained by calling the method

[2] Note that the outermost bracket around the latter printout denotes a `SingularStructure` of unspecified type, as indicated in eq. (2). Although the default behaviour does not distinguish among different orderings of the limits which appear at the same level, this can be tweaked using the keyword argument `orderless` of the comparison function `__eq__` for `SingularStructure`.

get_all_elementary_structures() on a base_objects.Process instance process for a given maximum number max_unresolved of unresolved particles. Internally, the generation proceeds as follows:

```
for n = 1 to max_unresolved do
    for each combination of n final-state legs do
        if the combination can become soft then
            add its soft structure to the list;
        end
    end
    for each combination of n + 1 final-state legs do
        if the combination can become collinear then
            add its collinear structure to the list;
        end
    end
    for each combination of n final-state legs do
        for each hadronic initial-state leg do
            if the set of these n + 1 legs can become collinear then
                add its collinear structure to the list;
            end
        end
    end
end
```

The conditional statements about whether a set of particles can become soft or collinear are implemented in the methods can_become_soft() and can_become_collinear() of IRSubtraction. They determine the candidate parent particles of the given legs using parent_PDGs_from_legs().[3]

2.2 Nesting structures

Lists of elementary structures can be assembled into nested structures of the type described in section 1.2, which are arguably more practical to process counterterms. This is achieved by grouping the list under a SingularStructure of unspecified type, and calling the method nest() on it. In turn, this relies on act_on().

Moreover, all possible combinations can be formed from a list of elementary structure using the method get_all_combinations of SingularStructure.

2.3 Assembling counterterms

References

[3] Note that in principle there might be more than one possible parent, as for instance in the case of a $q\bar{q}$ pair which might be produced from a gluon or a photon.