SAMURAI

Scattering AMplitudes from Unitarity-based Reduction Algorithm at Integrand-level

P. Mastrolia, G. Ossola, T. Reiter, and F. Tramontano

Abstract

These documentation notes provide a general description of the SAMURAI library, designed for the automated numerical evaluation of one-loop corrections to any scattering amplitudes. SAMURAI is based on the decomposition of the integrand according to the OPP-approach, extended to accommodate an implementation of the generalized d-dimensional unitarity-cuts technique, and uses a polynomial interpolation exploiting the Discrete Fourier Transform. It can process integrands with any number of external legs, written either as numerator of Feynman diagrams or as product of tree level amplitudes and it can be compiled in double or quadruple precision.

The package can be downloaded from:

`http://cern.ch/samurai`

1 Downloading and Installing SAMURAI

All the files are contained in the archive `samurai_v1.0.tar.gz`. The archive contains the files for the SAMURAI library [1], several examples of calculations, and also the routines for the evaluation scalar integrals QCDLoop [2] and OneLOop [3].

1. Download the archive `samurai_v1.0.tar.gz` and extract the files. They will be copied in a folder called `/samurai`.

2. Run the `Install` script. It will compile all useful routines and organize them. All routines are written in Fortran 90 and the default compiler is `gfortran`. In order to change compiler (or compiling options), the user should edit all the `makefile` commands.

   After running the `Install` script, you will find four subfolders within the `/samurai` directory: the subdirectory named `/libs` will contain all the libraries, namely the reduction routines in `libsamurai.a`, and three libraries for the calculation of scalar integrals.

   The folders named `/avh_olo_091110` and `/QCDLoop-1.9` contain the files required to produce the libraries for the scalar integrals. We include the latest version available (as of June 3rd, 2010). Updates can be downloaded from the webpages of their authors.

   Examples that reproduce all calculations described in the SAMURAI paper [1] can be found in separate subfolders in `/examples`. The `Install` script compiles all the examples, with the exception of the "Six Quarks" (that takes about 10 minutes to compile). The user can process it separately by typing `make` in the directory `/examples/uussbb`.

3. Run each process using the corresponding command `process.exe`. The description of the files contained in one specific example is given in Section 2.1.

2 Running SAMURAI

In this section we will provide an introduction about using SAMURAI and interfacing it with various kind on numerators. The sequence for each process should be as follow:

```
call initsamurai(imeth,isca,verbosity,itest)
call InitDenominators(nleg,Pi,msq,v0,m0,v1,m1,...,vlast,mlast)
call samurai(xnum,tot,totr,Pi,msq,nleg,rank,istop,scale2,ok)
call exitsamurai
```

To initialize the SAMURAI library, one needs to choose the arguments of the subroutine `initsamurai`

```
call initsamurai(imeth,isca,verbosity,itest)
```

specify the type of input to reduce (`imeth`), the routines for the numerical evaluation the scalar integrals (`isca`), the details of the output (`verbosity`), and the test to apply to the reconstruction:

`imeth` = `'diag'` or `'tree'`
`isca` = 1 (QCDloop); 2 (OneLOop)
`verbosity` = 0 (nothing); 1 (coeffs); 2 (coeffs+s.i.); 3(coeffs+s.i.+tests)
`itest` = 0 (none); 1 (powertest); 2 (nntest); 3 (lnntest)

- `imeth -` SAMURAI can reduce integrands of one-loop amplitudes defined as *numerator functions* sitting on products of denominators, by choosing `imeth=diag` (default option); or as *products of tree-level amplitudes* sewn along cut-lines, by choosing `imeth=tree`.

- `isca -` The user can trigger the use of QCDLoop [2] by assigning `isca=1`; or the use of OneLOop [3] with `isca=2`.

- `verbosity -` The level of information printed in the file `output.dat` can be chosen by increasing the value of `verbosity`:

  `verbosity=0`, no output from the library;

  `verbosity=1`, the coefficients are printed;

  `verbosity=2`, the value of the MI's are printed as well;

  `verbosity=3`, the outcome of the numerical test appears.

- `itest` - This option is used to select the test to monitoring the quality of the numerical reconstruction. The possibilities are `itest=0,1,2,3` to have respectively none, the global ($N = N$)-test, the local ($N = N$)-test, and the power-test.

  The thresholds for the reconstruction tests can be changed by setting their values in the file `ltest.dat`. The user is invited to copy this file from the directory `samurai` to the directory where there is the first call to `initsamurai` and edit it inserting the desired value for the limit of the tests. The phase-space points failing the tests are stored in a file called `bad_points.dat`. In principle they could be re-processed by enhancing the numerical precision obtained by compiling the SAMURAI library with quadruple precision options (ifort, lf95). While `imeth=diag` supports all the options for `itest`, the choice `imeth=tree` allows only `itest=0,2`. Details on the tests can be found in [1].

After selecting the routines for the scalar integrals and the reduction technique, the user should select a phase-space point (this can optionally be generated with `Rambo` [4]) and provide information about the integrand, that is specified by its own *numerator* and *denominators*.

Optionally the *denominators* of the diagram to be reduced can be filled with the help of the subroutine `InitDenominators` which generates the lists of internal momenta `Pi` and squared masses `msq` characterizing each propagator:

```
call InitDenominators(nleg,Pi,msq,v0,m0,v1,m1,...,vlast,mlast)
```

The arguments of the subroutine, labeled as input/output (`[i/o]`) according to their role, are:

- `nleg - [i]`. The integer number of the external legs of the diagram, corresponding to the number of denominators.

- `Pi - [o]`. The array `Pi(i,m)` contains the `nleg` four-vectors p_i present in the denominators of the integrand: we used the definition $\bar{D}_i = (\bar{q} + p_i)^2 - m_i^2 - \mu^2$. In the notation `Pi(i,m)`, the first index, `i=0,...,nleg-1`, runs on the set of the denominators; while the second index `m=1,...,4`, runs over the components of the vector, with the energy being given as 4th component.

- `msq - [o]`. The array `msq(i)`, is the list of the squared masses that appear in the propagators. The ordering `i=0,...,nleg-1` is bound to the list of momenta `Pi(i,m)`.
- `v0, m0 - [i]`. The vector `v0` and the mass `m0` are assigned to the first denominator.
- `vlast, mlast - [i]`. The vector `vlast` and the mass `mlast` are assigned to the last denominator.

The *numerator* of the diagram is defined an external function, whose name can be decided by the user, but with fixed arguments. Here we adopt the dummy name `xnum`.

- `xnum` - The complex function `xnum(icut,q,mu2)` is the integrand to be reduced. The arguments of the function `xnum(icut,q,mu2)` are: `icut`, an integer labeling the cut in which each digit corresponds to a cut-denominator in decreasing order (ex. 53210, 4210, 321); `q`, the virtual four-momentum, q (with the energy given as 4^{th} component); and `mu2` the extra-dimensional parameter, μ^2.

  When `imeth=diag`, `xnum` is expected to have the form of a numerator, hence being polynomial in q and μ^2. In this case `xnum` is a unique function to be processed at every level of the top-down reduction, and does not depend on the considered cut.

  When `imeth=tree`, `xnum` is expected to be formed by the product of tree-amplitudes, therefore it also contains the un-cut propagators. In this case, `xnum` is not unique, but should change according to the considered cut. Therefore, the value of `icut` yields a selective access to the proper integrand within the same function.

Having defined the integrand, `xnum`, and the corresponding denominators characterized by `Pi` and `msq`, the actual reduction is performed by the library SAMURAI,

```
call samurai(xnum,tot,totr,Pi,msq,nleg,rank,istop,scale2,ok)
```

which writes the total result of the reduction in `tot`. For convenience, the rational term that is part of `tot` is also separately stored in `totr`. Here is detailed description of each argument:

- `xnum - [i]`. Already defined.
- `tot - [o]`. The complex variable `tot` contains the final result for the integrated amplitude of numerator `xnum`. The finite part, that also includes the rational term, will be stored in `tot(0)`, while `tot(-1)` and `tot(-2)` contain the single and double poles, respectively.
- `totr - [o]`. For the purpose of comparisons and debugging, we also provide the rational part `totr` alone. This complex number is the sum of all contributions coming from integrals in shifted dimensions, namely all contributions that contain a dependence from μ^2 in the reconstructed integrand.
- `nleg - [i]`. Already defined.
- `Pi - [i]`. Already defined.
- `msq - [i]`. Already defined.
- `rank - [i]`. This integer value is the maximum rank of the numerator. This information is extremely valuable in order to optimize the reduction and improve the stability of the results. Using this information, we can simplify the reconstruction of the numerator by eliminating contributions that do not appear in the reduction. If the information about the rank of the integrand is not available, `rank` should be set equal to `nleg`.
- `istop - [i]`. This flag stops the reduction at the level requested by the user. `istop=5,4,3,2,1` will interrupt the calculation after determining pentagon-, box-, triangle-, bubble-, and tadpole-coefficients respectively. This procedure can be particularly useful to improve the precision of calculations when we know a priori that a particular set of integrals does not contribute.
- `scale2 - [i]`. This is the scale (squared) that is used in the evaluation of scalar integrals.
- `ok - [o]`. This logical variable carries information about the goodness of the reconstruction. The default values is `ok=true`, and it is set to `ok=false` when the reconstruction test fails.

As stated in Ref. [1], the generic one-loop integrand can be polynomial in ϵ up to the second-order. Each coefficient of the ϵ-decomposition can be assigned to a specific function, *i.e.* `xnum0`, `xnum1`, `xnum2`, which can be independently processed.

2.1 A simple example – Rank 6 tensor

We describe here the details of the simplest example that we provide with our code. This should be used as a first introduction to SAMURAI. More advanced options and more involved examples are discussed in the paper.

There are two files that are relevant for this example: `process.f90` is the main program in which we set up the arguments, we choose a phase space point, select the momenta and masses in the denominators and finally we call the reduction with a specific numerator function; `mxnum.f90` that contains the numerator function `xnum`.

Here is the list of operations contained in `process.f90`:

1. Choose the arguments: select the four values for `imeth` (method), `isca` (routines for scalar integrals), `verbosity` (printout), and `itest` (test on the reconstruction) as explained in the previous Section. The command

   ```
   call initsamurai(imeth,isca,verbosity,itest)
   ```

   will pass the information to the rest of the program.

2. Choose the configuration of the external/internal legs. The external momenta can be chosen by the user or generated with `Rambo`. They are called `vecs`. Starting from those, the user should define the internal momenta in the denominators, labeled by `Vi(:)`, and the corresponding squared masses `msq(i)`.

3. Call SAMURAI: the first argument is the name of the numerator function, to whom we gave the dummy name `xnum` (and it is contained in the file `mxnum.f90`). The command

   ```
   call samurai(xnum,tot,totr,Vi,msq,nleg,rank,istop,scale2,ok)
   ```

   will return the result in the array `tot`: `tot(0)` will contain the finite part, `tot(-1)` the pole $1/\epsilon$ and `tot(-2)` the double pole $1/\epsilon^2$.

The rational part is already summed within `tot(0)`; however, for the purpose of testing and debugging, it is also returned in the variable `totr`.

The logical variable `ok` contains the information about the outcome of the reconstruction test. If the value returned is `.false.`, the result is not safe and should be discarded or processed again with higher precision.

For practical calculations, this whole process should be included in a loop over a sample of phase space points. The procedure can be mimicked in the example by changing the parameter `nevt`.

Some examples make use of scalar and spinor products for the construction of the numerator. We provide some useful routines in the file `kinematic.f90`.

References

[1] P. Mastrolia, G. Ossola, T. Reiter, and F. Tramontano, SAMURAI.

[2] R. K. Ellis and G. Zanderighi, "Scalar one-loop integrals for QCD," *JHEP* **02** (2008) 002, `0712.1851`.

[3] A. van Hameren, C. G. Papadopoulos, and R. Pittau, "Automated one-loop calculations: a proof of concept," *JHEP* **09** (2009) 106, `0903.4665`.

[4] R. Kleiss, W. J. Stirling, and S. D. Ellis, "A NEW MONTE CARLO TREATMENT OF MULTIPARTICLE PHASE SPACE AT HIGH-ENERGIES," *Comput. Phys. Commun.* **40** (1986) 359.