NUMERICAL EVALUATION OF TWO-LOOP INTEGRALS WITH pySecDec[*]

S. Borowka

Theoretical Physics Department, CERN, Geneva, Switzerland

G. Heinrich, S. Jahn, S.P. Jones, M. Kerner

Max Planck Institute for Physics, Föhringer Ring 6, 80805 München, Germany

J. Schlenk

IPPP, University of Durham, South Road, Durham DH1 3LE, UK

(Received January 10, 2018)

We describe the program pySecDec, which factorises endpoint singularities from multi-dimensional parameter integrals and can serve to calculate integrals occurring in higher order perturbative calculations numerically. We focus on the new features and on frequently asked questions about the usage of the program.

DOI:10.5506/APhysPolBSupp.11.375

1. Introduction

At the CERN Large Hadron Collider (LHC), the exploration of the Higgs sector has just begun. Data with unprecedented precision are being and will be produced, allowing us to further explore fundamental questions like the nature of electroweak symmetry breaking, of which we only got a glimpse so far by the discovery of the Higgs boson. The comparison of these data to theoretical predictions is vital in order to identify effects of "New Physics", which may manifest themselves first indirectly, via loop effects. High precision theoretical predictions are, therefore, mandatory for the success of the LHC program, and even more so at future colliders.

[*] Presented at the Final HiggsTools Meeting, Durham, UK, September 11–15, 2017.

To increase the precision of the theoretical predictions, higher orders in the perturbative expansion in the strong and electroweak coupling constants need to be calculated. Such corrections often involve integrals depending on several kinematic/mass scales at two (or more) loops, where analytic results are hard to achieve.

In these cases, numerical approaches may offer a solution. A method which proved useful in the presence of dimensionally regulated singularities is sector decomposition [1–4], as it provides an algorithm to factorise such singularities in an automated way. The coefficients of the resulting Laurent series in the regulator are parametric integrals which can be integrated numerically. This algorithm has been implemented in the program SecDec [5–8], where from version 2.0 [6] the restriction to Euclidean kinematics was lifted by combining sector decomposition with a method to deform the multi-dimensional integration contour into the complex plane [9,10]. Other implementations of the sector decomposition algorithm can be found in Refs. [11–18].

In this article, we describe the new version of the SecDec program, called pySecDec [8]. We particularly focus on the user interface, providing answers to "Frequently Asked Questions".

2. Structure of the program

The program consists of two basic parts: an algebraic part, based on `python` and `FORM` [19–21], and a numerical part, based on `C++` code. The isolation of regulated endpoint singularities and the subsequent numerical integration can act on general polynomial functions, whereof Feynman integrals are a special case.

Loop integrals (after the Feynman parametrisation and momentum integration) can be considered as special cases of these more general polynomial integrands. In the `python` code, this is reflected by the following structure: The `python` function `make_package` accepts a list of polynomials raised to their individual powers as input — corresponding to the box (1b) in Fig. 1. In contrast, `loop_package` takes a loop integral (optionally from either its graph or its propagator representation), which corresponds to the box (1a). After constructing the Feynman parametrisation of the loop integral, `loop_package` calls `make_package` for further processing. The steps (1) to (7) are performed in `python` and `FORM`, where `FORM` produces optimized `C++` code. The compiled integrand functions are by default combined into a library. For the numerical integration, we provide a simple interface to integrators from the Cuba [22] library. The user also has direct access to the integrand functions, for example, to pass them to an external integrator.

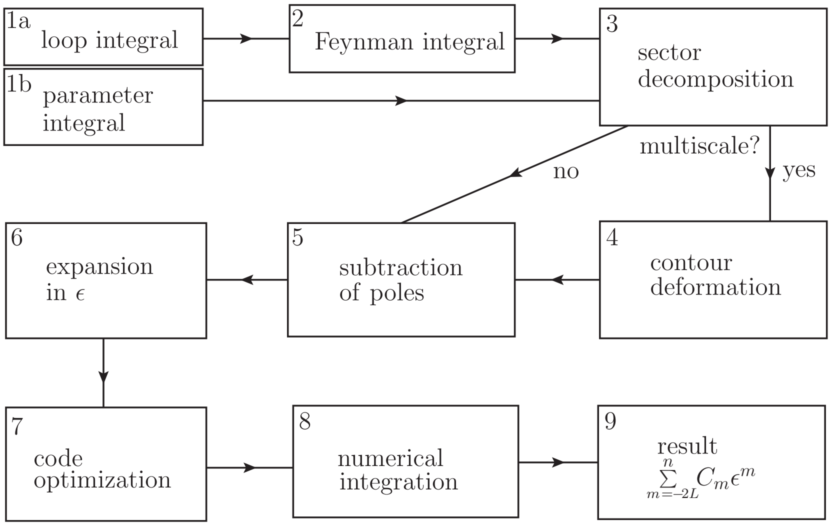

Fig. 1. Flowchart showing the main building blocks of pySecDec.

2.1. Installation and usage

The program can be downloaded from `http://secdec.hepforge.org` or `https://github.com/mppmu/secdec`. It relies on `python` and runs with versions 2.7 and 3. It also uses the packages `numpy` (`http://www.numpy.org`) and `sympy` (`http://www.sympy.org`).

It is easiest to install pySecDec from a release tarball available at `https://github.com/mppmu/secdec/releases/latest`. After download, pySecDec is installed by the following shell commands:

```
tar -xf pySecDec-<version>.tar.gz
cd pySecDec-<version>
make
<copy the highlighted output lines into your .bashrc>
```

The `make` command will automatically build further dependencies in addition to pySecDec itself. These are the Cuba library [22, 23] needed for multi-dimensional numerical integration, `FORM` [19–21] for the algebraic manipulation of expressions and to produce optimized `C++` code, Nauty [24] to find sector symmetries and the GSL library [25]. The lines to be copied into the `.bashrc` define environment variables which make sure that pySecDec and its dependencies are found. The pySecDec user is strongly encouraged to cite the additional dependencies when using the program.

2.1.1. Geometric sector decomposition strategies

The program Normaliz [26, 27] is needed for the geometric decomposition strategies `geometric` and `geometric_ku`. In pySecDec version 1.3, the versions 3.0.0, 3.1.0, 3.1.1, 3.3.0 and 3.4.0 of Normaliz are known to work. Precompiled executables for different systems can be downloaded from `https://www.normaliz.uni-osnabrueck.de`. We recommend to export its path to the environment of the terminal such that the *normaliz* executable is always found. Alternatively, the path can be passed directly to the functions that call it, using `normaliz_executable=[path_to_normaliz]`. The strategy `iterative` can be used without having Normaliz installed.

2.2. Usage

The program comes with detailed documentation in both `pdf` (`doc/pySecDec.pdf`) and `html` (`doc/html/index.html`) format. Online documentation can be found at `https://secdec.readthedocs.io/en/latest`. In the `examples` folder, we provide examples for several ways how to apply the program. One is to use pySecDec in a "standalone" mode to obtain numerical results for individual integrals. This corresponds to a large extent to the way previous SecDec versions were used. The other allows the generation of a library which can be linked to the calculation of amplitudes or other expressions, to evaluate the integrals contained in these expressions.

To get started, we recommend to read the section "getting started" in the online documentation. The basic steps can be summarised as follows:

1. Produce a `python` script to define the integral, the replacement rules for the kinematic invariants, the requested order in the regulator and some other options (see *e.g.* the one-loop box example `box1L/generate_box1L.py`).

2. Run the script using `python`. This will generate a subdirectory according to the `name` specified in the script.

3. Type `make -C <name>`, where `<name>` is your chosen name. This will create the `C++` libraries.

4. Produce a `python` script to perform the numerical integration using the `python` interface (see *e.g.* `box1L/integrate_box1L.py`).

Further usage options such as looping over multiple kinematic points are described in the documentation and in Ref. [8].

The algebra package can also be used for symbolic manipulations on integrals. This can be of particular interest when dealing with non-standard loop integrals, or if the user would like to interfere at intermediate stages of the algebraic part.

2.3. New features

In addition to the complete re-structuring and usage of open source software only, there are various new features compared to SecDec 3:

— The functions can have any number of regulators for endpoint singularities, not only the dimensional regulator ϵ.

— The treatment of numerators of loop integrals is more flexible. Numerators can be defined in terms of contracted Lorentz vectors or inverse propagators or a combination of both.

— The distinction between "general functions" and "loop integrands" is removed in the sense that all features are available for both, loop integrals and general polynomial functions (as far as they make sense outside the loop context).

— The inclusion of additional functions which do not enter the decomposition has been facilitated and extended.

— The treatment of poles which are higher than logarithmic has been improved.

— A procedure has been implemented to detect and remap singularities at $x_i = 1$ which result from special kinematic configurations.

— A symmetry finder [28] has been implemented which can detect isomorphisms between sectors.

— Diagrams can be drawn (optionally, based on `neato` [29]; the program will however run normally if `neato` is not installed).

— The evaluation of multiple integrals or even amplitudes is now possible, using the generated `C++` library.

2.4. Frequently asked questions

In the following, we list some questions which may come up during usage of the program, and give answers which should spare the user to search the manual.

— How can I adjust the numerical integration parameters?

If the `python` interface is used for the numerical integration, *i.e.* a `python` script like `examples/integrate_box1L.py`, the integration parameters can be specified in the argument list of the integrator call. For example, using `Vegas` as integrator:

```
box1L.use_Vegas(flags=2, epsrel=1e-3, epsabs=1e-12,
nstart=5000, nincrease=10000, maxeval=10000000,
real_complex_together=True)
```

or, using `Divonne` as integrator:

```
box1L.use_Divonne(flags=2, epsrel=1e-3, epsabs=1e-12,
maxeval=10000000, border=1e-8, real_complex_together=True)
```

The parameter `real_complex_together` tells the integrator to integrate real and imaginary parts simultaneously. A list of possible options for the integrators can be found at the end of Section 5.9 of the manual.

— How can I increase the numerical accuracy?

The integrator stops if any of the folllowing conditions is fulfilled: (1) `epsrel` is reached, (2) `epsabs` is reached, (3) `maxeval` is reached. Therefore, setting these parameters accordingly will cause the integrator to make more iterations to reach a more accurate result.

— How can I tune the contour deformation parameters?

You can specify the parameters in the argument of the integral call in the `python` script for the integration, see *e.g.* line 12 of `examples/integrate_box1L.py`:

```
str_integral_without_prefactor, str_prefactor,
str_integral_with_prefactor=box1L(real_parameters=[4.,-0.75,1.25,1.],
number_of_presamples=1000000,deformation_parameters_maximum=0.5)
```

This sets the number of presampling points to 10^6 (default: 10^5) and the maximum value for the contour deformation parameter λ, `deformation_parameters_maximum`, to 0.5 (default: 1). The user should make sure that `deformation_parameters_maximum` is always larger than `deformation_parameters_minimum` (default: 10^{-5}). These parameters are explained in Section 5.9. of the manual under “Parameters”.

— What can I do if the program stops with an error message containing *“sign_ check_ error”*?

This error occurs if the contour deformation leads to a wrong sign of the Feynman $i\,\delta$ prescription, usually due to the fact that the deformation parameter λ is too large.

Choose a larger value for `number_of_presamples` and a smaller value (*e.g.* 0.5) for `deformation_parameters_maximum` (see item above). If that does not help, you can try 0.1 instead of 0.5 for `deformation_parameters_maximum`.

— What does “additional_prefactor” mean exactly?

We should first point out that the conventions for additional prefactors defined by the user have been changed between SecDec 3 and pySecDec. The prefactor specified by the user will now be *included* in the numerical result.

To make clear what is meant by “additional”, we repeat our conventions for Feynman integrals here: A scalar Feynman graph G in D dimensions at L loops with N propagators, where the propagators can have arbitrary, not necessarily integer powers ν_j, has the following representation in momentum space:

$$\begin{aligned} G &= \int \prod_{l=1}^{L} \mathrm{d}^D\kappa_l \, \frac{1}{\prod\limits_{j=1}^{N} P_j^{\nu_j}\left(\{k\},\{p\},m_j^2\right)} \,, \\ \mathrm{d}^D\kappa_l &= \frac{\mu^{4-D}}{i\pi^{\frac{D}{2}}}\, \mathrm{d}^D k_l \,, \qquad P_j\left(\{k\},\{p\},m_j^2\right) = \left(q_j^2 - m_j^2 + i\delta\right) \,, \end{aligned} \tag{1}$$

where the q_j are linear combinations of external momenta p_i and loop momenta k_l. Introducing Feynman parameters leads to

$$\begin{aligned} G &= (-1)^{N_\nu} \frac{\Gamma(N_\nu - LD/2)}{\prod_{j=1}^{N}\Gamma(\nu_j)} \\ &\quad \times \int\limits_0^\infty \prod_{j=1}^{N} \mathrm{d}x_j \; x_j^{\nu_j - 1} \, \delta\left(1 - \sum_{l=1}^{N} x_l\right) \frac{\mathcal{U}^{N_\nu - (L+1)D/2}}{\mathcal{F}^{N_\nu - LD/2}} \,. \end{aligned} \tag{2}$$

The prefactor $(-1)^{N_\nu}\,\Gamma(N_\nu - LD/2)/\prod_{j=1}^{N}\Gamma(\nu_j)$ coming from the Feynman parametrisation will always be included in the numerical result, corresponding to `additional_prefactor=1` (default), *i.e.* the program will return the numerical value for G. If the user defines `additional_prefactor=‘gamma(3-2*eps)’`, this prefactor will be expanded in ϵ and included in the numerical result returned by pySecDec, in addition to the one coming from the Feynman parametrisation.

For general polynomials not related to loop integrals, *i.e.* in `make_package`, the prefactor provided by the user is the only prefactor, as there is no prefactor coming from a Feynman parametrisation in this case. This is the reason why in `make_package` the keyword for the prefactor defined by the user is `prefactor`, while in `loop_package` it is `additional_prefactor`.

— What can I do if I get ‘nan’?

This means that the integral does not converge and can have several reasons. When `Divonne` is used as an integrator, it is important to use a non-zero value for `border`, *e.g.* `border=1e-8`. `Vegas` is in general the most robust integrator. When using `Vegas`, try to increase the values for `nstart` and `nincrease`, for example `nstart=10000` (default: 1000) and `nincrease=5000` (default: 500).

— Can I include my own functions in the numerator of a loop integral?

Yes, as long as the functions are finite in the limit $\epsilon \to 0$. The numerator should be a sum of products of numbers, scalar products (*e.g.* `‘p1(mu)*k1(mu)*p1(nu)*k2(nu)’` and/or symbols (*e.g.* `‘m’`). The default numerator is 1. Examples:

```
p1(mu)*k1(mu)*p1(nu)*k2(nu) + 4*s*eps*k1(mu)*k1(mu)
p1(mu)*(k1(mu) + k2(mu))*p1(nu)*k2(nu)
p1(mu)*k1(mu)*my_function(eps)
```

More details can be found in Section 5.2.1 of the manual.

— How can I integrate just one coefficient of a particular order in ϵ?

You can pick a certain order in the `C++` interface (see Section 2.2.4 of the manual). To integrate only one order, change the line

```
const box1L::nested_series_t<secdecutil::UncorrelatedDeviation
<box1L::integrand_return_t» result_all = secdecutil::
deep_apply( all_sectors, integrator.integrate );
```

to

```
int order = 0;  // compute finite part only
const secdecutil::UncorrelatedDeviation<box1L::integrand_return_t>
result_all = secdecutil::deep_apply(
all_sectors.at(order), integrator.integrate );
```

where `box1L` is to be replaced by the name of your integral. In addition, you should remove the lines

```
std::cout << "-- prefactor -- " << std::endl;
const box1L::nested_series_t<box1L::integrand_return_t> prefactor =
 ↪  box1L::prefactor(real_parameters, complex_parameters);
std::cout << prefactor << std::endl << std::endl;

std::cout << "-- full result (prefactor*integral) -- " << std::endl;
std::cout << prefactor*result_all << std::endl;
```

because the expansion of the prefactor will in general mix with the pole coefficients and thus affect the finite part. We should point out however that deleting these lines also means that the result will not contain any prefactor, not even the one coming from the Feynman parametrisation.

— How can I use complex masses?

In the `python` script generating the expressions for the integral, define `mass_symbols` in the same way as for real masses, *e.g.*

```
Mandelstam_symbols = ['s', 't']
mass_symbols = ['msq']
```

In `loop_package` then define

```
real_parameters = Mandelstam_symbols,
complex_parameters = mass_symbols,
. . .
```

In the integration script (using the `python` interface), the numerical values for the complex parameters are given after the ones for the real parameters:

```
str_integral_without_prefactor, str_prefactor,
str_integral_with_prefactor = integral(
real_parameters=[4.,-1.25],complex_parameters=[1.-0.0038j])
```

Note that in `python`, the letter 'j' is used rather than 'i' for the imaginary part.

— When should I use the "split" option?

This option can be useful for integrals which do not have an Euclidean region. If certain kinematic conditions are fulfilled, for example, if the integral contains massive on-shell lines, it can happen that singularities at $x_i = 1$ remain in the $\mathcal{F}$ polynomial after the decomposition. The split option remaps these singularities to the origin of parameter space. If your integral is of this type, and with the standard approach, the numerical integration does not seem to converge, try the "split" option. It produces a lot more sectors, so it should not be used without need.

We also would like to mention that very often a change of basis can be beneficial if integrals of this type occur in the calculation.

3. Conclusions

We have described new features of the program pySecDec, which is publicly available at `https://github.com/mppmu/secdec`. pySecDec is entirely based on an open source software. The algebraic part can isolate end-point singularities in any number of regulators from general polynomial expressions, for example multi-loop Feynman integrals. For the numerical part, a library of `C++` functions is created, which allows very flexible usage, and, in general, outperforms SecDec 3 in the numerical evaluation times. In particular, it extends the functionality of the program from the evaluation of individual (multi-)loop integrals to the evaluation of larger expressions containing multiple integrals, as for example two-loop amplitudes. We have also provided answers to some questions which were frequently asked by pySecDec users.

This research was supported in part by the Research Executive Agency of the European Union under the Grant Agreement PITN-GA2012316704 (HiggsTools) and by the Munich Institute for Astro- and Particle Physics (MIAPP) of the DFG cluster of excellence "Origin and Structure of the Universe".

REFERENCES

[1] K. Hepp, *Commun. Math. Phys.* **2**, 301 (1966).

[2] M. Roth, A. Denner, *Nucl. Phys. B* **479**, 495 (1996) [`arXiv:hep-ph/9605420`].

[3] T. Binoth, G. Heinrich, *Nucl. Phys. B* **585**, 741 (2000) [`arXiv:hep-ph/0004013`].

[4] G. Heinrich, *Int. J. Mod. Phys. A* **23**, 1457 (2008) [`arXiv:0803.4177 [hep-ph]`].

[5] J. Carter, G. Heinrich, *Comput. Phys. Commun.* **182**, 1566 (2011) [`arXiv:1011.5493 [hep-ph]`].

[6] S. Borowka, J. Carter, G. Heinrich, *Comput. Phys. Commun.* **184**, 396 (2013) [`arXiv:1204.4152 [hep-ph]`].

[7] S. Borowka *et al.*, *Comput. Phys. Commun.* **196**, 470 (2015) [`arXiv:1502.06595 [hep-ph]`].

[8] S. Borowka *et al.*, *Comput. Phys. Commun.* **222**, 313 (2018) [`arXiv:1703.09692 [hep-ph]`].

[9] D.E. Soper, *Phys. Rev. D* **62**, 014009 (2000) [`arXiv:hep-ph/9910292`].

[10] S. Beerli, A New Method for Evaluating Two-loop Feynman Integrals and Its Application to Higgs Production, Ph.D. Thesis, ETC Zürich, 2008.

[11] C. Bogner, S. Weinzierl, *Comput. Phys. Commun.* **178**, 596 (2008) [`arXiv:0709.4092 [hep-ph]`].

[12] J. Gluza, K. Kajda, T. Riemann, V. Yundin, *Eur. Phys. J. C* **71**, 1516 (2011) [`arXiv:1010.1667 [hep-ph]`].

[13] T. Ueda, J. Fujimoto, *PoS* **ACAT08**, 120 (2008) [`arXiv:0902.2656 [hep-ph]`].

[14] T. Kaneko, T. Ueda, *PoS* **ACAT2010**, 082 (2010) [`arXiv:1004.5490 [hep-ph]`].

[15] A. Smirnov, M. Tentyukov, *Comput. Phys. Commun.* **180**, 735 (2009) [`arXiv:0807.4129 [hep-ph]`].

[16] A. Smirnov, V. Smirnov, M. Tentyukov, *Comput. Phys. Commun.* **182**, 790 (2011) [`arXiv:0912.0158 [hep-ph]`].

[17] A.V. Smirnov, *Comput. Phys. Commun.* **185**, 2090 (2014) [`arXiv:1312.3186 [hep-ph]`].

[18] A.V. Smirnov, *Comput. Phys. Commun.* **204**, 189 (2016) [`arXiv:1511.03614 [hep-ph]`].

[19] J.A.M. Vermaseren, `arXiv:math-ph/0010025`.

[20] J. Kuipers, T. Ueda, J.A.M. Vermaseren, *Comput. Phys. Commun.* **189**, 1 (2015) [`arXiv:1310.7007 [cs.SC]`].

[21] B. Ruijl, T. Ueda, J. Vermaseren, `arXiv:1707.06453 [hep-ph]`.

[22] T. Hahn, *Comput. Phys. Commun.* **168**, 78 (2005) [`arXiv:hep-ph/0404043`].

[23] T. Hahn, `arXiv:1408.6373 [physics.comp-ph]`.

[24] B.D. McKay, A. Piperno, `arXiv:1301.1493 [cs.DM]`.

[25] `https://www.gnu.org/software/gsl/`

[26] W. Bruns, B. Ichim, C. Söger, `arXiv:1206.1916 [math.CO]`.

[27] W. Bruns, B. Ichim, T. Römer, C. Söger, Normaliz. Algorithms for Rational Cones and Affine Monoids, available from `http://www.math.uos.de/normaliz`

[28] S.P. Jones, to appear in the proceedings of the 18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017), *J. Phys.: Conf. Series* (2017).

[29] `http://www.graphviz.org`