KA-TP-09-2023, P3H-23-035, IPPP/23/24

Numerical Scattering Amplitudes with pySecDec

G. Heinrich[a], S. P. Jones[b], M. Kerner[a], V. Magerya[a], A. Olsson[a], J. Schlenk[c]

[a]*Institute for Theoretical Physics, Karlsruhe Institute of Technology (KIT), 76128 Karlsruhe, Germany*

[b]*Institute for Particle Physics Phenomenology, Durham University, Durham DH1 3LE, UK*

[c]*ICS, University of Zurich, Winterthurerstrasse 190, 8057 Zurich, Switzerland*

Abstract

We present a major update of the program pySecDec, a toolbox for the evaluation of dimensionally regulated parameter integrals. The new version enables the evaluation of multi-loop integrals as well as amplitudes in a highly distributed and flexible way, optionally on GPUs. The program has been optimised and runs up to an order of magnitude faster than the previous release. A new integration procedure that utilises construction-free median Quasi-Monte Carlo rules is implemented. The median lattice rules can outperform our previous component-by-component rules by a factor of 5 and remove the limitation on the maximum number of sampling points. The expansion by regions procedures have been extended to support Feynman integrals with numerators, and functions for automatically determining when and how analytic regulators should be introduced are now available. The new features and performance are illustrated with several examples.

Keywords: Perturbation theory, Feynman diagrams, scattering amplitudes, multi-loop, numerical integration

PROGRAM SUMMARY

Manuscript Title: Numerical Scattering Amplitudes with pySecDec
Authors: G. Heinrich, S. P. Jones, M. Kerner, V. Magerya, A. Olsson, J. Schlenk
Program Title: pySecDec
Developer's repository: https://github.com/gudrunhe/secdec
Online documentation: https://secdec.readthedocs.io
Licensing provisions: GNU Public License v3
Programming language: Python, Form, C++, Cuda
Computer: from a single PC/Laptop to a cluster, depending on the problem; if the optional GPU support is used, Cuda compatible hardware is required.
Operating system: Unix, Linux
RAM: hundreds of megabytes or more, depending on the complexity of the problem
Keywords: Perturbation theory, Feynman diagrams, scattering amplitudes, multi-loop, numerical integration
Classification: 4.4 Feynman diagrams, 5 Computer Algebra, 11.1 General, High Energy Physics and Computing.
External routines/libraries: GSL [1], NumPy [2], SymPy [3], Nauty [4], Cuba [5], Form [6], GiNaC and CLN [7], Normaliz [8], GMP [9].
Journal reference of previous version: Comput. Phys. Commun. 273 (2022) 108267 [1].
Does the new version supersede the previous version?: yes
Nature of the problem:
Scattering amplitudes at higher orders in perturbation theory are typically represented as a linear combination of coefficients — containing the kinematic invariants and the space-time dimension — multiplied with loop integrals which contain singularities and whose analytic representation might be unknown.
Solution method:
Extraction of singularities in the dimensional regularization parameter as well as in analytic regulators for potential spurious singularities is done using sector decomposition. The combined evaluation of the integrals with their coefficients is performed in an efficient way.
Restrictions: Depending on the complexity of the problem, limited by memory and CPU/GPU time.
Running time: Between a few seconds and several days, depending on the complexity of the problem.
References:

[1] M. Galassi et al, GNU Scientific Library Reference Manual. ISBN:0954612078, http://www.gnu.org/software/gsl/.

[2] C. R. Harris, K. J. Millman, S. J. van der Walt, et al, Array programming with NumPy, Nature **585** (2020) 357–362. doi:10.1038/s41586-020-2649-2, http://www.numpy.org/.

[3] A. Meurer, et al., SymPy: symbolic computing in Python, PeerJ Comp. Sci. **3** (2017) e103. doi:10.7717/peerj-cs.103, http://www.sympy.org/.

[4] B. D. McKay and A. Piperno, Practical graph isomorphism, II, J. Symb. Comput. **60** (2014) 94–112. doi:10.1016/j.jsc.2013.09.003, http://pallini.di.uniroma1.it.

[5] T. Hahn, CUBA: A Library for multidimensional numerical integration, Comput. Phys. Commun. **168** (2005) 78. arXiv:hep-ph/0404043, http://www.feynarts.

de/cuba/.

[6] J. Kuipers, T. Ueda and J. A. M. Vermaseren, Code Optimization in FORM, Comput. Phys. Commun. **189** (2015) 1. arXiv:1310.7007, http://www.nikhef.nl/~form/.

[7] C. W. Bauer, A. Frink, and R. B. Kreckel, Introduction to the GiNaC framework for symbolic computation within the C++ programming language, J. Symb. Comput. **33** (2002) 1–12. arXiv:cs/0004015, https://www.ginac.de/.

[8] W. Bruns, B. Ichim, B. and T. Römer, C. Söger, Normaliz. Algorithms for rational cones and affine monoids. http://www.math.uos.de/normaliz/.

[9] T. Granlund et al, GMP: The GNU Multiple Precision Arithmetic Library. https://gmplib.org/.

1. Introduction

The calculation of scattering amplitudes beyond one loop is required in order to provide predictions for the increasingly precise measurements at the LHC, at B-factories and at other colliders. Furthermore, future lepton colliders require substantial progress in the calculation of higher order electroweak corrections, which usually involve several mass scales. The latter pose challenges for the evaluation of the corresponding integrals, in particular for analytic approaches. The program (py)SecDec [2, 3, 4, 5] offers the possibility to calculate multi-scale integrals beyond one loop numerically. Other public programs for the numerical evaluation of multi-loop integrals based on sector decomposition within dimensional regularisation [6, 7] are `sector_decomposition` [8] and Fiesta [9, 10, 11, 12, 13]. The program Feyntrop [14] provides a numerical approach for evaluating quasi-finite Feynman integrals using tropical sampling [15]. Other analytic/semi-analytic approaches include DiffExp [16, 17], AMFlow [18] and SeaSyde [19] which calculate Feynman integrals by solving differential equations using series expansions.

The program pySecDec has been upgraded recently with the ability to perform expansions by regions [1], a method pioneered in Refs. [20, 21, 22, 23]. Ref. [1] also describes an early implementation of an algorithm for efficiently calculating the weighted sum of integrals.

In this paper, we present pySecDec version 1.6, which is a major upgrade in several respects. One of the main changes is the fact that much more general coefficients of integrals than previously allowed are now supported. This feature is important for the calculation of amplitudes in a form resulting from IBP reduction, where the coefficients of the master integrals are usually sums of large rational polynomials containing kinematic invariants and the space-time dimension D. Furthermore, various changes in the code structure and numerical evaluation lead to a significant speed-up of the numerical evaluation. We present a new Quasi-Monte-Carlo (QMC) evaluator, called Disteval, which is optimised for a highly distributed evaluation. Another major improvement is achieved by the use of median generating vectors for the rank-1 lattice rules the QMC integration is based on. In addition, the feature of expansion by regions has been upgraded. For example, the program can automatically detect whether a regulator in addition to the dimensional regulator is needed in certain regions. In addition, the algebraic expressions multiplying each order of the expansion in a small parameter are provided to the user.

This article is structured as follows. In Section 2 the new features of version 1.6 are described. In Section 3 we present examples which demonstrate the usage of the program and the new features, as well as timings comparing previous pySecDec versions to the current version. Conclusions are presented in Section 4.

The release version of the code is available at `https://pypi.org/project/pySecDec/` and can be obtained via pip. The development version lives at `https://github.com/gudrunhe/secdec`. Online documentation can be found at `https://secdec.readthedocs.io/`.

2. New features of pySecDec

The main new features of pySecDec version 1.6 are a new integrator/importance sampling procedure (Disteval), support for construction-free median Quasi-Monte Carlo rules and improved support for expansion by regions.

The Disteval integrator is presented in Section 2.1, it implements a newly constructed Quasi-Monte-Carlo (QMC) integrator and is significantly faster and more configurable than our previous integrators. The Disteval integrator also comes with much better support for inputting complicated coefficients of the master integrals, including sums of rational functions resulting from the IBP reduction of amplitudes.

In Section 2.2, we describe and provide benchmarks of our implementation of *median Quasi-Monte Carlo rules*, a new QMC lattice construction based on Ref. [24]. The median QMC rules are made available in the Qmc and the Disteval integrators.

Improvements to the expansion by regions routines are described in Section 2.3. The new version of pySecDec supports Feynman integrals with numerators and provides functions for determining where an additional extra regulator, in addition to dimensional regularisation, is needed.

2.1. The new Quasi-Monte-Carlo evaluator DISTEVAL

pySECDEC traditionally comes with support for multiple integrators: `Qmc` based on the QMC library [5]; `Vegas`, `Suave`, `Divonne`, and `Cuhre` based on the CUBA library [25]; `CQuad` based on the GSL library [26]. Out of these we have recommended the usage of the `Qmc` integrator as the only one that achieves super-linear scaling of the integration precision with integration time for practical multidimensional integrals. All of these six integrators are available through a unified integration interface we shall call "INTLIB" (for lack of a better name).

With the new version of pySECDEC we introduce a new integration interface and an integrator "DISTEVAL". DISTEVAL implements a Randomized Quasi-Monte-Carlo (RQMC) integration method based on rank-1 shifted lattice rules [27, 28]. It is directly analogous to the INTLIB `Qmc` integrator, but with significantly higher performance, and the possibility of evaluation distributed across several computers. As with `Qmc`, DISTEVAL supports both CPUs and GPUs, with the latter ones being preferred due to their speed.

In Section 3 we provide a series of benchmarks demonstrating the speedup DISTEVAL provides over `Qmc` (usually between 3x and 10x) across a variety of integrals, on both CPUs and GPUs.

There are multiple sources of this speedup:

- While INTLIB integrands are compiled separately from the integration algorithms and are called indirectly by the integrators, DISTEVAL integrands fully include the integration loop. This enables the hoisting of the common code from the integration loop, the fusion of the lattice point generation and the integrand evaluation, and multiple micro-optimizations by the compiler. This however comes at the expense of flexibility in choosing integrators.
- The code for GPU integrands and CPU integrands are generated separately, allowing for separate optimization to be applied for each.
- On the GPU side DISTEVAL uses the highly optimized NVidia CUB library[1] to sum up the samples on the GPU (instead of performing the sum on the CPU), minimizing the data transfer between CPU and GPU.
- Modern CPUs are capable of executing multiple independent instructions in parallel. For example, an AMD EPYC 7F32 processor contains four floating-point execution units: two capable of performing

[1] `https://github.com/NVIDIA/cub`

one 256-bit Fused Multiply-Add (FMA) operation per cycle each, and two capable of one 256-bit addition operation per cycle each, for the total of 16 double-precision (i.e. 64-bit) operations per cycle. Saturating these executing units with work is essential in achieving optimal performance, and the best way to do that is to structure the code to operate on multiple values at the same time, packing 64-bit double-precision values into 256-bit arrays and utilizing SIMD[2] instructions that operate on the whole array at once.

The integrand kernels pySecDec generates for Disteval do exactly this: each mathematical operation is coded to work on 4 double-precision values simultaneously, and if the compiler is allowed to emit 256-bit SIMD instructions (i.e. via the AVX2 and FMA instructions sets on x86 processors), each such operation becomes a single instruction.

Note that while all modern x86 processors support AVX2 and FMA, some older ones do not, and because of this Disteval does not require their support. It is up to the user to check if all their target machines have this support,[3] and if so, to allow the compiler to use these instruction sets by e.g. setting `CXXFLAGS` to `-mavx2 -mfma` during compilation.[4] This is highly recommended.

Users that plan to perform integration on a single machine are advised to set `CXXFLAGS` to `-march=native`, so that the compiler would be allowed to auto-detect the capabilities of the processor it is running on, and use all the available instruction sets.

- Multiple smaller micro-optimizations on the CPU and the GPU sides to reduce the overhead for smaller integrands, and to speed up larger ones.

2.1.1. Using Disteval

Usage-wise, Disteval diverges from IntLib, during compilation and integration, but is similar enough that porting integration scripts should be easy.

As an example, let us consider a massless one-loop box. To generate the integration library for both integration interfaces, one can use the following Python script:

[2]"Single Instruction Multiple Data."

[3]This can be done by checking the presence of `avx2` and `fma` flags in `/proc/cpuinfo`.

[4]See e.g. https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html for a description of machine-specific options of GCC.

```python
import pySecDec as psd
if __name__ == "__main__":
    li = psd.LoopIntegralFromPropagators(
        loop_momenta=["l"],
        external_momenta=["p1","p2","p3"],
        propagators=["l**2","(l-p1)**2","(l-p1-p2)**2","(l-p1-p2-p3)**2"],
        replacement_rules=[
            ("p1*p1","0"),   ("p2*p2","0"),   ("p3*p3","0"),
            ("p1*p2","s/2"), ("p2*p3","t/2"), ("p1*p3","-s/2-t/2")])
    psd.loop_package(
        name="box1L",
        loop_integral=li,
        real_parameters=["s","t"],
        requested_orders=[0])
```

Then, to compile the IntLib library one can invoke `make` from the command shell:

```
make -C box1L -j4
```

Similarly, to compile the Disteval library one can use:[5]

```
make -C box1L -j4 disteval
```

The resulting library will be fully contained in the `box1L/disteval/` directory, meaning that the directory can be freely moved to a different location. The file `box1L/disteval/box1L.json` will contain the full description of the requested integral, and will work as the entry point to the library.

If one wants to use the resulting library on a GPU with “compute capability” 8.0, one should add `SECDEC_WITH_CUDA_FLAGS="-arch=sm_80"` to the arguments of the `make` call.[6] For IntLib this will build a library that can only be used on the GPU; for Disteval the resulting library will be able to work with and without a GPU.

To integrate using IntLib one can use the Python interface:

```python
from pySecDec.integral_interface import IntegralLibrary
lib = IntegralLibrary("box1L/box1L_pylink.so")
lib.use_Qmc()
_, _, result = lib(real_parameters=[4.0, -0.75], esprel=1e-3, epsabs=1e-8)
print(result)
```

Similarly, to integrate using Disteval one can use the Python interface:

[5] As noted earlier, adding `CXXFLAGS="-mavx2 -mfma"` to this `make` call is recommended.

[6] The list of NVidia “Compute Capability” codes for different GPUs is available at https://developer.nvidia.com/cuda-gpus.

```
from pySecDec.integral_interface import DistevalLibrary
lib = DistevalLibrary("box1L/disteval/box1L.json")
result = lib(parameters={"s": 4.0, "t": -0.75}, epsrel=1e-3, epsabs=1e-8)
print(result)
```

Alternatively, one can also use the new command-line interface:

```
python3 -m pySecDec.disteval box1L/disteval/box1L.json \
    s=4 t=-0.75 --epsrel=1e-3 --epsabs=1e-8
```

2.1.2. Distributed evaluation

The integrand evaluation under DISTEVAL is performed by worker processes, while the main process is responsible for distributing work among the workers and processing the results. Communication between the main and the worker processes is done via bidirectional bytestreams (i.e. pipes), using a custom `json`-based protocol, which means that the workers do not need to be located on the same machine as the main process.

By default, the Python interface of DISTEVAL will launch one worker process per locally available GPU, or one per locally available CPU. Each CPU worker is launched with the command

```
python3 -m pySecDecContrib pysecdec_cpuworker
```

and each GPU worker is launched with the command

```
python3 -m pySecDecContrib pysecdec_cudaworker -d <i>
```

where `<i>` is the (zero-based) index of the GPU this worker should use.

The default worker selection however can be overridden through the `workers` argument of `DistevalLibrary` to allow execution on different machines. For example, suppose that the integration is to be spread across two machines: `gpu1` with a single GPU, and `gpu2` with two GPUs; if both machines are reachable via `ssh`, then one could setup the integration library as follows:

```
lib = DistevalLibrary(
    "box1L/disteval/box1L.json",
    workers=[
        "ssh gpu1 python3 -m pySecDecContrib pysecdec_cudaworker -d 0",
        "ssh gpu2 python3 -m pySecDecContrib pysecdec_cudaworker -d 0",
        "ssh gpu2 python3 -m pySecDecContrib pysecdec_cudaworker -d 1"
    ])
```

2.1.3. Adaptive weighted sum evaluation

Since pySecDec version 1.5 IntLib supports adaptive integration of weighted sums of integrals (e.g. amplitudes) via the `sum_package()` function. Additionally, versions of `loop_package()` and `make_package()` implemented in terms of `sum_package()` have been added. Disteval implements a very similar adaptive sampling algorithm.

Suppose we have a set of integrals I_i, and we want to calculate a set of their weighted sums $A_k \equiv \sum_i C_{ki} I_i$. When evaluated under RQMC, each I_i can be thought of as a normally distributed random variable,

$$I_i \sim \mathcal{N}(\mathrm{mean}(I_i), \mathrm{var}(I_i)). \tag{1}$$

Let us assume that it takes τ_i of time to evaluate the integrand of I_i once, and that $\mathrm{var}(I_i)$ scales with the number of integrand evaluations n_i (a.k.a. the size of the lattice on which the integrand is evaluated) as

$$\mathrm{var}(I_i) = \frac{w_i}{n_i^\alpha}. \tag{2}$$

Our objective then is to choose n_i as functions of C_{ki}, w_i, τ_i, and α, to minimize the total integration time

$$T \equiv \sum_i \tau_i n_i, \tag{3}$$

while achieving the total variance V_k requested by the user:

$$\mathrm{var}(A_k) = \sum_i |C_{ji}|^2 \frac{w_i}{n_i^\alpha} = V_k \;\; (\forall k)\,. \tag{4}$$

We solve this optimization problem via the Lagrange multiplier method:

$$L \equiv T + \sum_k \lambda_k \left(\mathrm{var}(A_k) - V_k\right), \qquad \text{and} \qquad \frac{\partial L}{\partial \left\{n_i, \lambda_k\right\}} = 0. \tag{5}$$

If only one sum A_k needs to be evaluated, then these equations have a closed-form solution:

$$\begin{aligned} \lambda_k &= \frac{1}{\alpha}\left(\frac{1}{V_k}\sum_k \left(|C_{jk}|^2 w_k \tau_k^\alpha\right)^{\frac{1}{\alpha+1}}\right)^{\frac{\alpha+1}{\alpha}}, \\ n_i &= \left(\frac{\alpha w_i}{\tau_i}\lambda_k |C_{ji}|^2\right)^{\frac{1}{\alpha+1}}. \end{aligned} \tag{6}$$

If multiple sums are requested, Disteval uses this formula first for the first sum, then updates n_i and applies it to the next sum, and so on.

To make this work in practice, Disteval needs to estimate the integral evaluation speed τ_i, convergence constants w_i, and the power α. The evaluation speed τ_i is estimated on-line, by first benchmarking the relative performance of each worker, and then by tracking how fast a given integral is being evaluated on a given worker. The convergence constants w_i are first estimated by evaluating all integrals with some preset minimum lattice size (10^4 by default), and then updated each time an integration result is obtained. The parameter α is chosen conservatively to be 2, which is the minimum asymptotic scaling guaranteed by the use of QMC methods (for some examples see Figure 7 where $\alpha \approx 3$, and Figure 9 where $\alpha \approx 2$).

Here it is important to note that the scaling law of Eq. (2) is only asymptotic. In practice the usage of rank-1 lattice rules means that for each lattice size n_i we must construct a completely new lattice, and often larger n_i results in a larger error, instead of a smaller one — a phenomenon which we call *unlucky lattices*.

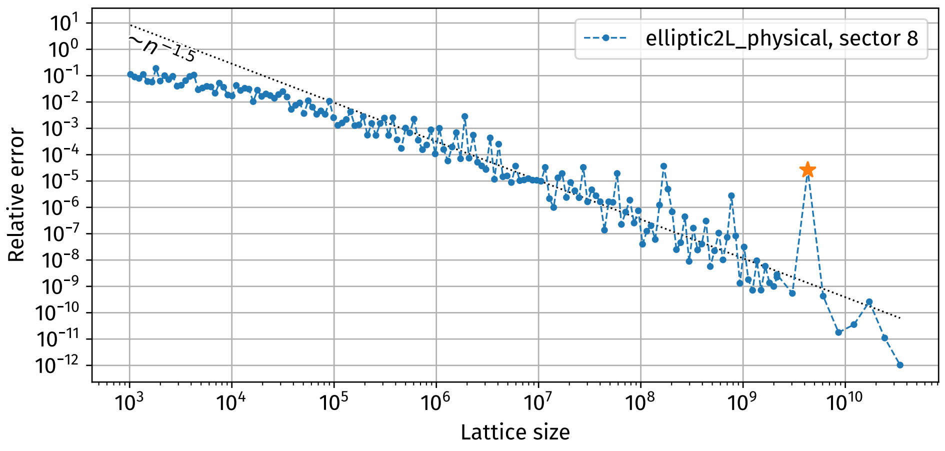
As an illustration, consider Figure 1: although the variance overall scales as $1/n^3$ (and thus the error as $1/n^{1.5}$), the progression is not monotonic, and one particularly unlucky lattice results in an integration error more than four orders of magnitude worse than lattices of similar size around it — but only for one of the integrals, for the other the same lattice gives a perfectly good result.

This scaling structure makes the integration times inherently unpredictable: if during the integration an integral is evaluated on an unlucky lattice, then Disteval will overestimate the integral's w_i parameter, and will assume that many more samples of this integral are needed to achieve the requested precision, wasting integration time. The practical impact of this is usually low to moderate, unless one encounters a very unlucky lattice such as the one marked with a star in Figure 1. To some extent, this effect can be tamed by the *median QMC rules*, introduced in the following section.

2.2. Median Quasi-Monte Carlo rules

The Quasi-Monte Carlo integration in previous versions of pySecDec was based on pre-computed generating vectors, provided with the Qmc library [5]. These generating vectors were constructed using the component-by-component (CBC) method [29], minimizing the worst-case error of the QMC integration, assuming arbitrary integrands belong to a Korobov space with smoothness $\alpha = 2$ and using product weights.

However, for a given integrand, a lattice of size n based on the above CBC construction might not be the optimal choice, resulting in the unlucky lattices illustrated in the previous section. Furthermore, constructing lattices via the CBC method is computationally expensive, and the largest



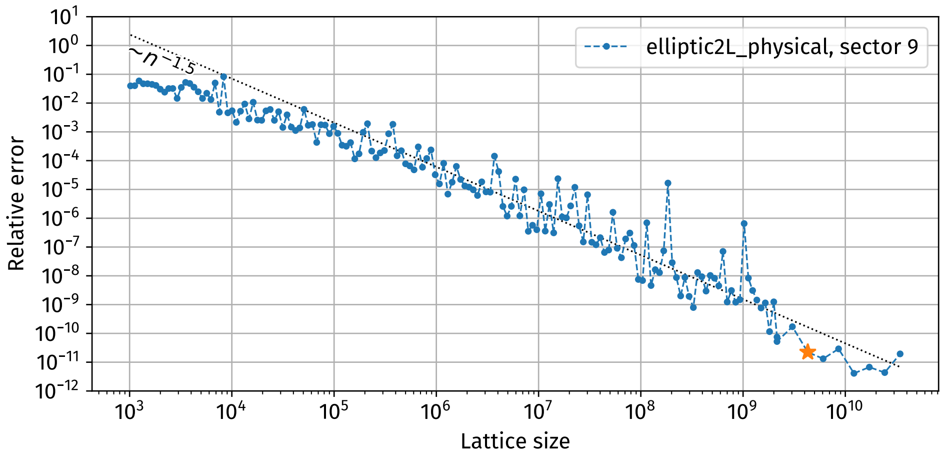


Figure 1: The RQMC integration error (i.e. $\sqrt{\mathrm{var}(I_i)/m}$) after $m = 32$ repetitions for lattices of different sizes. The integrals are sectors of the `elliptic2L_physical` example from Section 3.2. The lattices are taken from the QMC library, and are the same for both integrals. The result of one particularly unlucky lattice is marked with a star; note that this lattice is only unlucky for one of the sectors and performs normally for the others.

Lattices \ Accuracy	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
CBC	1.7 s	1.8 s	1.8 s	2.3 s	3.9 s	18 s	452 s	51.6 m	98.9 m
Median, $r = 3$	1.7 s	1.7 s	1.8 s	2.2 s	3.7 s	12 s	44 s	3.3 m	13.8 m
Median, $r = 5$	1.7 s	1.7 s	1.8 s	2.2 s	3.7 s	12 s	44 s	2.8 m	8.0 m
Median, $r = 7$	1.7 s	1.8 s	1.7 s	2.1 s	4.2 s	12 s	39 s	2.8 m	9.4 m
Median, $r = 11$	1.7 s	1.7 s	1.8 s	2.2 s	3.7 s	12 s	37 s	2.6 m	7.5 m
Median, $r = 15$	1.7 s	1.8 s	1.8 s	2.2 s	3.5 s	10 s	38 s	2.8 m	8.2 m
Median, $r = 23$	1.7 s	1.8 s	1.9 s	2.3 s	3.9 s	12 s	39 s	2.7 m	14.8 m
Median, $r = 31$	1.7 s	1.9 s	2.0 s	2.4 s	4.3 s	14 s	46 s	3.5 m	11.1 m
Median, $r = 63$	1.8 s	2.0 s	2.2 s	2.9 s	5.8 s	21 s	66 s	4.6 m	16.7 m

Table 1: Average integration times for the `elliptic2L_physical` example using the DISTEVAL integrator depending on the requested accuracy and the lattice construction method, comparing lattices derived via CBC and median QMC rules. The timings were taken using an NVidia A100 80G GPU, with the integrands compiled using CUDA 11.8.89.

such lattice currently provided by the QMC library has $\sim 7 \cdot 10^{10}$ sampling points. If the requested precision of the integral can not be achieved with the largest available lattice, the error can only be improved by repeated sampling of this lattice with random shifts, resulting in a $n^{-1/2}$ scaling of the integration error, negating the benefits of QMC integration.

An alternative to the CBC construction called *median QMC rules* has been proposed in [24]. This construction is based on the observation that most generating vectors are good choices, provided the components are chosen from the set

$$\mathbb{U}_n \in \{1 \leq z \leq n-1 \,|\, \gcd(z,n) = 1\}. \tag{7}$$

For r randomly selected generating vectors $\mathbf{z}_1, \ldots, \mathbf{z}_r$ satisfying this condition, it has been shown that using the median

$$M_{n,r}(f) = \text{median}(Q_{n,\mathbf{z}_1}(f), \ldots, Q_{n,\mathbf{z}_r}(f)) \tag{8}$$

as an integral estimate results in the same convergence rate as the CBC construction with high probability (the larger r is chosen, the higher the probability). Here, $Q_{n,\mathbf{z}}(f)$ is the estimate for the integral of f, obtained using the rank-1 lattice rule with generating vector $\mathbf{z}$.

In pySECDEC we now provide the possibility for an automated construction of generating vectors following this method. It can be enabled with the option `lattice_candidates=r`, which specifies the number r of randomly chosen generating vectors. After selecting the generating vector according to the median QMC rules, the uncertainty of the integration is then obtained by sampling the integrand on m different random shifts of this lattice, as in previous versions of pySECDEC. Using this method, the construction of lattices of arbitrary size n is possible, and since the generating vectors are

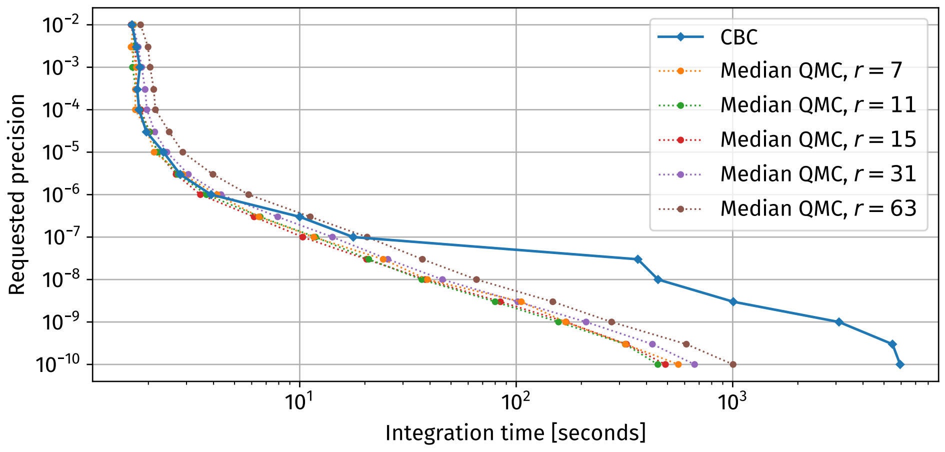


Figure 2: Integration time of the `elliptic2L_physical` example from Section 3.2 using the median QMC rules compared to the integration using CBC construction of the generating vectors. This plot uses the same benchmarking setup as Table 1.

chosen individually for each integrand, the problems due to unlucky lattices becomes less pronounced. With the default setting `lattice_candidates=0`, only the pre-computed generating vectors based on CBC construction are used.

As an illustration, consider the integration time of the `elliptic2L_physical` example presented in Table 1 and Figure 2. With CBC lattices, two of the most complicated sectors of the example share a particularly unlucky lattice at $n = 4.3 \cdot 10^9$, as depicted in Figure 1. Starting with the requested precision of around $3 \cdot 10^{-8}$ the evaluator consistently hits these lattices, and the integration time goes up significantly. On the other hand, the integration time with median QMC rules increases smoothly across the whole range.

A disadvantage of the median QMC rules is that, compared to using pregenerated lattices, an extra r samples of the integral are required in addition to the m samples used to estimate the integral uncertainty. In practice we typically find that despite this overhead, the integration time using the median QMC rules is either comparable to or improves upon that using lattices pregenerated via the CBC construction. This of course depends on the number of lattice candidates, r: with small r unlucky lattices are more likely, while large r means more overhead. For the `elliptic2L_physical` example, this behavior can be seen in Table 1: $r = 11$ and $r = 15$ seem to perform the best overall, while lower and higher r result in more integration time on average.

We have tested applying the generating vectors obtained for one particular integral using the median QMC rules to different integrals, thus lowering the overhead by avoiding the construction of new generating vectors for each integrand. However, we find that this typically leads to longer integration times, as a median lattice selected for a given integrand often does not turn out to be a high-quality choice for other integrands.

2.3. Extra regulators for Expansion by Regions

When expansion by regions is applied to a well-defined dimensionally regulated integral, new spurious singularities may be introduced which are not regulated by the original regulator. It is possible to detect geometrically which integrals will become ill-defined after expansion [1].

One way to handle the new singularities is to generalise the definition of the integral by adding new analytic regulators, $\delta_1, \ldots, \delta_N$. Commonly, this is done for Feynman integrals by altering the power of Feynman propagators according to $(\nu_1 \to \nu_1 + \delta_1, \ldots, \nu_N \to \nu_N + \delta_N)$, or, in the Feynman parametrisation, by multiplying the integrand by $x_1^{\delta_1} \cdots x_N^{\delta_N}$, where x_i are Feynman parameters. Introducing N independent new regulators can dramatically increase the complexity of the problem and is often unnecessary. Using the algorithms described in Ref. [1], several new routines for detecting and handling spurious divergences have been added to pySecDec, focusing on Feynman (loop) integrals.

The `loop_regions` function now accepts the argument `extra_regulator_name`. If a string or symbol is passed to this argument, pySecDec automatically determines if an extra regulator is required and, if so, introduces a single new regulator. The integrand is multiplied by $\mathbf{x}^{\boldsymbol{\nu}_\delta \delta}$ where δ is the extra regulator and $\boldsymbol{\nu}_\delta$ is a vector of integers automatically chosen such that the integral becomes well-defined. Alternatively, the user may pass a specific $\boldsymbol{\nu}_\delta$ as a list of integers or SYMPY rationals via the argument `extra_regulator_exponent`.

The function `suggested_extra_regulator_exponent`, which the user can call independently of `loop_regions`, automatically determines a vector of integers $\boldsymbol{\nu}_\delta$ sufficient to make a loop integral well-defined. Given a `loop_integral` object and the parameter in which it should be expanded, `smallness_parameter`, the function returns $\boldsymbol{\nu}_\delta$. There is considerable freedom in choosing the entries of $\boldsymbol{\nu}_\delta$. The only important property is that its entries must obey a set of inequalities which ensure it is not tangent to any of the hyperplanes spanned by the set of new (internal) facets, introduced by the expansion, which lead to spurious singularities. The `suggested_extra_regulator_exponent` function returns only one choice for $\boldsymbol{\nu}_\delta$, it obeys the additional constraint $\sum_i \boldsymbol{\nu}_{\delta,i} = 0$, which ensures that the new regulator does not appear in the power of the $\mathcal{U}$ or $\mathcal{F}$ polynomials.

The function `extra_regulator_constraints` provides the list of constraints which must be obeyed by the entries of $\boldsymbol{\nu}_\delta$ for it to regulate the new singularities. The user may call this function independently, for example, if they wish to impose additional constraints on the analytic regulators or if they want to understand the regions giving rise to spurious singularities and how they cancel. The function returns a dictionary of regions and constraints that must be obeyed in order to obtain regulated integrals, along with a complete list of all constraints (the `all` entry). Each set of constraints is provided as an array, each row of which can be interpreted as the elements of a vector $\mathbf{n}_f$ normal to an internal facet, f, which gives rise to a spurious singularity. The integral is regulated by any vector $\boldsymbol{\nu}_\delta$ which obeys $\langle \mathbf{n}_f, \boldsymbol{\nu}_\delta \rangle \neq 0 \ \forall f$.

The example `region_tools` demonstrates the use of each of the above functions on a 1-loop box integral with an internal mass.

2.4. New functionalities for coefficients of master integrals

To evaluate one or several weighted sums of integrals pySecDec provides the function `sum_package()` that takes a list of integrals I_i, and a matrix of coefficients C_{ki} (given as a list of its rows), so that in the end the weighted sums $A_k \equiv \sum_i C_{ki} I_i$ are evaluated. In version 1.5 the coefficients were required to be instances of the class `Coefficient`, and to be specified as a product of polynomials.

The new version of pySecDec now additionally supports more flexible ways to specify the coefficient matrix.

1. The coefficients themselves can now be arbitrary arithmetic expressions provided as strings. pySecDec now uses GiNaC [30] to parse these strings, so any syntax recognized by GiNaC is supported.

   The coefficient strings themselves are subsequently used in two ways: first during the integral library generation (i.e. inside `sum_package()`) pySecDec will try to determine the leading poles of the coefficients in the regulators, which is needed to determine the number of orders the integrals will need to be expanded to. Second, the strings will be saved to files as they are, and loaded back during the evaluation, at which point the symbolic variables will be substituted by the values provided by the user, and the resulting expressions will be expanded into a series in the regulators. This evaluation will be performed using arbitrary precision rational numbers so that no precision could be lost to numeric cancellations.

   This design was chosen to support expressions that are too big to be compiled to machine code or to be symbolically manipulated in non-trivial ways, such as coefficients arising after integration-by-parts reduction.

2. Each row of the coefficient matrix can be given either as a list of the same size as the number of integrals, or as a dictionary from integral

indices to coefficients. For example, `["a","0","b"]` and `{0:"a",2:"b"}` are now both valid ways to specify the same coefficient matrix row; the second way makes it easier to supply sparse matrices because zero coefficients can be omitted.

3. Each weighted sum can now be given a name. To this end, the coefficient matrix can be specified not as a list of rows, but rather as a dictionary from sum names (i.e. strings) to coefficient matrix rows. The supplied names are then used by Disteval in the integration log, and in its results, which can optionally be structured as dictionaries from the sum name to their values.

   The goal is making it easier to work with multiple sums at the same time.

3. Usage examples and comparison to the previous version

The examples described below can be found in the folder `examples/` of the pySecDec distribution. Unless stated otherwise, the default settings are used.

3.1. New and featured examples

We begin by describing the new examples introduced for the current release. These examples are primarily designed to demonstrate some of the new features. In Section 3.1.2 we demonstrate the flexible input syntax for amplitudes and in Section 3.1.3 we show how individual coefficients of the smallness parameter can be accessed when using expansion by regions. The remaining examples demonstrate the performance of the Disteval and IntLib integrators.

3.1.1. Simple jupyter notebook examples

The folder `examples/jupyter/` contains three examples in the form of a jupyter notebook where the whole workflow is demonstrated. These examples are

`easy.ipynb`: an easy function depending on two parameters;

`box.ipynb`: a one-loop box diagram with massive propagators;

`muon_production.ipynb`: the one-loop amplitude for $e^+e^- \to \mu^+\mu^-$ in massless QED.

Two of the examples are also available without jupyter format, in the folders `examples/easy/` and `examples/muon_production/`, respectively.

3.1.2. One-loop amplitude for $e^+e^- \to \mu^+\mu^-$

The example `muon_production` calculates the one-loop amplitude for muon production in electron-positron annihilation, $e^+e^- \to \mu^+\mu^-$, with massless leptons in QED. It evaluates a set of scalar master integrals and combines the results with the corresponding integral coefficients. The generation of the amplitude and the Passarino-Veltman reduction of the contributing integrals was done with FeynCalc [31]. This example is meant to highlight the improved handling of integral coefficients that increases the practicality of using pySecDec for full amplitude calculations.

The pySecDec result for the Born-virtual interference, proportional to α^3, where α is the QED fine structure constant, at $s = 3.0$, $t = -1.0$, $u = -2.0$ (subject to the physical constraint $s + t + u = 0$) reads[7]

$$\begin{aligned}\mathcal{A}^{(1)}\mathcal{A}^{(0)*} = &+ (-8.704559922781777(7) \cdot 10^4 + 7(5) \cdot 10^{-11}\, i) \cdot \varepsilon^{-2} \\ &+ (+6.1407633077(4) \cdot 10^4 - 2.73461815073(4) \cdot 10^5\, i) \cdot \varepsilon^{-1} \\ &+ (+3.45368951804(8) \cdot 10^5 + 3.98348633939(8) \cdot 10^5\, i) \\ &+ N_f \big[- 2.9015199742604458(3) \cdot 10^4 \cdot \varepsilon^{-1} \\ &\qquad + 3.574514829439898(2) \cdot 10^4 \\ &\qquad - 9.1153938353806605(8) \cdot 10^4\, i \big] + \mathcal{O}(\varepsilon)\,, \end{aligned} \tag{9}$$

where N_f is the number of lepton flavours. The result for the full amplitude has been validated with FeynCalc [31]. Since the building blocks of this reduced amplitude are only massless integrals, the integration time for one phase space point at the accuracy seen above is in the order of seconds.

3.1.3. Example from 2-loop muon decay with asymptotic expansion

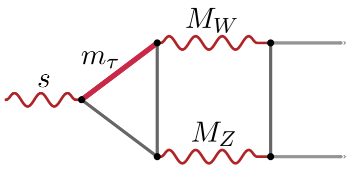


Figure 3: A 2-loop three point integral with three mass scales.

The example `muon_decay2L` demonstrates the possibility to produce Python output for each coefficient of an expansion in the smallness parameter within expansion by regions. The diagram in Figure 3 is expanded in the limit of

[7]Here and throughout the paper the numbers in the parentheses indicate the uncertainty of the final digits. For example, 1.2345(67) means 1.2345 ± 0.0067.

small τ-mass up to order 1, which generates terms with four different powers of m_τ^2: 0, 1, $1-\varepsilon$ and $1-2\varepsilon$. The result for this diagram reads

$$\begin{aligned}
&+(m_\tau^2)^0\big[+(-3.5410(2)+3.0610(3)\,i)\big]\\
&+(m_\tau^2)^1\big[+(-4.93694(1)\cdot 10^{-2}+2.237604(1)\cdot 10^{-1}\,i)\cdot\varepsilon^{-2}\\
&\qquad+(-5.0283(3)\cdot 10^{-1}-8.7873(3)\cdot 10^{-1}\,i)\cdot\varepsilon^{-1}\\
&\qquad+(+2.6476(2)-1.2090(2)\,i)\big]\\
&+(m_\tau^2)^{1-\varepsilon}\big[+(+9.873890(5)\cdot 10^{-2}-4.4752040(5)\cdot 10^{-1}\,i)\cdot\varepsilon^{-2}\\
&\qquad+(+2.14024(8)\cdot 10^{-1}+1.97848(7)\cdot 10^{-1}\,i)\cdot\varepsilon^{-1}\\
&\qquad+(-7.9370(4)\cdot 10^{-1}+4.6869(5)\cdot 10^{-1}\,i)\big]\\
&+(m_\tau^2)^{1-2\varepsilon}\big[+(-4.93694(1)\cdot 10^{-2}+2.237604(1)\cdot 10^{-1}\,i)\cdot\varepsilon^{-2}\\
&\qquad+(+2.8875(3)\cdot 10^{-1}+6.8082(4)\cdot 10^{-1}\,i)\cdot\varepsilon^{-1}\\
&\qquad+(+9.855(1)\cdot 10^{-1}+2.5875(2)\,i)\big].
\end{aligned} \tag{10}$$

To obtain the result in this form — mixing the symbolic prefactors of the form $(m_\tau^2)^k$ with numeric coefficients — one can generate the integration libraries as in Figure 4 and use them for integration as in Figure 5. The generation script here is similar to code example 2 in [1]. Note that the individual regions in Eq. (10) are divergent, however the sum is finite.

On line 4 of Figure 4, `LoopIntegralFromGraph()` is used to define a loop integral. On line 20 this integral is asymptotically expanded in the smallness parameter `mtsq` $\equiv m_\tau^2$ via the `loop_regions()` function up to order 1. Then, on line 28 the powers of m_τ^2 are extracted from the prefactors of the terms of the expansion, and each term has its prefactor modified to no longer include m_τ. On line 31 a mapping between each unique power of the smallness parameter and the corresponding modified terms is added to a dictionary. Note that several terms may be attributed to the same smallness parameter power. The final part of the generation script creates the integral libraries corresponding to each unique power of the smallness parameter via the `sum_package()` call on line 36. On line 42 the dictionary mapping powers of the smallness parameter to names of the corresponding integration libraries is saved in a JSON file; this file will later be used by the integration script.

The integration script of Figure 4 demonstrates how the DISTEVAL integrator can be called to produce a result of the form given in Eq. (10). On lines 10 and 11 respectively, each integration library is loaded and called with the kinematic variables $s = 3.0$, $M_W^2 = 0.78$, $M_Z^2 = 1.0$. Some commonly configured parameters are set explicitly in the library call: `epsrel` is the relative accuracy, `points` is the initial QMC lattice size, `format` is the output format of the result (`"sympy"`, `"mathematica"`, or `"json"`), `number_of_presamples` is the number of samples used for the initial contour deformation parameter selection, and `timeout` is the maximal allowed integration time in seconds.

```python
import pySecDec as psd, sympy as sp, json

if __name__ == "__main__":
    li = psd.LoopIntegralFromGraph(
        internal_lines = [['mt',[1,4]], ['mw',[4,2]], ['0',[2,3]],
                          ['0',[4,5]], ['0',[1,5]], ['mz',[5,3]]],
        external_lines = [['p1',1], ['p2',2], ['p3',3]],
        regulators = ['eps'],
        replacement_rules = [
            ('p1', '-p2-p3'),
            ('p2*p2', '0'),
            ('p3*p3', '0'),
            ('p2*p3', 's/2'),
            ('mw**2', 'mwsq'),
            ('mz**2', 'mzsq'),
            ('mt**2', 'mtsq')])

    terms = psd.loop_regions(name = 'muon_decay2L',
                             loop_integral = li,
                             smallness_parameter = 'mtsq',
                             decomposition_method = 'geometric',
                             expansion_by_regions_order = 1)

    term_by_prefactor_exponent = {}
    for term in terms:
        coefficient, exponent = sp.sympify(str(term.prefactor)).
            as_coeff_exponent(sp.sympify('mtsq'))
        term = term._replace(prefactor = coefficient)
        term_by_prefactor_exponent.setdefault(str(exponent), [])
        term_by_prefactor_exponent[str(exponent)].append(term)

    prefactor_exponent_by_name = {}
    for i, (exponent, term) in enumerate(sorted(
        term_by_prefactor_exponent.items())):
        prefactor_exponent_by_name[f'prefactor_{i+1}'] = exponent
        psd.sum_package(f'prefactor_{i+1}',
                        term,
                        regulators = ['eps'],
                        requested_orders = [0],
                        real_parameters = ['s', 'mwsq', 'mzsq'])

    with open('prefactor_exponent_by_name.json', 'w') as f:
        json.dump(prefactor_exponent_by_name, f)
```

Figure 4: Generation script for the two-loop muon decay example.

```python
from pySecDec.integral_interface import DistevalLibrary
import json
import sympy as sp

with open('prefactor_exponent_by_name.json') as f:
    prefactor_exponent_by_name = json.load(f)

result_by_prefactor_exponent = {}
for name, exponent in prefactor_exponent_by_name.items():
    loop_integral = DistevalLibrary(f'{name}/disteval/{name}.json')
    result_by_prefactor_exponent[exponent] = loop_integral(
            parameters = {'s': 3, 'mwsq': 0.78, 'mzsq': 1.0},
            epsrel = 1e-4, points = 1e4, format = 'sympy',
            number_of_presamples = 1e4, timeout = None)

print('Result:')
for exponent, str_result in result_by_prefactor_exponent.items():
    result = sp.sympify(str_result)
    val = result[0].subs({"plusminus": 0})
    err = result[0].coeff("plusminus")
    print(f'''\
  +mtsq^({exponent})*(
    +1/eps^2*(({val.coeff("eps",-2)}) +/- ({err.coeff("eps",-2)}))
    +1/eps^1*(({val.coeff("eps",-1)}) +/- ({err.coeff("eps",-1)}))
    +  eps^0*(({val.coeff("eps",0)}) +/- ({err.coeff("eps",0)}))
  )''')
```

Figure 5: Integration script for the two-loop muon decay example.

The full list of parameters is available in the pySecDec documentation on the `DistevalLibrary` class. The integration script keeps track of which integration library corresponds to which smallness parameter power via the dictionary previously created by the generation script.

3.1.4. 2-loop 5-point hexatriangle example with several mass scales

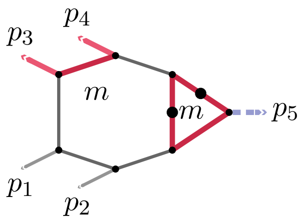


Figure 6: A 2-loop 5-point integral with massive propagators and massive legs. The integral is evaluated in $6-2\varepsilon$ space-time dimensions. The configuration being tested is $p_1^2 = p_2^2 = 0$, $p_3^2 = p_4^2 = m^2 = 1$, $p_5^2 = 12/23$, $(p_1+p_2)^2 = 262/35$, $(p_2+p_3)^2 = -92/53$, $(p_3+p_5)^2 = 491/164$, $(p_5+p_4)^2 = 373/124$, $(p_4+p_1)^2 = -65/36$.

The example `hexatriangle` is a 2-loop 5-point integral depicted in Figure 6. This is a master integral for the amplitude of $q\bar{q} \to t\bar{t}H$ production at two loops. The integral is dimensionally shifted to $6-2\varepsilon$ space-time dimensions; the dimensional shift and additional dots were chosen to make it finite in ε and fast to evaluate.

The value of the integral at the point specified in Figure 6 is

$$1.454919812(7) \cdot 10^{-7} - 1.069797219(8) \cdot 10^{-7}\, i + \mathcal{O}(\varepsilon). \tag{11}$$

The convergence rate of the integral is depicted in Figure 7. Overall the obtained precision scales with the integration time t approximately as $1/t^{1.6}$. We want to emphasise that such scaling is made possible by the use of the QMC integration methods; traditional Monte Carlo methods only scale as fast as $1/t^{0.5}$.

A more detailed list of integration timings is given in Table 2.

3.1.5. 2-loop 5-point offshell pentabox example

The example `pentabox_offshell` is an integral depicted in Figure 8. It is a 2-loop pentabox with an internal mass, massive legs, and the total of 7 scales. The integral is evaluated in $6-2\varepsilon$ space-time dimensions (where it is finite in ε) up to $\mathcal{O}(\varepsilon^4)$; a prefactor of $\Gamma(2+2\varepsilon)$ is divided out to match the configuration of Section 6.4 of [14], where the same integral is calculated numerically via tropical integration.

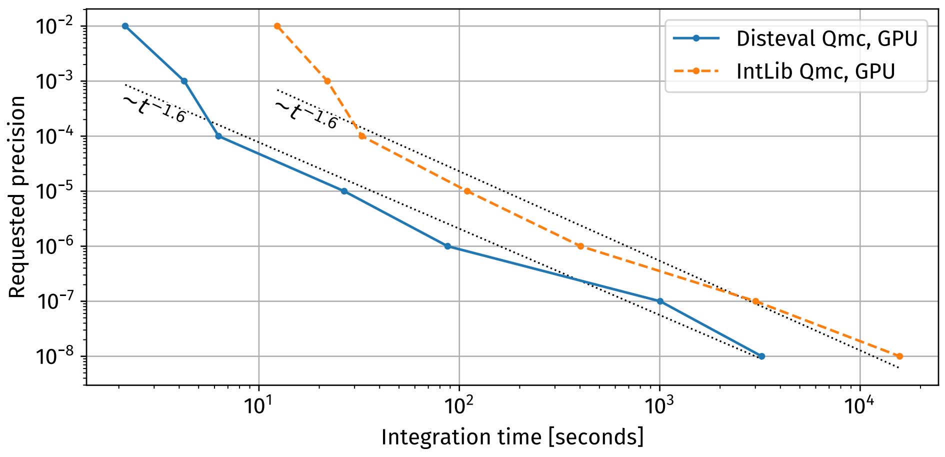


Figure 7: The obtained precision by integration time for the `hexatriangle` example. This plot is based on the data from Table 2.

	Integrator \ Accuracy	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
GPU	DISTEVAL	2.2 s	4.2 s	6.3 s	27 s	1.5 m	17 m	54 m
	INTLIB	12.3 s	22.0 s	32.6 s	110 s	6.7 m	50 m	263 m
	Ratio	5.6	5.2	5.2	4.1	5.6	3.0	4.9
CPU	DISTEVAL	3.5 s	5.1 s	14 s	1.6 m	8.3 m	57 m	4.7 h
	INTLIB	8.5 s	20.8 s	86 s	14.2 m	62.2 m	480 m	43.1 h
	Ratio	2.4	4.1	6.1	8.7	7.5	8.4	9.2

Table 2: Integration timings for the `hexatriangle` example (Figure 6) depending on the requested accuracy using two integrators: DISTEVAL and INTLIB `Qmc`. The GPU timings were taken using an NVidia A100 80G, with the integrands compiled using CUDA 11.8.89. The CPU timings are for an AMD EPYC 7F32 processor with 16 cores (32 threads), with the integrands compiled using GCC 12.2.1 with `CXXFLAGS` set to `-O3 -mavx2 -mfma`.

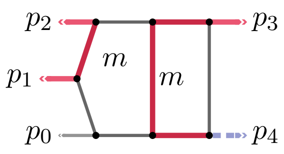


Figure 8: A 2-loop 5-point pentabox integral with massive propagators and massive legs. The configuration being tested is $p_0^2 = 0$, $p_1^2 = p_2^2 = p_3^2 = m^2 = 1/2$, $(p_0 + p_1)^2 = 2.2$, $(p_0 + p_2)^2 = 2.3$, $(p_0 + p_3)^2 = 2.4$, $(p_1 + p_2)^2 = 2.5$, $(p_1 + p_3)^2 = 2.6$, $(p_2 + p_3)^2 = 2.7$.

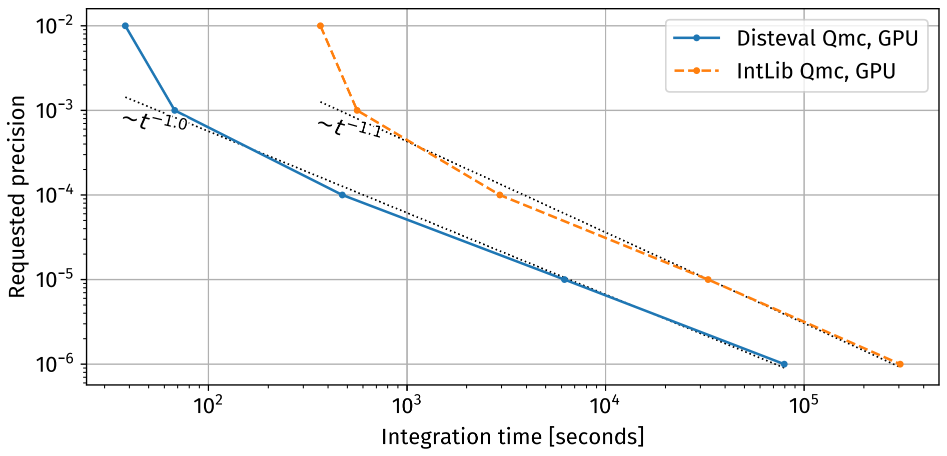


Figure 9: The obtained precision by integration time for the `pentabox_offshell` example. This plot is based on the data from Table 3.

The value of the integral at the point specified in Figure 8 is

$$\begin{aligned}
&+ (+6.443869(7) \cdot 10^{-2} - 8.267759(7) \cdot 10^{-2}\, i)\, \varepsilon^0 \\
&+ (+4.043397(2) \cdot 10^{-1} + 3.189607(2) \cdot 10^{-1}\, i)\, \varepsilon^1 \\
&+ (-7.771389(2) \cdot 10^{-1} + 9.370171(2) \cdot 10^{-1}\, i)\, \varepsilon^2 \\
&+ (-1.3220709(6) \cdot 10^{0} - 1.2139678(6) \cdot 10^{0}\, i)\, \varepsilon^3 \\
&+ (+1.3789155(10) \cdot 10^{0} - 1.2118956(10) \cdot 10^{0}\, i)\, \varepsilon^4 \\
&+ \mathcal{O}(\varepsilon^5)
\end{aligned} \tag{12}$$

These values match the ones given in [14] within the uncertainty limits.

The convergence rate of the integral is depicted in Figure 9. Overall the obtained precision scales with the integration time t approximately as $1/t$.

A more detailed list of integration timings is given in Table 3.

3.1.6. 4-loop triangle diagram

The example `gminus2_4L` is a four-loop diagram contributing to the electron or muon anomalous magnetic moment. The diagram is depicted in Figure 10.

	Integrator \ Accuracy	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
GPU	DISTEVAL	38 s	1.1 m	7.9 m	1.7 h	22 h
	INTLIB	366 s	9.3 m	48.9 m	9.1 h	85 h
	Ratio	9.6	8.3	6.2	5.3	3.8
CPU	DISTEVAL	13 s	2.4 m	43 m	7.9 h	—
	INTLIB	67 s	18.9 m	299 m	65.0 h	—
	Ratio	5.0	7.8	7.0	8.2	—

Table 3: Integration timings for the `pentabox_offshell` example (Figure 8) depending on the requested accuracy using two integrators: DISTEVAL and INTLIB `Qmc`. Same benchmarking conditions as in Table 2.

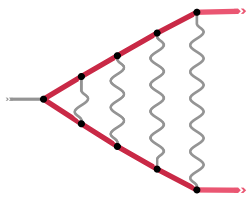


Figure 10: A 4-loop diagram with kinematics inspired by contributions to the electron or muon anomalous magnetic moment.

The massive lines (coloured in red) denote on-shell massive fermion lines, $p^2 = m^2$. For the grey external line with momentum q, the limit $q \to 0$ needs to be taken, such that the diagram is characterised by $q^2 = 0, q \cdot p = 0, p^2 = m^2$. Therefore the corresponding integral becomes a single scale integral, depending only on m^2.

The pySECDEC result for `gminus2_4L` reads

$$\begin{aligned}
&+ 2.60420(2) \cdot 10^{-3} \cdot \varepsilon^{-4} \\
&+ 2.5237(2) \cdot 10^{-2} \cdot \varepsilon^{-3} \\
&+ 3.8721(4) \cdot 10^{-1} \cdot \varepsilon^{-2} \\
&+ 3.9116(4) \cdot \varepsilon^{-1} \\
&+ 39.256(4) + \mathcal{O}(\varepsilon).
\end{aligned} \tag{13}$$

3.1.7. 6-loop two-point function

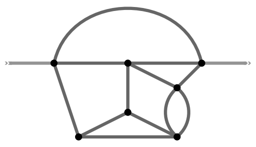


Figure 11: A 6-loop two-point integral.

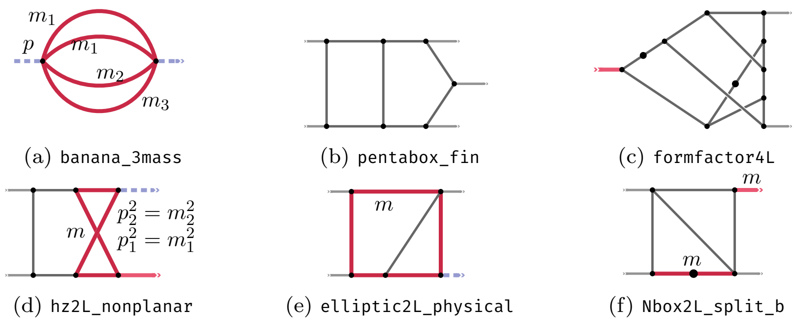


Figure 12: All diagrams of Table 4 except for `bubble6L`, which is described in detail in Section 3.1.7.

The `bubble6L` example consists of the 6-loop 2-point integral shown in Figure 11. The pole coefficients are given analytically in Eq. (A3) of Ref. [32] (at $p^2 = -p_E^2 = -1$, where p_E is the external momentum in Euclidean space). The decomposition method `'geometric'` is the default and recommended decomposition method in version 1.6. In this example it is mandatory to use a geometric decomposition because `'iterative'` leads to an infinite recursion. Usually the geometric decomposition method produces the fewest sectors. However, for graphs with very high symmetry, the iterative method can occasionally produce fewer sectors than the geometric method as it does not destroy symmetries when one of the Feynman parameters is eliminated using the δ-constraint. More information about the various decomposition methods can be found in Refs. [3, 4, 33] and in the code documentation.

The analytic result is given by

$$\begin{aligned} B_{6L}^{\text{analyt.}} &= \frac{1}{\varepsilon^2}\frac{147}{16}\zeta_7 - \frac{1}{\varepsilon}\left(\frac{147}{16}\zeta_7 + \frac{27}{2}\zeta_3\zeta_5 + \frac{27}{10}\zeta_{3,5} - \frac{2063}{504000}\pi^8\right) + \mathcal{O}(\varepsilon^0) \\ &= \frac{9.264208985946416}{\varepsilon^2} + \frac{91.73175282208716}{\varepsilon} + \mathcal{O}(\varepsilon^0)\ . \end{aligned} \tag{14}$$

The pySecDec result at $p^2 = -1$ obtained with the Disteval integrator reads

$$\begin{aligned} B_{6L}^{\text{num.}} = &+ 9.26420902(3)\cdot\varepsilon^{-2} \\ &+ 9.17317528(8)\cdot 10^1\cdot\varepsilon^{-1} \\ &+ 1.11860698(1)\cdot 10^3 + \mathcal{O}(\varepsilon)\ . \end{aligned} \tag{15}$$

3.2. Previously existing examples

Several previously existing pySecDec examples, shown in Figure 12, have been benchmarked in [5]. In Table 4 and Figure 13 we provide a comparison of the integration time of those examples using the Disteval integrator

	Integrator \ Accuracy	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
`banana_3mass`	Disteval	2.1 s	2.1 s	2.4 s	2.6 s	2.6 s	2.9 s	3.6 s
	IntLib	5.0 s	4.9 s	6.4 s	7.2 s	8.5 s	8.5 s	13.8 s
	Ratio	2.3	2.3	2.7	2.7	3.2	3.0	3.9
`bubble6L`	Disteval	1.8 m	1.8 m	1.8 m	2.1 m	3.8 m	10.2 m	1.2 h
	IntLib	39.5 m	38.8 m	39.6 m	43.8 m	85.1 m	170.7 m	11.6 h
	Ratio	22	22	22	21	22	17	10
`formfactor4L`	Disteval	4.1 m	4.1 m	4.1 m	4.4 m	7.7 m	14.6 m	0.96 h
	IntLib	74 m	73 m	73 m	74 m	136 m	246 m	10.9 h
	Ratio	18	18	18	17	18	17	11
`elliptic2L_physical`	Disteval	1.6 s	1.5 s	1.7 s	1.9 s	4.0 s	19 s	7.6 m
	IntLib	3.1 s	4.8 s	4.9 s	7.3 s	13.8 s	53 s	4.3 m
	Ratio	1.9	3.1	2.8	3.9	3.4	2.9	0.6
`hz2L_nonplanar`	Disteval	2.1 s	2.6 s	4.6 s	30.4 s	2.2 m	5.1 m	27.1 m
	IntLib	9 s	17 s	41 s	163 s	9.6 m	16.0 m	27.3 m
	Ratio	1.8	3.4	4.6	4.4	4.2	3.0	1.0
`Nbox2L_split_b`	Disteval	2.7 s	9.8 s	16.8 s	0.58 m	2.4 m	9.1 m	20 m
	IntLib	24 s	73 s	223 s	6.6 m	26 m	43 m	93 m
	Ratio	3.0	4.6	9.7	9.9	10.5	4.8	4.7
`pentabox_fin`	Disteval	5 s	8 s	11 s	0.71 m	3.7 m	18.5 m	1.1 h
	IntLib	45 s	65 s	88 s	3.2 m	11.3 m	74.8 m	4.6 h
	Ratio	8.6	7.9	7.7	4.5	3.1	4.0	4.2

Table 4: Integration timings on a GPU for different examples using the IntLib `Qmc` integrator and the new Disteval integrator. All timings are using the CBC generating vectors from the previous release, meaning the ratios between IntLib and Disteval are purely due to the improvements described in Section 2.1. The significantly improved timings achieved by using the new median generating vectors are shown in Table 1.

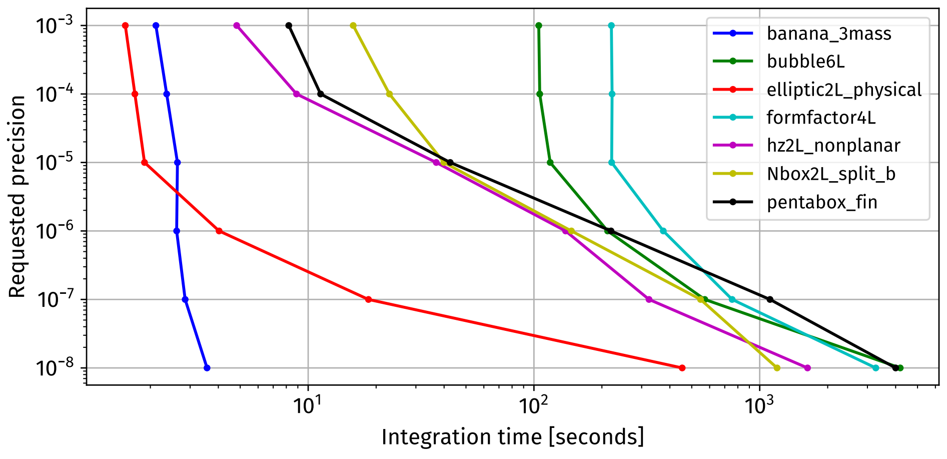


Figure 13: Convergence rates of the Disteval timings from Table 4

(new in v1.6) and the IntLib `Qmc` integrator (the default of v1.5.6), all on an NVidia A100 80G GPU (using Cuda version 11.8).

The reported integration times correspond to the wall clock times for running the integration via the Python interface of pySecDec. In particular, the numerical integration of *all* orders in ε up to the finite order is included in the timings. The precision refers to the relative error which in this case is defined as $\epsilon_{\text{rel}} = \sqrt{\frac{(\Delta R)^2+(\Delta I)^2}{R^2+I^2}}$, R and I are the real and imaginary parts of a coefficient in the ε-expansion, and ΔR and ΔI are the corresponding uncertainties. The examples `formfactor4L` and `bubble6L` have been calculated using the `baker` integral transformation, for the other examples the default transformation `korobov3` has been used.

The overall conclusion is that Disteval is 3×-5× faster than IntLib `Qmc` with equivalent settings on a GPU at higher accuracies, with the exception of the Euclidean integrals `bubble6L` and `formfactor4L`. They contain a large number of sectors, each very simple, so that the execution time is mostly dominated by overhead. Disteval has up to 20× less overhead.

Of particular note is the benchmark of the `elliptic2L_physical` example: at the requested precision of 10^{-8}, the speedup of Disteval is 0.6, so it is slower than IntLib `Qmc`. The reason for this is exactly the unlucky lattice at $n = 4.3 \cdot 10^9$ depicted in Figure 1: Disteval reaches it first at this requested precision, while IntLib does not hit this particular lattice because its algorithm of selecting n_i differs just slightly enough to land on a nearby lattice instead. In any case, this problem is circumvented by the use of median QMC rules, and we have investigated the `elliptic2L_physical` example in detail in Section 2.2.

4. Conclusions

We have presented version 1.6 of pySecDec, featuring a major upgrade targeted at the evaluation of loop amplitudes through a novel, highly distributed Quasi-Monte-Carlo (QMC) evaluation method. Compared to the previous version, the virtues of the new method applied to individual multi-loop integrals are particularly manifest for multi-scale integrals and when high precision is requested. Very importantly, the calculation of *amplitudes* rather than individual integrals is facilitated. This is achieved through several improvements, for instance, new functionalities to treat the coefficients of master integrals, which are typically large expressions after IBP reduction. Furthermore, amplitudes are calculated as weighted sums of integrals with coefficients, with an overall precision goal that can be specified by the user. A new integrator based on median QMC rules avoids the limitations of the component-by-component construction of generating vectors for lattice rules. It also remedies the intermediate loss of QMC-typical scaling that has been observed for some fixed individual lattices.

The release contains improvements to the expansion by regions functionality, including the treatment of integrals with numerators within expansion by regions and the automated detection of whether and where additional regulators are needed, making this information completely transparent to the user. The coefficients of each order of the expansion in the small parameter are now also easily accessible to the user.

With these new features pySecDec is significantly faster, more flexible, and easier to use than previous versions. It is better equipped to analyse and tackle a wide range of problems including previously intractable multi-loop amplitudes needed for precision phenomenology, problems requiring multiple dimensional regulators, and integrals/amplitudes where higher numerical precision than previously possible is required.

Acknowledgements

We would like to thank Goutam Das, Joshua Davies, Christoph Greub, Andrey Pikelner, Vladyslav Shtabovenko and Yannick Ulrich for discussions and raising issues that helped to improve the program. This research was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 - TRR 257. SJ is supported by a Royal Society University Research Fellowship (Grant URF/R1/201268).

References

[1] G. Heinrich, S. Jahn, S. P. Jones, M. Kerner, F. Langer, V. Magerya et al., *Expansion by regions with pySecDec*, Comput. Phys. Commun. **273** (2022) 108267, [2108.10807].

[2] S. Borowka, J. Carter and G. Heinrich, *Numerical Evaluation of Multi-Loop Integrals for Arbitrary Kinematics with SecDec 2.0*, Comput. Phys. Commun. **184** (2013) 396–408, [1204.4152].

[3] S. Borowka, G. Heinrich, S. P. Jones, M. Kerner, J. Schlenk and T. Zirke, *SecDec-3.0: numerical evaluation of multi-scale integrals beyond one loop*, Comput. Phys. Commun. **196** (2015) 470–491, [1502.06595].

[4] S. Borowka, G. Heinrich, S. Jahn, S. P. Jones, M. Kerner, J. Schlenk et al., *pySecDec: a toolbox for the numerical evaluation of multi-scale integrals*, Comput. Phys. Commun. **222** (2018) 313–326, [1703.09692].

[5] S. Borowka, G. Heinrich, S. Jahn, S. P. Jones, M. Kerner and J. Schlenk, *A GPU compatible quasi-Monte Carlo integrator interfaced to pySecDec*, *Comput. Phys. Commun.* **240** (2019) 120–137, [1811.11720], https://github.com/mppmu/qmc.

[6] T. Binoth and G. Heinrich, *An automatized algorithm to compute infrared divergent multiloop integrals*, *Nucl. Phys. B* **585** (2000) 741–759, [hep-ph/0004013].

[7] G. Heinrich, *Sector Decomposition*, *Int. J. Mod. Phys. A* **23** (2008) 1457–1486, [0803.4177].

[8] C. Bogner and S. Weinzierl, *Resolution of singularities for multi-loop integrals*, *Comput. Phys. Commun.* **178** (2008) 596–610, [0709.4092].

[9] A. V. Smirnov and M. N. Tentyukov, *Feynman Integral Evaluation by a Sector decomposiTion Approach (FIESTA)*, *Comput. Phys. Commun.* **180** (2009) 735–746, [0807.4129].

[10] A. V. Smirnov, V. A. Smirnov and M. Tentyukov, *FIESTA 2: Parallelizeable multiloop numerical calculations*, *Comput. Phys. Commun.* **182** (2011) 790–803, [0912.0158].

[11] A. V. Smirnov, *FIESTA 3: cluster-parallelizable multiloop numerical calculations in physical regions*, *Comput. Phys. Commun.* **185** (2014) 2090–2100, [1312.3186].

[12] A. V. Smirnov, *FIESTA4: Optimized Feynman integral calculations with GPU support*, *Comput. Phys. Commun.* **204** (2016) 189–199, [1511.03614].

[13] A. V. Smirnov, N. D. Shapurov and L. I. Vysotsky, *FIESTA5: Numerical high-performance Feynman integral evaluation*, *Comput. Phys. Commun.* **277** (2022) 108386, [2110.11660].

[14] M. Borinsky, H. J. Munch and F. Tellander, *Tropical Feynman integration in the Minkowski regime*, 2302.08955.

[15] M. Borinsky, *Tropical Monte Carlo quadrature for Feynman integrals*, 2008.12310.

[16] M. Hidding, *DiffExp, a Mathematica package for computing Feynman integrals in terms of one-dimensional series expansions*, *Comput. Phys. Commun.* **269** (2021) 108125, [2006.05510].

[17] F. Moriello, *Generalised power series expansions for the elliptic planar families of Higgs + jet production at two loops*, *JHEP* **01** (2020) 150, [1907.13234].

[18] X. Liu and Y.-Q. Ma, *AMFlow: A Mathematica package for Feynman integrals computation via auxiliary mass flow*, Comput. Phys. Commun. **283** (2023) 108565, [2201.11669].

[19] T. Armadillo, R. Bonciani, S. Devoto, N. Rana and A. Vicini, *Evaluation of Feynman integrals with arbitrary complex masses via series expansions*, Comput. Phys. Commun. **282** (2023) 108545, [2205.03345].

[20] V. Smirnov, *Renormalization and asymptotic expansions*, vol. 14. Birkhäuser, 1991.

[21] M. Beneke and V. A. Smirnov, *Asymptotic expansion of Feynman integrals near threshold*, Nucl. Phys. B **522** (1998) 321–344, [hep-ph/9711391].

[22] A. Pak and A. Smirnov, *Geometric approach to asymptotic expansion of Feynman integrals*, Eur. Phys. J. C **71** (2011) 1626, [1011.4863].

[23] B. Jantzen, *Foundation and generalization of the expansion by regions*, JHEP **12** (2011) 076, [1111.2589].

[24] T. Goda and P. L'Ecuyer, *Construction-free median quasi-monte carlo rules for function spaces with unspecified smoothness and general weights*, SIAM Journal on Scientific Computing **44** (2022) A2765–A2788, [2201.09413].

[25] T. Hahn, *Concurrent Cuba*, Comput. Phys. Commun. **207** (2016) 341–349.

[26] M. Galassi et al., *GNU Scientific Library Reference Manual (3rd Ed.)*. Network Theory Ltd, Bristol, England, 2009, http://www.gnu.org/software/gsl/.

[27] J. Dick, F. Y. Kuo and I. H. Sloan, *High-dimensional integration: The quasi-monte carlo way*, *Acta Numerica* **22** (2013) 133–288.

[28] Z. Li, J. Wang, Q.-S. Yan and X. Zhao, *Efficient Numerical Evaluation of Feynman Integral*, Chinese Physics C **40, No. 3** (2016) 033103, [1508.02512].

[29] D. Nuyens and R. Cools, *Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel hilbert spaces*, *Mathematics of Computation* **75** (2006) 903–920, http://www.jstor.org/stable/4100318.

[30] C. W. Bauer, A. Frink and R. Kreckel, *Introduction to the* GiNaC *framework for symbolic computation within the C++ programming language*, J. Symb. Comput. **33** (2002) 1–12, [cs/0004015].

[31] V. Shtabovenko, R. Mertig and F. Orellana, *FeynCalc 9.3: New features and improvements*, *Comput. Phys. Commun.* **256** (2020) 107478, [2001.04407].

[32] M. V. Kompaniets and E. Panzer, *Minimally subtracted six loop renormalization of $O(n)$-symmetric ϕ^4 theory and critical exponents*, *Phys. Rev. D* **96** (2017) 036016, [1705.06483].

[33] J. Schlenk and T. Zirke, *Calculation of Multi-Loop Integrals with SecDec-3.0*, *PoS* **RADCOR2015** (2016) 106, [1601.03982].