

PAPER • OPEN ACCESS

Loop integral evaluation and asymptotic expansion with pySecDec

To cite this article: Vitaly Magerya 2023 *J. Phys.: Conf. Ser.* **2438** 012143

View the [article online](#) for updates and enhancements.

You may also like

- [Uncertainty quantification by ensemble learning for computational optical form measurements](#)
Lara Hoffmann, Ines Fortmeier and Clemens Elster
- [Glacial isostatic adjustment: physical models and observational constraints](#)
W Richard Peltier, Patrick Pak-Cheuk Wu, Donald F Argus et al.
- [A Clear View of a Cloudy Brown Dwarf Companion from High-resolution Spectroscopy](#)
Jerry W. Xuan, Jason Wang, Jean-Baptiste Ruffio et al.



245th ECS Meeting
San Francisco, CA
May 26–30, 2024

PRiME 2024
Honolulu, Hawaii
October 6–11, 2024

Bringing together industry, researchers, and government across 50 symposia in electrochemistry and solid state science and technology

Learn more about ECS Meetings at
<http://www.electrochem.org/upcoming-meetings>

 Save the Dates for future ECS Meetings!

Loop integral evaluation and asymptotic expansion with pySecDec

Vitaly Magerya

Institute for Theoretical Physics, Karlsruhe Institute of Technology,
Wolfgang-Gaede-Str. 1, Geb. 30.23, 76131 Karlsruhe

E-mail: vitalii.maheria@kit.edu

Abstract. The evaluation of higher-loop Feynman integrals is at the core of the quest to reduce the uncertainty of theoretical predictions and match experimental data from the LHC and future colliders. pySECDEC is a program to evaluate such integrals numerically based on the sector decomposition approach; its new release version 1.5 introduces features significantly improving its performance: automatic adaptive evaluation of weighted sums of integrals (e.g. amplitudes) and asymptotic expansion in kinematic ratios. Here we briefly review both, illustrating the expected performance benefits.

1. Introduction

The increasing amount of data from the Large Hadron Collider (LHC) allows the experimental groups to push the precision boundaries of the knowledge of scattering cross-sections, and poses an ongoing challenge to the theoreticians to make the theoretical predictions at least as precise as the experimental ones. Already now at least 2-loop QCD corrections are needed for this, and future colliders will require 3-loop QCD and mixed QCD-electroweak corrections [1].

The two major bottlenecks for these corrections are phase-space integration and loop integration of the amplitudes. While at 1-loop level the required loop integrals are all known analytically, and the phase-space integration can be done using e.g. the well known Passarino-Veltman reduction [2], no such general solutions exist starting at the 2-loop level: the existing phase-space integration methods are computationally challenging and process-dependent, only a subset of the 2-loop Feynman integrals of interest are known analytically (mostly in the massless cases), and the numerical evaluation methods have long-standing problems with both performance and precision.

pySECDEC [3, 4, 5] comes into this picture as a tool for numerical evaluation of loop integrals, implementing the sector decomposition approach [6, 7] (with FIESTA [8] being the other major implementation of the same method). It has a long history, and has started as the Mathematica code SECDEC [9, 10, 11]. The current iteration is written in Python and C++, and is available from GitHub¹.

Recently version 1.5 of pySECDEC was released bringing performance improvements through adaptive sampling of weighted sums of integrals (i.e. amplitudes) and an implementation of the expansion-by-regions method of the asymptotic expansion of integrals in kinematic invariants to help with e.g. high-energy regions where the numerical integration performance is known to be

¹ <https://github.com/gudrunhe/secdec>




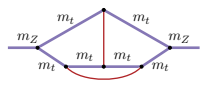

| Diagram \ Relative precision | | 10^{-3} | 10^{-4} | 10^{-5} | 10^{-6} | 10^{-7} | 10^{-8} |
|---|-----|-----------|-----------|-----------|-----------|-----------|-----------|
|  | GPU | 15s | 20s | 40s | 200s | 13m | 50m |
| | CPU | 10s | 50s | 400s | 4000s | 180m | 1200m |
|  | GPU | 18s | 19s | 30s | 20s | 1.2m | 2m |
| | CPU | 5s | 14s | 60s | 50s | 12m | 16m |
|  | GPU | 6s | 11s | 12s | 30s | 3m | 24m |
| | CPU | 5s | 10s | 50s | 800s | 60m | 800m |

Table 1. pySECDEC 1.5 integration time with Quasi Monte-Carlo (QMC) integrator for a selection of massive 3-loop electroweak self-energy integrals taken from [13]. The CPU is AMD Epyc 7302 with 32 threads; the GPU is NVidia A100.

a problem. A more detailed description can be found in [5]; here we would like to report on the current status of pySECDEC performance when applied to practical higher-loop calculations.

2. The expected performance

pySECDEC comes with multiple configurable integrators. Possibly the most well-known one is the VEGAS integrator (provided by CUBA [12]); the recommended one however is the Quasi Monte-Carlo integrator (QMC) described in [4]. The major advantage QMC has over the classical Monte-Carlo techniques, even the advanced ones such as the VEGAS integrator, is that its precision scales as $1/N$ or better, with N being the number of integrand evaluations (“samples”), while the classical approaches scale as $1/\sqrt{N}$. In other words, to get 10x precision with QMC one needs to wait at most 10x as long, while with classical Monte-Carlo one would have to wait 100x as long.

To illustrate the scaling and the generally expected performance, let us turn to Table 1 where integration times of several 3-loop massive integrals are displayed. In short: it takes seconds to minutes per integral to achieve sufficient precision for practical purposes. Note that the precision scaling for the first and the second integrals is even better than $1/N$: it is closer to $1/N^{1.5}$ and $1/N^3$ respectively, i.e. it takes less than 10x time to achieve 10x precision.

Naturally, the performance depends on the employed computer hardware. See Figure 1 for a rough comparison between different CPU and GPU models. The takeaway is that a top consumer-grade GPU brings the performance of a single server-grade CPU, while a top server-grade GPU is 10x faster. For this reason we advocate the use of server-grade GPUs for complicated integration problems. Note that the factor limiting pySECDEC performance on GPUs is the use of double-precision floating point variables—something that is not needed by the majority of GPU users, and is not optimized for by GPU manufacturers outside of specific server-grade GPU models like NVidia A100 or AMD MI200.²

2.1. On the importance of a good integral selection

While the integration times shown in Table 1 are representative of many 7- and 8-line integrals, if one needs to evaluate a full amplitude, many more integrals would need to be calculated, and in our experience a fraction of them will converge much slower. For this reason it is important to remove linearly dependent integrals by using reduction via the integration-by-parts (IBP) relations, and it is important to apply an effort to the selection of the master integral basis:

² Note that pySECDEC relies on NVidia’s CUDA libraries and therefore does not currently work on AMD GPUs.

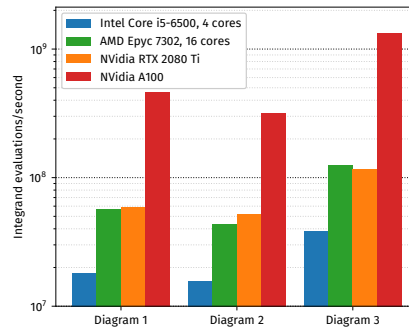


Figure 1. The number of integrand evaluations per second of the integrands of the diagrams from Table 1 on a desktop-grade CPU (Intel Core i5-6500), a server-grade CPU (AMD Epyc 7302), a consumer-grade GPU (NVidia RTX 2080 Ti), and a server-grade GPU (NVidia A100).

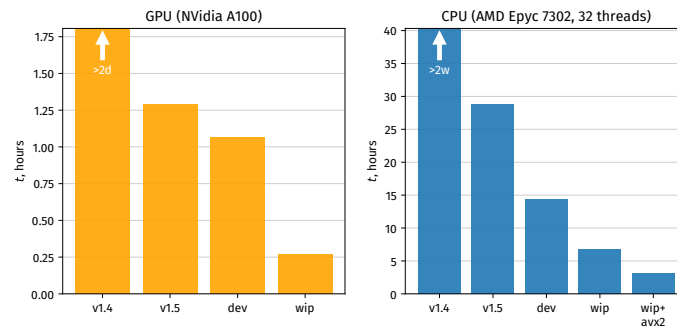


Figure 2. Integration time of the first diagram from Table 1 to 7 digits of precision by pySECDEC version. Here `dev` stands for the current public development version (v1.5.3+), `wip` stands for the work-in-progress code to be released in the future, and `avx2` stands for compilation with AVX2 and FMA processor instruction sets allowed (the work-in-progress code has special provisions to use them).

choosing a good basis can be the difference between having the answer in minutes and not having it at all.

While we do not have a general recipe to predict which integrals will converge poorly, but the rule of thumb is that one should choose integrals with a positive power of the U polynomial (in the Feynman parametrization), and the power of F should not be lower than -2 . To achieve this one would sometimes need to raise the powers of the propagators, and other times to perform dimensional shifts (see e.g. [14]). This does of course shift the complexity from the numerical evaluation to the IBP reduction, but often this is an acceptable tradeoff.

3. Adaptive sampling of amplitudes

Despite the best efforts to select a good basis of integrals, complicated integrals can always remain. For those, pySECDEC v1.5 brings a major performance improvement, and subsequent work strives to incrementally improve on that: see Figure 2 for a rough performance progression of pySECDEC versions.

The biggest contributor of the improvement between v1.4 and v1.5 is adaptive sampling of weighted sums of integrals. To see how it works, suppose one wants to evaluate

$$1 \text{ --- } \text{circle} \text{ ---} + 10 \text{ --- } \text{diamond} \text{ ---} + 50 \text{ --- } \text{two circles} \text{ ---} \quad (1)$$




| Amplitude term | Naive sampling | Naive error | Better sampling | Better error |
|--|----------------|--------------------|----------------------------------|--------------------|
| 1  | 10^6 samples | $1 \cdot 10^{-6}$ | $\frac{1}{2} \cdot 10^6$ samples | $2 \cdot 10^{-6}$ |
| 10  | 10^6 samples | $10 \cdot 10^{-6}$ | $\frac{1}{2} \cdot 10^6$ samples | $20 \cdot 10^{-6}$ |
| 50  | 10^6 samples | $50 \cdot 10^{-6}$ | $2 \cdot 10^6$ samples | $25 \cdot 10^{-6}$ |
| Total: | $3 \cdot 10^6$ | $51 \cdot 10^{-6}$ | $3 \cdot 10^6$ | $32 \cdot 10^{-6}$ |

Table 2. Integration uncertainty of a sum of integrals for two different distributions of the total number of integrand evaluations. For illustration purposes this example assumes that the integration error is exactly $1/N_{\text{samples}}$.

to some fixed precision. The naive approach would be to evaluate all three integrals to this precision and then to add them. However, because the last one has the largest coefficient and contributes to the overall error the most, it is best to spend more time evaluating this integral, and spend less time evaluating the others; this way more precision can be achieved using the same integration time. See [Table 2](#) for a worked out example.

In practice pySECDEC will try to determine the optimal sampling distribution not only based on the coefficients of integrals in a sum, but also on how well they converge and how fast can their integrand be evaluated—all determined automatically during the integration. Note that this optimization applies to both user-specified sums of integrals (e.g. full amplitudes) and to single integrals too, because under the sector decomposition method each integral is split into a sum of sectors, each sector being a separate integral. This is why [Figure 2](#) shows a big improvement at v1.5 even for a single integral.

We would additionally like to note that this technology is not new, and pySECDEC equipped with it has previously been successfully used in multiple 2-loop calculation such as [[15](#), [16](#), [17](#)].

4. Asymptotic expansion

Aside from some integrals being intrinsically complicated, another source of performance problems are kinematic limits: if an integral depends on kinematic invariants, the more extreme their ratios get the slower such an integral converges. The reason is that in this case the majority of the integral’s value becomes progressively concentrated in a smaller region of the integration space, and more evaluations are needed to sample this small space precisely.

As an illustration see [Figure 3](#): the integration time needed to reach a given precision increases with the increase of the m^2/s ratio, eventually making it impractical to obtain a result.

One solution to this problem is to note that since the kinematic ratio is large, it is possible to expand the integral in the inverse of this ratio, treating it as a smallness parameter, e.g.

$$\langle \text{triangle} \rangle^s = (\dots + \dots) \left(\frac{s}{m^2}\right)^{-1} + (\dots + \dots + \dots + \dots) \left(\frac{s}{m^2}\right)^0 + \mathcal{O}\left(\frac{s}{m^2}\right). \quad (2)$$

The method for such expansions, “expansion-by-regions”, has been worked out in [[18](#), [19](#)]. It consists of splitting the whole integration region into subregions such that the integration variables in each are of the specific order in the smallness parameter, making straightforward Taylor expansion possible. Then, each of the obtained integrals can be integrated over the whole integration region (not just its specific subregion), with the justification that the contribution of the overlapping subregions consist of scaleless integrals only (and therefore is zero).

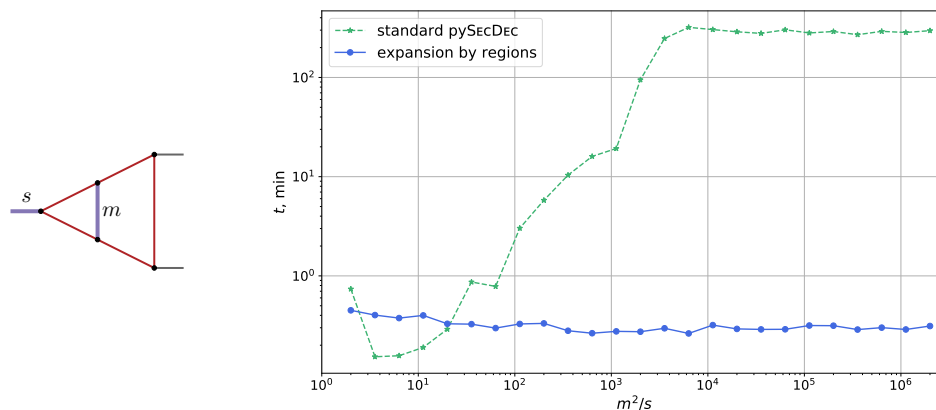


Figure 3. Time to evaluate the depicted integral to 3 digits of precision depending on the m^2/s ratio. The integration time is capped at 5 hours, and when $m^2/s > 3000$ the precision target can not be reached within that time.

Expansion by regions was previously implemented in ASY.M [20] and ASY2.M [21]; this implementation is currently shipped as a part of FIESTA. As of version 1.5 pySECDEC provides the function `loop_regions()` that expands a given loop integral by regions; the result of it can be directly turned into a standard pySECDEC integration library. The integration time of this library is shown on Figure 3 in comparison to the integration time of the unexpanded integral: as expected after the expansion the integration time no longer depends on m^2/s .

5. Conclusions

The recently released pySECDEC version 1.5 comes with two major new features: automatic adaptive evaluation of the weighted sums of integrals (e.g. amplitudes) and an implementation of asymptotic expansion of integrals in kinematic ratios. The first one brings a major speedup in the evaluation of single integrals and allows using pySECDEC to evaluate whole amplitudes in an optimal way. The second provides an essential tool in handling extreme kinematics (e.g. high-energy regions).

Since this release the pySECDEC team continues incrementally improving its performance, hoping to make it applicable to even more challenging integrals, and to establish it as a tool to optimally evaluate whole amplitudes. In our view sector decomposition and pySECDEC are the tools of the last resort: when analytic evaluation is impossible, when differential equations are unsolvable or unavailable, pySECDEC will still remain a viable option to get a result. Since the need for higher-loop corrections is higher than ever, we believe that the time for the last resort is nigh.

Acknowledgements

This research was supported in part by the COST Action CA16201 (Particleface) of the European Union and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 (TRR 257).

References

- [1] Freitas A 2021 *Acta Phys. Polon. B* **52** 929–946
- [2] Passarino G and Veltman M J G 1979 *Nucl. Phys. B* **160** 151–207
- [3] Borowka S, Heinrich G, Jahn S, Jones S P, Kerner M, Schlenk J and Zirke T 2018 *Comput. Phys. Commun.* **222** 313–326 (*Preprint* [1703.09692](#))
- [4] Borowka S, Heinrich G, Jahn S, Jones S P, Kerner M and Schlenk J 2019 *Comput. Phys. Commun.* **240** 120–137 (*Preprint* [1811.11720](#))

- [5] Heinrich G, Jahn S, Jones S P, Kerner M, Langer F, Magerya V, Pöldaru A, Schlenk J and Villa E 2022 *Comput. Phys. Commun.* **273** 108267 (*Preprint* [2108.10807](#))
- [6] Binoth T and Heinrich G 2000 *Nucl. Phys. B* **585** 741–759 (*Preprint* [hep-ph/0004013](#))
- [7] Heinrich G 2008 *Int. J. Mod. Phys. A* **23** 1457–1486 (*Preprint* [0803.4177](#))
- [8] Smirnov A V, Shapurov N D and Vysotsky L I 2021 (*Preprint* [2110.11660](#))
- [9] Carter J and Heinrich G 2011 *Comput. Phys. Commun.* **182** 1566–1581 (*Preprint* [1011.5493](#))
- [10] Borowka S, Carter J and Heinrich G 2013 *Comput. Phys. Commun.* **184** 396–408 (*Preprint* [1204.4152](#))
- [11] Borowka S, Heinrich G, Jones S P, Kerner M, Schlenk J and Zirke T 2015 *Comput. Phys. Commun.* **196** 470–491 (*Preprint* [1502.06595](#))
- [12] Hahn T 2005 *Comput. Phys. Commun.* **168** 78–95 (*Preprint* [hep-ph/0404043](#))
- [13] Dubovyk I, Usovitsch J and Grzanka K 2021 *Symmetry* **13** 975
- [14] von Manteuffel A, Panzer E and Schabinger R M 2015 *JHEP* **02** 120 (*Preprint* [1411.7392](#))
- [15] Chen L, Heinrich G, Jones S P, Kerner M, Klappert J and Schlenk J 2021 *JHEP* **03** 125 (*Preprint* [2011.12325](#))
- [16] Chen L, Heinrich G, Jahn S, Jones S P, Kerner M, Schlenk J and Yokoya H 2020 *JHEP* **04** 115 (*Preprint* [1911.09314](#))
- [17] Jones S P, Kerner M and Luisoni G 2018 *Phys. Rev. Lett.* **120** 162001 (*Preprint* [1802.00349](#))
- [18] Beneke M and Smirnov V A 1998 *Nucl. Phys. B* **522** 321–344 (*Preprint* [hep-ph/9711391](#))
- [19] Jantzen B 2011 *JHEP* **12** 076 (*Preprint* [1111.2589](#))
- [20] Pak A and Smirnov A 2011 *Eur. Phys. J. C* **71** 1626 (*Preprint* [1011.4863](#))
- [21] Jantzen B, Smirnov A V and Smirnov V A 2012 *Eur. Phys. J. C* **72** 2139 (*Preprint* [1206.0546](#))