Loop integral evaluation with pySecDec

Vitaly Magerya

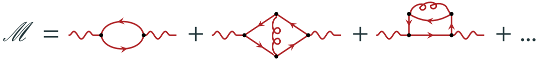
Institute for Theoretical Physics,
Karlsruhe Institute of Technology

ACAT 2021,
Daejeon, South Korea

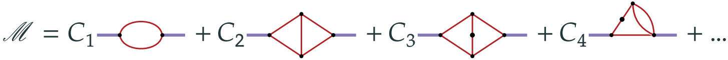
Where does pySecDec fit in?

Basic steps of calculating scattering matrix elements:

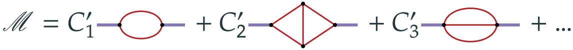
1. Generate Feynman diagrams for the process.

   * 

2. Project the diagrams onto scalar integrals.

   * 

3. Reduce the number of integrals using IBP relations.

   * 

4. *Evaluate the loop integrals.*

   * Analytically: hard to impossible at two loops if masses are present.
   * *Numerically:*
     * *Sector decomposition:* pySecDec, Fiesta.
     * Mellin-Barnes representation: MB, Ambre.
     * Numerical differential equations: e.g. DiffExp + pySecDec/Fiesta/MB.

5. Integrate over kinematics.

   * A whole field of study by itself.

Sector decomposition in short

$$I = \int_0^1 \mathrm{d}x \int_0^1 \mathrm{d}y \, \left(x+y\right)^{-2+\varepsilon} = ?$$

Problem: the integrand diverges at $x, y \to 0$, can't integrate numerically.

Solution: [Heinrich '08; Binoth, Heinrich '00]

1. Factorize the divergence in x and y with sector decomposition:
   - $I = \int \cdots \times \Big(\underbrace{\theta\left(x > y\right)}_{\text{Sector 1}} + \underbrace{\theta\left(x < y\right)}_{\text{Sector 2}}\Big) = \int_0^1 \mathrm{d}x \int_0^x \mathrm{d}y \, \left(x+y\right)^{-2+\varepsilon} + \begin{pmatrix} x \\ \updownarrow \\ y \end{pmatrix}$
2. Rescale the integration region in each sector back to a hypercube:
   - $I \overset{y \to x\,y}{=} \int_0^1 \mathrm{d}x \, \underbrace{x^{-1+\varepsilon}}_{\text{Factorized pole}} \int_0^1 \mathrm{d}y \left(1+y\right)^{-2+\varepsilon} + \begin{pmatrix} x \\ \updownarrow \\ y \end{pmatrix}$
3. Extract the pole at $x \to 0$ analytically, expand in ε:
   - $I = -\frac{2}{\varepsilon} \int_0^1 \mathrm{d}y \left(1+y\right)^{-2+\varepsilon} = -\frac{2}{\varepsilon} \int_0^1 \mathrm{d}y \left(\frac{1}{\left(1+y\right)^2} - \frac{\ln\left(1+y\right)}{\left(1+y\right)^2} \varepsilon + \mathcal{O}\left(\varepsilon^2\right) \right)$
4. Integrate each term in ε numerically (they all converge now).

pySecDec overview

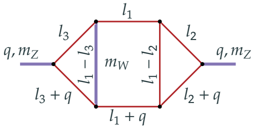
pySecDec: library for *numerically evaluating parametric integrals* via sector decomposition and Monte Carlo integration. [Heinrich et al '21, '18, '17]

- https://github.com/gudrunhe/secdec
- Written in *Python*, C++, Form.
- Multiple sector decomposition methods: iterative, geometric.
- Multiple integration algorithms:
  - Best: *Randomized Quasi-Monte Carlo* (Qmc); [Borowka et al '18]
    - Integration error $\sim \frac{1}{N_{\text{samples}}}$ (classical Monte Carlo has $\sim \frac{1}{N_{\text{samples}}^{1/2}}$);
    - Works on CPUs and GPUs (with CUDA);
  - Classic: Vegas/Suave/Divonne/Cuhre (Cuba), Cquad (Gsl).
- Installation via the *standard Python package* installer (Linux, Mac):
  - `pip3 install --user pySecDec`

Recent new features:

- Adaptive sampling of whole amplitudes (weighted sums of integrals).
- Builtin asymptotic expansion of integrals.

Using pySECDEC for a single Feynman integral



$= ?$

Generate the integration library:

```
import pySecDec as psd
if __name__ == "__main__":
    psd.loop_package(
        name="dia1",
        loop_integral=
            psd.LoopIntegralFromPropagators(
                loop_momenta=["l1", "l2", "l3"],
                propagators=[
                    "(l1)**2",
                    "(l2)**2",
                    "(l3)**2",
                    "(l1 + q)**2",
                    "(l2 + q)**2",
                    "(l3 + q)**2",
                    "(l1 - l2)**2",
                    "(l1 - l3)**2 - mw2"
                ],
                powerlist=[1,1,1,1,1,1,1,1],
                replacement_rules=[
                    ("q*q", "mz2")
                ]),
        real_parameters=["mz2", "mw2"],
        requested_orders=[0],
        decomposition_method="geometric")
```

Inspect the generated code:

```
$ ls dia1/
dia1_data/              dia1_integral/  pylink/
src/                    Makefile        Makefile.conf
README                  dia1.hpp        integral_names.txt
integrate_dia1.cpp
```

Compile it for the CPU:

```
$ cd dia1 && make
```

or for both CPU and GPU (with CUDA):

```
$ export SECDEC_WITH_CUDA_FLAGS="-arch=sm_80" CXX=nvcc
$ cd dia1 && make
```

Run it:

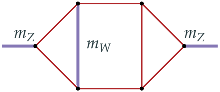
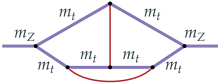
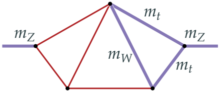
```
from pySecDec.integral_interface import IntegralLibrary
if __name__ == "__main__":
    lib = IntegralLibrary("dia1/dia1_pylink.so")
    lib.use_Qmc(verbosity=1)
    mz2 = 1.0
    mw2 = 0.78
    _, _, result = lib([mz2, mw2], epsrel=1e-7, verbose=True)
    print(result)
```

The result:

```
+ ((6.82645729748523067e+00,-1.60711801420445148e+01)
+/- (8.07205205949069617e-07,7.55815020343424309e-07))
+ O(eps)
```

Expected performance for 3-loop EW integrals

pySecDec[1] + QMC *integration times* for 3-loop self-energy integrals:[2]

<table>
<tr><th colspan="2">Diagram \ Relative precision</th><th>10^{-3}</th><th>10^{-4}</th><th>10^{-5}</th><th>10^{-6}</th><th>10^{-7}</th><th>10^{-8}</th></tr>
<tr><td rowspan="2"></td><td>GPU</td><td>15s</td><td>20s</td><td>40s</td><td>200s</td><td>13m</td><td>50m</td></tr>
<tr><td>CPU</td><td>10s</td><td>50s</td><td>400s</td><td>4000s</td><td>180m</td><td>1200m</td></tr>
<tr><td rowspan="2"></td><td>GPU</td><td>18s</td><td>19s</td><td>30s</td><td>20s</td><td>1.2m</td><td>2m</td></tr>
<tr><td>CPU</td><td>5s</td><td>14s</td><td>60s</td><td>50s</td><td>12m</td><td>16m</td></tr>
<tr><td rowspan="2"></td><td>GPU</td><td>6s</td><td>11s</td><td>12s</td><td>30s</td><td>3m</td><td>24m</td></tr>
<tr><td>CPU</td><td>5s</td><td>10s</td><td>50s</td><td>800s</td><td>60m</td><td>800m</td></tr>
</table>

[Same diagrams as in Dubovyk, Usovitsch, Grzanka '21]

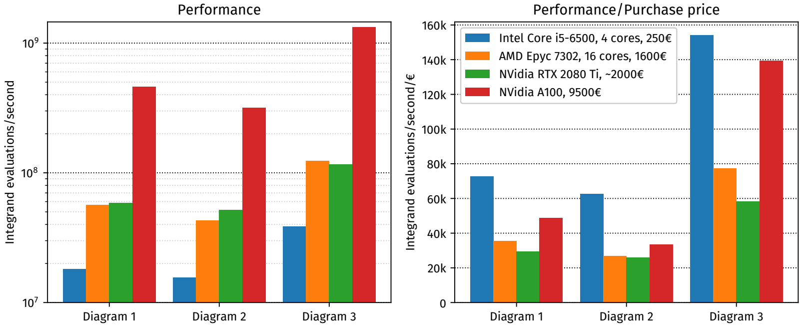
In short: *seconds to minutes per integral* to achieve practical precision.

[1] Version 1.5 + work-in-progress + AVX2.

[2] *GPU*: NVidia A100 40GB; *CPU*: AMD Epyc 7302 with 32 threads.

CPU vs GPU with pySecDec

pySecDec[3] integrand sampling speed on different devices:

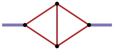


In short:

- Top consumer-grade GPU (RTX 2080 Ti) ≈ server-grade CPU.
- Top server-grade GPU (A100) ≈ $10\times$ server-grade CPU.

[3]Version 1.5 + work-in-progress + AVX2.

Adaptive sampling of amplitudes

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:	$3 \cdot 10^6$	$51 \cdot 10^{-6}$	$3 \cdot 10^6$	$32 \cdot 10^{-6}$

[Example assumes integration error = $1/n$]

pySecDec now automatically optimizes the total integration time based on

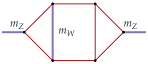
- how fast each integral can be sampled,
- how well it converges,
- how large its coefficient is.

$\Rightarrow$ Automatic *speedup for amplitudes* (weighted sums of integrals).

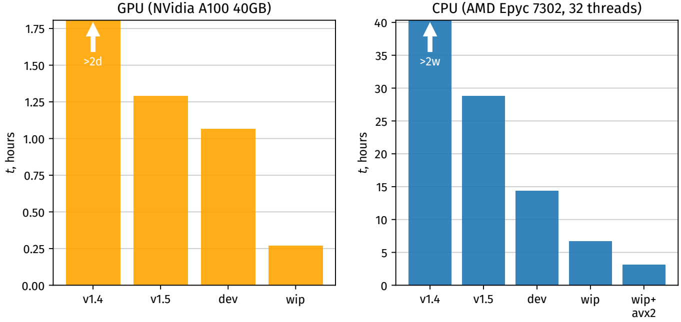
$\Rightarrow$ Automatic speedup for single integrals too (sums of sectors).

$\Rightarrow$ Used in e.g. 2-loop $gg \to ZZ$ (2011.15113) and $gg \to ZH$ (2011.12325); see also: talk by Chaitanya Paranjape on Thursday.

Performance improvements by pySecDec version

Time to integrate 

to 7 digits of precision with pySecDec + QMC:

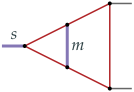


Speedup sources:

- *v1.5:* adaptive sampling, automatic contour deformation adjustment;
- *dev:* separation of real and complex variables in the integrand code;
- *wip:* removal of the indirection from the integrand code, vectorization.

The latest release is fast; *the next release will be faster.*

Asymptotic expansion, briefly

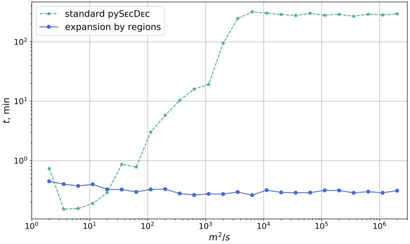
$I \equiv$  $= ?$

Problem: when $s/m^2 \ll 1$ numerical integration converges poorly.

- General problem when scale ratios are not ≈ 1.

Solution: asymptotic expansion in scale ratios. [Beneke, Smirnov '98, Jantzen '11]

- Takes out extreme ratios from the integrand:
  - $I = (\cdots + \cdots)\left(\frac{s}{m^2}\right)^{-1} + (\cdots + \cdots + \cdots + \cdots)\left(\frac{s}{m^2}\right)^{0} + \mathcal{O}\left(\frac{s}{m^2}\right)$
- Now implemented in pySecDec (through `pySecDec.loop_regions`).
- E.g. time to integrate the above triangle to 10^{-3} accuracy:



Summary

pySecDec provides:

- Numerical evaluation of *massive multi-loop integrals*.
  - 3-loop massive integrals at 6 digits in seconds to minutes.
- Dependable results when other methods fail.

The latest release (v1.5) introduces: [Heinrich '21]

- Optimized evaluation of *amplitudes* (weighted sums of integrals).
  - Proven in several 2-loop calculations.
  - Automatically applicable to single integrals too.
- *Asymptotic expansion* of integrals for extreme kinematics.

Future releases will bring:

- Even better performance.

Summary

pySecDec provides:

- Numerical evaluation of *massive multi-loop integrals*.
  - 3-loop massive integrals at 6 digits in seconds to minutes.
- Dependable results when other methods fail.

The latest release (v1.5) introduces: [Heinrich '21]

- Optimized evaluation of *amplitudes* (weighted sums of integrals).
  - Proven in several 2-loop calculations.
  - Automatically applicable to single integrals too.
- *Asymptotic expansion* of integrals for extreme kinematics.

Future releases will bring:

- Even better performance.

Thank you for your attention.

Backup slides

The need for higher loop corrections

Error predictions on key electroweak parameters: [Freitas '21]

Errors		Γ_Z, MeV	m_W, MeV	R_b, 10^{-5}	$\sin^2\theta^l_{\text{eff}}$, 10^{-5}
Experimental	current	2.3	12	66	14
	FCC-ee	0.1	0.7	6.0	0.5
	CEPC	0.5	1.0	4.3	2.3
Theoretical	current, ≤ 2 loops	0.4	4	10	4.5
	future?, ≤ 3 loops	0.15	1.0	5	1.5

Up to *3-loop theory is needed* to match the experimental precision of the future colliders.

Using pySecDec for amplitudes

Generate the integration library:

```
import pySecDec as psd
if __name__ == "__main__":
    # First term
    term1 = psd.LoopPackage(
        name="integral1",
        loop_integral=
            psd.LoopIntegralFromPropagators(
                ...
            ),
        real_parameters=[...])
    coeff1 = psd.Coefficient(
        numerators=['1 + 2*eps^3', ...],
        denominators=['-3 + 2*eps', '-1 + 2*eps', ...],
        parameters=[])
    # Second term
    term2 = ...
    coeff2 = ...
    ...
    # The amplitude
    psd.sum_package('amplitude',
        [term1, term2, ...],
        regulators=['eps'],
        requested_orders=[0],
        coefficients=[[coeff1, coeff2, ...]],
        real_parameters=[...]
    )
```

Compile an *run*: same as for a single integral.

$\Longrightarrow$ Under the hood single integrals are implemented the same as sums.

Using pySecDec with asymptotic expansion

Generate the integration library:

```
import pySecDec as psd
if __name__ == "__main__":
    # find the regions and expand the integral up to O(s)
    terms = psd.loop_regions(
        name="triangle2L",
        loop_integral=
            psd.LoopIntegralFromPropagators(
                loop_momenta=["l1", "l2"],
                external_momenta=["p1", "p2"],
                propagators=[
                    "(l1 + p1)**2",
                    "(l1 - p2)**2",
                    "(l2 + p1)**2",
                    "(l2 - p2)**2",
                    "l2**2",
                    "(l1 - l2)**2 - msq"
                ],
                replacement_rules=[
                    ("p1*p1", 0),
                    ("p2*p2", 0),
                    ("p1*p2", "s/2")
                ]),
        smallness_parameter="s",
        decomposition_method="geometric",
        expansion_by_regions_order=0)
    # generate the library
    psd.sum_package("triangle2L_by_regions",
        terms,
        regulators=["eps"],
        requested_orders=[0],
        real_parameters=["s", "msq"])
```

Compile an *run*: same as for a single integral.

⇒ Adaptive amplitude optimization is enabled automatically.