pySecDec and the integration challenges at multiloop-EW

Vitaly Magerya

Institute for Theoretical Physics,
Karlsruhe Institute of Technology

The 2021 International Workshop on the
High Energy Circular Electron Positron Collider
Nanjing, 2021

The need for higher loop corrections

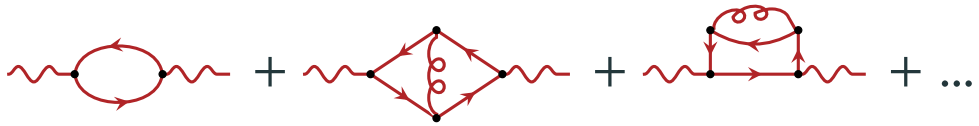
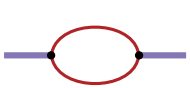
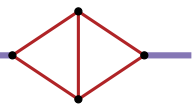
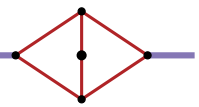
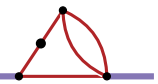
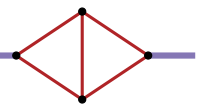
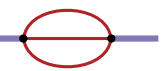
Error predictions on key electroweak parameters: [Freitas '21]

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
	CEPC	0.5	1.0	4.3	2.3
Theoretical	current, $\leq$ 2 loops	0.4	4	10	4.5
	future?, $\leq$ 3 loops	0.15	1.0	5	1.5

Up to *3-loop theory is needed to match CEPC experimental precision.*

Where does pySecDec fit in?

Basic steps of calculating scattering matrix elements:

1. Generate Feynman diagrams for the process.
   * $\mathscr{M} =$  $+ \ldots$
2. Project the diagrams onto scalar integrals.
   * $\mathscr{M} = C_1$  $+ C_2$  $+ C_3$  $+ C_4$  $+ \ldots$
3. Reduce the number of integrals using IBP relations.
   * $\mathscr{M} = C'_1$  $+ C'_2$  $+ C'_3$  $+ \ldots$
4. *Evaluate the loop integrals.*
   * Analytically: not feasible already at two loops if masses are present.
     $\Rightarrow$ Limited application to electroweak calculations (m_W, m_Z, m_t, m_H).
   * *Numerically:*
     * *Sector decomposition:* pySecDec, Fiesta.
     * Mellin-Barnes representation: MB, Ambre.
     * Numerical differential equations: DiffExp + pySecDec/Fiesta/MB.

Sector decomposition in short

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

Problem: 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:
   - $I = \int \cdots \times \left(\theta\left(x > y\right) + \theta\left(x < y\right)\right) = \int_0^1 \mathrm{d}x \int_0^x \mathrm{d}y \left(x+y\right)^{-2+\varepsilon} + \left(\begin{smallmatrix} x \\ \updownarrow \\ y \end{smallmatrix}\right)$
   - $I \overset{y \to x\, y}{=} 2 \int_0^1 \mathrm{d}x\, x^{-1+\varepsilon} \int_0^1 \mathrm{d}y \left(1+y\right)^{-2+\varepsilon}$
2. Extract the pole at $x \to 0$ analytically, expand in ε:
   - $I = -\frac{2}{\varepsilon} \int_0^1 \mathrm{d}y \left(1+y\right)^{-2+\varepsilon} = -\frac{2}{\varepsilon} \int_0^1 \mathrm{d}y \frac{1-\ln\left(1+y\right)\varepsilon}{\left(1+y\right)^2} + \mathscr{O}\left(\varepsilon\right)$
3. Integrate each term in ε numerically (they all converge).

pySecDec overview

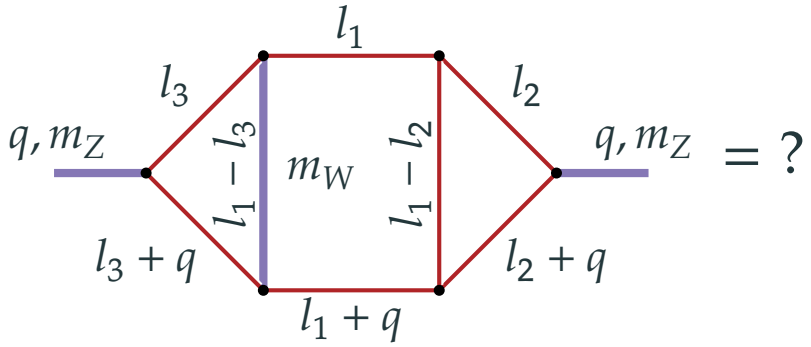
pySecDec: program for *numerically evaluating parametric integrals* via sector decomposition. [Heinrich et al '21, '18, '17]

* `https://github.com/gudrunhe/secdec`
* Written in *Python*, C++, Form.
* Good for dimensionally regularized Feynman integrals, but also arbitrary scalar or tensor parametric integrals.
* Multiple sector decomposition methods: iterative, geometric.
* Multiple integration algorithms:
  * Best: Randomized Quasi-Monte Carlo (Qmc);
  * Classic: Vegas/Suave/Divonne/Cuhre (Cuba), Cquad (Gsl).
* Works on *CPUs* and *GPUs* (with Qmc and CUDA).
* Installation via the *standard Python package* installer (Linux, Mac):
  * `pip3 install --user pySecDec`

Recent new features:

* Adaptive evaluation 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__":
    li = 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")
        ])
    psd.loop_package(
        name="dia1",
        loop_integral=li,
        real_parameters=["mz2", "mw2"],
        requested_orders=[0],
        decomposition_method="geometric")
```

Compile it for the CPU:

```
$ cd dia1 && make
$ ls
Makefile            Makefile.conf  README          dia1.hpp
dia1_data           dia1_integral  dia1_pylink.so  integral_names
integrate_dia1.cpp  pylink         src
```

… or for the GPU (with CUDA):

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

Run it:

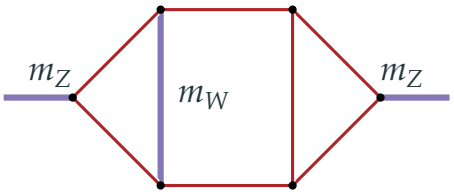
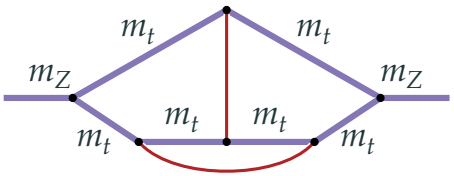
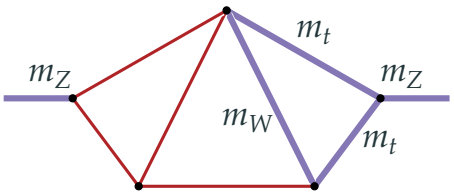
```
from pySecDec.integral_interface import IntegralLibrary
if __name__ == "__main__":
    lib = IntegralLibrary("dia1/dia1_pylink.so")
    lib.use_Qmc(verbosity=1)
    mz2 = 1.0
    mw2 = 0.78
    _, _, result = lib([mz2, mw2], epsrel=1e-7, verbose=True)
    print(result)
```

The result will read:

```
+ ((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]

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	—
	CPU	5s	10s	50s	800s	60m	800m

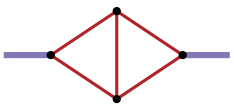
[Same diagrams as in Dubovyk, Usovitsch, Grzanka '21]

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

[1] Development version.

[2] *GPU*: NVidia A100; *CPU*: AMD Epyc 2.8GHz, 32 threads.

Adaptive amplitude evaluation

Amplitude	Naive sampling	Nave error	Better sampling	Better error
300 	10^6 samples	$300 \cdot 10^{-6}$	$2 \cdot 10^6$ samples	$150 \cdot 10^{-6}$
20 	10^6 samples	$20 \cdot 10^{-6}$	$\frac{1}{2} \cdot 10^6$ samples	$40 \cdot 10^{-6}$
1 	10^6 samples	$1 \cdot 10^{-6}$	$\frac{1}{2} \cdot 10^6$ samples	$2 \cdot 10^{-6}$
Total:	$3 \cdot 10^6$	$321 \cdot 10^{-6}$	$3 \cdot 10^6$	$192 \cdot 10^{-6}$

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

More generally: pySecDec can optimize the total integration time based on:

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

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

$\Rightarrow$ 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).

Using pySecDec for amplitudes

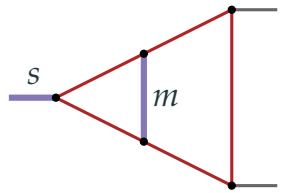
Generate the integration library:

```
import pySecDec as psd
if __name__ == "__main__":
    # First term
    li1 = psd.LoopIntegralFromPropagators(
        ...
    )
    term1 = psd.LoopPackage(
        name="integral1",
        loop_integral=li1,
        real_parameters=[...])
    coeff1 = psd.Coefficient(
        numerators=['1 + 2*eps^3', ...],
        denominators=['-3 + 2*eps', '-1 + 2*eps'],
        parameters=[])
    # Second term
    li2 = ...
    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.

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

Asymptotic expansion, briefly

$$I \equiv$$ 

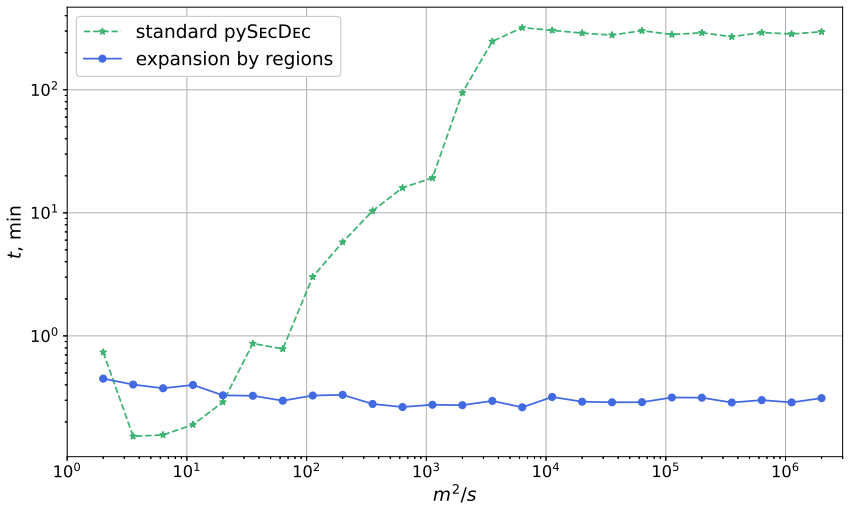
$$= ?$$

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.
- E.g. time to integrate the above triangle to 10^{-3} accuracy:



Using pySecDec with asymptotic expansion

Generate the integration library:

```
import pySecDec as psd
if __name__ == "__main__":
    # define the integral
    li = 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")
        ])
    # find the regions and expand the integral up to O(s)
    terms = psd.loop_regions(
        name="triangle2L",
        loop_integral=li,
        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.

$\Longrightarrow$ Adaptive amplitude optimization is enabled automatically.

Summary

- CEPC requires amplitudes with 2- and 3-loop massive integrals.
- pySecDec provides:
  1. Numerical evaluation of *massive multi-loop integrals*.
     - 3-loop EW integrals at 6 digits in seconds to minutes.
  2. Optimized evaluation of *amplitudes*.
     - Proven in multiple 2-loop calculations.
  3. *Asymptotic expansion* for extreme kinematics.
- Latest pySecDec release (v1.5, 2108.10807): faster than ever.
  - Next release: even faster.
- When all else fails, pySecDec is still there for you.

Summary

* CEPC requires amplitudes with 2- and 3-loop massive integrals.
* pySecDec provides:
  1. Numerical evaluation of *massive multi-loop integrals*.
     * 3-loop EW integrals at 6 digits in seconds to minutes.
  2. Optimized evaluation of *amplitudes*.
     * Proven in multiple 2-loop calculations.
  3. *Asymptotic expansion* for extreme kinematics.
* Latest pySecDec release (v1.5, 2108.10807): faster than ever.
  * Next release: even faster.
* When all else fails, pySecDec is still there for you.

Thank you for your attention.