

# Two-loop virtual amplitudes for $q\bar{q} \rightarrow t\bar{t}H$ via modular arithmetic and sector decomposition

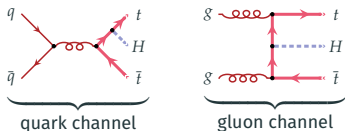
Vitaly Magerya

Institute for Theoretical Physics,  
Karlsruhe Institute of Technology,  
CRC TRR 257

June 13 2024, CERN

# $t\bar{t}H$ production at the LHC

At the tree level:

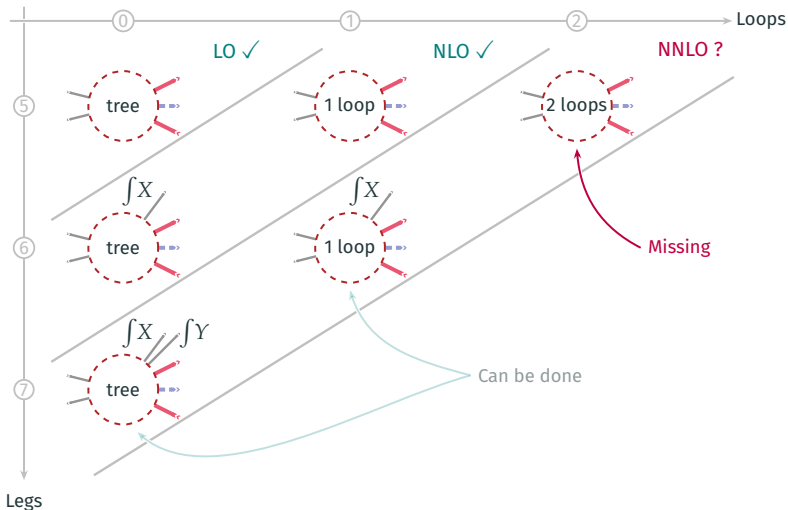


First observation at LHC reported in 2018. [ATLAS '17, '17, '18, '20, '23; CMS '18, '18, '20, '20, '22]  
 Measurements based on data from LHC Run 2 (2015–2018):

		$\sigma_{t\bar{t}H}/\sigma_{t\bar{t}H,SM}$	$\mathcal{L}$	$H$ decay channels
ATLAS '18	1.32	$+0.18_{-0.18}$ (stat) $+0.21_{-0.19}$ (syst)	$79.8 \text{ fb}^{-1}$	$\gamma\gamma, bb, WW, ZZ$
ATLAS '20	1.43	$+0.33_{-0.31}$ (stat) $+0.21_{-0.15}$ (syst)	$139 \text{ fb}^{-1}$	$\gamma\gamma$
CMS '20	1.38	$+0.29_{-0.27}$ (stat) $+0.21_{-0.11}$ (syst)	$137 \text{ fb}^{-1}$	$\gamma\gamma$
CMS '20	0.92	$+0.19_{-0.19}$ (stat) $+0.17_{-0.13}$ (syst)	$137 \text{ fb}^{-1}$	$WW, \tau\tau, ZZ$

HL-LHC will have  $\mathcal{L} \sim 3000 \text{ fb}^{-1}$ , reducing *statistical* uncertainty by 4-5x.  
 To reduce *systematic* uncertainty: *NNLO calculation is needed.*

# Parts of an NNLO calculation



Big missing part for NNLO: *two-loop virtual amplitudes*.

# Theory results for $t\bar{t}H$ production

## NLO:

- \* NLO QCD

[Beenakker, Dittmaier, Krämer, Plümper, Spira, Zerwas '01]

[Reina, Dawson '01]

[Reina, Dawson, Wackerath '01]

[Beenakker, Dittmaier, Krämer, Plümper, Spira, Zerwas '02]

[Dawson, Orr, Reina, Wackerath '02]

[Dawson, Jackson, Orr, Reina, Wackerath '03]

- \* NLO QCD, parton shower

[Frederix, Frixione, Hirschi, Maltoni, Pittau, Torrielli '11]

[Garzelli, Kardos, Papadopoulos, Trocsanyi '11]

[Hartanto, Jager, Reina, Wackerath '15]

- \* NLO EW

[Frixione, Hirschi, Pagani, Shao, and Zaro '14]

- \* NLO QCD+EW, NWA

[Zhang, Ma, Zhang, Chen, Guo '14]

[Frixione, Hirschi, Pagani, Shao, and Zaro '15]

- \* NLO QCD, off-shell

[Denner, Feger '15]

[Stremmer, Worek '21]

[Denner, Lang, Pellen '20]

[Bevilacqua, Bi, Hartanto, Kraus, Lupattelli, Worek '22]

# Theory results for $t\bar{t}H$ production, II

NLO, contd.:

- \* NLO+NLL QCD [Kulesza, Motyka, Stebel, Theeuwes '15]  
[Ju, Yang '19]
- \* NLO+NNLL QCD [Broggio, Ferroglia, Pecjak, Signer, Yang '15]  
[Broggio, Ferroglia, Pecjak, Yang '16]  
[Kulesza, Motyka, Stebel, Theeuwes '17]  
[Kulesza, Motyka, Schwartländer, Stebel, Theeuwes '20]
- \* NLO QCD+SMEFT [Maltoni, Vryonidou, Zhang '16]
- \* NLO QCD+EW, off-shell [Denner, Lang, Pellen, Uccirati '16]
- \* NLO+NNLL QCD+EW [Broggio, Ferroglia, Frederix, Pagani, Pecjak, Tsinikos '19]
- \* NLO QCD to  $\mathcal{O}(\varepsilon^2)$  [Buccioni, Kreer, Liu, Tancredi '23]
- \*  $t \rightarrow H$  fragmentation functions at  $\mathcal{O}(y_t^2 \alpha_s)$  [Brancaccio, Czakon, Generet, Krämer '21]

# Theory results for $t\bar{t}H$ production, III

## NNLO:

- \* NNLO QCD, flavour off-diagonal [Catani, Fabre, Grazzini, Kallweit '21]
- \* NNLO QCD total cross-section, soft Higgs [Catani, Devoto, Grazzini, Kallweit, Mazzitelli, Savoini '22]
- \* Two-loop QCD virtual amplitude, IR poles [Chen, Ma, Wang, Yang, Ye '22]
- \* Leading  $N_c$  two-loop QCD master integrals,  $n_l$ -part [Cordero, Figueiredo, Kraus, Page, Reina '23]
- \* Two-loop QCD virtual amplitude, high-energy boosted limit [Wang, Xia, Yang, Ye '24]
- \* *Two-loop QCD virtual amplitude,  $q\bar{q}$  channel,  $n_l$ - and  $n_h$ -parts* [Agarwal, Heinrich, Jones, Kerner, Klein, Lang, V.M., Olsson '24]

# The amplitude

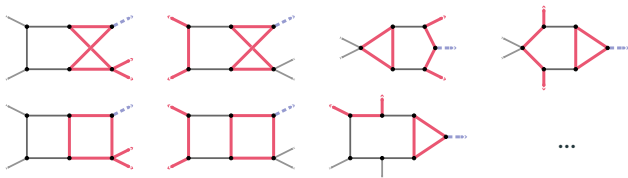
Model: QCD with a scalar  $H$ ,  $n_l$  light (massless) quarks,  $n_h$  heavy (top) quarks.  
Amplitude of  $q\bar{q} \rightarrow t\bar{t}H$  projected onto Born, and decomposed in  $\alpha_s$  as

$$\langle \text{AMP} | \text{AMP}_{\text{tree}} \rangle = \mathcal{A} + \left( \frac{\alpha_s}{2\pi} \right) \mathcal{B} + \left( \frac{\alpha_s}{2\pi} \right)^2 \mathcal{C}.$$

As a proof-of-concept: only parts proportional to  $n_l$  or  $n_h$  in  $\mathcal{C}$  for now.

*Why is the calculation complicated?*

1. IBP reduction of the amplitude to master integrals is too complicated to be computed symbolically (at the moment).
  - \* 5 legs and 2 masses ( $m_t, m_H$ )  $\Rightarrow$  7 scales (6 scaleless variables).
2. Massive two-loop integrals contributing to  $\mathcal{C}$  are not known analytically.



# Calculation method

1. Generate all Feynman diagrams for  $q\bar{q} \rightarrow t\bar{t}H$  at two loops. [QGRAF]  
⇒ 249 non-zero diagrams (of 702 for the full  $q\bar{q}$  channel).
2. Insert Feynman rules, apply the projector  $|\text{AMP}_{\text{tree}}\rangle$ . [ALIBRARY]
3. Sum over the spinor and color tensors. [FORM; COLOR.H]  
⇒ ~20000 scalar integrals (of ~90000);  
⇒ 9 structures:  $\{n_h|n_l\} C_A C_F N_c$ ,  $\{n_h|n_l\} C_F^2 N_c$ ,  $\{n_h|n_l\} d_{33}$ ,  $\{n_h|n_l\}^2 C_F N_c$ ;  
\* 6 structures not included:  $C_A^2 C_F N_c$ ,  $C_A C_F^2 N_c$ ,  $C_F^3 N_c$ ,  $C_A d_{33}$ ,  $C_F d_{33}$ ,  $d_{44}$ .
4. Resolve integral symmetries, construct integral families. [FEYNCON; ALIBRARY]  
⇒ 44 families, 28 up to external leg permutation (of 89 and 39).
5. Figure out master integral count in each sector. [KIRA]  
⇒ 831 master integrals in total (of 3005 for the full  $q\bar{q}$  channel);  
⇒ up to 8 integrals per sector (up to 13 for the full  $q\bar{q}$  channel).

...



# Calculation method, II

6. Choose a *good master integral basis*, allowing raised denominator powers and dimensional shifts.
7. Generate IBP relations, dimensional recurrence relations. [KIRA; ALIBRARY]
8. *Precompute* (“trace”) the *IBP solution* for each family with Rational Tracer. [RATRACER]
9. *Precompile* the pySECDEC *integration library* for the amplitude pieces. [pySECDEC]
  - \* Each color structure as a separate weighted sum of the master integrals.
10. *For each point* in the phase space:
  - 10.1 *Solve IBP relations* using the precomputed trace (with RATRACER).
    - \* Each Mandelstam variable set to a rational number.
  - 10.2 *Evaluate the amplitudes* as weighted sums of masters (with pySECDEC).
    - \* The weights are taken from the IBP solution.
  - 10.3 Apply renormalization and pole subtraction.  
[Ferroglia, Neubert, Pecjak, Yang '09; Bärnreuther, Czakon, Fiedler '13]
  - 10.4 Save the result.





# Choosing the master integrals

A good basis of master integrals optimizes the IBP solution time and the pySECDEC evaluation time. Our choice:

- \* is quasi-finite, [von Manteuffel, Schabinger '14]
- \* is  $d$ -factorizing, [Smirnov, Smirnov '20; Usovitsch '20]
- \* results in IBP coefficients with small denominators,
- \* avoids  $\varepsilon$  poles in the coefficients of top-level sectors,
- \* avoids  $\varepsilon$  poles in the differential equation matrix,
- \* is fast to evaluate with pySECDEC.

⇒ Need to consider denominator powers raised up to 6,  
and dimensional shifts to  $d = 6 - 2\varepsilon$  and  $d = 8 - 2\varepsilon$ .

To illustrate, pySECDEC integration time to  $10^{-3}$  precision:<sup>1</sup>

	$\varepsilon^{-2} \dots \varepsilon^0$	>2h		$\varepsilon^{-2} \dots \varepsilon^0$	20m
	$\varepsilon^{-2} \dots \varepsilon^0$	1m		$\varepsilon^{-3} \dots \varepsilon^0$	27s

<sup>1</sup>pySECDEC 1.5.3, NVidia A100 GPU.

# IBP relations with RATRACER

# IBP relations

An *IBP integral family* with  $L$  *loop momenta*  $l_i$ , and  $E$  *external momenta*  $p_i$ , is the set of Feynman integrals

$$I_{\nu_1, \nu_2, \dots, \nu_N} \equiv \int \frac{d^d l_1 \cdots d^d l_L}{D_1^{\nu_1} \cdots D_N^{\nu_N}}, \quad D_i \equiv (l_j \pm p_k \pm \dots)^2 - m_i^2 + i0,$$

where  $\nu_i$  are the “indices”, the  $D_i$  are the “denominators”.

The idea: shifting  $l_k$  by *any vector*  $v$  should not change  $I$ :

$$\lim_{\alpha \rightarrow 0} \frac{\partial}{\partial \alpha} I(l_k \rightarrow l_k + \alpha v) = \int d^d l_1 \cdots d^d l_L \frac{\partial}{\partial l_k^\mu} \frac{v^\mu}{D_1^{\nu_1} \cdots D_N^{\nu_N}} = 0.$$

These are the *IBP relations*, valid for all  $k$  and all  $v$ ,  $L(L + E)$  in total.

\* \* \*

*Example.* For  $I_{a,b,c} \equiv \text{---} \left\langle \begin{array}{l} b \\ a \\ c \end{array} \right\rangle$ , if we choose  $k = 1$  and  $v = l_1$ , we will get

$$(d - 2a - b - c)I_{a,b,c} - c I_{a-1,b,c+1} - b I_{a-1,b+1,c} = 0.$$

# Laporta algorithm

To solve IBP relations in practice use the *Laporta algorithm*: [Laporta '00]

1. Substitute integer values for the indices  $\nu_i$  into the IBP relations, obtaining a large linear system with many different  $I_{\nu_1 \dots \nu_N}$ .
2. Define an ordering on  $I_{\nu_1 \dots \nu_N}$  from “simple” to “complex” integrals.
3. Perform Gaussian elimination on the linear system, eliminating the most “complex” integrals first.
4. A small number of “simple” integrals will remain uneliminated.  
⇒ These are the *master integrals*. The rest will be expressed as their linear combinations.

# Software for solving IBPs

## IBP solvers not using modular arithmetic:

- \* **REDUZE 2.** [von Manteuffel, Studerus '12]
- \* **LITERED** (useful Mathematica functions, required by FIRE). [Lee '13]
- \* **FORCER** (for massless 2-point functions). [Ruijl, Ueda, Vermaseren '17]

## IBP solvers that use modular arithmetic:

- \* **FINRED** (a private implementation). [von Manteuffel et al]
- \* **FIRE6.** [Smirnov, Chuharev '19]
  - \* Does not provide multivariate reconstruction.
- \* **KIRA** when used with **FIREFLY.** [Klappert, Lange, Maierhöfer, Usovitsch '20; Klappert, Klein, Lange '20]
- \* **FINITEFLOW** (a library for arbitrary computations). [Peraro '19]
- \* **CARAVEL** (a library for amplitude computations). [Cordero, Sotnikov et al '20]
- \* **RATRACER** (with KIRA and FIREFLY). [V.M. '22]

... and multiple others.

# Modular arithmetic methods

To find a symbolic form of a rational function  $f(x_1, \dots, x_N)$ :

- \* *Evaluate*  $f$  modulo a prime number *many times*, with  $x_i$  set to integers.
- \* *Reconstruct* the exact symbolic form of  $f$  from the obtained values.

*Example:* if we have an unknown  $f(x)$ , and we have evaluated

$$\begin{aligned} f(11) &= 139 \pmod{997}, & f(65) &= 479 \pmod{997}, \\ f(38) &= 350 \pmod{997}, & f(92) &= 115 \pmod{997}, \end{aligned}$$

then we can use *polynomial interpolation* to find a polynomial form of  $f$ :

$$f(x) = 618 + 979x + 486x^2 + 41x^3 \pmod{997},$$

and then *rational function reconstruction* to find an equivalent rational form:

$$f(x) = \frac{996 + 333x}{1 + x} \pmod{997},$$

and finally *rational number reconstruction* to find the rational coefficients:

$$f(x) = \frac{-1 + \frac{2}{3}x}{1 + x} \pmod{997}.$$

*Guess* that this is the true form of  $f(x)$ ; evaluate more times to verify.











# Optimizing the modular Gaussian elimination

When performing Gaussian elimination one needs to:

- \* Represent the equation set as a sparse matrix data structure.
  - \* Keep the equations sorted.
  - \* Keep terms in each equation sorted.
  - \* Adjust the layout (and maybe reallocate memory) after each operation.
    - \* *This is not much work, but modular arithmetic is even less work!*

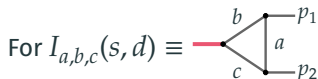
IBP solvers using modular arithmetics will:

- \* Recreate the same data structures, same memory allocations, in the same order during each evaluation, many times.
  - \* Only the modular values (small integers) change between evaluations.
- \* Spend relatively little time on actual modular arithmetic.
  - \* Because it is so fast!

How to speed this up? Eliminate the data structure overhead:

- \* *Record the list of arithmetic operations* performed during the first evaluation (“*a trace*”).
- \* Simply *replay this list* for subsequent evaluations.

# Rational traces



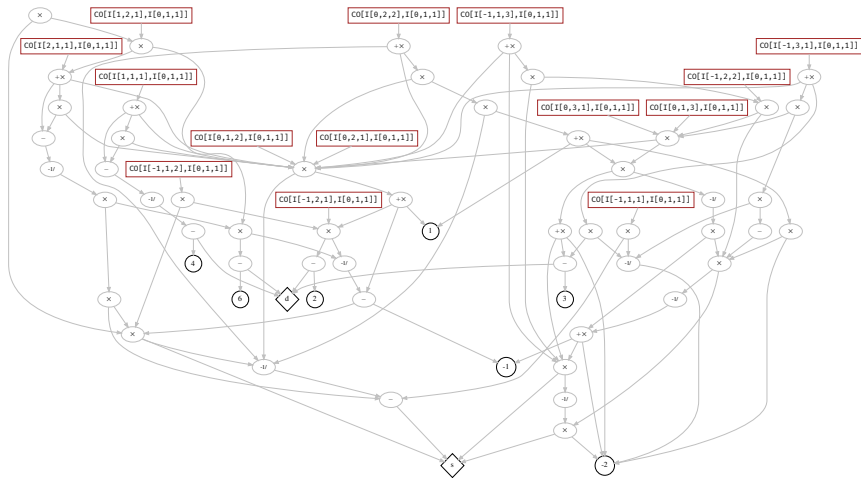
the *trace* of the IBP solution might look like:

```
t0 = var 'd'
t1 = int 4
t2 = sub t0 t1
t3 = int 1
t4 = var 's'
t5 = neg t4
t6 = int 6
t7 = sub t0 t6
t8 = int -1
t9 = int 2
t10 = int -2
t11 = sub t0 t9
t12 = int 3
t13 = sub t0 t12
t14 = mul t4 t10
t15 = neginv t5
t16 = mul t4 t15
t17 = sub t8 t16
t18 = mul t5 t16
t19 = neginv t17
t20 = mul t7 t19
[...]

t54 = addmul t53 t27 t44
t55 = mul t25 t44
t56 = addmul t55 t25 t44
t57 = mul t23 t44
t58 = addmul t57 t23 t44
t59 = mul t20 t58
t60 = mul t16 t59
save t60 as CO[I[1,1,2],I[0,1,1]]
save t59 as CO[I[1,2,1],I[0,1,1]]
save t58 as CO[I[2,1,1],I[0,1,1]]
save t56 as CO[I[1,1,1],I[0,1,1]]
save t54 as CO[I[-1,1,3],I[0,1,1]]
save t52 as CO[I[-1,2,2],I[0,1,1]]
save t51 as CO[I[-1,3,1],I[0,1,1]]
save t46 as CO[I[0,1,3],I[0,1,1]]
save t49 as CO[I[0,2,2],I[0,1,1]]
save t47 as CO[I[-1,1,2],I[0,1,1]]
save t46 as CO[I[0,3,1],I[0,1,1]]
save t42 as CO[I[-1,2,1],I[0,1,1]]
save t44 as CO[I[0,1,2],I[0,1,1]]
save t44 as CO[I[0,2,1],I[0,1,1]]
save t45 as CO[I[-1,1,1],I[0,1,1]]
```

# Rational traces

For  $I_{a,b,c}(s,d) \equiv$   the *trace* of the IBP solution might look like:



# RATRACER overview

RATRACER (“Rational Tracer”): a program for *solving systems of linear equations* using modular arithmetic based on rational traces. [V.M. '22]





- \* Can trace the solution of arbitrary systems of linear equations: IBP relations, dimensional recurrence relations, amplitude definitions, etc.
- \* Can optimize and transform traces.
- \* Always keeps traces on disk, not limited by RAM size.
- \* Uses FIREFLY for reconstruction. [Klappert, Klein, Lange '20, '19]
- \* Initially created for solving IBPs for massive 5-point 2-loop diagrams.
- \* Available at [github.com/magv/ratracr](https://github.com/magv/ratracr).

Intended usage:

1. Use KIRA (or LITERED, or custom code) to export IBP relations to text files.
2. Use RATRACER to load them and solve them.

# RATRACER benchmarks

For IBP reduction of every integral (i.e. not single amplitudes):

	Evaluation speedup vs. KIRA+FIREFLY	$\frac{t_{\text{reconstruction}}}{t_{\text{evaluation}}}$	Total speedup vs. KIRA+FIREFLY	Total speedup vs. KIRA+FERMAT	Total speedup vs. FIRE6
	20	3.3	5.2	1.2	$\infty?$
	7.8	1/3.3	6.0	37	$\infty?$
	26	25	1.7	1/3.3	1.8
	9.6	8.8	5.2	2.6	8.8

[[github.com/magy/ibp-benchmark](https://github.com/magy/ibp-benchmark)]

Resulting performance:

- \* Consistent ~10x speedup in modular evaluation over KIRA+FIREFLY.
- \* Up to ~5x *speedup in total reduction time* over KIRA+FIREFLY for complicated examples, 1x-30x over KIRA+FERMAT,  $\infty$ x over FIRE6.

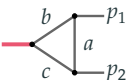


# Trace transformations

Given a trace, RATRACER can:

- \* *Set some of the variables to expressions* or numbers.
  - \* E.g. set  $mh2$  to “ $12/23*mt2$ ”,  $d$  to “ $4-2*eps$ ”, or  $s$  to “13600”.
  - \* No need to remake the IBP system just to set a variable to a number.
- \* *Select any subset of the outputs*, and drop operations that don't contribute to them (via dead code elimination).
  - \* Can be used to split the trace into parts.
    - \* Each part can be reconstructed separately (e.g. on a different machine).
  - \* See master-wise and sector-wise reduction in other solvers.
- \* *Expand the result into a series* in any variable.
  - \* By evaluating the trace while treating each value as a series, and saving the trace of that evaluation.
  - \* Done before the reconstruction, so one less variable to reconstruct in, but potentially more expressions (depending on the truncation order).
  - \* In practice only few leading orders in  $\varepsilon$  are needed, so expand in  $\varepsilon$  up to e.g.  $\mathcal{O}(\varepsilon^0)$ , and *don't waste time on reconstructing the higher orders*.

# Truncated series expansion

For  $I_{a,b,c}(s, d) \equiv$   before expansion:

- \* Variables to reconstruct in:  $s$  and  $d$ .
- \* Trace outputs: “CO [I [1, 1, 1], I [0, 1, 1]]”, etc:

$$I_{1,1,1} = \text{CO} [I [1, 1, 1], I [0, 1, 1]] I_{0,1,1}.$$





After expansion in  $\varepsilon$  to  $\mathcal{O}(\varepsilon^0)$ :

- \* Variables to reconstruct in: only  $s$ .
- \* Trace outputs: “ORDER [CO [I [1, 1, 1], I [0, 1, 1], eps<sup>-1</sup>]]”, etc:

$$I_{1,1,1} = \text{ORDER} [\text{CO} [I [1, 1, 1], I [0, 1, 1], \text{eps}^{-1}] \varepsilon^{-1} I_{0,1,1} \\ + \text{ORDER} [\text{CO} [I [1, 1, 1], I [0, 1, 1], \text{eps}^0] \varepsilon^0 I_{0,1,1}.$$

- \* Might be slower to evaluate, but fewer evaluations are needed.  
⇒ The more complicated the problem, the higher the speedup.

# RATRACER + series expansion benchmarks

	probe time speedup			total time speedup		
	$\mathcal{O}(\varepsilon^0)$	$\mathcal{O}(\varepsilon^1)$	$\mathcal{O}(\varepsilon^2)$	$\mathcal{O}(\varepsilon^0)$	$\mathcal{O}(\varepsilon^1)$	$\mathcal{O}(\varepsilon^2)$
	1/1.3	1/1.5	1/1.8	3.2	2.4	1.9
	1/2.0	1/2.5	1/3.0	2.7	1.4	1/1.3
	1/1.4	1/2.4	1/2.9	2.3	1.7	1.4
	1/1.0	1/1.6	1/2.1	4.3	2.3	1.6

[[github.com/magv/ibp-benchmark](https://github.com/magv/ibp-benchmark)]

Resulting performance:

- \* A *~3x speedup* with  $\varepsilon$  expansion up to  $\mathcal{O}(\varepsilon^0)$ .
- \* The higher the expansion, the less the benefit.

# RATRACER for $q\bar{q} \rightarrow t\bar{t}H$

For the calculation of 2-loop  $q\bar{q} \rightarrow t\bar{t}H$ :

- \* Reduction is done for each phase-space point separately.
  - ⇒ Mandelstam variables are set to rational numbers.
- \* Coefficients are expanded into a series in  $\varepsilon$ .
  - ⇒ No need to reconstruct in  $\varepsilon$ .
- ⇒ RATRACER outputs rational numbers, no need for function reconstruction.
- ⇒ Trace sizes of 0.4–90MB per family, 500MB in total (compressed).
- ⇒ Reduction in under *2 CPU minutes per phase-space point*.
  - \* Down from ~1 hour on 16 cores with KIRA 2.3+FIREFLY!
  - \* Around 10x improvement from faster evaluation.
  - \* Around 3x-4x improvement from expansion in  $\varepsilon$ .
  - \* Fast enough that we don't need symbolic IBP solutions.

# Feynman integrals with pySECDEC

# Amplitude evaluation with pySECDEC

pySECDEC: library for numerically evaluating Feynman integrals via *sector decomposition* and *(Quasi-) Monte Carlo integration*. [Heinrich et al '23, '21, '18, '17]

- \* [github.com/gudrunhe/secdec](https://github.com/gudrunhe/secdec)
- \* Takes a specification for *weighted sum of integrals* (i.e. amplitudes), decomposes integrals into sectors to isolate divergences, produces an Quasi-Monte Carlo integration library.
  - \* We use one sum per color structure.
  - \* Integrals sampled adaptively to reach the requested precision of the sums.
  - \* The 831 masters decompose into  $\sim 18000$  sectors ( $\sim 28000$  integrals).
- \* In the latest release version 1.6: [Heinrich et al '23]
  - \* New “median QMC lattice” construction, with unlimited maximum size.
  - \* New RQMC integrator “disteval” with 4x-5x speedup across the board.
- \* Integration time to get 0.3% precision for this calculation on a GPU:
  - \* from *5 minutes in the bulk* of the phase-space,
  - \* to  $\infty$  near boundaries (e.g. high-energy region) due to growing cancellations and spiky integrals (capped at 1 day).

# Sector decomposition in short

$$I = \int_0^1 dx \int_0^1 dy (x+y)^{-2+\varepsilon} = ?$$

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

Solution:

[Heinrich '08; Binoth, Heinrich '00]

1. Factorize the divergence in  $x$  and  $y$  with sector decomposition:

$$* I = \int \cdots \times \underbrace{(\theta(x > y))}_{\text{Sector 1}} + \underbrace{\theta(y > x)}_{\text{Sector 2}} = \int_0^1 dx \int_0^x dy (x+y)^{-2+\varepsilon} + \left( \begin{array}{c} x \\ \updownarrow \\ y \end{array} \right)$$

2. Rescale the integration region in each sector back to a hypercube:

$$* I \stackrel{y \rightarrow xy}{=} \int_0^1 dx \underbrace{x^{-1+\varepsilon}}_{\text{Factorized pole}} \int_0^1 dy (1+y)^{-2+\varepsilon} + \left( \begin{array}{c} x \\ \updownarrow \\ y \end{array} \right)$$

3. Extract the pole at  $x \rightarrow 0$  analytically, expand in  $\varepsilon$ :

$$* I = -\frac{2}{\varepsilon} \int_0^1 dy (1+y)^{-2+\varepsilon} = -\frac{2}{\varepsilon} \int_0^1 dy \left( \frac{1}{(1+y)^2} - \frac{\ln(1+y)}{(1+y)^2} \varepsilon + \mathcal{O}(\varepsilon^2) \right)$$

4. Integrate each term in  $\varepsilon$  numerically (they all converge now).

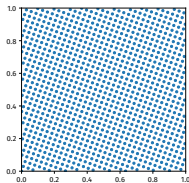
In practice: geometric sector decomposition.

[Bogner, Weinzierl '07; Kaneko, Ueda, '09]

# Randomized Quasi Monte Carlo integration

Randomized Quasi Monte-Carlo integration:

1. Let  $\vec{x}_{1\dots N}^{(k)}$  be  $K$  *low-discrepancy sequences* in  $[0;1]^n$ .
2. Estimate  $\int I(\vec{x}) d^n\vec{x} \approx \text{mean}_k \text{mean}_i I(\vec{x}_i^{(k)})$ .
3. Estimation error  $\approx \text{stdev}_k \text{mean}_i I(\vec{x}_i^{(k)}) / \sqrt{K-1}$ .



Low-discrepancy sequences:

[Dick, Kuo, Sloan '13]

- \* Digital sequences (Sobol', Faure, etc), with error  $\sim N^{-1}$ .
- \* Scrambled nets, with error  $\sim N^{-1.5}$  if  $\partial_x I$  is square-integrable.
- \* Lattice rules, with error  $\sim N^{-\alpha}$ , if  $\partial_x^{(\alpha)} I$  is square-integrable and periodic.
  - \* To enforce periodicity, use a variable transformation ( $x \rightarrow y$ ):
    - \* baker's:  $y = 1 - |2x - 1|$ ;
    - \* Korobov $_{l,r}$ :  $dy/dx \sim x^l (1-x)^r$ ; etc.

In pySECDEC: *rank-1 lattice rules*:  $\vec{x}_i^{(k)} = \left( \frac{i \cdot \vec{g}_N}{N} + \overrightarrow{\text{random}}_k \right) \bmod 1$ .

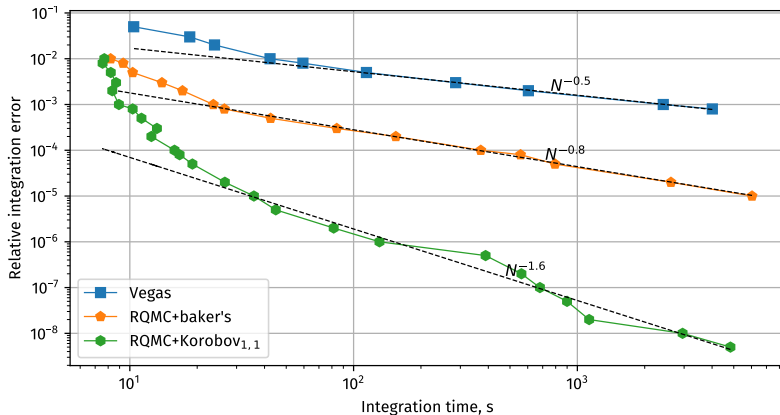
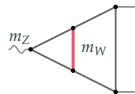
- \* Generating vectors  $\vec{g}_N$  constructed online via “median QMC lattice” construction.

[Heinrich et al '23; Goda, L'Ecuyer '22]



# Monte Carlo vs RQMC

Integration time scaling for Monte Carlo (VEGAS)  
vs Randomized Quasi Monte Carlo (QMC).<sup>2</sup>



# Dealing with large cancellations

Large cancellations in parts of the high-energy region, e.g.:

$$\begin{aligned} \mathcal{E} = & 10^{29} \text{ (diagram)} + 10^{29} \text{ (diagram)} \\ & + 10^{24} \text{ (diagram)} + 10^{24} \text{ (diagram)} + 10^{24} \text{ (diagram)} \\ & + 10^{19} \text{ (diagram)} + 10^{19} \text{ (diagram)} + 10^{18} \text{ (diagram)} \\ & + \dots \approx 10^{-3} \end{aligned}$$

- \* Knowing the integrals at full double precision (16 digits) is not enough!
  - \* The cancelling integrals converge well with QMC.
    - \* The precision is limited by the use of double floats more than convergence.
- ⇒ Make pySECDEC use *double-double* (32 digits) for integrals that need it:
- \* *20+ digits of precision* for 4-propagator integrals reachable;
  - \* custom implementation for CPUs and GPUs;
  - \* around 20x performance hit compared to doubles.

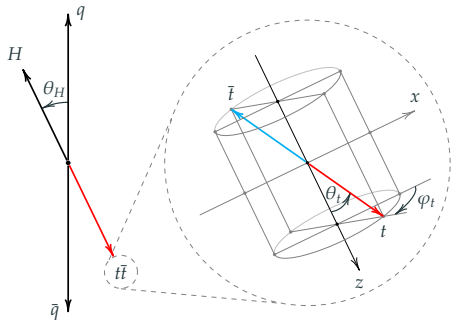
Results for  $q\bar{q} \rightarrow t\bar{t}H$

# Phase-space parameters

We parameterize the  $q\bar{q} \rightarrow t\bar{t}H$  phase space as chained decay, and instead of

$$s = (p_q + p_{\bar{q}})^2 \in [(2m_t + m_H)^2; \infty],$$
$$s_{t\bar{t}} = (p_t + p_{\bar{t}})^2 \in [(2m_t)^2; (\sqrt{s} - m_H)^2 - (2m_t)^2],$$

introduce:



$$\beta^2 \equiv 1 - \frac{s_{\min}}{s} \in [0; 1],$$

$$\text{frac}_{s_{t\bar{t}}} \equiv \frac{s_{t\bar{t}} - s_{t\bar{t},\min}}{s_{t\bar{t},\max} - s_{t\bar{t},\min}} \in [0; 1],$$

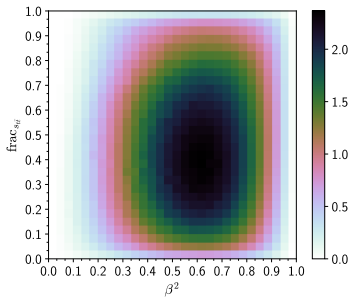
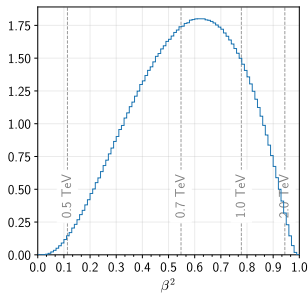
$$\theta_H \in [0; \pi],$$

$$\theta_t \in [0; \pi],$$

$$\varphi_t \in [0; 2\pi].$$

# Which parts of the phase-space are relevant?

Event density at the LHC according to the tree-level amplitude:



To cover 90% of events:  $\beta^2 \in [0.24, 0.88]$ , that is  $\sqrt{s} \in [540 \text{ GeV}, 1.4 \text{ TeV}]$ .

\* \* \*

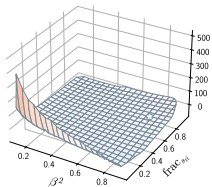
Example results as two-dimensional slices around the center point of:

$$\begin{aligned} \beta^2 &= 0.8, & \text{frac}_{S_{Ht}} &= 0.7, \\ \cos \theta_H &= 0.8, & \cos \theta_t &= 0.9, & \cos \varphi_t &= 0.7, \\ m_H^2 &= 12/23 m_t^2, & \mu &= s/2. \end{aligned}$$

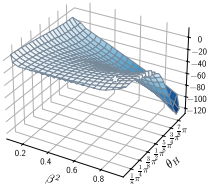
# Resulting slices in $\beta^2$ and $\text{frac}_{s_{\overline{H}}}$ , $\theta_H$ , $\theta_t$ , $\varphi_t$

$N_f$  part of the two-loop amplitude (*our result*):

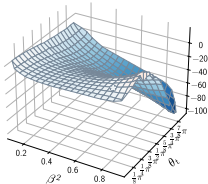
$C/A$



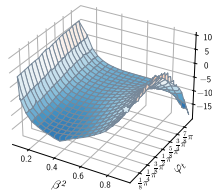
$C/A$



$C/A$

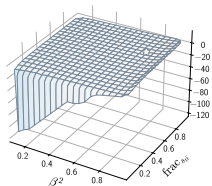


$C/A$

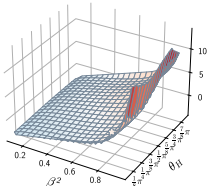


One-loop amplitude (*already known*):

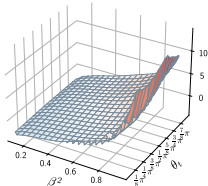
$B/A$



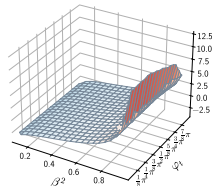
$B/A$



$B/A$

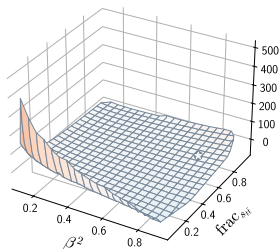


$B/A$

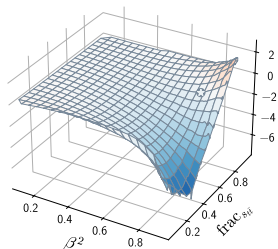


# Resulting slices in $\beta^2$ and $\text{frac}_{s\bar{f}}$

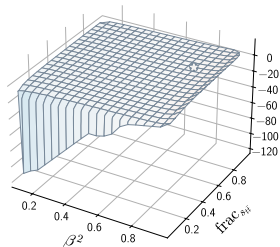
$C/A$



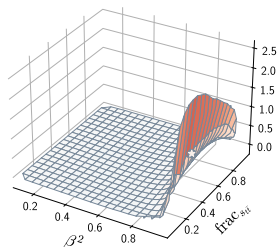
$C \times (\text{phase-space density}) \times 10^3$



$B/A$

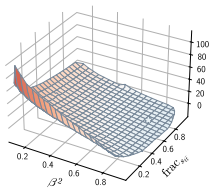


$B \times (\text{phase-space density}) \times 10^3$

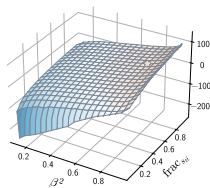


# Resulting slices in $\beta^2$ and $\text{frac}_{s\bar{f}\bar{f}}$ by color factor

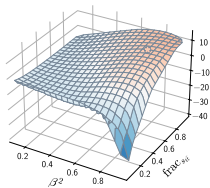
$C_{hC_A}/A$



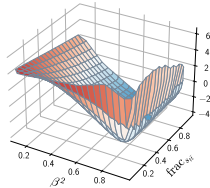
$C_{hC_F}/A$



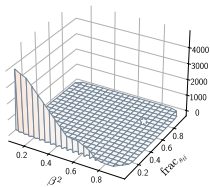
$C_{hd_{33}}/A$



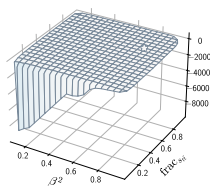
$C_{hh}/A$



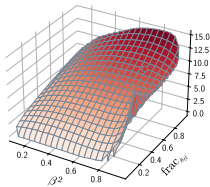
$C_{lC_A}/A$



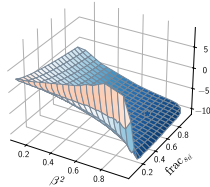
$C_{lC_F}/A$



$C_{ld_{33}}/A$



$C_{lh}/A$





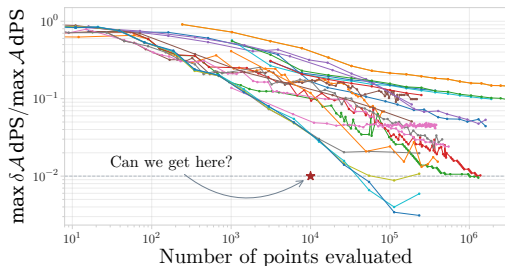
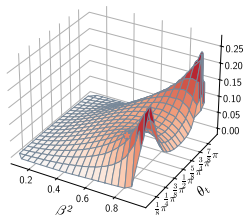
# How to use the results?

Goal: precompute points on a 5-dimensional grid, interpolate in between.

- \* How few points do we need to evaluate for 1% approximation error?
- \* Which interpolation method fits best?
  - \* Splines, polynomials, rationals, sparse grids, radial basis functions, low-rank decompositions, neural networks?
- \* At which points to sample?
  - \* Random unweighted samples, RAMBO samples, regular grids, sparse grids, lattices, Padua points, Fekete points, locally adaptive points?

Maximal approximation error of  $\mathcal{A}$  by various methods (work in progress):

$\mathcal{A} \times (\text{phase-space density}) \times 10^3$



# Summary & Outlook

## Done:

- \* IBP performance improvements with RATRACER.
- \* Performance and precision improvements in pySECDEC.
- \*  $N_f$ -part of the two-loop virtual amplitude for  $q\bar{q} \rightarrow t\bar{t}H$ .

## In progress:

- \* The rest of the two-loop virtual amplitude for  $q\bar{q} \rightarrow t\bar{t}H$ .
- \* Interpolation for the results.

## Future plans:

- \* Two-loop virtual amplitude for  $gg \rightarrow t\bar{t}H$ .
- \* Combination with real radiation.
- \* Phenomenological applications.

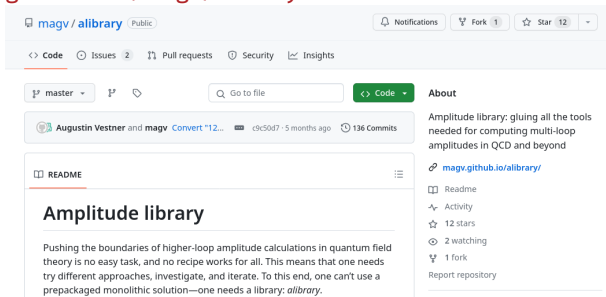
# Backup slides

# Amplitude library

Most of this work is glued by ALIBRARY (“*Amplitude Library*”). It provides functions and interfaces to tools for multiloop calculations in Mathematica:

- \* Diagram generation and visualization (QGRAF, GRAPHVIZ, TIKZ).
- \* Feynman rule insertion.
- \* Tensor trace summation (FORM, COLOR.H).
- \* Integral symmetries, IBP families (FEYNSON).
- \* Export to/from IBP solvers (KIRA, FIRE+LITERED).
- \* Export to/from pySECDEC.

[github.com/magv/alibrary](https://github.com/magv/alibrary)



magv / alibrary Public

Notifications Fork 1 Star 12

<> Code Issues 2 Pull requests Security Insights

master Go to file Code

Augustin Vestner and magv Convert \*12... c9c50d7 · 5 months ago 136 Commits

README

## Amplitude library

Pushing the boundaries of higher-loop amplitude calculations in quantum field theory is no easy task, and no recipe works for all. This means that one needs try different approaches, investigate, and iterate. To this end, one can't use a prepackaged monolithic solution—one needs a library: *alibrary*.

About

Amplitude library: gluing all the tools needed for computing multi-loop amplitudes in QCD and beyond

magv.github.io/alibrary/

Readme

Activity

12 stars

2 watching

1 fork

Report repository

# RATRACER trace optimizations

Given a trace, RATRACER can optimize it using:

\* *Constant propagation:*

$$\begin{cases} t11 = \text{int } 2 \\ t12 = \text{int } 3 \\ t13 = \text{mul } t11 \ t12 \end{cases} \Rightarrow \begin{cases} t11 = \text{int } 2 \\ t12 = \text{int } 3 \\ t13 = \text{int } 6 \end{cases}$$

\* *Trivial operation simplification:*

$$\begin{cases} t11 = \text{int } -1 \\ t12 = \text{mul } t11 \ t7 \end{cases} \Rightarrow \begin{cases} t11 = \text{int } -1 \\ t12 = \text{neg } t7 \end{cases}$$

\* *Common subexpression elimination:*

$$\begin{cases} t11 = \text{add } t5 \ t7 \\ t12 = \text{add } t5 \ t7 \end{cases} \Rightarrow \begin{cases} t11 = \text{add } t5 \ t7 \\ t12 = t11 \end{cases}$$

\* *Dead code elimination:*

$$\begin{cases} t11 = \text{add } t5 \ t7 \\ [\dots, t11 \text{ is unused}] \end{cases} \Rightarrow \begin{cases} \text{nop} \\ [\dots] \end{cases}$$

\* Especially useful if a user wants to select a subset of the outputs.

\* *“Finalization”:*

$$\begin{cases} t11 = \text{add } t5 \ t6 \\ t12 = \text{add } t11 \ t7 \\ [\dots, t11 \text{ is unused}] \end{cases} \Rightarrow \begin{cases} t11 = \text{add } t5 \ t6 \\ t11 = \text{add } t11 \ t7 \\ [\dots] \end{cases}$$

\* Needed to minimize the temporary memory needed for the evaluation.

# RATRACER usage for IBP reduction

## 1. Use KIRA to generate the IBP equations.

```
$ cat >config/integralfamilies.yaml <<EOF
integralfamilies:
  - name: "I"
    loop_momenta: [1]
    top_level_sectors: [b111]
    propagators:
      - ["1", 0]
      - ["1-p1", 0]
      - ["1+p2", 0]
```

EOF

```
$ cat >config/kinematics.yaml <<EOF
```

```
kinematics:
  outgoing_momenta: [p1, p2]
  kinematic_invariants: [[s, 2]]
  scalarproduct_rules:
    - [[p1,p1], 0]
    - [[p2,p2], 0]
    - [[p1,p2], "s/2"]
```

```
# symbol_to_replace_by_one: s
```

EOF

```
$ cat >export-equations.yaml <<EOF
jobs:
```

```
  - reduce_sectors:
      reduce:
        - {sectors: [b111], r: 4, s: 1}
      select_integrals:
        select_mandatory_recursively:
          - {sectors: [b111], r: 4, s: 1}
      run_symmetries: true
      run_initiate: input
```

EOF

```
$ kira export-equations.yaml
```

## 2. Use RATRACER to create a trace with the solution.

```
$ ratracer \
  load-equations input_kira/I/SYSTEM_I_0000000007.kira.gz \
  load-equations input_kira/I/SYSTEM_I_0000000006.kira.gz \
  solve-equations choose-equation-outputs --maxr=4 --maxs=1 \
  optimize finalize save-trace I.trace.gz
```

## 3. Optionally expand the outputs into a series in $\epsilon$ .

```
$ ratracer \
  set d '4-2*eps' load-trace I.trace.gz \
  to-series eps 0 \
  optimize finalize save-trace I.eps0.trace.gz
```

## 4. Use RATRACER (+FIREFLY) to reconstruct the solution.

```
$ ratracer \
  load-trace I.eps0.trace.gz \
  reconstruct --to=I.solution.txt --threads=8 --inmem
```

# RATRACER usage as a library

RATRACER is built to support custom user-defined traces.

*Any rational algorithm* can be turned into a trace (via the C++ API).

Usage:

```
#include <ratracer.h>
int main() {
    Tracer tr = tracer_init();

    Value x = tr.var(tr.input("x"));
    Value y = tr.var(tr.input("y"));

    Value x_sqr =
        tr.pow(x, 2);
    Value expr =
        tr.add(x_sqr, tr.mulint(y, 3));

    /* expr = x^2 + 3y */
    tr.add_output(expr, "expr");

    tr.save("example.trace.gz");
    return 0;
}
```

API:

```
struct Value { uint64_t id; uint64_t val; };
struct Tracer {
    Value var(size_t idx);
    Value of_int(int64_t x);
    Value of_fmpz(const fmpz_t x);
    bool is_zero(const Value &a);
    bool is_minus1(const Value &a);
    Value mul(const Value &a, const Value &b);
    Value mulint(const Value &a, int64_t b);
    Value add(const Value &a, const Value &b);
    Value addint(const Value &a, int64_t b);
    Value sub(const Value &a, const Value &b);
    Value addmul(const Value &a,
                const Value &b1,
                const Value &b2);
    Value inv(const Value &a);
    Value neginv(const Value &a);
    Value neg(const Value &a);
    Value pow(const Value &base, long exp);
    Value div(const Value &a, const Value &b);
    void assert_int(const Value &a, int64_t n);
    void add_output(const Value &src, const char *name);
    size_t input(const char *name, size_t len);
    size_t input(const char *name);
    int save(const char *path);
    void clear();
};
Tracer tracer_init();
```

# pySECDEC contour deformation

$$I \equiv \int d^n \vec{x} \frac{U^\alpha(\vec{x})}{F^\beta(\vec{x}, \dots) + i0}$$

What to do if  $F(\vec{x}) = 0$  inside the integration region?

⇒ Deform  $\vec{x}$  into the complex plane!

$$\vec{x} \rightarrow \vec{x} + i \vec{\delta}(\vec{x}),$$

$$F \rightarrow F + i \delta \partial F - \delta^2 \partial^2 F - i \delta^3 \partial^3 F + \dots$$

Choose  $\vec{\delta}(\vec{x})$  to enforce the  $+i0$  prescription:

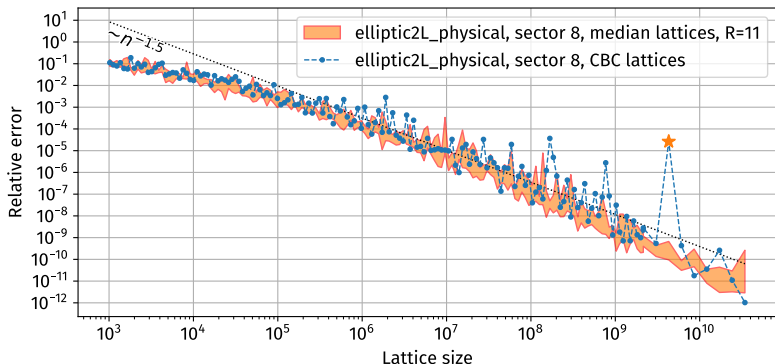
$$\text{Im} F = \delta \partial F - \delta^3 \partial^3 F + \dots \stackrel{\delta \rightarrow 0}{\approx} \delta \partial F > 0 \quad \Rightarrow \quad \vec{\delta}(\vec{x}) = \lambda \vec{\partial} F(\vec{x}).$$

- \* There is always  $\lambda$  small enough that  $\delta \partial F > \delta^3 \partial^3 F$ , and  $\text{Im} F > 0$ .
- \* The larger  $\lambda$  is, the further the pole is, the better the convergence is.
- \* In practice: choose  $\lambda$  heuristically, but decrease it if  $\text{Im} F(\vec{x}_i) < 0$ .
  - \* Gradient-based  $\lambda$  optimization can be useful.



# pySECDEC median QMC lattices in practice

Precision by lattice for a 2-loop massive box:



In short:

- \* median lattices are on average competitive with CBC lattices;
- \* at higher precisions the worst unlucky lattices are avoided;
- \* no limitation on the lattice size;
- \* but: need to be constructed on the fly, interleaved with integration.