Computer Physics Communications 306 (2025) 109384

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Computer Programs in Physics

ftint: Calculating gradient-flow integrals with pySecDec ☆

Robert V. Harlander [a,*], Theodoros Nellopoulos [a], Anton Olsson [b], Marius Wesle [c]

[a] TTK, RWTH Aachen University, 52056 Aachen, Germany
[b] Institute for Theoretical Physics, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
[c] Department of Mathematics, University of Tübingen, 72076 Tübingen, Germany

ARTICLE INFO

Dataset link: https://gitlab.com/ftint/ftint

Keywords:
Gradient flow
Perturbation theory
Feynman integrals

ABSTRACT

The program ftint is introduced which numerically evaluates dimensionally regularized integrals as they occur in the perturbative approach to the gradient-flow formalism in quantum field theory. It relies on sector decomposition in order to determine the coefficients of the individual orders in $\epsilon = (4 - D)/2$, where D is the space-time dimension. For that purpose, it implements an interface to the public library pySecDec. The current version works for massive and massless integrals up to three-loop level with vanishing external momenta, but the underlying method is extendable to more general cases.

Contents

☆ The review of this paper was arranged by Prof. Z. Was.

* Corresponding author.
E-mail address: robert.harlander@rwth-aachen.de (R.V. Harlander).

https://doi.org/10.1016/j.cpc.2024.109384
Received 25 July 2024; Received in revised form 6 September 2024; Accepted 17 September 2024
Available online 20 September 2024

PROGRAM SUMMARY

Program title: ftint
CPC Library link to program files: https://doi.org/10.17632/bt6bkffxh9.1
Developer's repository link: https://gitlab.com/ftint/ftint
Licensing provisions: MIT license
Programming language: Python
Supplementary material: README.md
Nature of problem: The perturbative approach to the gradient-flow formalism in quantum field theory leads to integrals which closely resemble regular Feynman integrals. However, they involve exponential factors which depend on the loop and external momenta, as well as on so-called flow-time variables. In general, the latter are also integrated over a finite interval. These integrals cannot be solved immediately with standard tools.
Solution method: The flow-time integrals are transformed to integrals over a hypercube by introducing Schwinger parameters. The latter are numerically evaluated using the public program pySecDec, which performs a sector decomposition and calculates the coefficients of the poles in the parameter $\epsilon = (4-D)/2$ by numerical integration, where D is the space-time dimension which occurs in dimensional regularization.
Additional comments including restrictions and unusual features: In its current form, ftint is restricted to one-, two-, and three-loop integrals with vanishing external momenta. It does allow for massive propagators though, raised to virtually arbitrary integer powers.

1. Introduction

The gradient-flow formalism (GFF) [1–5] is a useful tool for practical calculations in Quantum Field Theory (QFT). Its main feature is to suppress the high-momentum modes of quantum fields. In lattice Quantum Chromodynamics (QCD), this leads to a smoothing of the gauge field. Among many other applications, this allows for efficient ways to determine the lattice spacing, for example [3,6].

On the other hand, various applications of the GFF have been suggested that involve also perturbative calculations, among them the so-called short-flow-time expansion (SFTX) [4], where composite operators of flowed fields are expressed in terms of regular operators via matching coefficients which can be determined perturbatively. This approach has proven viable for evaluating matrix elements of the energy-momentum tensor in QCD, for example, or for the calculation of observables in flavor physics [7–14]. For more applications, see Refs. [15–21], for example.

The form of the integrals that occur in the perturbative approach to the GFF is remarkably close to the Feynman integrals of regular QFT. The only modifications are: (i) an exponential factor in the integrand which depends on the loop momenta, the external momenta and the masses, in general, as well as on so-called flow-time variables, and (ii) additional integrations over these flow-time variables [3,4].

It has been shown that many of the tools that have been developed for higher-order calculations in the perturbative approach to regular QFT can be applied or extended to flowed QFT [22]. In particular, this holds for the automatic generation of the associated Feynman diagrams, their simplification to scalar integrals, as well as their reduction to master integrals using integration-by-parts (IbP) relations. However, while there are powerful publicly available software tools for the numerical evaluation of the master integrals in regular QFT, this is not the case for flow-time integrals. In this paper, we will close this gap by providing the program ftint, which constitutes an interface for such integrals to pySecDec [23–25], allowing to evaluate them via the sector decomposition algorithm [26–28]. In this first version of the program we put the focus on integrals which depend only on a single mass scale, the flow-time t. This is certainly one of the most important cases, as the calculation of the matching coefficients in the SFTX leads to exactly this class of integrals [10]. However, ftint also allows for non-vanishing masses in the propagators, which may prove useful in many applications such as the calculation of mass effects to the action density $\langle G_{\mu\nu}G_{\mu\nu}\rangle$.

The structure of the remainder of this paper is as follows. After defining the problem in Section 2, we describe the method of our calculation in Section 3. The way to use the program ftint is presented in Section 4, including some examples and checks. Section 5 contains conclusions and an outlook for future work. In the appendix, we describe the possibility to adjust the input and output format of ftint.

2. General outline

We work in Euclidean space throughout this paper, unless stated otherwise. The generic form of the integrals which we will consider is [22,29]

$$f(\mathbf{c},\mathbf{a},\mathbf{b}) = (4\pi t)^{lD/2}\, t^{-b} \int\limits_{[0,1]^f} \mathrm{d}\mathbf{u}\, \mathbf{u}^{\mathbf{c}} \int\limits_{\mathbf{p}} \frac{\exp[-t\sum_{i=1}^{k} a_i P_i^2]}{(P_1^2+m_1^2)^{b_1}\cdots(P_k^2+m_k^2)^{b_k}} \tag{1}$$

where $\mathbf{p} = \{p_1,\ldots,p_l\}$ is the set of loop momenta,

$$\mathbf{b} = \{\{b_1,m_1\},\ldots,\{b_k,m_k\}\} \tag{2}$$

collects the so-called indices $b_i \in \mathbb{Z}$ and masses $m_i \in \mathbb{R}$. The prefactor t^{-b}, with $b = \sum_{i=1}^{k} b_i$, in Eq. (1) thus compensates the mass dimension of the integral, such that $f(\mathbf{c},\mathbf{a},\mathbf{b})$ is dimensionless. We furthermore define the simplified notation $\{b_i,0\} \to b_i$ for massless propagators.

The k functions $a_i \in \mathbf{a}$ are real-valued polynomials of the (dimensionless) flow-time variables $\mathbf{u} = \{u_1,\ldots,u_f\}$ that are non-negative on the hypercube $[0,1]^f$. Furthermore, we define

$$\mathbf{u}^{\mathbf{c}} := u_1^{c_1}\cdots u_f^{c_f}\,, \tag{3}$$

where $\mathbf{c} = \{c_1,\ldots,c_f\}$ is a set of non-negative integers.

The integration measure over the loop momenta is

$$\int\limits_{\mathbf{p}} := \int\limits_{p_1}\cdots\int\limits_{p_l}\,, \quad \text{with} \quad \int\limits_{p} = \int \frac{\mathrm{d}^D p}{(2\pi)^D}\,, \qquad D = 4-2\epsilon\,. \tag{4}$$

The P_i in Eq. (1) are linear combinations of the loop momenta. External momenta will be neglected in this paper. This is justified for the calculation of the perturbative matching coefficients of the SFTX to which ftint is tailored.[1] It implies that the number of propagators k in Eq. (1) is related to the number of loops l as

$$k = \begin{cases} 1 & \text{for} \quad l=1\,, \\ 3\,(l-1) & \text{for} \quad l \geq 2\,. \end{cases} \tag{5}$$

In principle, there could be propagator factors in Eq. (1) which differ only in their mass term, i.e., $P_i = P_j$ but $m_i \neq m_j$. However, using partial fractioning, these can always be re-written to integrals where $m_i = m_j$ if $P_i = P_j$.

For later purposes, it is convenient to define a symmetric square matrix $A(\mathbf{a})$ through the condition

$$\mathbf{p}^T A(\mathbf{a})\mathbf{p} = \sum_{i=1}^{k} a_i P_i^2\,, \tag{6}$$

where $\mathbf{p}$ are the loop momenta. Its explicit form depends on the choice of linear combinations P_i. To be specific, we choose

one-loop:

$$P_1 = p_1\,, \qquad \Rightarrow\ A(\mathbf{a}) = a_1\,. \tag{7a}$$

[1] The matching coefficients are most conveniently determined by using the method of projectors [30,31], which results in integrals whose only dimensional scale is the external flow-time t, see Ref. [10], for example.

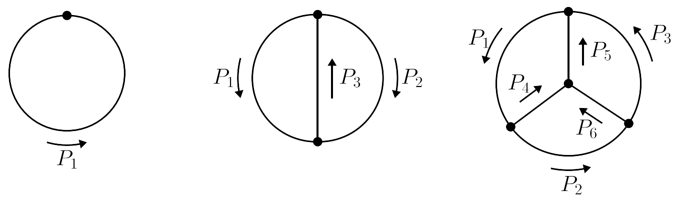

Fig. 1. One-, two-, and three-loop topologies of the integrals considered in this paper. All Feynman diagrams are produced with the help of `FeynGame` [32,33].

two-loop:

$$P_1 = p_1\,, \qquad P_2 = p_2\,, \qquad P_3 = p_1 + p_2\,, \tag{8a}$$

$$\Rightarrow\; A(\mathbf{a}) = \begin{pmatrix} a_1 + a_3 & a_3 \\ a_3 & a_2 + a_3 \end{pmatrix}. \tag{8b}$$

three-loop:

$$\begin{aligned} &P_1 = p_1\,, && P_2 = p_2\,, && P_3 = p_3\,, \\ &P_4 = p_1 - p_2\,, && P_5 = p_1 - p_3\,, && P_6 = p_2 - p_3\,, \end{aligned} \tag{9a}$$

$$\Rightarrow\; A(\mathbf{a}) = \begin{pmatrix} a_1 + a_4 + a_5 & -a_4 & -a_5 \\ -a_4 & a_2 + a_4 + a_6 & -a_6 \\ -a_5 & -a_6 & a_3 + a_5 + a_6 \end{pmatrix}. \tag{9b}$$

This choice defines the so-called integral topologies shown in Fig. 1. Note that there is only a single topology at each loop order up to the three-loop level. This pattern does not extend to higher orders though.

3. Calculation of the integrals

The method we pursue for the evaluation of the integrals closely follows Ref. [29]. However, while this paper employed the public software tools `FIESTA` [34–37] and the `MPFR` integration library,[2] we use `pySecDec` for the main calculational steps.

3.1. One loop

In order to describe the method that we apply for the calculation of the integrals, it is instructive to consider the one-loop level first. This is a particularly simple case which, however, already introduces the main concepts. The formal step to a general number of loops can then be achieved rather easily. Furthermore, it is useful to neglect all masses in a first outline. It turns out that their inclusion is almost trivial in the approach described below.

3.1.1. Massless case

At the one-loop level and neglecting the masses, Eq. (1) takes the form

$$f(\mathbf{c},\{a\},\{b\}) = (4\pi t)^{D/2}\, t^{-b} \int\limits_{[0,1]^f} \mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}} \int\limits_p \frac{\exp[-tap^2]}{(p^2)^b}\,. \tag{10}$$

It is useful to distinguish three cases.

Vanishing index. For $b = 0$, evaluating the integral over p gives

$$f(\mathbf{c},\{a\},\{0\}) = \int\limits_{[0,1]^f} \mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}}\,[a]^{-D/2}\,. \tag{11}$$

Recall that a is a polynomial in the flow-time variables u_i. This can be passed directly to `pySecDec` for numerical integration, see Section 3.4.

Positive index. For $b > 0$, we apply Schwinger parameters to make the p integral Gaußian:

$$\begin{aligned} f(\mathbf{c},\{a\},\{b\}) &= \frac{(4\pi t)^{D/2}}{(b-1)!} \int\limits_0^\infty \mathrm{d}x\, x^{b-1} \int\limits_{[0,1]^f} \mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}} \int\limits_p \exp\{-t[a+x]p^2\} = \\ &= \frac{1}{(b-1)!} \int\limits_0^\infty \mathrm{d}x\, x^{b-1} \int\limits_{[0,1]^f} \mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}}\,[a+x]^{-D/2}\,. \end{aligned} \tag{12}$$

In order to evaluate the integral over Schwinger parameters with `pySecDec`, we map them to the interval $[0,1]$. Unfortunately, the mapping

$$x \to \frac{1-x}{x} \tag{13}$$

introduces singularities at $x = 1$, while `pySecDec` expects them to occur only at $x = 0$. Therefore, we split the integration intervals of the Schwinger parameters as $[0,\infty) = [0,1] \cup (1,\infty)$, and map $x \to 1/x$ in the second interval, leading to

$$\begin{aligned} f(\mathbf{c},\{a\},\{b\}) &= \frac{1}{(b-1)!} \int\limits_{[0,1]^f} \mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}} \\ &\times \int\limits_0^1 \mathrm{d}x \left(x^{b-1}\,[a+x]^{-D/2} + x^{D/2-1-b}\,[x\,a+1]^{-D/2}\right). \end{aligned} \tag{14}$$

`pySecDec` can now treat the x and $\mathbf{u}$ integration on the same footing.

Negative index. In the case $b < 0$, we can use

$$(p^2)^{-b} = (-t)^b \frac{\partial^{-b}}{\partial x^{-b}} e^{-xtp^2}\bigg|_{x=0} \tag{15}$$

and obtain

$$\begin{aligned} f(\mathbf{c},\{a\},\{b\}) &= (-1)^b \frac{\partial^{-b}}{\partial x^{-b}} \int\limits_{[0,1]^f} \mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}}[a+x]^{-D/2}\bigg|_{x=0} \\ &= P(-b)\,(-1)^b \int\limits_{[0,1]^f} \mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}}[a]^{-D/2+b}\,, \end{aligned} \tag{16}$$

where

$$P(m) = \prod_{k=1}^{m} (1-k-D/2) = \frac{\Gamma(1-D/2)}{\Gamma(1-D/2-m)}\,. \tag{17}$$

This expression can again be directly passed to `pySecDec`.

Let us look at a few simple examples which can be calculated analytically:

$$\begin{aligned} f(\{\},\{1\},\{0\}) &= 1\,, \\ f(\{\},\{a\},\{n\}) &= (4\pi t)^{D/2}\, t^{-n} \int\limits_p \frac{\exp[-tap^2]}{(p^2)^n} = \\ &= \frac{a^{n-D/2}}{(n-1)!} \int\limits_0^\infty \mathrm{d}x\, x^{n-1} (1+x)^{-D/2} = \\ &= \frac{a^{n-D/2}}{(n-1)!} \frac{\Gamma(n)\Gamma(D/2-n)}{\Gamma(D/2)}\,, \qquad \text{for} \quad n \geq 1\,, a \in \mathbb{R}\,, \end{aligned} \tag{18}$$

$$\begin{aligned} f(\{\},\{a\},\{-n\}) &= (4\pi t)^{D/2}\, t^n \int\limits_p p^{2n} \exp[-tap^2] = \\ &= (4\pi t)^{D/2} (-a)^{-n} \frac{\partial^n}{\partial x^n} \int\limits_p \exp[-ta(1+x)p^2]\bigg|_{x=0} = \end{aligned}$$

[2] See http://www.holoborodko.com/pavel/mpfr/ and https://www.mpfr.org/.

$$= (-1)^n a^{-n-D/2} \frac{\partial^n}{\partial x^n}(1+x)^{-D/2}\bigg|_{x=0},$$

from which it follows that

$$f(\{\},\{1\},\{1\}) = \frac{1}{D/2-1} = 1+\epsilon+\epsilon^2+\cdots,$$
$$f(\{\},\{1\},\{2\}) = -\frac{1}{\epsilon}\left(1+\epsilon+\epsilon^2+\dots\right),$$
$$f(\{0\},\{u_1\},\{1\}) = (4\pi t)^{D/2}t^{-1}\int_0^1 \mathrm{d}u_1 \int_p \frac{\exp[-tu_1p^2]}{p^2} = \tag{19}$$
$$= -(4\pi t)^{D/2}t^{-2}\int_p \frac{\exp[-tp^2]}{(p^2)^2} = -f(\{0\},\{1\},\{2\}),$$
$$f(\{0\},\{u_1\},\{-1\}) = -1,$$

for example, where we have used the fact that scaleless integrals are zero in dimensional regularization.

3.1.2. Massive propagators

We may generalize the integral in Eq. (10) by allowing for massive propagators. Specifically, we consider the integral

$$f(\mathbf{c},\{a\},\{\{b,m\}\}) \equiv (4\pi t)^{D/2}t^{-b}\int_{[0,1]^f}\mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}}\int_p \frac{\exp[-tap^2]}{(p^2+m^2)^b}. \tag{20}$$

Note that we only need to consider positive b, because the other cases can be algebraically reduced to already known integrals via the binomial formula:

$$(p^2+m^2)^{-b} = \sum_{n=0}^{-b}\binom{-b}{n}(p^2)^n(m^2)^{-b-n} \qquad \text{for } b\le 0. \tag{21}$$

Following the same steps as above for $b>0$, on the other hand, we arrive at

$$f(\mathbf{c},\{a\},\{\{b,m\}\}) =$$
$$= \frac{1}{(b-1)!}\int_0^\infty \mathrm{d}x\, x^{b-1}e^{-xtm^2}\int_{[0,1]^f}\mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}}\,[a+x]^{-D/2}. \tag{22}$$

We again split the integration interval into $x\in[0,1]\cup(1,\infty)$ and perform the substitution $x\to 1/x$ in the second interval. This leads to a singularity at $x=0$ in the argument of the exponential, which is spurious, however, as the exponential vanishes at this point. Therefore, the only effect on the integrals due to propagators being massive is the multiplication of a positive function. Since it is factorized and does not contribute to the true singularity structure, it has no impact on pole extraction during sector decomposition. Section 3.4 describes how such factors are treated in pySecDec.

As mentioned above, the more general case of several masses,

$$\int_{[0,1]^f}\mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}}\int_p \frac{\exp[-tap^2]}{(p^2+m_1^2)^{b_1}\cdots(p^2+m_n^2)^{b_n}} \tag{23}$$

can be reduced to integrals of the form Eq. (22) by partial fractioning.

Let us consider a particularly simple example for a massive one-loop integral which can be solved analytically:

$$f(\{\},\{1\},\{\{1,1\}\}) = 1 - ze^z\Gamma(0,z), \qquad z=m^2t, \tag{24}$$

where $\Gamma(n,z)=\int_z^\infty \mathrm{d}x\,x^{n-1}e^{-x}$ is the incomplete Γ function.

3.2. Higher orders

Let us now move on to the multi-loop level, first focusing on the massless case. It is helpful to define the auxiliary function

$$F(\mathbf{a},\mathbf{x}) = (4\pi t)^{lD/2}\int_{\mathbf{p}}\exp\left\{-t\sum_{i=1}^k[a_i+x_i]P_i^2\right\}. \tag{25}$$

Using Eq. (6), one can perform the Gaußian integral over the loop momenta to obtain

$$F(\mathbf{a},\mathbf{x}) = [\det A(\mathbf{a}+\mathbf{x})]^{-D/2}. \tag{26}$$

We refer to the $\mathbf{x}=\{x_1,\dots,x_k\}$ as Schwinger parameters in the following, even if they are not integrated over. It is helpful to note that each $x_i\in\mathbf{x}$ occurs only linearly in $\det A(\mathbf{a}+\mathbf{x})$.

Consider now a flow-time integral with indices $b_1,\dots,b_k$, which we divide up as follows:

$$b_i>0 \quad \text{for} \quad i\in I_{\text{int}},$$
$$b_i<0 \quad \text{for} \quad i\in I_{\text{diff}}, \tag{27}$$
$$b_i=0 \quad \text{for} \quad i\in I_0.$$

For the vanishing indices $b_i=0$, we can simply set the corresponding Schwinger parameters to zero in Eq. (26), $x_i=0$. For the negative indices $b_i<0$, on the other hand, we use Eq. (15), meaning that we need to take the derivative w.r.t. $-x_i$ at $x_i=0$. Note that n derivatives acting on Eq. (26) produce n terms of the form

$$[\det A(\mathbf{a}+\mathbf{x})]^{-D/2-k}\,g_k(\mathbf{a},\mathbf{x},D), \qquad k\in\{1,\dots,n\}, \tag{28}$$

where g_k is polynomial in its arguments at most of order x_i^k for each x_i. Finally, for the positive indices $b_i>0$, we integrate over x_i and multiply by $1/(b_i-1)!$. In summary,

$$f(\mathbf{c},\mathbf{a},\mathbf{b}) = \left[\prod_{j\in I_{\text{int}}}\frac{1}{(b_j-1)!}\int_0^\infty \mathrm{d}x_j\,x_j^{b_j-1}\right]\times$$
$$\times\int_{[0,1]^f}\mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}}\left[\left(\prod_{i\in I_{\text{diff}}}\frac{\partial^{-b_i}}{\partial(-x_i)^{-b_i}}\right)[\det A(\mathbf{a}+\mathbf{x})]^{-D/2}\right]_{x_k=0 \text{ for } k\in(I_{\text{diff}}\cup I_0)}. \tag{29}$$

Again, we split the integration region for the Schwinger parameters into $x_j\in[0,1]\cup(1,\infty]$. The integrand in Eq. (29) thus consists of polynomials of the x_i with $i\in I_{\text{int}}$ and the $\mathbf{u}$, raised to non-integer powers which can be passed to pySecDec for integration.

Non-vanishing masses $\mathbf{m}$ can be taken into account in a straightforward way. As pointed out above, only massive propagators with positive indices need to be considered, because non-positive indices can be reduced algebraically to known integrals. Also, we can assume that each independent momentum is associated with only a single mass, which can always be achieved by partial fractioning as pointed out in the one-loop case. In this case, the only modification is to include a factor

$$\exp\left(-t\sum_{j\in I_{\text{int}}}x_jm_j^2\right) \tag{30}$$

in the integrand of Eq. (29).

3.3. Symmetries of the flow-time integrals

The representation of a flow-time integral in the form of Eq. (1) is not unique. The integrals remain invariant under certain combined permutations of the parameters $\mathbf{c}$, $\mathbf{a}$, $\mathbf{u}$, and $\mathbf{b}$. Employing such symmetries may significantly reduce the number of integrals that need to be evaluated. ftint provides an option to map any flow-time integral to a standard form which we refer to as *normal form*. This section briefly describes our basic strategy to determine the normal form.

The first symmetry we employ corresponds to re-naming the flow-time integration variables. In general, the integral is preserved when permuting the variables $\mathbf{u}$ in the polynomials $\mathbf{a}(\mathbf{u})$ and applying the inverse permutation to $\mathbf{c}$, i.e.

$$f(\mathbf{c}, \mathbf{a}(\mathbf{u}), \mathbf{b}) = f(p\mathbf{c}, \mathbf{a}(p^{-1}\mathbf{u}), \mathbf{b}), \tag{31}$$

where p denotes a permutation and p^{-1} its inverse. For example, one arrives at the identity

$$f(\{0,1\}, \{u_1u_2, u_2, u_1\}, \{3,1,2\}) = f(\{1,0\}, \{u_1u_2, u_1, u_2\}, \{3,1,2\}) \tag{32}$$

by just interchanging the names of u_1 and u_2.

The second symmetry is related to permutations of the momenta P_i (modulo signs) which leave the topologies in Fig. 1 (i.e., the momentum conservation relations) invariant. Since each line of these topologies corresponds to a momentum P_i, an index b_i and a polynomial a_i, permutations of the P_i correspond to simultaneous permutations of the b_i and the a_i. In the one-loop case, there is only a single line and thus no additional symmetry results from these considerations. The two-loop topology, on the other hand, is symmetric under any permutation of lines, and thus any simultaneous permutations of $\mathbf{a}$ and $\mathbf{b}$. For example,

$$f(\{1,0\}, \{u_1u_2, u_1, u_2\}, \{3,1,2\}) = f(\{1,0\}, \{u_1u_2, u_2, u_1\}, \{3,2,1\}). \tag{33}$$

At three-loop level, there are $4! = 24$ permutations of the P_i that preserve the topology shown in Fig. 1 (corresponding to the permutations of the four vertices). For example,

$$(P_1, P_2, P_3, P_4, P_5, P_6) \to (-P_3, -P_2, -P_1, P_6, P_5, P_4) \tag{34}$$

corresponds to the mirror symmetry along the vertical axis, while

$$(P_1, P_2, P_3, P_4, P_5, P_6) \to (-P_3, P_6, P_5, -P_2, -P_1, -P_4) \tag{35}$$

implies a non-trivial continuous deformation of the diagram. Each such transformation results in a permutation p, for which it holds that[3]

$$f(\mathbf{c}, \mathbf{a}, \mathbf{b}) = f(\mathbf{c}, p\mathbf{a}, p\mathbf{b}). \tag{36}$$

Using these symmetries, we can map any flow-time integral I onto an equivalent standard form. To achieve this, we generate a list of equivalent integrals by applying all combinations of permutations from each of the two symmetries to I. This list then gets sorted according to a lexicographical criterion, and the first element is defined to be the normal form of I. Our procedure guarantees that two flow-time integrals can be transformed into one another by the symmetry operations described above if and only if they have the same normal form; they will then obviously integrate to the same result. Note that the converse is not true: Integrals with different normal forms cannot be transformed into one another by the discussed symmetry operations; however, they may still integrate to the same result.

A specific example and how to use `ftint` in order to map an integral to its normal form will be discussed in Section 4.4.

3.4. Implementation with `pySecDec`

`pySecDec` [23–25] is a toolbox for the evaluation of dimensionally regularized parameter integrals. It utilizes the sector decomposition algorithm to isolate and subtract overlapping endpoint singularities, and produces an integration library to evaluate the coefficients of an expansion in the dimensional regulator. The parameter integrals `pySecDec` targets are of the form

$$I = \int_{[0,1]^d} \mathrm{d}\mathbf{x}\, f_1^{\alpha_1}(\mathbf{x}) \cdots f_k^{\alpha_k}(\mathbf{x}), \tag{37}$$

where the f_l are functions of the parameters and the α_l are linear in the space-time dimension D. These integrals are divergent in general, but can be evaluated in dimensional regularization by taking $D = 4 - 2\epsilon$ and extracting the poles in a Laurent series in ϵ. Performing such an expansion requires defining adequate subtraction terms, which can be highly non-trivial for integrands with nested singularity structures. The sector decomposition approach offers an algorithmic procedure of decomposing the integral into sectors with factorized singularity structures, where it is straightforward to define subtractions [26–28]. `pySecDec` provides implementations of several decomposition algorithms, based on either iterative or geometric strategies. The interface in `ftint` uses geometric sector decomposition as it usually leads to fewer sectors than the iterative approaches [38,39]. After sector decomposition, the integral is represented as a sum of Laurent series in α sectors

$$I = \sum_{l=1}^{\alpha} \sum_{n=-r}^{p} I_{l,n} \frac{1}{\epsilon^n} + \mathcal{O}(\epsilon^{r+1}), \tag{38}$$

where p is the order of the highest pole and the expansion coefficients $I_{l,n}$ are sector integrals that are finite at the integration boundaries. A simple example of a sector integral $I_{j,0}$ making up the finite part of the expansion in a sector j where a logarithmic divergence in x_1 has been extracted is

$$I_{j,0} = \int_{[0,1]^d} \mathrm{d}\mathbf{x}\, x_1^{-1-\epsilon} \left[\mathcal{I}(x_1, \ldots, x_d) - \mathcal{I}(x_1 = 0, \ldots, x_d) \right]. \tag{39}$$

The subtraction term $\mathcal{I}(x_1 = 0, \ldots, x_d)$ ensures that the sector integral is finite as $x_1 \to 0$. For more severe divergences, the power of the extracted pole is raised through a number of integration-by-parts iterations until there are only logarithmic divergences remaining. In this form, the sector integrals are well suited for numerical integration. The latest release of `pySecDec` [25] introduced `Disteval`, a new integration library. It implements a quasi-Monte Carlo integrator which has yielded significant performance increases compared to previous versions. The integration interface in `ftint` exclusively uses the `Disteval` integrator as it supersedes all older integrators.

The massive flow-time integrals described in Section 3.1.2 include factors $e^{-tm_j^2 x_j}$. After the substitution $x \to 1/x$ in the second interval (to map $(1,\infty) \to (0,1)$) they transform into $e^{-tm_j^2/x_j}$. The $1/x$ pole in the argument is spurious and does not need to be extracted through sector decomposition. To minimize the work of the decomposition algorithm, `pySecDec` allows for the definition of finite functions that only enter at subtraction level. Factors like these effectively scale the magnitude of the integrand, which means they still need to be included in subtraction terms such as $\mathcal{I}(x_1 = 0, \ldots, x_d)$ in Eq. (39). In order to avoid practical issues of having a spurious $1/x$ pole in the exponent, a regulator δ is added to the transformed exponential functions such that

$$e^{-tm_j^2/x_j} \to e^{-tm_j^2/(x_j+\delta)}. \tag{40}$$

By default, δ is set to 10^{-10} at integration. In order to ensure that this parameter does not affect the accuracy of the integration result, the user may change it using the option `--delta`, see Section 4.2. Since the exponential function does not affect sector decomposition, this additional parameter has virtually no impact on performance.

4. Using `ftint`

Following the spirit of `pySecDec`, the evaluation of an integral with `ftint` is divided into two steps. In the first step, the integral is decomposed into sectors where the boundary singularities have been isolated and subtracted with the help of `pySecDec`. It creates and compiles a `C++` integration library which is used in the second step to numerically evaluate the integrals. The motivation for splitting the program into two parts is that the user may want to perform the second step several times, for example with different target errors or for several mass parameters. Since this only affects the numerical integration, one can save on com-

[3] Since only the squares of the P_i enter the integral, sign changes of the P_i like in Eqs. (34) and (35) do not matter here.

puting time for the decomposition and compilation step which needs to be done only once in this case.

It is important to note that the current version of `ftint` is limited to evaluating integrals of the form specified in Eq. (1). Specifically, this implies that all external momenta must be zero, all massive propagators must have a positive index b, and no two propagators may carry the same momentum while having different masses. As previously mentioned, the latter two constraints do not really represent limitations, as any integral that does not meet these criteria can be algebraically transformed into a linear combination of integrals that do. While future versions of `ftint` may include built-in support for these transformations, the current version requires the user to apply these modifications before passing them to `ftint`. This can be readily accomplished using built-in functions in `Mathematica` [40] or `FORM` [41,42], for example, or via the `Python` library `SymPy`.

4.1. Sector decomposition

The decomposition part of the program is implemented in `ftint_pySecDec.py`. It is called from the command line as

```
$ python3 <ftint_path>/ftint_pySecDec.py <ft_integrals> [<
    ↪ options>]
```

Listing 1: Command for sector decomposition.

where, here and in the following, `<ftint_path>` is to be replaced by the actual path to the source code of `ftint`. In the simplest case, `<ft_integrals>` is a string encoding a single flow-time integral. Its format follows closely the definition in Eq. (1), with some adjustments. The most significant one is the representation of the masses. For the sector decomposition, the actual value of a mass is irrelevant, as long as it is non-zero. In the input for `ftint_pySecDec.py`, non-zero masses are indicated only by their index, i.e. $m_1 \to$ `1`, $m_2 \to$ `2`, etc.[4] Let us consider a specific two-loop example[5]:

$$\begin{aligned}&\texttt{f[\{0,1\},\{u2-u1*u2,2,u2\},\{-1,\{2,2\},\{1,3\}\}]} =\\&= f(\{0,1\},\{u_2-u_1\,u_2,2,u_2\},\{-1,\{2,m_2\},\{1,m_3\}\})=\\&(4\pi t)^D\int_{p,k}\int_{u_{1,2}} u_2\,\frac{p^2}{(k^2+m_2^2)^2((p+k)^2+m_3^2)}e^{-t[(-u_1u_2+u_2)p^2+2k^2+u_2(p+k)^2]}\end{aligned} \tag{41}$$

The first line contains the input for `ftint_pySecDec.py`, the second line corresponds to the notation of Eq. (1). The flow-time integration variables must be named `u1`, `u2`, ..., and the multiplication symbol "`*`" must be given explicitly. The actual numerical values for the masses must be provided only upon numerical integration, using the `--masses` option of `ftint_integrate.py`, as will be discussed in more detail in Section 4.2.

If a propagator is massless, the mass argument is `0`, or it can be left out altogether. For example, for $m_2 = 0$, the third argument of `f` in the first line of Eq. (41) could be given either as `{-1,{2,0},{1,3}}` or as `{-1,2,{1,3}}`. Recall that an integral may contain additional infrared singularities in the limit where a mass is zero. This is indeed the case in the example considered here: while for $m_2 \neq 0$ the integral contains only poles up to $1/\epsilon$ (see Listing 5 below), there are also $1/\epsilon^2$ poles for $m_2 = 0$. It is important to indicate all massless propagators already at the decomposition stage in this case.

The command to run the decomposition for the integral in Eq. (41), assuming default settings, is

```
$ python3 <ftint_path>/ftint_pySecDec.py \
    'f[{0,1},{u2-u1*u2,2,u2},{-1,{2,2},{1,3}}]'
```

`ftint` will first convert the flow-time integral from the momentum representation of Eq. (1) to the parameter form of Eq. (37), and then pass it on to `pySecDec`. In this particular case, sector decomposition and subsequent compilation take of the order of one and ten seconds on a regular modern desktop computer, respectively.[6] By default, the output files are written to the directory `ftint_out_<n>`, created by `ftint` in the current working directory. Initially, the parameter `<n>` is set to `0`; it is recursively increased by one if the directory `ftint_out_<n>` already exists. In the following, we will assume `<n>=0`, unless indicated otherwise.

The content of the output directory `ftint_out_0` is

```
|-- integral_information.json
\-- secdec
    \-- secdec_ft_integral_1
        |-- secdec.out
        |-- compilation.out
        |-- ftint_data.json
        \-- disteval
```

Listing 2: Contents of the output directory `ftint_out_0`.

The file `integral_information.json` collects the relevant parameters of the specific run, while the directory `secdec` contains one `secdec_ft_integral_<n>` for each compiled flow-time integral (in this case the input consisted of only one integral). Inside there are logfiles `secdec.out` and `compilation.out` of the decomposition and compilation respectively, a data-file `ftint_data.json` with integral-specific information, as well as the compiled integral library `disteval`.

The input `<ft_integrals>` in Listing 1 can also be one or several strings containing several flow-time integrals, or even one or several file names. `ftint` will extract the flow-time integrals given in the proper format and perform the decomposition and the compilation of the integration library for each integral (duplicates are removed).[7] The output for all of them will be written to a single output directory `ftint_out_<n>`, whose contents will look as in Listing 2, but with a separate library `secdec_ft_integral_*` for each integral of the input. An example will be presented in Section 4.3.

`ftint_pySecDec.py` provides a number of options which can be displayed using the command

```
$ python3 <ftint_path>/ftint_pySecDec.py --help
```

These options are

- `--help`: Print this list of options.
- `--list=STR_NUM_LIST`: Specify a sublist of integrals to be evaluated. The format is either a comma-separated list of integers, or an interval of the form `3-10`, for example.
- `--print_list`: Print the list of integrals to be calculated and exit.
- `--exclude=EXCLUDE [EXCLUDE ...]`: Do not evaluate the integrals in `EXCLUDE`, which is of the same format as `<ft_integrals>` in Listing 1.
- `--normalform`: Map the integrals to their normal form.
- `--normalform_only`: Map the integrals to their normal form and stop.

[4] The reason why we do not adopt the more intuitive notation `f[{0,1},{u2-u1*u2,2,u2},{-1,{2,m2},{1,m3}}]` is not to interfere with any locally defined symbols `m2`, `m3` in the user's code.

[5] See `examples/2L_massive.in` in the code repository.

[6] This can change with suitable options for the compiler and for `make`, see below.

[7] The regular expression searched for is `f\[\{*.?\}\]`. The user is advised to make sure that this cannot be confused with any other objects in the input. See also Appendix A.

`--normalform_list`: Return the list of equivalent integrals and stop.
`--normal_key=NORMAL_KEY`: `0/-1`: define first/last element of `normalform_list` as normal form, see Section 4.4.
`--count_only`: Return the number of integrals to be calculated and stop.
`--latex`: Return the input diagrams in LaTeX format and stop.
`--epsorder=EPSORDER`: Required order in $\epsilon = (4 - D)/2$.
`--input_format INPUT_FORMAT`: Specification of input format, see Appendix A.
`--outdir=OUT_DIR`: Name of output directory.
`--CXX_flags=CXX_FLAGS`: `C++` compiler flags, example: `--CXX_flags="-mavx2 -mfma"`.
`--make_flags=MAKE_FLAGS`: `Makefile` flags, example: `--make_flags=-j4`.
`--CUDA_flags=CUDA_FLAGS`: `CUDA` flags to compile with GPU support, example: `--CUDA_flags=-arch=sm_XX` where `sm_XX` should be replaced with the target `NVidia` GPU architecture.
`--integrate`: Automatically calls `ftint_integrate.py` after sector decomposition to perform the numerical integration (with default options).
`--dimension=DIMENSION`: The integer part N of the space-time dimension $D = N - 2\epsilon$.
`--overwrite`: Overwrite existing directory. Otherwise, appends `_<n>` to existing output directory.
`--append`: Append new integrals from the input to the compiled integrals in the output directory. This flag can not be passed together with `--overwrite`.
`--ibp_power_goal=IBP_POWER_GOAL`: Before defining subtraction terms, `pySecDec` performs a number of integration by parts iterations to raise the order of the factorized poles to `IBP_POWER_GOAL`.

4.2. Numerical integration

The integration part of the program is implemented in `ftint_integrate.py`. It is called from the command line as:

```
$ python3 <ftint_path>/ftint_integrate.py <integral_directory>
    ↪ [<options>]
```

Listing 3: Command for numerical integration.

where `<integral_directory>` is the name of the output directory created by `ftint_pySecDec.py`, i.e. `ftint_out_<n>` by default. `ftint` will pass the integral to `pySecDec` for numerical integration. If not specified otherwise, the output files will be written to the directory `<integral_directory>/result_<m>`, where by default `<m>` is set to "`0`", but is recursively increased by one if the output directory exists. The numerical result of the integral is stored in the file `mathf_out.m` in the form of a `Mathematica` replacement rule. For example, the following command would evaluate the integral in Eq. (41) with $m_2^2 = 2.5/t$ and $m_3^2 = 3/t$ with otherwise default settings:

```
$ python3 <ftint_path>/ftint_integrate.py ftint_out_0 --masses
    ↪ =0,2.5,3
```

Listing 4: Numerical evaluation of the integral in Eq. (41).

The numerical values for the *squared* masses have to be specified via the `--masses` option in units of the inverse flow time $1/t$.[8] Upon completion, `ftint` has added a directory `result_0` to `ftint_out_0` which, aside from some other information on the integration run, contains the file `mathf_out.m` with the following content:

```
1  (*
2   produced by ftint, version 1.0, Fri May 24 13:50:34 2024
3  *)
4  {
5  (* integral 1 [ m2**2 = 2.5/t, m3**2 = 3/t ]: *)
6  f[{0,1},{u2-u1*u2,2,u2},{-1,{2,2},{1,3}}] ->  (
7       +eps
         ↪ ^-2*(+0.0000000000000000*10^+00+0.0000000000000000*10^+00*
         ↪ I)
8       +eps
         ↪ ^-2*(+0.0000000000000000*10^+00+0.0000000000000000*10^+00*
         ↪ I)*plusminus
9       +eps
         ↪ ^-1*(+1.1266529421611552*10^-02+0.0000000000000000*10^+00*
         ↪ I)
10      +eps
         ↪ ^-1*(+2.3027953579787037*10^-10+0.0000000000000000*10^+00*
         ↪ I)*plusminus
11      +eps^0*(+1.7797179680747620*10^-04+0.0000000000000000*10^+00*
         ↪ I)
12      +eps^0*(+1.0762786605309655*10^-09+0.0000000000000000*10^+00*
         ↪ I)*plusminus
13      +eps^1*(+6.8615123921316060*10^-03+0.0000000000000000*10^+00*
         ↪ I)
14      +eps^1*(+1.1513874186746525*10^-09+0.0000000000000000*10^+00*
         ↪ I)*plusminus
15      +eps^2*(-5.5497372202654837*10^-03+0.0000000000000000*10^+00*
         ↪ I)
16      +eps^2*(+7.0664076906102803*10^-09+0.0000000000000000*10^+00*
         ↪ I)*plusminus
17      +eps^3*(+5.2760242194534335*10^-03+0.0000000000000000*10^+00*
         ↪ I)
18      +eps^3*(+3.2253823531270829*10^-08+0.0000000000000000*10^+00*
         ↪ I)*plusminus
19    )
20 }
```

Listing 5: Output file of the numerical integration.

The integration uncertainties are marked by the variable `plusminus`. The mass values are specified only in the comment on line 5. This makes it easy to use the replacement rule in order to evaluate the same expression for different mass values. For example, if the result of the calculation is stored in a `Mathematica` variable `result` which depends on the integral under consideration, one can obtain a numerical value for `result` as

```
replace = Get["ftint_out_0/result_0/mathf_out.m"]; result /.
    ↪ replace
```

Listing 6: Inserting the numerical value for the integral within `Mathematica`.

One may now call `ftint_integrate.py` again with different mass parameters, and the result would be stored in `ftint_out_0/result_1/mathf_out.m`. One can then use again the code in Listing 6, simply replacing `result_0` by `result_1`.

In addition to the `Mathematica` output file, `ftint` provides the result also as YAML[9] and JSON[10] files named `sympyf_out.yml` and `out.json`. The `ftint` distribution includes the files `read_yaml.py` and `read_json.py` as examples on how to read these files in `Python`.

To see an overview of the optional parameters related to the integration, together with their defaults, one can run

```
$ python3 ftint_integrate.py --help
```

The options are

[8] Recall that Eq. (1) is dimensionless and thus only depends on $m_1^2 t, \cdots, m_k^2 t$. It is instructive to note that, if we had specified the l.h.s. of Eq. (41) as `f[{0,1},{u2-u1*u2,2,u2},{-1,{2,3},{1,2}}]`, the command in Listing 4 would evaluate the r.h.s. of that equation with $m_3^2 = 2.5/t$ and $m_2^2 = 3/t$.

[9] https://yaml.org/.

[10] https://www.json.org/.

`--help`: Print this list of options.
`--masses=MASSES`: Values for masses of each propagator, provided as comma-separated list. The n^{th} value of that list is inserted for the mass labeled n in the input of `ftint_pySecDec.py`; see also Footnote 8.
`--epsrel=EPSREL`: Stop if this relative precision is reached.
`--epsabs=EPSABS`: Stop if this absolute precision is reached.
`--delta=DELTA`: Cut-off parameter for mass exponential, see Eq. (40).
`--output_format=OUTPUT_FORMAT`: Specification of output format, see Appendix A.
`--points=POINTS`: Begin integration with this lattice size.[11]
`--presamples=PRESAMPLES`: Use this many points for presampling.[11]
`--shifts=LATTICE_SHIFTS`: Use this many lattice shifts per integral.[11]
`--lattice_candidates=LATTICE_CANDIDATES`: Number of median lattice candidates.[11]
`--outfile=OUTFILE`: Name of the individual integration output files.
`--timeout=TIMEOUT`: The maximum number of seconds the integrator will spend on each integral. If `TIMEOUT` is reached a result that may not meet the desired numerical accuracy will be returned.

In addition, the options `--list`, `--print_list`, `--overwrite` and `--outdir` are available, with the same meaning as in `ftint_pySecDec.py`, see above.

4.3. Example: checking an integration-by-parts relation

As already pointed out above, like regular Feynman integrals in dimensional regularization, flow-time integrals obey certain IbP relations which can be derived by considering integrals over total derivatives w.r.t. the loop momenta or the flow-time variables; details can be found in Ref. [22]. Let us numerically check such an IbP relation in order to give an example on how `ftint` can be used in practice. We formulate the relation in terms of a `Mathematica` replacement rule which we assume is contained in a file[12] `ibp_rule.in`, see Listing 7.

```
{f[{},{0,0,0,1,1,1},{-1,1,1,1,1,1}] ->
-f[{},{0,0,0,1,1,1},{0,1,1,1,1,0}]/2
-(f[{},{0,0,0,1,1,1},{1,0,1,1,0,0}])/(1-n/4)
-(f[{},{0,0,0,1,1,1},{1,1,0,0,0,0}])/(2*(1+(-7/12+n/12)*n))}
```

Listing 7: A three-loop IbP identity in `Mathematica` format.

Here, $\mathrm{n} = 4 - 2\epsilon$. Since some of the integrals on the r.h.s. have a prefactor $\sim 1/\epsilon$, we need to evaluate them through $\mathcal{O}(\epsilon)$ in order to check this relation through $\mathcal{O}(\epsilon^0)$, while the other integrals are needed only through $\mathcal{O}(\epsilon^0)$. In realistic cases, it is advisable to split the set of integrals according to the required power in ϵ. For this simple example though, we evaluate all integrals to $\mathcal{O}(\epsilon)$.

This is done by first performing the sector decomposition:

```
$ python3 <ftint_path>/ftint_pySecDec.py ibp_rule.in --epsorder
    ↪ =1
```

This will first report that `ftint` finds four different flow-time integrals in the file. It will then perform the sector decomposition for the first diagram and compile the corresponding integration library, before turning to the second diagram etc. After completion, `ftint` has created a directory named `ftint_out_0` with the following structure:

```
\-- ftint_out_0
    |-- integral_information.json
    \-- secdec
        |-- secdec_ft_integral_1
        |-- secdec_ft_integral_2
        |-- secdec_ft_integral_3
        \-- secdec_ft_integral_4
```

The directories `secdec_ft_integral_*` contain the integration libraries for each of the four integrals, as well as other information required by `ftint` for the numerical integration. The latter is performed through the command

```
$ python3 <ftint_path>/ftint_integrate.py ftint_out_0
```

This again first reports that four integrals will be evaluated. The corresponding numerical results will be printed to the screen. After completion, `ftint` will have created a subdirectory named `results_0` in `ftint_out_0`, which, among other information, contains the file `mathf_out.m`, whose contents are shown in Listing 8.

```
(*
 produced by ftint, version 1.0, Fri May 24 09:21:05 2024
*)
{
(* integral 1 : *)
f[{},{0,0,0,1,1,1},{-1,1,1,1,1,1}] ->  (
    +eps
     ↪ ^-1*(+1.4384102482242656*10^-01+0.0000000000000000*10^+00*
     ↪ I)
    +eps
     ↪ ^-1*(+7.5901486511209429*10^-09+0.0000000000000000*10^+00*
     ↪ I)*plusminus
    +eps^0*(+9.1030039859574119*10^-01+0.0000000000000000*10^+00*
     ↪ I)
    +eps^0*(+7.7645751796859680*10^-06+0.0000000000000000*10^+00*
     ↪ I)*plusminus
    +eps^1*(+3.6458111371273918*10^+00+0.0000000000000000*10^+00*
     ↪ I)
    +eps^1*(+4.2456809764811359*10^-05+0.0000000000000000*10^+00*
     ↪ I)*plusminus
  ),
(* integral 2 : *)
f[{},{0,0,0,1,1,1},{0,1,1,1,1,0}] ->  (
    +eps
     ↪ ^-1*(+2.8768207244049038*10^-01+0.0000000000000000*10^+00*
     ↪ I)
    +eps
     ↪ ^-1*(+1.2597382329360634*10^-11+0.0000000000000000*10^+00*
     ↪ I)*plusminus
    +eps^0*(+1.4717502294126590*10^+00+0.0000000000000000*10^+00*
     ↪ I)
    .
    .
    .
```

Listing 8: Result for the integrals of Listing 7. Only the first few lines are shown.

One can now check the IbP relation within `Mathematica` using the following code[13]:

```
rule = Get["ibp_rule.in"][[1]];
replace = Get["ftint_out_0/result_0/mathf_out.m"];
check = Normal[Series[(rule[[1]]-rule[[2]]) /. n -> 4-2*eps /.
     ↪ replace,
        {eps,0,0}]];
```

The result is

[11] These are QMC parameters, see e.g. Ref. [25] for a more detailed explanation. The default settings are fine for most examples.
[12] See `examples/ibp_rule.in` in the code repository.
[13] See `examples/check_ibp_rule.m` in the code repository.

```
                    -15              -15
         -3.66374 10    + 4.85345 10    plusminus
Out[19]= --------------------------------------- +
                            2
                         eps

              -9
     7.64685 10   (-1.48995 + 1. plusminus)
>    ------------------------------------- +
                         eps

              -6
>    7.90387 10   (2.47724 + 1. plusminus)
```

meaning that the l.h.s. and the r.h.s. of the relation agree within the default numerical precision. It is now easy to increase this precision by running

```
$ python3 <ftint_path>/ftint_integrate.py ftint_out_0 \
      --epsrel=1e-8 --epsabs=1e-8
```

4.4. Mapping to the normal form

If one needs to compute a large list of integrals, it may be advantageous to map them to their normal form before integration, see Section 3.3. This can be done by calling `ftint_pySecDec.py` with the option `--normalform`. `ftint` will then produce the file `ftint_out_<n>/normalmap.m` which contains the mapping of each integral to its normal form (unless it already is in the normal form) in `Mathematica` format. `ftint` will then proceed with the calculation only for the normal-form integrals.

For example, if the file[14] `normal_form.in` contains the following list of flow-time integrals

```
1 f[{},{2,1,0},{2,2,1}]
2 f[{1},{2*u1,u1,u1},{2,1,3}]
3 f[{2,1},{u1*u2,2,u1},{3,2,1}]
4 f[{1,2},{u1*u2,u2,2},{3,1,2}]
```

Listing 9: List of flow-time integrals to be brought to normal form.

then the call

```
$ python3 <ftint_path>/ftint_pySecDec.py integrals.m --
   ↪ normalform
```

will produce the file `ftint_out_0/normal_form.in` with the following content:

```
{f[{},{2,1,0},{2,2,1}] -> f[{},{2,1,0},{{2,0},{2,0},{1,0}}],
f[{1},{2*u1,u1,u1},{2,1,3}] -> f[{1},{2*u1,u1,u1
    ↪ },{{2,0},{3,0},{1,0}}],
f[{2,1},{u1*u2,2,u1},{3,2,1}] -> f[{2,1},{u1*u2,u1
    ↪ ,2},{{3,0},{1,0},{2,0}}],
f[{1,2},{u1*u2,u2,2},{3,1,2}] -> f[{2,1},{u1*u2,u1
    ↪ ,2},{{3,0},{1,0},{2,0}}]}
```

The first thing to notice is that `ftint` does not use the abbreviated notation for massless propagators, simply to ensure a unique output format. Furthermore, aside from this notational aspect, the integral in line 1 of Listing 9 is already in normal form. The integrals in line 3 and line 4 are mapped to the same normal form. Thus, `ftint` will do the sector decomposition only for the *three* different normal-form integrals. As usual, it will write the integration libraries to `ftint_out_0`, compile them, and the user can evaluate them numerically using `ftint_integrate.py`.

[14] See `examples/normal_form.in` in the code repository.

If the user is only interested in the normal-form mappings, one may use the `--normalform_only` option instead. In this case, `ftint` will write the file `ftint_out_0/normalform.m` and stop.

For integrals with non-vanishing masses, the procedure works in the very same way. The only subtlety here is that the order of the masses may change. Consider, for example, the integral

$$\texttt{f[\{1,2\},\{3*u1,2,u1*u2\},\{\{1,1\},\{4,2\},\{2,3\}\}]} = \\ = \int_{p,k}\int_{u_{1,2}} u_1 u_2^2 \, \frac{e^{-t[3u_1p^2+2k^2+u_1u_2(p+k)^2]}}{(p^2+m_1^2)(k^2+m_2^2)^4((p+k)^2+m_3^2)^2}\,. \tag{42}$$

Mapping it to normal form will result in

$$\texttt{f[\{1,2\},\{u1*u2,3*u1,2\},\{\{2,3\},\{1,1\},\{4,2\}\}]} = \\ = \int_{p,k}\int_{u_{1,2}} u_1 u_2^2 \, \frac{e^{-t[u_1u_2p^2+3u_1k^2+2(p+k)^2]}}{(p^2+m_3^2)^2(k^2+m_1^2)((p+k)^2+m_2^2)^4}\,. \tag{43}$$

In order to numerically evaluate these integrals with `ftint_integrate.py`, one must use the same order of arguments m_1, m_2, m_3 in the `--masses` option. This means that the command for the integration is independent of whether one evaluates the original integral or its normal form (aside from the fact that their integration libraries may be located in different directories).

As described in Section 3.3, if the option `--normalform` is given, `ftint` generates a list of equivalent integrals and continues the calculation with the *first* element of this list. The user may alter this behavior by adding the option `--normal_key=-1`, in which case `ftint` will continue with the *last* element of the sorted list. The full (sorted) list can be viewed by calling `ftint` with the option `--normalform_list`. We have not observed any significant differences in computing times among these integrals.

4.5. Checks

4.5.1. Analytic solutions, symmetries, and simple identities

There are a number of rather straightforward checks that we have used to validate `ftint`:

- Some simple integrals can be solved analytically, see, e.g., Eqs. (19) and (24). We have compared a number of them to the numerical result from `ftint` and found agreement.
- An integral and its normal form, see Section 3.3, must lead to the same numerical result, of course. We have confirmed this symmetry with `ftint`, which both checks the numerical evaluation of the integral, as well as the algorithm and implementation for mapping the integrals to their normal form.
- Certain multi-loop integrals can be written as products of integrals at lower loop order. For example, it is easy to see that

  $$f(\{\},\{1,0,1\},\{-1,0,0\}) = f(\{\},\{1\},\{-1\})\cdot f(\{\},\{1\},\{0\}) \tag{44}$$

  All such checks were passed by `ftint`.

4.5.2. Flow-time derivatives

In a dimensionally regularized integral, one can interchange integration and differentiation without changing the result. In the massless case, one may derive non-trivial relations from this that can be easily checked. For example, since the integral as defined in Eq. (1) is dimensionless, in the massless case it follows that

$$0 = t\frac{\partial}{\partial t} f(\mathbf{c},\mathbf{a},\mathbf{b}) = \left(\frac{lD}{2} - b\right) f(\mathbf{c},\mathbf{a},\mathbf{b}) \\ - (4\pi t)^{lD/2}\, t^{-b+1} \int_{[0,1]^f} \mathrm{d}\mathbf{u}\,\mathbf{u}^{\mathbf{c}} \sum_{j=1}^{k} a_j \int_{\mathbf{p}} P_j^2 \, \frac{\exp[-t\sum_{i=1}^{k} a_i P_i^2]}{(P_1^2)^{b_1}\cdots(P_k^2)^{b_k}}\,. \tag{45}$$

Canceling the power of P_j^2 and inserting the explicit polynomials a_j of the flow-time variables turns the r.h.s. into the sum of regular flow-time integrals of the form Eq. (1) with varied parameters. As an example, consider again a two-loop case:

$$(-6+2\epsilon)\, f(\{\},\{1,1,0\},\{0,-2,0\}) = \\ = -f(\{\},\{1,1,0\},\{-1,-2,0\}) - f(\{\},\{1,1,0\},\{0,-3,0\})\,. \tag{46}$$

This relation can be checked in close analogy to the example discussed in Section 4.3.

4.5.3. Integration-by-parts identities

In Section 4.3, we used the check of a two-loop IbP relation in order to demonstrate the operation of `ftint`. In fact, we have used hundreds of such relations at three-loop level, derived in the context of calculations performed in Ref. [16], for example, in order to check `ftint`.

5. Conclusions and outlook

With more and more potential applications of the perturbative approach to the GFF identified, the demand for suitable software tools is increasing. In this paper, we described the application of the sector-decomposition algorithm to flow-time integrals up to the three-loop level in the form of a `Python` program named `ftint`. It transforms a flow-time integral without external momenta to a multidimensional parameter integral over a unit hypercube, which is then passed to the public library `pySecDec` for sector decomposition and numerical integration.

We have performed a number of checks on the program and made an effort towards user-friendliness and flexibility. Future releases of the program will support partial fractioning for propagators with identical momenta but different masses, as well as non-vanishing external momenta. Furthermore, we plan to make use of `pySecDec`'s main strengths of evaluating complete amplitudes rather than individual integrals.

CRediT authorship contribution statement

Robert V. Harlander: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Funding acquisition, Conceptualization. **Theodoros Nellopoulos:** Writing – review & editing, Software, Methodology. **Anton Olsson:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology. **Marius Wesle:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Anton Olsson reports financial support and travel were provided by German Research Foundation. Robert Harlander reports administrative support was provided by German Research Foundation. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The program is available at https://gitlab.com/ftint/ftint.

Acknowledgements

We would like to thank Janosch Borgulat, Nils Felten, Gudrun Heinrich, Stephen Jones, Matthias Kerner, Jonas Kohnen, Fabian Lange, Henri Lindlahr, Vitaly Magerya, Tobias Neumann, Johannes Schlenk, and Henry Werthenbach for helpful input and comments. This research was supported by the *Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants 396021762 - TRR 257* and *460791904*.

Appendix A. In- and output format

By default, `ftint` assumes the format defined in Eq. (1), as exemplified by Eq. (41), for the flow-time integrals, both in the input and the output. `ftint` provides a basic way to convert between other formats and the `ftint`-format by editing the file `user_format.py`. This defines three functions. For the sake of clarity, let us assume that the user would like to perform a mapping

```
f[{a},{b},{c}] <-> g([a],[b],[c])
```

The purpose of the three functions in `user_format.py` is given as follows.

- `user_patterns` allows the user to specify the pattern which defines a gradient-flow integral. In the example above, the function could be defined as

```
def user_patterns(input_format):
    if input_format==2:
        out = [r"g\(\[.*?\]\)"]
    else:
        out = [r"f\[\{.*?\}\]"]
    return(out)
```

- `from_user` defines how to translate the user's format to the `ftint` format. In this case, one could define

```
def from_user(input_format,string):
    if input_format==2:
        out = re.sub(r'g\(\[(.*?)\],\[(.+?)\],\[(.+?)\]\)',r'f[{\1},{\2},{\3}]',string)
    else:
        out = string
    return(out)
```

- `to_user` defines how to translate the `ftint` format to the user's format. For the current example, this could be achieved through

```
def to_user(output_format,string):
    if output_format==2:
        out = re.sub(r'f\[\{(.*?)\},\{(.+?)\},\{(.+?)\}\]',r'g([\1],[\2],[\3])',string)
    else:
        out = string
    return(out)
```

The user can now switch to the new input format by providing the option `--input_format=2` to `ftint_pySecDec.py`. The new output format is obtained by providing the option `--output_format=2` to `ftint_integrate.py`. Without these options, the default format will be adopted. Let us stress that this is a very rudimentary implementation. The user is advised to use it with care. It may be safer to convert all input to the default `ftint` notation before using `ftint`.

References

[1] R. Narayanan, H. Neuberger, Infinite N phase transitions in continuum Wilson loop operators, J. High Energy Phys. 03 (2006) 064, arXiv:hep-th/0601210.

[2] M. Lüscher, Trivializing maps, the Wilson flow and the HMC algorithm, Commun. Math. Phys. 293 (2010) 899–919, arXiv:0907.5491 [hep-lat].

[3] M. Lüscher, Properties and uses of the Wilson flow in lattice QCD, J. High Energy Phys. 08 (2010) 071, arXiv:1006.4518 [hep-lat], Erratum: J. High Energy Phys. 03 (2014) 092.

[4] M. Lüscher, P. Weisz, Perturbative analysis of the gradient flow in non-abelian gauge theories, J. High Energy Phys. 02 (2011) 051, arXiv:1101.0963 [hep-th].

[5] M. Lüscher, Chiral symmetry and the Yang–Mills gradient flow, J. High Energy Phys. 04 (2013) 123, arXiv:1302.5246 [hep-lat].

[6] BMW collaboration, S. Borsányi, S. Dürr, Z. Fodor, C. Hoelbling, S.D. Katz, S. Krieg, T. Kurth, L. Lellouch, T. Lippert, C. McNeile, High-precision scale setting in lattice QCD, J. High Energy Phys. 09 (2012) 010, arXiv:1203.4469 [hep-lat].

[7] H. Suzuki, Energy–momentum tensor from the Yang–Mills gradient flow, PTEP 2013 (2013) 083B03, arXiv:1304.0533 [hep-lat], Erratum: PTEP 2015 (2015) 079201.

[8] H. Makino, H. Suzuki, Lattice energy–momentum tensor from the Yang–Mills gradient flow—inclusion of fermion fields, PTEP 2014 (2014) 063B02, arXiv:1403.4772 [hep-lat], Erratum: PTEP 2015 (2015) 079202.

[9] T. Iritani, M. Kitazawa, H. Suzuki, H. Takaura, Thermodynamics in quenched QCD: energy–momentum tensor with two-loop order coefficients in the gradient-flow formalism, PTEP 2019 (2) (2019) 023B02, arXiv:1812.06444 [hep-lat].

[10] R.V. Harlander, Y. Kluth, F. Lange, The two-loop energy–momentum tensor within the gradient-flow formalism, Eur. Phys. J. C 78 (11) (2018) 944, arXiv:1808.09837 [hep-lat], Erratum: Eur. Phys. J. C 79 (2019) 858.

[11] A. Suzuki, Y. Taniguchi, H. Suzuki, K. Kanaya, Four quark operators for kaon bag parameter with gradient flow, Phys. Rev. D 102 (3) (2020) 034508, arXiv:2006.06999 [hep-lat].

[12] H. Suzuki, H. Takaura, $t \to 0$ extrapolation function in the small flow time expansion method for the energy–momentum tensor, PTEP 2021 (7) (2021) 073B02, arXiv:2102.02174 [hep-lat].

[13] R.V. Harlander, F. Lange, Effective electroweak Hamiltonian in the gradient-flow formalism, Phys. Rev. D 105 (7) (2022) L071504, arXiv:2201.08618 [hep-lat].

[14] M. Black, R. Harlander, F. Lange, A. Rago, A. Shindler, O. Witzel, Using gradient flow to renormalise matrix elements for meson mixing and lifetimes, PoS LATTICE2023 (2024) 263, arXiv:2310.18059 [hep-lat].

[15] SymLat collaboration, M.D. Rizik, C.J. Monahan, A. Shindler, Short flow-time coefficients of CP-violating operators, Phys. Rev. D 102 (3) (2020) 034509, arXiv:2005.04199 [hep-lat].

[16] R.V. Harlander, F. Lange, T. Neumann, Hadronic vacuum polarization using gradient flow, J. High Energy Phys. 08 (2020) 109, arXiv:2007.01057 [hep-lat].

[17] E. Mereghetti, C.J. Monahan, M.D. Rizik, A. Shindler, P. Stoffer, One-loop matching for quark dipole operators in a gradient-flow scheme, J. High Energy Phys. 04 (2022) 050, arXiv:2111.11449 [hep-lat].

[18] R. Harlander, M.D. Rizik, J. Borgulat, A. Shindler, Two-loop matching of the chromomagnetic dipole operator with the gradient flow, PoS LATTICE2022 (2023) 313, arXiv:2212.09824 [hep-lat].

[19] J. Borgulat, R.V. Harlander, J.T. Kohnen, F. Lange, Short-flow-time expansion of quark bilinears through next-to-next-to-leading order QCD, J. High Energy Phys. 05 (2024) 179, arXiv:2311.16799 [hep-lat].

[20] A. Shindler, Moments of parton distribution functions of any order from lattice QCD, arXiv:2311.18704 [hep-lat].

[21] J. Dragos, T. Luu, A. Shindler, J. de Vries, A. Yousif, Confirming the existence of the strong CP problem in lattice QCD with the gradient flow, Phys. Rev. C 103 (1) (2021) 015202, arXiv:1902.03254 [hep-lat].

[22] J. Artz, R.V. Harlander, F. Lange, T. Neumann, M. Prausa, Results and techniques for higher order calculations within the gradient-flow formalism, J. High Energy Phys. 06 (2019) 121, arXiv:1905.00882 [hep-lat], Erratum: J. High Energy Phys. 10 (2019) 032.

[23] S. Borowka, G. Heinrich, S. Jahn, S.P. Jones, M. Kerner, J. Schlenk, T. Zirke, pySecDec: a toolbox for the numerical evaluation of multi-scale integrals, Comput. Phys. Commun. 222 (2018) 313–326, arXiv:1703.09692 [hep-ph].

[24] S. Borowka, G. Heinrich, S. Jahn, S.P. Jones, M. Kerner, J. Schlenk, A GPU compatible quasi-Monte Carlo integrator interfaced to pySecDec, Comput. Phys. Commun. 240 (2019) 120–137, arXiv:1811.11720 [physics.comp-ph].

[25] G. Heinrich, S.P. Jones, M. Kerner, V. Magerya, A. Olsson, J. Schlenk, Numerical scattering amplitudes with pySecDec, Comput. Phys. Commun. 295 (2024) 108956, arXiv:2305.19768 [hep-ph].

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

[27] T. Binoth, G. Heinrich, Numerical evaluation of multiloop integrals by sector decomposition, Nucl. Phys. B 680 (2004) 375–388, arXiv:hep-ph/0305234.

[28] G. Heinrich, Sector decomposition, Int. J. Mod. Phys. A 23 (2008) 1457–1486, arXiv:0803.4177 [hep-ph].

[29] R.V. Harlander, T. Neumann, The perturbative QCD gradient flow to three loops, J. High Energy Phys. 06 (2016) 161, arXiv:1606.03756 [hep-ph].

[30] S.G. Gorishnii, S.A. Larin, F.V. Tkachov, The algorithm for OPE coefficient functions in the MS scheme, Phys. Lett. B 124 (1983) 217–220.

[31] S.G. Gorishnii, S.A. Larin, Coefficient functions of asymptotic operator expansions in minimal subtraction scheme, Nucl. Phys. B 283 (1987) 452.

[32] R.V. Harlander, S.Y. Klein, M. Lipp, FeynGame, Comput. Phys. Commun. 256 (2020) 107465, arXiv:2003.00896 [physics.ed-ph].

[33] R. Harlander, S.Y. Klein, M.C. Schaaf, FeynGame-2.1 – Feynman diagrams made easy, PoS EPS-HEP2023 (2024) 657, arXiv:2401.12778 [hep-ph].

[34] A.V. Smirnov, M.N. Tentyukov, Feynman Integral Evaluation by a Sector decomposiTion Approach (FIESTA), Comput. Phys. Commun. 180 (2009) 735–746, arXiv:0807.4129 [hep-ph].

[35] A.V. Smirnov, V.A. Smirnov, M. Tentyukov, FIESTA 2: parallelizeable multiloop numerical calculations, Comput. Phys. Commun. 182 (2011) 790–803, arXiv:0912.0158 [hep-ph].

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

[37] A.V. Smirnov, FIESTA4: optimized Feynman integral calculations with GPU support, Comput. Phys. Commun. 204 (2016) 189–199, arXiv:1511.03614 [hep-ph].

[38] T. Kaneko, T. Ueda, A geometric method of sector decomposition, Comput. Phys. Commun. 181 (2010) 1352–1361, arXiv:0908.2897 [hep-ph].

[39] G. Heinrich, S. Jahn, S.P. Jones, M. Kerner, F. Langer, V. Magerya, A. Pöldaru, J. Schlenk, E. Villa, Expansion by regions with pySecDec, Comput. Phys. Commun. 273 (2022) 108267, arXiv:2108.10807 [hep-ph].

[40] Wolfram Research, Inc., Mathematica, version 14.1, https://www.wolfram.com/mathematica, 2024, Champaign, IL.

[41] J.A.M. Vermaseren, New features of FORM, arXiv:math-ph/0010025.

[42] J. Kuipers, T. Ueda, J.A.M. Vermaseren, J. Vollinga, FORM version 4.0, Comput. Phys. Commun. 184 (2013) 1453–1467, arXiv:1203.6543 [cs.SC].