COLLIER: a fortran-based Complex One-Loop LIbrary in Extended Regularizations version 1.2.8

Ansgar Denner[a], Stefan Dittmaier[b], Lars Hofer[c]

[a] *Universität Würzburg, Institut für Theoretische Physik und Astrophysik, D-97074 Würzburg, Germany*
[b] *Albert-Ludwigs-Universität Freiburg, Physikalisches Institut, D-79104 Freiburg, Germany*
[c] *Department de Física Quàntica i Astrofísica (FQA), Institut de Ciències del Cosmos (ICCUB), Universitat de Barcelona (UB), Martí Franquès 1, E-08028 Barcelona, Spain*

Abstract

We present the library COLLIER for the numerical evaluation of one-loop scalar and tensor integrals in perturbative relativistic quantum field theories. The code provides numerical results for arbitrary tensor and scalar integrals for scattering processes in general quantum field theories. For tensor integrals either the coefficients in a covariant decomposition or the tensor components themselves are provided. COLLIER supports complex masses, which are needed in calculations involving unstable particles. Ultraviolet and infrared singularities are treated in dimensional regularization. For soft and collinear singularities mass regularization is available as an alternative.

Keywords: NLO computations; radiative corrections; one-loop integrals; higher orders; tensor reduction; scalar integrals

[☆] The program is available from http://collier.hepforge.org.

Email addresses: ansgar.denner@physik.uni-wuerzburg.de (Ansgar Denner), stefan.dittmaier@physik.uni-freiburg.de (Stefan Dittmaier), hofer@ecm.ub.edu (Lars Hofer)

Contents

1. Introduction

The exploitation of experiments at high-energy colliders like the LHC heavily relies on theoretical predictions for scattering processes within quantum field theories, which are basically evaluated within perturbation theory. Next-to-leading-order (NLO) perturbative corrections of the strong interaction are a crucial ingredient for decent predictions, but also NLO electroweak corrections are required in many cases (see, for instance, Refs. [1, 2]).

In the calculation of NLO QCD corrections huge progress has been made in recent years (see e.g. Refs. [1–5]), and automated tools have become available such as BLACKHAT [6], NGLUON [7], HELAC-NLO [8], GOSAM [9], and MADGRAPH5_AMC@NLO [10]. For the calculation of the one-loop matrix elements different approaches have been developed and implemented like CUTTOOLS [11], HELAC-1LOOP [12], SAMURAI [13], MADLOOP [14], OPENLOOPS [15], or RECOLA [16]. Electroweak NLO corrections, whose automation for multi-leg computations started with the development of the recursive generator RECOLA [16], have recently been implemented also in MADGRAPH5_AMC@NLO [17, 18] and OPENLOOPS [19, 20]. Finally, the packages FEYNARTS [21], FEYNCALC [22, 23], and FORMCALC [24–26] provide flexible tools to evaluate NLO amplitudes within and beyond the Standard Model, with more emphasis on generality than on high multiplicities.

The great progress in the evaluation of multi-leg one-loop amplitudes was triggered both by systematic improvements in the traditional Feynman-diagrammatic approach and by the development of new field-theoretical techniques based on generalized unitarity relations. In the latter approaches [27–33], one-loop amplitudes are directly expressed in terms of scalar integrals. The direct reduction of the amplitude to the fixed set of scalar integrals leads to numerical problems in specific regions of phase space, which are usually overcome by resorting to quadruple precision in the numerical calculation. The Feynman-diagrammatic approach and also the recent recursive methods [15, 16, 34] instead rely on tensor integrals. This allows to adapt the reduction method to the respective region of phase space, and an optimal choice avoids numerical instabilities to a large extent. The COLLIER library presented here represents a comprehensive tool for the evaluation of scalar and tensor integrals that is applicable in either of the two complementary types of approaches.

The reduction of tensor integrals to a small set of basic integrals goes back to Brown and Feynman [35], Melrose [36], and Passarino and Velt-

man [37]. Over the decades, the methods have been refined and improved by various authors [38–51]. The complete set of reduction methods presented in Refs. [43, 50] serves as the basis of the code Collier.

Eventually, the reduction of tensor integrals leads to a set of scalar integrals, which were systematically investigated for the first time in Ref. [52] by 't Hooft and Veltman. Subsequently, results were published for specific IR-singular cases both in mass and dimensional regularization [40, 53, 54], and a simplified result for the general 4-point function with real masses was derived [55]. For processes with unstable particles, in particular in the complex-mass scheme [56, 57], expressions valid for complex masses of internal particles are required. A corresponding code for the regular 4-point function has been published in Ref. [58]. More compact results for the general scalar 4-point function with complex masses were presented in Ref. [59], where results for all relevant (regular and IR-singular) cases in scattering processes, both in mass and dimensional regularization, have been presented. These results have been encoded in Collier, while the expressions implemented for the scalar 2- and 3-point functions are based on Refs. [45, 52].

Several integral libraries are already available for the calculation of one-loop scalar and tensor integrals: FF [60], LoopTools [24], QCDLoop [61], OneLOop [62], Golem95C [63], PJFry [51], and Package-X [64]. The Collier library presented here includes the complete set of tensor integrals necessary for processes with complex masses with no a-priori restriction on the number of external particles.

Collier has been developed and applied in the course of several cutting edge NLO QCD + electroweak calculations to scattering processes, comprising for instance $e^+e^- \to WW \to 4$ fermions [57, 65], $H \to WW/ZZ \to 4$ fermions [66, 67], $pp \to t\bar{t}b\bar{b}$ [68–71], $pp \to WWb\bar{b}$ [72–74], $pp \to 2\ell+\leq 2j$ [20, 75], $pp \to WW \to 4\ell$ in double-pole approximation [76], $pp \to W+\leq 3j$ [19], $pp \to t\bar{t}jj$ [77], $pp \to WWb\bar{b}H$ [78], $pp \to \mu^+\mu^-e^+e^-$ [79], and various applications with lower multiplicities in the final state. The code has also been used to calculate one-loop amplitudes in NNLO calculations [80–82], where the evaluation of one-loop integrals for collinear and/or soft external momenta is required. It is further integrated in the NLO generators OpenLoops [15] and Recola [16].

This article is organized as follows: After setting the relevant conventions used by Collier in Section 2, we outline the methodology applied to calculate the tensor integrals in Section 3. Section 4 describes the internal structure of the Collier library and Section 5 its actual usage. Our con-

clusions are given in Section 6. Finally, Appendix A provides some further details on the kinematical input used to define one-loop integrals.

2. Conventions

We consistently use the conventions of Refs. [50, 59]. The methods used for the reduction of tensor integrals have been described in Refs. [43, 50], while the implemented results for the scalar 4-point functions can be found in Ref. [59], those for the scalar 1-, 2-, and 3-point functions are based on Refs. [45, 52].

In D dimensions, one-loop tensor N-point integrals have the general form

$$T^{N,\mu_1\cdots\mu_P}(p_1,\ldots,p_{N-1},m_0,\ldots,m_{N-1}) = \frac{(2\pi\mu)^{4-D}}{\mathrm{i}\pi^2}\int \mathrm{d}^D q\,\frac{q^{\mu_1}\cdots q^{\mu_P}}{N_0 N_1\ldots N_{N-1}} \tag{1}$$

with the denominator factors

$$N_k = (q+p_k)^2 - m_k^2 + \mathrm{i}\epsilon, \qquad k=0,\ldots,N-1, \qquad p_0 = 0, \tag{2}$$

where $\mathrm{i}\epsilon$ ($\epsilon > 0$) is an infinitesimally small imaginary part. For $P = 0$, i.e. with a factor 1 instead of integration momenta in the numerator of the loop integral, (1) defines the scalar N-point integral T_0^N. Following the notation of Ref. [52], we set $T^1 = A$, $T^2 = B$, $T^3 = C$, $T^4 = D$, $T^5 = E$, $T^6 = F$, and $T^7 = G$.

In order to be able to write down tensor decompositions in a concise way, we use a notation for the basic tensor structures in which curly brackets denote symmetrization with respect to Lorentz indices in such a way that all non-equivalent permutations of the Lorentz indices on metric tensors g and a generic momentum p contribute with weight one. In covariants with n_p momenta $p_{i_j}^{\mu_j}$ ($j = 1,\ldots,n_p$) only one representative out of the $n_p!$ permutations of the indices i_j is kept. Thus, we have for example

$$\begin{aligned}
\{p\ldots p\}_{i_1\ldots i_P}^{\mu_1\ldots\mu_P} &= p_{i_1}^{\mu_1}\ldots p_{i_P}^{\mu_P},\\
\{gp\}_{i_1}^{\mu\nu\rho} &= g^{\mu\nu}p_{i_1}^{\rho} + g^{\nu\rho}p_{i_1}^{\mu} + g^{\mu\rho}p_{i_1}^{\nu},\\
\{gpp\}_{i_1 i_2}^{\mu\nu\rho\sigma} &= g^{\mu\nu}p_{i_1}^{\rho}p_{i_2}^{\sigma} + g^{\mu\rho}p_{i_1}^{\sigma}p_{i_2}^{\nu} + g^{\mu\sigma}p_{i_1}^{\nu}p_{i_2}^{\rho}\\
&\quad + g^{\nu\rho}p_{i_1}^{\sigma}p_{i_2}^{\mu} + g^{\rho\sigma}p_{i_1}^{\nu}p_{i_2}^{\mu} + g^{\nu\sigma}p_{i_1}^{\rho}p_{i_2}^{\mu},\\
\{gg\}^{\mu\nu\rho\sigma} &= g^{\mu\nu}g^{\rho\sigma} + g^{\mu\sigma}g^{\nu\rho} + g^{\mu\rho}g^{\nu\sigma}.
\end{aligned} \tag{3}$$

These basic tensor structures can be recursively defined according to

$$\{p\dots p\}^{\mu_1\dots\mu_P}_{i_1\dots i_P} = p^{\mu_1}_{i_1}\dots p^{\mu_P}_{i_P}, \tag{4}$$

$$\{\underbrace{g\dots g}_{n}p\dots p\}^{\mu_1\dots\mu_P}_{i_{2n+1}\dots i_P} = \frac{1}{n}\sum_{\substack{k,l=1\\k<l}}^{P} g^{\mu_k\mu_l}\{\underbrace{g\dots g}_{n-1}p\dots p\}^{\mu_1\dots\mu_{k-1}\mu_{k+1}\dots\mu_{l-1}\mu_{l+1}\dots\mu_P}_{i_{2n+1}\dots i_P}.$$

We decompose the general tensor integral into Lorentz-covariant structures as

$$\begin{aligned}
T^{N,\mu_1\dots\mu_P} &= \sum_{n=0}^{\left[\frac{P}{2}\right]}\sum_{i_{2n+1},\dots,i_P=1}^{N-1}\{\underbrace{g\dots g}_{n}p\dots p\}^{\mu_1\dots\mu_P}_{i_{2n+1}\dots i_P}T^{N}_{\underbrace{0\dots0}_{2n}i_{2n+1}\dots i_P}\\
&= \sum_{i_1,\dots,i_P=1}^{N-1}p^{\mu_1}_{i_1}\dots p^{\mu_P}_{i_P}T^{N}_{i_1\dots i_P} + \sum_{i_3,\dots,i_P=1}^{N-1}\{gp\dots p\}^{\mu_1\dots\mu_P}_{i_3\dots i_P}T^{N}_{00i_3\dots i_P}\\
&\quad + \sum_{i_5,\dots,i_P=1}^{N-1}\{ggp\dots p\}^{\mu_1\dots\mu_P}_{i_5\dots i_P}T^{N}_{0000i_5\dots i_P}+\dots\\
&\quad + \begin{cases}\displaystyle\sum_{i_P=1}^{N-1}\{g\dots gp\}^{\mu_1\dots\mu_P}_{i_P}T^{N}_{\underbrace{0\dots0}_{P-1}i_P}, & \text{for } P \text{ odd},\\ \{g\dots g\}^{\mu_1\dots\mu_P}T^{N}_{\underbrace{0\dots0}_{P}}, & \text{for } P \text{ even},\end{cases}
\end{aligned} \tag{5}$$

where $[P/2]$ is the largest integer number smaller or equal to $P/2$. For each metric tensor in the Lorentz-covariant tensor structure, the corresponding coefficient carries an index pair "00" and for each momentum p_{i_r} it carries the corresponding index i_r. By definition, the tensor coefficients $T^N_{i_1\dots i_P}$ are totally symmetric in the indices $i_1,\dots,i_P$.

For tensor integrals up to rank six, the decompositions more explicitly read

$$T^{N,\mu} = \sum_{i_1=1}^{N-1}p^{\mu}_{i_1}T^{N}_{i_1}, \qquad T^{N,\mu\nu} = \sum_{i_1,i_2=1}^{N-1}p^{\mu}_{i_1}p^{\nu}_{i_2}T^{N}_{i_1i_2} + g^{\mu\nu}T^{N}_{00},$$

$$T^{N,\mu\nu\rho} = \sum_{i_1,i_2,i_3=1}^{N-1}p^{\mu}_{i_1}p^{\nu}_{i_2}p^{\rho}_{i_3}T^{N}_{i_1i_2i_3} + \sum_{i_1=1}^{N-1}\{gp\}^{\mu\nu\rho}_{i_1}T^{N}_{00i_1},$$

$$
\begin{aligned}
T^{N,\mu\nu\rho\sigma} &= \sum_{i_1,i_2,i_3,i_4=1}^{N-1} p_{i_1}^{\mu} p_{i_2}^{\nu} p_{i_3}^{\rho} p_{i_4}^{\sigma} T^N_{i_1 i_2 i_3 i_4} + \sum_{i_1,i_2=1}^{N-1} \{gpp\}^{\mu\nu\rho\sigma}_{i_1 i_2} T^N_{00 i_1 i_2} \\
&\quad + \{gg\}^{\mu\nu\rho\sigma} T^N_{0000}, \\
T^{N,\mu\nu\rho\sigma\tau} &= \sum_{i_1,i_2,i_3,i_4,i_5=1}^{N-1} p_{i_1}^{\mu} p_{i_2}^{\nu} p_{i_3}^{\rho} p_{i_4}^{\sigma} p_{i_5}^{\tau} T^N_{i_1 i_2 i_3 i_4 i_5} \\
&\quad + \sum_{i_1,i_2,i_3=1}^{N-1} \{gppp\}^{\mu\nu\rho\sigma\tau}_{i_1 i_2 i_3} T^N_{00 i_1 i_2 i_3} + \sum_{i_1=1}^{N-1} \{ggp\}^{\mu\nu\rho\sigma\tau}_{i_1} T^N_{0000 i_1}, \\
T^{N,\mu\nu\rho\sigma\tau\alpha} &= \sum_{i_1,i_2,i_3,i_4,i_5,i_6=1}^{N-1} p_{i_1}^{\mu} p_{i_2}^{\nu} p_{i_3}^{\rho} p_{i_4}^{\sigma} p_{i_5}^{\tau} p_{i_6}^{\alpha} T^N_{i_1 i_2 i_3 i_4 i_5 i_6} \\
&\quad + \sum_{i_1,i_2,i_3,i_4=1}^{N-1} \{gpppp\}^{\mu\nu\rho\sigma\tau\alpha}_{i_1 i_2 i_3 i_4} T^N_{00 i_1 i_2 i_3 i_4} \\
&\quad + \sum_{i_1,i_2=1}^{N-1} \{ggpp\}^{\mu\nu\rho\sigma\tau\alpha}_{i_1 i_2} T^N_{0000 i_1 i_2} + \{ggg\}^{\mu\nu\rho\sigma\tau\alpha} T^N_{000000}. \qquad (6)
\end{aligned}
$$

UV- or IR-singular integrals are represented in dimensional regularization, where $D = 4 - 2\epsilon$, as

$$
\begin{aligned}
T^N &= \tilde{T}^N_{\rm fin} + a^{\rm UV} \left(\Delta_{\rm UV} + \ln \frac{\mu^2_{\rm UV}}{Q^2} \right) \\
&\quad + a_2^{\rm IR} \left(\Delta^{(2)}_{\rm IR} + \Delta^{(1)}_{\rm IR} \ln \frac{\mu^2_{\rm IR}}{Q^2} + \frac{1}{2} \ln^2 \frac{\mu^2_{\rm IR}}{Q^2} \right) + \tilde{a}_1^{\rm IR} \left(\Delta^{(1)}_{\rm IR} + \ln \frac{\mu^2_{\rm IR}}{Q^2} \right) \\
&= T^N_{\rm fin}(\mu^2_{\rm UV}, \mu^2_{\rm IR}) + a^{\rm UV} \Delta_{\rm UV} + a_2^{\rm IR} \left(\Delta^{(2)}_{\rm IR} + \Delta^{(1)}_{\rm IR} \ln \mu^2_{\rm IR} \right) + a_1^{\rm IR} \Delta^{(1)}_{\rm IR} \qquad (7)
\end{aligned}
$$

with

$$
\begin{aligned}
\Delta_{\rm UV} &= \frac{c(\epsilon_{\rm UV})}{\epsilon_{\rm UV}}, \quad c(\epsilon) = \Gamma(1+\epsilon)(4\pi)^{\epsilon}, \\
\Delta^{(2)}_{\rm IR} &= \frac{c(\epsilon_{\rm IR})}{\epsilon^2_{\rm IR}}, \quad \Delta^{(1)}_{\rm IR} = \frac{c(\epsilon_{\rm IR})}{\epsilon_{\rm IR}}. \qquad (8)
\end{aligned}
$$

We make explicit all UV and IR poles as well as the terms involving the corresponding mass scales $\mu_{\rm UV}$ and $\mu_{\rm IR}$. We further factor out the term

$c(\epsilon) = \Gamma(1+\epsilon)(4\pi)^\epsilon = 1 + \mathcal{O}(\epsilon)$ and absorb it into the definitions of $\Delta_{\rm UV}$, $\Delta^{(2)}_{\rm IR}$, and $\Delta^{(1)}_{\rm IR}$. In order to avoid logarithms of dimensionful quantities in the first equation in (7), we have singled out an auxiliary scale Q which is implicitly fixed by the masses and momenta entering the respective loop integral. The output delivered by COLLIER corresponds to the last line of (7), including the terms proportional to $a^{\rm UV}$, $a_2^{\rm IR}$, and $a_1^{\rm IR}$. The parameters $\mu^2_{\rm UV}$, $\mu^2_{\rm IR}$, $\Delta_{\rm UV}$, $\Delta^{(2)}_{\rm IR}$, and $\Delta^{(1)}_{\rm IR}$ can be chosen freely by the user, but do not influence UV- and IR-finite quantities. Note that we distinguish between singularities of IR and UV origin. By default $\Delta_{\rm UV}$, $\Delta^{(2)}_{\rm IR}$, and $\Delta^{(1)}_{\rm IR}$ are set to zero and the output equals $T^N_{\rm fin}(\mu^2_{\rm UV}, \mu^2_{\rm IR})$. Setting the Δ's different from zero, the impact of the poles in ϵ can be numerically simulated in the results.

By default IR- and UV-singular integrals are calculated in dimensional regularization. Collinear singularities can also be regularized with masses. To this end, the corresponding masses, called $\overline{m}_i$ in the following, must be declared *small* in the initialization. Moreover, in all subroutine calls the respective mass parameters must have exactly the same (not necessarily small) numerical value as specified in the initialization. The *small* masses are treated as infinitesimally small in the scalar and tensor functions, and only in mass-singular logarithms the finite values are kept. For soft singularities of Abelian type, i.e. when $a_2^{\rm IR} = 0$ and collinear singularities are regularized with masses $\overline{m}_i$, the parameter $\mu_{\rm IR}$ can be interpreted as an infinitesimal photon or gluon mass after setting the parameter $\Delta^{(1)}_{\rm IR}$ to zero.[1]

Varying the parameters $\mu^2_{\rm UV}$, $\mu^2_{\rm IR}$, $\Delta_{\rm UV}$, $\Delta^{(2)}_{\rm IR}$, and $\Delta^{(1)}_{\rm IR}$ allows to check the cancellation of singularities. Moreover, choosing appropriate values for $\Delta_{\rm UV}$, $\Delta^{(2)}_{\rm IR}$, and $\Delta^{(1)}_{\rm IR}$ allows the user to switch to different conventions concerning the extraction of the prefactor $c(\epsilon)$ in (8). For instance, in Ref. [61] the ϵ-dependent prefactor in the one-loop integral is π^ϵ/r_Γ with $r_\Gamma = \Gamma^2(1-\epsilon)\Gamma(1+\epsilon)/\Gamma(1-2\epsilon)$ in contrast to the prefactor $(2\pi)^{2\epsilon}$ in our convention (1). We thus have to replace our factor $c(\epsilon)$ by

$$\frac{c(\epsilon)}{r_\Gamma(4\pi)^\epsilon} = \frac{\Gamma(1+\epsilon)}{r_\Gamma} = \frac{\Gamma(1-2\epsilon)}{\Gamma^2(1-\epsilon)} = 1 + \epsilon^2\frac{\pi^2}{6} + \mathcal{O}(\epsilon^3), \tag{9}$$

in order to obtain the singular integrals in the conventions of Ref. [61]. This is equivalent to the recipe of replacing $\Delta^{(2)}_{\rm IR} \to \Delta^{(2)}_{\rm IR} + \pi^2/6$ while keeping $\Delta_{\rm UV}$

[1] Formally this means that the scales obey the hierarchy $\mu_{\rm IR} \ll \overline{m}_i \ll M$ in the analytical derivation of the integrals, where M is any other scale involved in the calculation.

and $\Delta_{\rm IR}^{(1)}$ unchanged in our calculation. After this change, our parameters $\Delta_{\rm UV}$, $\Delta_{\rm IR}^{(2)}$, and $\Delta_{\rm IR}^{(1)}$ simply correspond to the poles $1/\epsilon$, $1/\epsilon^2$, and $1/\epsilon$ of Ref. [61], respectively.

3. Implemented methods

3.1. Calculation of tensor coefficients

The method used to evaluate a tensor integral depends on the number N of its propagators. For $N = 1, 2$, we use explicit numerically stable expressions [37, 50].

For $N = 3, 4$, all tensor integrals are numerically reduced to the basic scalar integrals, which are calculated using analytical expressions given in Refs. [45, 52, 59]. By default, the reduction is performed via standard Passarino–Veltman reduction [37]. In regions of the phase space where a Gram determinant becomes small, Passarino–Veltman reduction becomes unstable. In these regions we use the dedicated recursive expansion methods described in Ref. [50] plus some additional variants. All these methods have been implemented in COLLIER to arbitrary order in the expansion parameter. In order to decide on the method to use for a certain phase-space point, the following procedure is applied:

1. Passarino–Veltman reduction is used by default, and its reliability is assessed by first assigning appropriate accuracies for the scalar integrals and subsequently estimating the error propagation during the reduction. If the resulting error estimate $\Delta T^N(\widehat{P})$ for the integrals of the highest required rank $\widehat{P}$ is smaller than a predefined precision tag $\eta_{\rm req}$ (required precision), the result is kept and returned to the user.
2. In case step 1 does not provide sufficient accuracy, which typically means that the tensor integral involves small Gram determinants of external momenta, COLLIER switches to dedicated expansions. In order to decide which expansion is appropriate, an a-priori error estimate $\Delta T^N_{\rm prelim}(P)$ for the coefficients $T^N_{i_1\ldots i_P}$ is constructed up to the highest required rank $\widehat{P}$ for the different methods. The estimates are based on an assessment of the expected accuracy of the expansion and a simplified propagation of errors from the required scalar integrals. The expansion method with the smallest $\Delta T^N_{\rm prelim}(\widehat{P})$ is chosen. During the actual calculation of this expansion a more realistic precision $\Delta T^N(P)$

is assessed by analysing the correction of the last iteration. If the predefined precision tag $\eta_{\rm req}$ is reached, the result is kept and returned to the user. Otherwise the expansion stops if either a predefined iteration depth is reached, or if the accuracy does not increase anymore from one iteration step to the next.

3. If step 2 does not provide sufficient accuracy for the selected method, it is repeated for the other expansion methods that promise convergence by sufficiently small $\Delta T^N_{\rm prelim}(\widehat{P})$. If the predefined precision tag is reached after any of these repetitions, the result is kept and returned to the user.
4. If neither Passarino–Veltman reduction nor any of the tried expansions delivers results that match the target accuracy, the results of the method with the smallest error estimate $\Delta T^N(P)$ is returned to the user.

In this way stable results are obtained for almost all phase-space points, ensuring reliable Monte Carlo integrations.

For $N = 5, 6$, tensor integrals are directly reduced to integrals with lower rank and lower N following Refs. [43, 50], i.e. without involving inverse Gram determinants. For $N \geq 7$ a modification of the reduction of 6-point tensor integrals is applied as described in Section 7 of Refs. [50] [see text after (7.10) there].

3.2. Calculation of full tensors

While the methods described so far are formulated in the literature in terms of the Lorentz-invariant coefficients $T^N_{i_1\ldots i_P}$, a new generation of NLO generators, such as OpenLoops and Recola, needs the components of the full tensors $T^{N,\mu_1\cdots\mu_P}$. To this end, an efficient algorithm has been implemented in Collier to construct the tensors $T^{N,\mu_1\cdots\mu_P}$ from the coefficients $T^N_{i_1\ldots i_P}$. It performs a recursive calculation of those tensor structures (4) which are built exclusively from momenta. Non-vanishing components of tensor structures involving metric tensors are then obtained recursively by adding pairwise equal Lorentz indices to tensor structures with less metric tensors, taking into account the combinatorics of the indices and the signs induced by the metric tensors. The relevant combinatorial factors are calculated and tabulated during the initialization of Collier.

The numbers of invariant coefficients $T^N_{i_1\ldots i_P}$ and tensor components $T^{N,\mu_1\cdots\mu_P}$ are compared in Table 1. For $N \leq 4$ the number of invariant

coefficients for	$\widehat{P}=0$	$\widehat{P}=1$	$\widehat{P}=2$	$\widehat{P}=3$	$\widehat{P}=4$	$\widehat{P}=5$	$\widehat{P}=6$
$N=3$	1	3	7	13	22	34	50
$N=4$	1	4	11	24	46	80	130
$N=5$	1	5	16	40	86	166	296
$N=6$	1	6	22	62	148	314	610
$N=7$	1	7	29	91	239	553	1163
components	1	5	15	35	70	126	210

Table 1: Number $n_c(N,\widehat{P})$ of invariant coefficients $T^N_{i_1\dots i_P}$ for $N=3,\dots,7$ and rank $P\leq\widehat{P}=0,\dots,6$ (rows 2–6) and number $n_t(\widehat{P})$ of independent tensor components $T^{N,\mu_1\cdots\mu_P}$ for rank $P\leq\widehat{P}=0,\dots,6$ (last row).

$$B_{i_1\dots i_P} \qquad C_{i_1\dots i_P} \qquad D_{i_1\dots i_P} \qquad E_{i_1\dots i_P} \qquad F_{i_1\dots i_P} \qquad G_{i_1\dots i_P} \qquad \dots$$

$$B^{\mu_1\cdots\mu_P} \qquad C^{\mu_1\cdots\mu_P} \qquad D^{\mu_1\cdots\mu_P} \qquad E^{\mu_1\cdots\mu_P} \qquad F^{\mu_1\cdots\mu_P} \qquad G^{\mu_1\cdots\mu_P} \qquad \dots$$

Figure 1: Reduction chains in COLLIER: For $N\geq 6$ reduction can be performed at the tensor level.

coefficients is smaller than the number of tensor components, which is a basic precondition of the Passarino–Veltman reduction method. For $N\geq 5$, on the other hand, the situation is reversed. Actually, the reduction for $N\geq 6$ presented in (7.7) of Ref. [50] has been derived in terms of full tensors. Its translation to tensor coefficients requires a symmetrization and the resulting coefficients are not unique because of the redundant number of tensor structures. Therefore, for the calculation of the tensors $T^{N,\mu_1\cdots\mu_P}$ the reduction for $N\geq 6$ has been implemented in COLLIER also directly at the tensor level without resorting to a covariant decomposition.

Whereas for $N\leq 5$ the recursion exclusively proceeds at the coefficient level, with $T^{N,\mu_1\cdots\mu_P}$ constructed afterwards from the respective coefficients $T^N_{i_1\dots i_P}$, for $N\geq 6$ the reduction can alternatively be performed at the level of the tensors. This means that, in order to calculate a tensor integral with $N\geq 6$, any N_{tenred} with $5<N_{\text{tenred}}\leq N$ can be chosen such that the recursive calculation is performed at the coefficient level for $N<N_{\text{tenred}}$ and

at tensor level for $N \geq N_{\text{tenred}}$. The transition from coefficients to tensors then takes place at $N_{\text{tenred}} - 1$. The possible reduction chains are illustrated in Figure 1.

4. Structure of the library

The structure of COLLIER is illustrated schematically in Figure 2. The core of the library consists of the building blocks `COLI` and `DD`. They constitute two independent implementations of the scalar integrals T_0^N and the Lorentz-invariant coefficients $T^N_{i_1\ldots i_P}$ employing the methods described in the previous section. The building block `tensors` provides routines for the construction of the tensors $T^{N,\mu_1\ldots\mu_P}$ from the coefficients $T^N_{i_1\ldots i_P}$, as well as for a direct reduction of N-point integrals for $N \geq 6$ at the tensor level. The user interacts with the basic routines of `COLI`, `DD`, and `tensors` via the global interface of COLLIER. It provides routines to set or extract values of the parameters in `COLI` and `DD` as well as routines to calculate the tensor coefficients $T^N_{i_1\ldots i_P}$ or tensor components $T^{N,\mu_1\ldots\mu_P}$. The user can choose whether the `COLI` or the `DD` branch shall be used. It is also possible to calculate each integral with both branches and cross-check the results.

In a typical evaluation of a one-loop matrix element, a tensor integral is called several times with the same kinematical input: On the one hand, a single user call of an N-point integral with $P \geq 2$ leads to recursive internal calls of lower N'-point integrals, and for $N' \leq N - 2$ the same integral is reached through more than one path in the reduction tree. On the other hand, different user calls and their reductions typically involve identical tensor integrals. In order to avoid multiple calculations of the same integral, the sublibraries of COLLIER are linked to a global cache system which works as follows: A parameter N_{ext} numerates external integral calls, while for the book-keeping of internal calls a binary identifier id is propagated during the reduction. A pointer is assigned to each index pair (N_{ext}, id). During the evaluation of the first phase-space points the arguments of the corresponding function calls are compared, and pairs (N_{ext}, id) with identical arguments are pointed to the same address in the cache. For later phase-space points the result of the first call of an integral is written to the cache and read out in subsequent calls pointing to the same address. Use of the external cache system is optional and crucially requires that all calls for tensor integrals are made exactly in the same order for each phase-space point in a Monte Carlo integration, after an initialization that signals the beginning of the matrix-

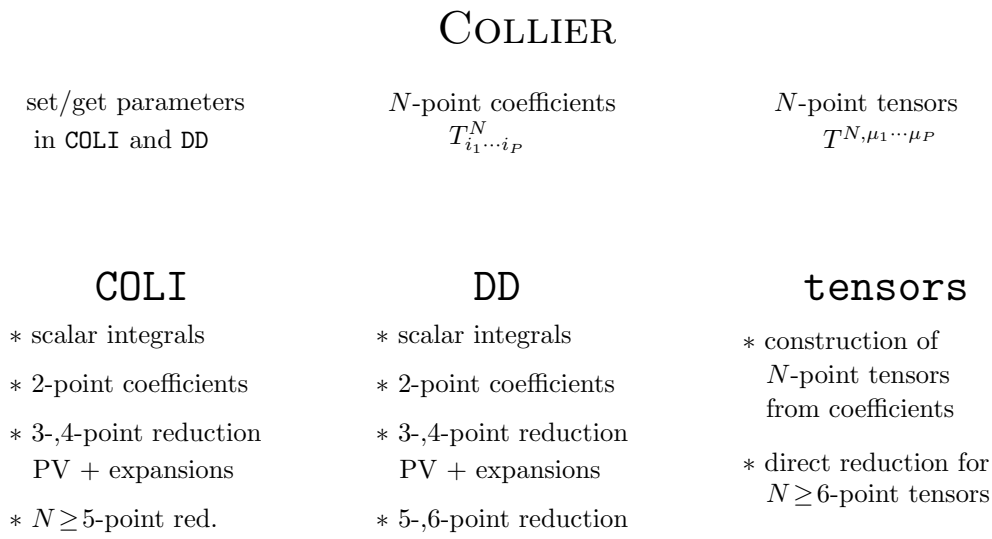

Figure 2: Structure of the library COLLIER.

element calculation for the respective phase-space point. Moreover, internal parameters must not be changed between the first and the last integral call in each event.

5. Usage of the library

5.1. Installation

For the installation of the COLLIER library the package `collier-`v`.tar.gz` is needed[2] (where v stands for the version of the library, e.g. $v = 1.2$), and the CMAKE build system should be installed. Since COLLIER is a stand-alone Fortran95 code, no additional libraries are required.

To start installation, `gunzip` and `untar collier-`v`.tar.gz` which will unpack into the directory `./COLLIER-`v containing the following files and directories

[2]The package can be downloaded from http://collier.hepforge.org.

- `CMakeLists.txt`: CMake makefile to produce the Collier library,
- `src`: Collier source directory, containing the main source files of Collier and further source files in the subdirectories
  - `COLI`: containing the files of the `COLI` branch,
  - `DDlib`: containing the files of the `DD` branch,
  - `tensors`: containing the files for tensor construction and direct tensor reduction,
  - `Aux`: containing auxiliary files,
- `build`: build directory, where CMake puts all necessary files for the creation of the library, such as object files,
- `modules`: empty directory for fortran module files,
- `demos`: directory with demo routines illustrating the use of Collier,
- `COPYING`: file with copyright information.

The Collier library is generated by changing to the directory `build` and issuing `"cmake .."` followed by `"make"`:

```
"cd build"
"cmake .."
"make" .
```

This requires CMake to be installed. Some information on individual configurations can be found at the top of the file `CMakeLists.txt`. By default `cmake` sets up the makefile in such a way that a dynamic library will be generated. If a static library is desired, this can be controlled by issuing

```
"cmake -Dstatic=ON .."
```

in the directory `COLLIER-`v.

If no options are specified, `cmake` automatically searches for installed Fortran compilers and chooses a suited one. The use of a particular compiler, e.g. the `ifort` compiler, can be enforced by

```
"cmake -DCMAKE_Fortran_COMPILER=ifort .." .
```

The full path to a compiler may be given.

Once the `Makefile` has been generated, the command `make` will generate the dynamic library `libcollier.so` or the static library `libcollier.a` in the directory `COLLIER-`v which can be linked to the user's program.

To create the executables for the sample programs (see Section 5.7) in the directory `demos`, the commands

```
"make demo"
"make democache"
```

should be issued in the directory `COLLIER-`v`/build`.

All files created by the `"make"` command can be discarded with

```
"make clean"
```

in the directory `COLLIER-`v`/build`, and all files produced by `"cmake"` can be eliminated by removing all files in the directory `COLLIER-`v`/build`. To clean up completely, all files generated by `"cmake"` should be removed.

5.2. General usage instructions

In order to use COLLIER in a `Fortran` program, the corresponding modules located in `COLLIER-`v`/modules` have to be loaded by including the line

```
use COLLIER
```

in the preamble of the respective code, and the library `libcollier.so` or `libcollier.a` in the directory `COLLIER-`v has to be supplied to the linker. This gives access to the public functions and subroutines of the COLLIER library described in the following subsections. The names of all these routines end with the suffix "`_cll`". This name convention is supposed to avoid conflicts with routine names present in the master program and increases readability by allowing for an easy identification of command lines referring to the COLLIER library.

Before COLLIER can be used to calculate tensor integrals, it must be initialized by calling

```
subroutine Init_cll(Nmax,rin,folder_name,noreset)
integer Nmax                    : maximal # of loop propagators
```

`integer,optional rin`	: maximal rank of loop integrals
`character,optional folder_name`	: name of folder for output
`logical,optional noreset`	: no new output folder and files ,

where the first argument `Nmax` is mandatory, while the other arguments `rin`, `folder_name` and `noreset` are optional. With the argument `Nmax` the user must specify the maximal number $N_{\max}$ of loop propagators of the tensor integrals $T^{N,P}$ he is going to calculate ($N \leq N_{\max}$), while with the optional second argument `rin` he can specify the maximal rank $P_{\max}$ ($P \leq P_{\max}$).[3] If the argument `rin` is omitted, the maximal rank is set to `Nmax` which is sufficient for integrals in renormalizable theories. The arguments `Nmax` and `rin` determine the size of internal tables generated by `Collier`; very large values for these parameters can thus affect the amount of allocated memory and the computing time for the integrals. As an optional third argument the user can pass a string `folder_name` to `Init_cll`, which then creates (or overwrites) a folder with the specified name in the current directory. The output of COLLIER will be directed to that folder. If the second argument is left out, the folder is named `'output_cll'` by default. The creation of an output folder can be suppressed by passing an empty string `folder_name=''` to `Init_cll`. In this case no output will be created by COLLIER except for the one related to the initialization and to fatal errors written to the standard output channel `stdout_cll=6`. In a second or any subsequent call of `Init_cll`, the optional fourth argument `noreset`, if present and set to `.true.`, causes that output folder and files are not re-created, but that the program continues writing in the existing files. The optional flag `noreset` is ignored in the very first call of `Init_cll`.

The call of `Init_cll` sets all internal parameters to default values specified in Table 2. In later calls certain parameters specified in Section 5.6 are prevented from a reinitialization if the optional argument `noreset` is flagged `.true.`. After the initialization, many of these parameters can be freely set to different values according to the needs of the user. To this end, COLLIER provides a subroutine `Set`X`_cll` for every parameter X, and subroutines `SwitchOn`Y`_cll`, `SwitchOff`Y`_cll` for every flag Y. To read out the current

[3] In the `DD` branch of COLLIER, N-point integrals are presently only implemented up to $N_{\max} = 6$, with 5-point functions supported up to rank 5 and 6-point functions supported up to rank 6. This is sufficient for any N-point integral with $N \leq 6$ appearing in renormalizable theories.

value of a parameter X a routine `get`X`_cll` is available for most parameters. The parameters that can be modified by the user and the respective routines are described in detail in Section 5.4.

After the initialization and a potential redefinition of internal parameters, COLLIER is ready to calculate tensor integrals. The generic subroutine `TN_cll` calculates the coefficients $T^N_{i_1\ldots i_P}$ of the Lorentz-covariant decomposition (5) of the tensor integrals $T^{N,P}$, while `TNten_cll` returns the tensor components $T^{N,\mu_1\cdots\mu_P}$. Alternatively, there are specific subroutines `A_cll`, `B_cll`, ..., `G_cll` and `Aten_cll`, `Bten_cll`, ..., `Gten_cll` for the 1-, 2-, ..., 7-point integrals, as well as `A0_cll`, `B0_cll`, ..., `D0_cll` for the scalar integrals. Momentum derivatives of 2-point coefficients, typically needed to calculate renormalization constants, can be calculated up to arbitrary rank using the generic subroutine `DB_cll`, or for the lowest ranks using the specific subroutines `DB0_cll`, `DB1_cll`, `DB00_cll`, and `DB11_cll`. More information on the subroutines for the calculation of tensor integrals is given in Section 5.3.

A typical application of COLLIER is to provide the one-loop tensor integrals within an NLO Monte Carlo generator. In this case, the master program performs a loop over Monte Carlo events, and for each event it calculates the one-loop matrix element employing COLLIER to evaluate the corresponding set of tensor integrals. In this context, the subroutine

```
subroutine InitEvent_cll(cacheNr)
integer,optional cacheNr : # of cache
```

should be called for each event before the tensor integrals are computed. This call will reinitialize the error flag and the accuracy flag of COLLIER which can be read out at the end of the sequence of tensor integral calls to obtain global information on the status of the calculations. If the cache system is used, the call of `InitEvent_cll` is mandatory to reinitialize the cache for every Monte Carlo event. In the case of multiple caches the respective cache number `cacheNr` has to be passed to `InitEvent_cll` as an optional argument. More information on the use of the cache system can be found in Section 5.5.

To help the user to get familiar with the basic usage of COLLIER, two sample programs `demo` and `democache` are distributed together with the library. They are described in Section 5.7.

parameter	type	set with	default
`mode`	`integer` $\in \{1,2,3\}$	`SetMode_cll`	1
$\eta_{\rm req}$	`double precision`	`SetReqAcc_cll`	`1d-8`
$\eta_{\rm crit}$	`double precision`	`SetCritAcc_cll`	`1d-1`
$\eta_{\rm check}$	`double precision`	`SetCheckAcc_cll`	`1d-2`
$\mu^2_{\rm UV}$	`double precision`	`SetMuUV2_cll`	`1d0`
$\mu^2_{\rm IR}$	`double precision`	`SetMuIR2_cll`	`1d0`
$\Delta_{\rm UV}$	`double precision`	`SetDeltaUV_cll`	`0d0`
$\Delta^{(1)}_{\rm IR}, \Delta^{(2)}_{\rm IR}$	`double precision`	`SetDeltaIR_cll`	`0d0,0d0`
$\{\overline{m}^2_1, \ldots, \overline{m}^2_{n_{\rm reg}}\}$	`double complex` ($n_{\rm reg}$)	`SetMinf2_cll`	$\{\}$
$\sigma_{\rm stop}$	`integer` < 0	`SetErrStop_cll`	-8
$N_{\rm tenred}$	`integer` ≥ 6	`SetTenRed_cll`	6
$n_{\rm cache}$	`integer` ≥ 0	`InitCacheSystem_cll`	0
$N^{\rm max}_{\rm cache}$	`integer` ($n_{\rm cache}$) ≥ 1	`SetCacheLevel_cll`	–
$n_{\rm err}$	`integer` ≥ 0	`SetMaxErrOut_cll`	100
$n_{\rm err,COLI}$	`integer` ≥ 0	`SetMaxErrOutCOLI_cll`	100
$n_{\rm err,DD}$	`integer` ≥ 0	`SetMaxErrOutDD_cll`	100
$n_{\rm inf}$	`integer` ≥ 0	`SetMaxInfOut_cll`	1000
$n^{N,\rm max}_{\rm check}$	`integer` ($N_{\rm max}$) ≥ 0	`SetMaxCheck_cll`	$\{50, ..., 50\}$
$n^{B',\rm max}_{\rm check}$	`integer` ≥ 0	`SetMaxCheckDB_cll`	50
$n^{N,\rm max}_{\rm crit}$	`integer` ($N_{\rm max}$) ≥ 0	`SetMaxCrit_cll`	$\{50, ..., 50\}$
$n^{B',\rm max}_{\rm crit}$	`integer` ≥ 0	`SetMaxCritDB_cll`	50
$\widehat{P}^{\rm max}$	`integer` ≥ 6	`SetRitmax_cll`	14
`outlev`	`integer` $\in \{0,1,2\}$	`SetInfOutLev_cll`	2

Table 2: List of COLLIER parameters.

5.3. Calculation of tensor integrals

For the tensor integrals, COLLIER provides routines that deliver the coefficients $T^N_{i_1\ldots i_P}$ of the Lorentz-covariant decomposition (5) and routines that deliver the components of the tensors $T^{N,\mu_1\ldots\mu_P}$.

The tensor coefficients $T^N_{i_1\ldots i_P}$ are represented by N-dimensional arrays of type `double complex` with the following convention:

$$\texttt{TN}(n_0,n_1,n_2,\ldots,n_{N-1}) = T^N_{\underbrace{0\ldots0}_{2n_0}\underbrace{1\ldots1}_{n_1}\underbrace{2\ldots2}_{n_2}\ldots\underbrace{N-1\ldots N-1}_{n_{N-1}}}. \tag{10}$$

In this way, all tensor coefficients $T^N_{i_1\ldots i_P}$ with $P=0,\ldots,\widehat{P}$ up to a given rank $\widehat{P}$ are stored within the same array

$$\texttt{double complex TN}(0\!:\![\widehat{P}/2],\underbrace{0\!:\!\widehat{P},\ldots,0\!:\!\widehat{P}}_{N-1}).$$

Note that identical coefficients $T^N_{i_1\ldots i_P}$, related to each other through a permutation of the indices $\{i_1,\ldots,i_P\}$, are represented by the same entry of the array `TN`. As an example, the mapping between the tensor coefficients $D_{i_1\ldots i_P}$ and the array `D` is explicitly written down in Table 3 for the case of the 4-point coefficients up to rank $\widehat{P}=4$.

Alternatively, the tensor coefficients of the N-point integrals up to rank $\widehat{P}$ can be obtained as a one-dimensional array

$$\texttt{double complex TN1}(n_c(N,\widehat{P})),$$

where $n_c(N,\widehat{P})$ is the total number of coefficients $T^N_{i_1\ldots i_P}$ with $i_1\le i_2\le\ldots\le i_P$ and $P\le\widehat{P}$. For $N=1,\ldots,7$ and $\widehat{P}=0,\ldots,6$ the explicit values of $n_c(N,\widehat{P})$ are given in Table 1. The tensor coefficients are inserted in the array `TN1` with ascending rank P from $P=0$ to $P=\widehat{P}$. Coefficients $T^N_{i_1\ldots i_P}$ and $T^N_{j_1\ldots j_P}$ of equal rank are ordered according to the first indices i_k,j_k in which they differ. For the 4-point coefficients up to rank $\widehat{P}=4$, the ordering can be read off from Table 3. Note that due to the `Fortran` limitation of arrays to rank 7, the coefficients of N-point integrals with $N\ge8$ can only be represented in the format of a one-dimensional array.

The full tensor integrals $T^{N,\mu_1\ldots\mu_P}$ are represented by 4-dimensional arrays of type `double complex` with the following convention:

$$\texttt{TNten}(n_0,n_1,n_2,n_3) = T^{N,\overbrace{0\ldots0}^{n_0}\overbrace{1\ldots1}^{n_1}\overbrace{2\ldots2}^{n_2}\overbrace{3\ldots3}^{n_3}}. \tag{11}$$

D_0	D(0,0,0,0)	D_{113}	D(0,2,0,1)	D_{1112}	D(0,3,1,0)
D_1	D(0,1,0,0)	D_{122}	D(0,1,2,0)	D_{1113}	D(0,3,0,1)
D_2	D(0,0,1,0)	D_{123}	D(0,1,1,1)	D_{1122}	D(0,2,2,0)
D_3	D(0,0,0,1)	D_{133}	D(0,1,0,2)	D_{1123}	D(0,2,1,1)
D_{00}	D(1,0,0,0)	D_{222}	D(0,0,3,0)	D_{1133}	D(0,2,0,2)
D_{11}	D(0,2,0,0)	D_{223}	D(0,0,2,1)	D_{1222}	D(0,1,3,0)
D_{12}	D(0,1,1,0)	D_{233}	D(0,0,1,2)	D_{1223}	D(0,1,2,1)
D_{13}	D(0,1,0,1)	D_{333}	D(0,0,0,3)	D_{1233}	D(0,1,1,2)
D_{22}	D(0,0,2,0)	D_{0000}	D(2,0,0,0)	D_{1333}	D(0,1,0,3)
D_{23}	D(0,0,1,1)	D_{0011}	D(1,2,0,0)	D_{2222}	D(0,0,4,0)
D_{33}	D(0,0,0,2)	D_{0012}	D(1,1,1,0)	D_{2223}	D(0,0,3,1)
D_{001}	D(1,1,0,0)	D_{0013}	D(1,1,0,1)	D_{2233}	D(0,0,2,2)
D_{002}	D(1,0,1,0)	D_{0022}	D(1,0,2,0)	D_{2333}	D(0,0,1,3)
D_{003}	D(1,0,0,1)	D_{0023}	D(1,0,1,1)	D_{3333}	D(0,0,0,4)
D_{111}	D(0,3,0,0)	D_{0033}	D(1,0,0,2)		
D_{112}	D(0,2,1,0)	D_{1111}	D(0,4,0,0)		

Table 3: Mapping between tensor coefficients $D_{i_1 \ldots i_P}$ ($P \leq 4$) and elements $\mathtt{D}(n_0, n_1, n_2, n_3)$ of the array $\mathtt{D}(0\!:\!2, 0\!:\!4, 0\!:\!4, 0\!:\!4)$. The mapping onto the elements of the one-dimensional array representation $\mathtt{D1}(46)$ is obtained by numerating the coefficients in the table starting from the top left entry downwards.

In this way, all tensor components $T^{N,\mu_1 \cdots \mu_P}$ with $P = 0, \ldots, \widehat{P}$ up to a given rank $\widehat{P}$ are stored within the same array

$$\texttt{double complex TNten}(0\!:\!\widehat{P}, 0\!:\!\widehat{P}, 0\!:\!\widehat{P}, 0\!:\!\widehat{P}).$$

Note that identical components $T^{N,\mu_1 \cdots \mu_P}$, related to each other through a permutation of the indices $\{\mu_1, \ldots, \mu_P\}$, are represented by the same entry of the array `TNten`. The mapping between the tensor components $T^{\mu_1 \cdots \mu_P}$ and the array `TNten` is explicitly written down in Table 4 up to rank $\widehat{P} = 3$.

Alternatively, the tensor components of the N-point integrals up to rank $\widehat{P}$ can be obtained as a one-dimensional array

$$\texttt{double complex TNten1}(n_t(\widehat{P})),$$

T_0	TNten(0,0,0,0)	T^{22}	TNten(0,0,2,0)	T^{033}	TNten(1,0,0,2)
T^0	TNten(1,0,0,0)	T^{23}	TNten(0,0,1,1)	T^{111}	TNten(0,3,0,0)
T^1	TNten(0,1,0,0)	T^{33}	TNten(0,0,0,2)	T^{112}	TNten(0,2,1,0)
T^2	TNten(0,0,1,0)	T^{000}	TNten(3,0,0,0)	T^{113}	TNten(0,2,0,1)
T^3	TNten(0,0,0,1)	T^{001}	TNten(2,1,0,0)	T^{122}	TNten(0,1,2,0)
T^{00}	TNten(2,0,0,0)	T^{002}	TNten(2,0,1,0)	T^{123}	TNten(0,1,1,1)
T^{01}	TNten(1,1,0,0)	T^{003}	TNten(2,0,0,1)	T^{133}	TNten(0,1,0,2)
T^{02}	TNten(1,0,1,0)	T^{011}	TNten(1,2,0,0)	T^{222}	TNten(0,0,3,0)
T^{03}	TNten(1,0,0,1)	T^{012}	TNten(1,1,1,0)	T^{223}	TNten(0,0,2,1)
T^{11}	TNten(0,2,0,0)	T^{013}	TNten(1,1,0,1)	T^{233}	TNten(0,0,1,2)
T^{12}	TNten(0,1,1,0)	T^{022}	TNten(1,0,2,0)	T_{333}	TNten(0,0,0,3)
T^{13}	TNten(0,1,0,1)	T^{023}	TNten(1,0,1,1)		

Table 4: Mapping between tensor components $T^{\mu_1\cdots\mu_P}$ ($P \leq 3$) and elements $\texttt{Tten}(n_0, n_1, n_2, n_3)$ of the array $\texttt{TNten}(0:3, 0:3, 0:3, 0:3)$. The mapping onto the elements of the one-dimensional array representation $\texttt{TNten1}(35)$ is obtained by numerating the coefficients in the table starting from the top left entry downwards.

where $n_t(\widehat{P})$ is the total number of tensor components $T^{N,\mu_1\cdots\mu_P}$ with $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_P$ and $P \leq \widehat{P}$. For $\widehat{P} = 0, \ldots, 6$ the explicit values of $n_t(\widehat{P})$ are given in Table 1. The tensor components are inserted in the array TNten1 with ascending rank P from $P = 0$ to $P = \widehat{P}$. Components $T^{N,\mu_1\cdots\mu_P}$ and $T^{N,\nu_1\cdots\nu_P}$ of equal rank are ordered according to the first indices μ_k, ν_k in which they differ. For tensors up to rank $\widehat{P} = 3$, the ordering can be read off from Table 4.

The routines for the tensor integrals provide both results for the full expressions and separately the coefficients of the UV-singular poles, $1/\epsilon_{\rm UV}$. The latter are useful to determine rational terms of UV origin. Rational terms of IR origin cancel in one-loop diagrams [68] as long as the loop is not inserted into an external line and only show up in wave-function renormalization constants which are easily calculated explicitly.

The coefficients $T^N_{i_1\ldots i_P}$ of the Lorentz-covariant decomposition (5) of the tensor integrals $T^{N,P}$ with $N = 1, \ldots, 7$ can be computed by calling the respective subroutines A_cll, ..., G_cll. The argument structure of these subroutines N_cll (N_cll=A_cll, ..., G_cll) is given by

```
subroutine N_cll(TN,TNuv,MomInv,mass2,R,TNerr)
double complex(0:R/2,0:R,...,0:R) TN      : T^{N,P}_{i1,...,iP} with P ≤ R
                                  (N−1)
double complex(0:R/2,0:R,...,0:R) TNuv    : T^{N,P UV}_{i1,...,iP} with P ≤ R
                                  (N−1)
double complex(1:n_P) MomInv              : momentum invariants
double complex(0:N-1) mass2               : squared masses
integer R                                 : maximal rank
double precision(0:R),optional TNerr      : error estimates .
```

The set of $n_{\mathcal{P}} = \binom{N}{2}$ momentum invariants $\mathcal{P}_N$, represented by the symbolic argument `MomInv`, is ordered such that the first N invariants correspond to the squares k_i^2 of the N incoming momenta k_i, the following N invariants to the squares $(k_i + k_{i+1})^2$ of pairs of adjacent momenta and so on. In terms of the off-set momenta $p_i = k_1 + \ldots + k_i$ entering the loop propagators given in (2), the set of momentum invariants reads

$$\begin{aligned}\mathcal{P}_{2k} = \big\{&(p_1-p_0)^2,(p_2-p_1)^2,\ldots,(p_{2k-1}-p_{2k-2})^2,(p_0-p_{2k-1})^2,\\ &(p_2-p_0)^2,(p_3-p_1)^2,\ldots,(p_0-p_{2k-2})^2,(p_1-p_{2k-1})^2,\\ &\ldots\\ &(p_{k-1}-p_0)^2,(p_k-p_1)^2,\ldots,(p_{k-3}-p_{2k-2})^2,(p_{k-2}-p_{2k-1})^2,\\ &(p_k-p_0)^2,(p_{k+1}-p_1)^2,\ldots,(p_{2k-2}-p_{k-2})^2,(p_{2k-1}-p_{k-1})^2\big\},\end{aligned} \tag{12}$$

$$\begin{aligned}\mathcal{P}_{2k+1} = \big\{&(p_1-p_0)^2,(p_2-p_1)^2,\ldots,(p_{2k}-p_{2k-1})^2,(p_0-p_{2k})^2,\\ &(p_2-p_0)^2,(p_3-p_1)^2,\ldots,(p_0-p_{2k-1})^2,(p_1-p_{2k})^2,\\ &\ldots\\ &(p_k-p_0)^2,(p_{k+1}-p_1)^2,\ldots,(p_{k-2}-p_{2k-1})^2,(p_{k-1}-p_{2k})^2\big\}.\end{aligned} \tag{13}$$

Note that the first $k-1$ lines in (12) each contain $N = 2k$ elements, while the kth line only contains $k = N/2$ elements. In (13) each of the k lines consists of $N = 2k+1$ elements. Within each line all momentum indices increase by one unit going from one element to the next one; an index that has already reached the maximum value $i = N-1$ passes on to $i = 0$. For $N = 2,..,7$, the sets of momentum invariants $\mathcal{P}_N$ are explicitly listed in Appendix A. They have to be provided to the subroutine *N*`_cll` in terms of parameters of type `double complex`, either as an array of length $n_{\mathcal{P}}$ or as

$n_{\mathcal{P}}$ single parameters. We stress that, although the variable type is `double complex`, non-vanishing imaginary parts of momentum invariants are not yet supported by the current version of COLLIER. It is further important to ensure that momentum invariants corresponding to invariant squared masses of single external particles assume their exact numerical value, avoiding any deviation that can occur for example if these momentum squares are evaluated numerically. Otherwise, problems can appear in IR-divergent integrals where momentum and mass arguments are compared internally in order to decide on the correct analytic expression. Obviously, the argument `MomInv` is absent in the case of one-point integrals `A_cll`.

The set of squared masses

$$\mathcal{M}_N \;=\; \left\{m_0^2, m_1^2, \ldots, m_{N-1}^2\right\}, \tag{14}$$

entering the loop propagators given in (2), is represented by N parameters of type `double complex` denoted symbolically as `mass2`. These parameters with possible non-vanishing (negative) imaginary parts have to be passed to the subroutine N`_cll` either as a single array or as individual arguments depending on which of the two formats (single parameters or array as described above) is chosen for the momentum invariants `MomInv`.

The `integer` argument `R` represents the maximum rank $\widehat{P}$ up to which the tensor integrals are calculated.It thus defines the size of the output arrays `TN` and `TNuv` of type `double complex`. As described before, they can be obtained either as an N-dimensional array with the first component of range $(0\!:\![\widehat{P}/2])$ and the other ones of range $(0\!:\!\widehat{P})$, or as a one-dimensional array of length $n_c(N,\widehat{P})$. The respective numbers $n_c(N,\widehat{P})$ are tabulated by `Collier` during the initialization and can be obtained with help of the function

```
function GetNc_cll(N,R) result(nc)
integer N,R,nc  : N, P̂, n_c(N, P̂) .
```

Finally, there is the option to add an additional output array `TNerr` consisting of $(0\!:\!\widehat{P})$ entries of type `double precision` to the list of arguments. If present, the elements of this array deliver an estimate of the absolute error size for the tensor coefficients $T^N_{i_1\cdots i_P}$ with all $i_k \neq 0$ of the corresponding rank P. The error estimate $\Delta T^N(P)$ is determined by following in a rough way the propagation of errors through the recursion algorithms and by estimating the approximate size of neglected higher-order terms in the expansions (see Section 3.1). We stress that the returned values should not be interpreted

as precise and reliable error specifications, but should rather be regarded as approximate order-of-magnitude estimates on the underlying uncertainties.

Instead of employing the individual subroutines `A_cll`, ..., `G_cll`, tensor coefficients can be calculated for (in principle) arbitrary N by means of the generic subroutine

```
subroutine TN_cll(TN,TNuv,MomInv,mass2,Nn,R,TNerr) .
```

The argument structure of the generic `TN_cll` differs from the one of the specific `A_cll`, ..., `G_cll` only by the presence of the additional `integer Nn`,

`integer Nn` : # of loop propagators ($= N$) ,

defining the number of loop propagators. In the case of the routine `TN_cll`, momentum invariants `MomInv`, squared masses `mass2`, and resulting coefficients `TN`, `TNuv` can only be handled in the format of one-dimensional arrays of length $n_{\mathcal{P}}$, N, and $n_c(N, \widehat{P})$, respectively.

The tensor components $T^{N,\mu_1\cdots\mu_P}$ of the integrals $T^{N,P}$ with $N = 1,\ldots,7$ can be computed by calling the respective subroutines `Aten_cll`, ..., `Gten_cll`. The argument structure of these subroutines N`ten_cll` = `Aten_cll`, ..., `Gten_cll` is given by

```
subroutine
    Nten_cll(TNten,TNtenuv,MomVec,MomInv,mass2,R,TNtenerr) .
```

In addition to the momentum invariants `MomInv` and the squared masses `mass2`, which enter the tensor subroutine N`ten_cll` exactly in the same way as the coefficient subroutine N`_cll`, also the $N-1$ four-vectors p_i appearing in the loop propagators in (2) have to be passed to the subroutine N`ten_cll`. Represented by the symbolic argument `MomVec`,

`double complex MomVec(...)` : momentum components ,

they have to be provided either as $N-1$ individual arrays of range $(0:3)$ corresponding to the momentum components p_i^μ with $\mu = 0,\ldots,3$, or combined into a single array of format $(0:3, N-1)$. Note that the same format must be chosen for the three arguments `MomVec`, `MomInv`, and `mass2`, i.e. either all of them must be provided as collections of individual arguments or all of them must be given as combined arrays. As in the case of `MomInv`, the variable type is `double complex` (although non-vanishing imaginary parts

are not supported by the current version of COLLIER), and the argument `MomVec` is absent for the one-point integrals `A_cll`.

The `integer` argument `R` representing the maximum rank $\widehat{P}$ up to which the tensor integrals are calculated defines the size of the output arrays `TNten` and `TNtenuv` of type `double complex`. As described before, they can be obtained either as a 4-dimensional array of format $(0\!:\!\widehat{P}, 0\!:\!\widehat{P}, 0\!:\!\widehat{P}, 0\!:\!\widehat{P})$, or as a one-dimensional array of length $n_t(\widehat{P})$. The respective numbers $n_t(\widehat{P})$ are tabulated by COLLIER during the initialization and can be obtained with help of the function

```
function GetNt_cll(R) result(nt)
integer R,nt  : P̂, n_t(P̂) .
```

An error estimate for the tensor components can be accessed adding the optional output array `TNtenerr` to the list of arguments. Its entries $(0\!:\!\widehat{P})$ of type `double precision` provide estimates on the absolute error size for the tensor components of the corresponding rank, in a similar manner as in the case of the subroutine N`_cll`.

Instead of employing the individual subroutines `Aten_cll`, ..., `Gten_cll`, tensor components can be calculated for (in principle) arbitrary N by means of the generic subroutine

```
subroutine
  TNten_cll(TNten,TNtenuv,MomVec,MomInv,mass2,Nn,R,TNtenerr) .
```

The argument structure of the generic `TNten_cll` differs from the one of the specific `Aten_cll`, ..., `Gten_cll` only by the presence of the additional `integer Nn` defining the number of loop propagators. In the case of the subroutine `TNten_cll`, momenta `MomVec`, momentum invariants `MomInv`, and squared masses `mass2` can only be handled as single arrays of format $(0\!:\!3, 1\!:\!N-1)$, $(1:n_{\mathcal{P}})$, and $(0:N-1)$, respectively, whereas for the outputs `TNten` and `TNtenuv` the user is still free to choose between the array structures $(0\!:\!\widehat{P}, 0\!:\!\widehat{P}, 0\!:\!\widehat{P}, 0\!:\!\widehat{P})$ and $(1\!:\!n_t(\widehat{P}))$.

Obviously, any call of a coefficient subroutine `A_cll`, ..., `G_cll` or `TN_cll`, as well as of a tensor subroutine `Aten_cll`, ..., `Gten_cll` or `TNten_cll`, delivers the result for the respective scalar integral within its output array. A user exclusively interested in the scalar 1-, ...,4-point master integrals can either restrict the rank argument `R` to $\widehat{P} = 0$, or can employ the alternative routines N`0_cll` = `A0_cll`, ..., `D0_cll`:

```
subroutine N0_cll(TN0,MomInv,mass2)
double complex TN0  : T_0^N .
```

These routines provide the respective scalar integral as a single output variable `TN0` of type `double complex`, while for the inputs `MomInv` and `mass2` choice can be made between the usual options. Note that the routines `A0_cll`, ..., `D0_cll` are not linked to the cache system and may fail if the Gram determinant for the 3-point function or the Cayley determinant of the 4-point function vanishes.

Finally, `COLLIER` provides also routines for the calculation of momentum derivatives of 2-point coefficients, needed for the wave-function renormalization of external particles. The subroutine in charge, `DB_cll`, is structured as

```
subroutine DB_cll(DB,DBuv,MomInv,mass2,R,DBerr)
double complex DB(...)
double complex DBuv(...)
double complex DBerr(...) .
```

The derivatives $B'_{i_1\cdots i_{\widehat{P}}}(p_1^2) \equiv \partial B_{i_1\cdots i_{\widehat{P}}}(p_1^2)/\partial p_1^2$ are returned via the output arrays `DB` and `DBuv`, with an optional error estimate via `DBerr`. The conventions for the in- and output arguments are in complete analogy with the ones of the subroutine `B_cll`. The results of the functions B'_0, B'_1, B'_{00}, and B'_{11} can further be obtained as single `double complex` variables with help of the subroutines

```
subroutine DB0_cll(DB0,MomInv,mass2)
double complex DB0 ,
subroutine DB1_cll(DB1,MomInv,mass2)
double complex DB1 ,
subroutine DB00_cll(DB00,MomInv,mass2)
double complex DB00 ,
subroutine DB11_cll(DB11,MomInv,mass2)
double complex DB11 .
```

Since derivatives $B'_{i_1\cdots i_{\widehat{P}}}$ are not cached, calls of the subroutines `DB_cll`, `DB0_cll`, `DB1_cll`, `DB00_cll`, and `DB11_cll` do not interfere with the cache system of COLLIER.

5.4. Setting and getting parameters

The results for the tensor integrals do not only depend on the explicit mass and momentum arguments, but also on the choices made concerning *regularization parameters*, as well as on *technical parameters* governing the selection of reduction schemes and the number of iterations for expansion methods. The parameters of the latter two groups are typically kept at fixed values for a certain set of integral calls. Therefore they do not form part of the argument lists of the individual integral calls, but are rather gathered as a set of global parameters. They are initialized to default values specified in Table 2 during the initialization procedure of COLLIER and can be modified later on. In the following we give details on these parameters, as well as on the subroutines that allow the user to change or read out their values.

First, we note that the version number of COLLIER can be retrieved upon calling

```
subroutine GetVersionNumber_cll(version)
character(len=5) version .
```

5.4.1. Regularization parameters

COLLIER uses dimensional regularization to handle UV divergences. The results for UV-divergent integrals thus depend on the regulator $\Delta_{\rm UV}$ defined in (8) and the scale of dimensional regularization, $\mu_{\rm UV}$, more precisely on the combination $\Delta_{\rm UV} + \ln(\mu^2_{\rm UV}/Q^2)$, where Q^2 is some scale of the respective tensor integral. At fixed order in perturbation theory, physical S-matrix elements do not depend on $\Delta_{\rm UV}$ and $\mu_{\rm UV}$. In COLLIER, $\Delta_{\rm UV}$ and $\mu^2_{\rm UV}$ are treated as numerical parameters of type `double precision` with default values $\Delta_{\rm UV} = 0$ and $\mu^2_{\rm UV} = 1$, which can be modified employing the subroutines

```
subroutine SetDeltaUV_cll(delta)
double precision delta ,
subroutine SetMuUV2_cll(mu2)
double precision mu2 .
```

On the one hand, by varying these parameters one can verify the UV finiteness of S-matrix elements numerically. On the other hand, in renormalization schemes like MS or $\overline{\rm MS}$, the scale $\mu_{\rm UV}$ of dimensional regularization can be identified with the renormalization scale $\mu_{\rm ren}$ of a running coupling $g(\mu_{\rm ren})$.

In this case it gains a physical interpretation with impact on S-matrix elements. The current values of $\Delta_{\rm UV}$ and $\mu^2_{\rm UV}$ can be read out with help of the subroutines

```
subroutine GetDeltaUV_cll(delta)
double precision delta ,
subroutine GetMuUV2_cll(mu2)
double precision mu2 .
```

By default, also IR divergences are regularized dimensionally. The results of IR-divergent integrals thus depend on $\Delta^{(1)}_{\rm IR}$ and $\Delta^{(2)}_{\rm IR}$ defined in (8), and on the scale of dimensional regularization, $\mu_{\rm IR}$. At fixed order in perturbation theory, IR-finite observables do not depend on $\Delta^{(1)}_{\rm IR}$, $\Delta^{(2)}_{\rm IR}$, and $\mu_{\rm IR}$, once contributions from virtual and real corrections are combined. In COLLIER, $\Delta^{(1)}_{\rm IR}$, $\Delta^{(2)}_{\rm IR}$, and $\mu^2_{\rm IR}$ are represented by numerical parameters of type `double precision` with default values $\Delta^{(1)}_{\rm IR} = \Delta^{(2)}_{\rm IR} = 0$ and $\mu^2_{\rm IR} = 1$, which can be modified employing the subroutines

```
subroutine SetDeltaIR_cll(delta1,delta2)
double precision delta1,delta2 ,
subroutine SetMuIR2_cll(mu2)
double precision mu2 .
```

Note, in particular, that $\Delta^{(1)}_{\rm IR}$ and $\Delta^{(2)}_{\rm IR}$ can be varied independently. A variation of $\Delta^{(1)}_{\rm IR}$, $\Delta^{(2)}_{\rm IR}$, and $\mu^2_{\rm IR}$ can be performed as a numerical check on the IR finiteness of observables. The current values are retrieved calling

```
subroutine GetDeltaIR_cll(delta1,delta2)
double precision delta1,delta2 ,
subroutine GetMuIR2_cll(mu2)
double precision mu2 .
```

Collinear divergences can also be regulated introducing a list of mass regulators,

$$\mathcal{R}_{n_{\rm reg}} = \left\{\overline{m}^2_1, \overline{m}^2_2, \ldots, \overline{m}^2_{n_{\rm reg}}\right\}. \tag{15}$$

To this end, the user must call the subroutine

```
subroutine SetMinf2_cll(nminf,minf2)
double complex minf2(nminf)
integer nminf ,
```

where the `integer` variable `nminf` stands for the number $n_{\rm reg}$ of different regulator masses and the array `minf2` contains their squared values $\overline{m}_i^2$ of type `double complex`. Alternatively, regulator masses can be added successively calling the subroutine

```
subroutine AddMinf2_cll(m2)
double complex m2 ,
```

which increments $n_{\rm reg}$ by one and adds the `double complex` value `m2` to the list $\mathcal{R}_{n_{\rm reg}}$. When a tensor integral is called, its arguments (squared masses and momentum invariants) are numerically compared to the elements of $\mathcal{R}_{n_{\rm reg}}$. Identified entries are treated as infinitesimally small throughout the calculation and their (not necessarily small) numerical values are only kept in otherwise singular logarithms. It is crucial that the small masses have exactly the same values in the calls of all subroutines. The number of mass regulators $n_{\rm reg}$ and the list of their squared values can be read out with

```
subroutine GetNminf_cll(nminf) ,
subroutine GetMinf2_cll(minf2)
double complex minf2(nminf)
integer nminf ,
```

respectively. Finally, the subroutine

```
subroutine ClearMinf2_cll
```

allows to clear the list $\mathcal{R}_{n_{\rm reg}}$ and to reset $n_{\rm reg}$ to zero.

5.4.2. Technical parameters

COLLIER can be run in three different modes that are chosen employing

```
subroutine SetMode_cll(mode)
integer mode
```

with the `integer` argument `mode=1,2,3`. This subroutine switches between the different branches implemented. For `mode=1` (the default value) the `COLI` branch is used, for `mode=2` the `DD` branch is used, and for `mode=3` the integrals are calculated in both branches and compared. In the latter case, COLLIER returns from the two results the one for which a higher precision is estimated internally. The error estimate delivered (if added as optional argument to the subroutine call) is determined from the internal estimate of the respective library and the difference between the `COLI` and `DD` results, as the maximum of these two estimators. Differences between `COLI` and `DD` that exceed a certain threshold value are further reported to the file `CheckOut.cll` (see Section 5.6 for details). The chosen value of `mode` can be retrieved with

```
subroutine GetMode_cll(mode)
integer mode .
```

The target precision $\eta_{\rm req}$ aimed at in the calculation of the tensor integrals can be set calling

```
subroutine SetReqAcc_cll(acc)
double precision acc
```

with the argument `acc` of type `double precision`. For the calculation of tensor integrals, COLLIER will choose a reduction scheme that is expected to reach the required precision $\eta_{\rm req}$, if necessary trying different choices and performing expansions up to the order at which the target precision is achieved. Hence, the choice of $\eta_{\rm req}$ affects the precision of the results as well as the run time, and has to be made in light of the desired balance between the two. The default value is $\eta_{\rm req} = 10^{-8}$, and the library has been optimized in particular for this setting. The current values of $\eta_{\rm req}$ can be inquired with the subroutine

```
subroutine GetReqAcc_cll(acc)
double precision acc .
```

To which extent results can really be obtained within the precision $\eta_{\rm req}$ depends on the complexity of the problem. As a second precision threshold, a critical precision $\eta_{\rm crit}$, which should be larger than $\eta_{\rm req}$, can be set via the subroutine

```
subroutine SetCritAcc_cll(acc)
double precision acc .
```

The argument `acc` is again of type `double precision`. The critical precision does not influence the actual calculation, it is a mere book-keeping device: If within the sequence of computed integrals for a certain phase-space point the estimated uncertainty for at least one integral fails to reach $\eta_{\rm crit}$, an accuracy flag is raised to indicate a warning. The user can consult this flag at any time and thus dynamically decide how to proceed (e.g. if he wants to discard the respective phase-space point, recalculate it with a different branch/different settings, etc.). Moreover, critical integrals can be monitored. If this option is selected, their arguments and results are automatically written to an output file. More information on the accuracy flag and the monitoring of critical integrals is given in Section 5.6. The critical precision is initialized as $\eta_{\rm crit} = 10^{-1}$; its value can be obtained by

```
subroutine GetCritAcc_cll(acc)
double precision acc .
```

Finally, a third precision parameter $\eta_{\rm check}$, which should be chosen larger than $\eta_{\rm req}$, governs the comparison between the results obtained from `COLI` and `DD`. The default value of this variable is $\eta_{\rm check} = 10^{-4}$; and it can be modified and read calling

```
subroutine SetCheckAcc_cll(acc) ,
subroutine GetCheckAcc_cll(acc)
double precision acc .
```

For `mode`=3, integral calls yielding results with a relative deviation between `COLI` and `DD` of more than $\eta_{\rm check}$ are logged in the file `CheckOut.cll`. In `mode`=1 and `mode`=2 the parameter $\eta_{\rm check}$ is irrelevant.

Instead of invoking the described subroutines to set individually the precision thresholds $\eta_{\rm req}$, $\eta_{\rm crit}$, and $\eta_{\rm check}$, the subroutine

```
subroutine SetAccuracy_cll(acc0,acc1,acc2)
double precision acc0,acc1,acc2
```

can be used to set all of them at the same time. The `double precision` arguments `acc0`, `acc1`, and `acc2` represent in this order $\eta_{\rm req}$, $\eta_{\rm crit}$, and $\eta_{\rm check}$.

A further technical parameter is given by the maximal rank $\widehat{P}^{\max}$ up to which tensors are calculated in iterative methods and which thus defines a cut-off order for the expansion methods. To set $\widehat{P}^{\max}$, the subroutine

```
subroutine SetRitmax_cll(ritmax)
integer ritmax
```

can be invoked with the argument `ritmax` bigger or equal to 7. In turn, the parameter `ritmax` can be retrieved with the help of

```
subroutine GetRitmax_cll(ritmax) .
```

The parameter $\widehat{P}^{\max} \geq 7$ acts as maximal rank for 4-point integrals in the expansion methods, the maximal rank for 3- and 2-point integrals is then internally set to $\widehat{P}^{\max}+2$ and $\widehat{P}^{\max}+4$, respectively. The value of $\widehat{P}^{\max}$ thus affects the precision and computing time of N-point integrals with $N \leq 4$ (from external and internal calls), and the library has been optimized in particular for the default setting $\widehat{P}^{\max} = 14$. Note further that in order to facilitate the calculation of all tensor integrals $T^{N,P}$ up to $N = N_{\max}$ and $P = P_{\max}$ (with $N_{\max}, P_{\max}$ as specified in the initialization call of `Init_cll`), $\widehat{P}^{\max}$ cannot be chosen smaller than $P_{\max} + 4 - N_{\max}$.

As explained in Section 3, for $N \geq 6$, reduction methods are implemented in terms of the coefficients $T^N_{i_1\dots i_P}$ as well as in terms the tensor components $T^{N,\mu_1\dots\mu_P}$. A general calculation of a tensor integral $T^{N,\mu_1\dots\mu_P}$ with $N \geq 6$ thus proceeds in three steps: First, for $5 \leq \bar{N} = N_{\text{tenred}} - 1 \leq N$ the coefficients $T^{\bar{N}}_{i_1\dots i_{P_{\bar{N}}}}$ are calculated recursively starting from 2-point coefficients. Then the tensors $T^{\bar{N},\mu_1\dots\mu_{P_{\bar{N}}}}$ are built from the coefficients $T^{\bar{N}}_{i_1\dots i_{P_{\bar{N}}}}$. Finally, the tensor $T^{N,\mu_1\dots\mu_{P_N}}$ is calculated recursively from the tensors $T^{\bar{N},\mu_1\dots\mu_{P_{\bar{N}}}}$ (see Figure 1 for an illustration). The threshold N_{tenred} from which on tensor reduction is used can be set and retrieved with the subroutines

```
subroutine SetTenRed_cll(Ntenred) ,
subroutine GetTenRed_cll(Ntenred)
integer Ntenred ,
```

where the argument `Ntenred` represents the parameter N_{tenred}. The subroutine

```
subroutine SwitchOnTenRed_cll,
```

being equivalent to `SetTenRed_cll(Ntenred)` with `Ntenred`=6, opts for the maximal level of tensor reduction, while the subroutine

```
subroutine SwitchOffTenRed_cll
```

switches it off completely. The default setting corresponds to maximal tensor reduction $N_{\text{tenred}} = 6$, which is favoured compared to other choices regarding run time.

As of COLLIER version 1.2.8 a new option for the calculation of high-rank tensor integrals via `TNten_cll` is provided. It improves the accuracy of some critical tensor integrals but does not work with the global cache of collier. This feature is disabled by default but can be switched on upon calling

```
subroutine SwitchOnArgPerm_cll,
```

which switches of the global cache system (see Section 5.5). It can be switched off via the subroutine

```
subroutine SwitchOffArgPerm_cll.
```

5.5. Using the cache system

COLLIER disposes of a cache system which is used to avoid repeated calculations of identical tensor integrals and thus serves to speed up computations. It can operate in a local or in a global mode: In the local mode, integrals are only stored during the processing of a single subroutine call from Section 5.3. The cache detects identical integrals that are reached several times in the course of the reduction algorithm via different paths of the reduction tree and avoids their recalculation. In the global mode, in addition also identical integrals belonging to different user calls of subroutines from Section 5.3 are linked. Whereas the local mode of the cache is always at work, the global mode has to be activated explicitly. To this end, a number n_{cache} of separate caches can be created to store the results of the calculated coefficients or tensors. This is done calling

```
subroutine InitCacheSystem_cll(ncache,Nmax)
integer ncache,Nmax ,
```

where the parameters `ncache` and `Nmax` represent the total number n_{cache} of caches and the maximal $N = N_{\text{cache}}^{\text{max}}$ up to which N-point integrals are cached, respectively. In order to run the cache in the global mode, it is compulsory

that for each phase-space point the sequence of integral calls assigned to a certain cache `cacheNr` is preceded by a call of `InitEvent(cacheNr)`. We further stress that these integral calls must be realized for each phase-space point in the same order, and global parameters (like $\mu^2_{\rm UV}$, the `mode` of COLLIER, etc.) must not be reset within a phase-space point, because integrals are identified by their position in the sequence of user calls. Note that there is no hard limit concerning the size of the caches `cacheNr` and that the required memory is determined and allocated dynamically during the first phase-space points. Depending on the application in question, running the cache in the global mode can lead to a high use of memory resources.[4]

Instead of fixing the total number of caches $n_{\rm cache}$ right from the beginning, it is also possible to subsequently add caches to the cache system by calling the subroutine

```
subroutine AddNewCache_cll(cache_no,Nmax)
integer cache_no,Nmax .
```

The new cache is initialized to store N-point integrals up to the value `Nmax` received as input, while the number assigned to it is returned as output argument `cache_no`. A call of `AddNewCache_cll` without a previous initialization of the cache system is equivalent to the call of `InitCacheSystem_cll(ncache,Nmax)` with argument `ncache` $= 1$.

The threshold $N_{\rm cache} \leq N^{\rm max}_{\rm cache}$ up to which integrals are cached can be adjusted individually for each cache. For this purpose the subroutine

```
subroutine SetCacheLevel_cll(cache_no,Nmax)
integer cache_no,Nmax
```

is provided. Note that the level $N^{\rm max}_{\rm cache}$ of a cache `cache_no` can only be changed *before* the first phase-space point for the respective cache is evaluated (i.e. before `InitEvent_cll` is evaluated for the first time with the argument `cache_no`). Later calls of `SetCacheLevel_cll` with argument `cache_no` are ignored.

With help of the subroutine

[4]While a single cache is in principle sufficient, different caches are needed if different subprocesses of a particle reaction are calculated simultaneously, but not always in exactly the same order.

```
subroutine SwitchOffCacheSystem_cll
```

the global cache system can be switched off temporarily. This option can for instance be useful if an exceptional situation makes it necessary to depart from the fixed sequence of integrals by inserting additional integral calls. The subroutine

```
subroutine SwitchOnCacheSystem_cll
```

switches the global cache on again, which will take up its work at the point it was interrupted.

It is also possible to switch off only one particular cache calling

```
subroutine SwitchOffCache_cll(cache_no)
integer cache_no .
```

In this case, when switched on again via the subroutine

```
subroutine SwitchOnCache_cll(cache_no)
integer cache_no ,
```

the cache either continues at the point where it was paused, or, if in the meantime the subroutine `InitEvent` has been employed, with the first integral of the list of cache `cache_no`.

5.6. Error treatment and output files

Internal errors as well as possible failures in precision are handled by COLLIER in a twofold way: On the one hand, corresponding global flags for errors and accuracy are set and can be read out by the user, permitting to take measures on the flight. On the other hand, corresponding error messages and problematic integral calls are recorded in output files.

The error flag $\sigma_{\rm err}$ is obtained calling

```
subroutine GetErrFlag_cll(errflag)
integer errflag .
```

It is represented by the integer `errflag` assuming values in the range from $\sigma_{\rm err} = 0$ in the case of faultless processing to $\sigma_{\rm err} = -10$ reserved for the most fatal errors. The flag $\sigma_{\rm err}$ keeps its current value until it is overwritten by a more negative one implying that an error has occurred that is considered

worse than all errors encountered so far. In this way, σ_{err} indicates the severest error that has appeared since its initialization. It is automatically reinitialized to $\sigma_{\mathrm{err}} = 0$ with the call of `InitEvent_cll` for a new phase-space point, and can in addition be reset at any time invoking

```
subroutine InitErrFlag_cll .
```

If σ_{err} falls below a certain threshold σ_{stop}, execution of the program is stopped automatically. The default value $\sigma_{\mathrm{stop}} = -8$ is chosen, so that errors that could be related to the particular characteristics of an individual phase-space point do not lead to a stop, whereas systematic errors expected to be common to all phase-space points trigger the termination of the program. It is possible to select a different value for σ_{stop} or to retrieve its value, committing the corresponding `integer` argument `stopflag` to the subroutines

```
subroutine SetErrStop_cll(stopflag) ,
subroutine GetErrStop_cll(stopflag)
integer stopflag .
```

A stop of the program can be suppressed completely upon calling

```
subroutine SwitchOffErrStop_cll() .
```

The accuracy flag σ_{acc} works in a very similar manner. It serves as an indicator reflecting the precision of the results and is retrieved as `integer` argument `accflag` via the subroutine

```
subroutine GetAccFlag_cll(accflag)
integer accflag .
```

Initialized with $\sigma_{\mathrm{acc}} = 0$, it is changed to $\sigma_{\mathrm{acc}} = -1$ as soon as an integral calculation does not reach the target precision η_{req}, and to $\sigma_{\mathrm{acc}} = -2$ if it does not fulfil the critical precision η_{crit} (see Section 5.4.2 for details on these parameters). As in the case of the error flag, σ_{acc} is only overwritten by a more negative value and thus signals the accuracy of the most critical integral calculation since its initialization. A reinitialization to $\sigma_{\mathrm{acc}} = 0$ is automatically performed with the call of `InitEvent_cll` for a new phase-space point, and can in addition be achieved with the help of the subroutine

```
subroutine InitAccFlag_cll .
```

With the initialization of COLLIER, the user chooses how messages on errors and accuracy as well as additional information should be returned. By default, COLLIER stores this information in separate files which are deposited in the directory `output_cll` created within the working directory during initialization. As described in Section 5.2, the user can define a different path or name for the output folder by adding a corresponding string as a second optional argument to the subroutine `Init_cll`, or he can suppress the creation of file output by choosing the empty string as folder name. This predefined setting can be modified later, switching off or on the file output via the subroutines

```
subroutine SwitchOffFileOutput_cll ,
subroutine SwitchOnFileOutput_cll ,
```

or creating a new output directory employing the subroutine

```
subroutine SetOutputFolder_cll(fname)
character(len=*) fname .
```

The path of the output folder is represented by the string `fname`, and it can be read out calling

```
subroutine GetOutputFolder_cll(fname)
character(len=*) fname .
```

Error messages are directed to the files `ErrOut.coli`, `ErrOut.dd`, and `ErrOut.cll` depending on whether the source of error is located in the `COLI` library, the `DD` library, or the global interface (or the module `tensors`), respectively. During the initialization of COLLIER these files are created within the above-mentioned output folder, and a free output channel (with number > 100) is automatically assigned to each of them. Output channels can also be attributed manually by the user calling the corresponding subroutines

```
subroutine SetnerroutCOLI_cll(outchan) ,
subroutine SetnerroutDD_cll(outchan) ,
subroutine Setnerrout_cll(outchan)
integer outchan
```

with the channel number `outchan` as `integer` argument. Most notably, this possibility permits to redirect the error output to the standard channel by choosing `outchan`=6. Note that, if the file output is switched off via the subroutine `SwitchOffFileOutput_cll`, the standard channel is not closed and, moreover, COLLIER output directed to it (whether it be the terminal or a dedicated standard output file) continues to be delivered. The output channels currently selected can be retrieved by the subroutines

```
subroutine GetnerroutCOLI_cll(outchan) ,
subroutine GetnerroutDD_cll(outchan) ,
subroutine Getnerrout_cll(outchan)
integer outchan .
```

The number of displayed error messages is limited to $n_{\rm err}^{\rm max} = 100$ by default in order to protect the error files from growing in size without control. This predefined limit can be modified individually for the three types of errors with help of the subroutines

```
subroutine SetMaxErrOutCOLI_cll(nout) ,
subroutine SetMaxErrOutDD_cll(nout) ,
subroutine SetMaxErrOut_cll(nout)
integer nout
```

specifying a corresponding `integer` number `nout`. By default the error limits as well as the respective counters are reset in case of a reinitialization of COLLIER, but this reset is suppressed if `noreset=.true.` is passed as additional argument to `Init_cll`. The counters can be reinitialized by hand calling

```
subroutine InitErrCntCOLI_cll ,
subroutine InitErrCntDD_cll ,
subroutine InitErrCnt_cll .
```

The error output can be dis- and enabled by calling

```
subroutine SetErrOutLev_cll(outlev)
integer outlev
```

with `integer` argument `outlev`=0 and `outlev`=1, respectively. In case COLLIER is initialized with an empty string as output-folder name, error output is disabled by default, while in all other cases it is switched on.

Additional information and status messages not related to errors are recorded in the log-file `InfOut.cll`, created as well during the initialization of COLLIER in the designated output folder. Also in this case a free output channel (with number > 100) is automatically generated, and can be adapted by the user passing the corresponding `integer` number `outchan` to the subroutine

```
subroutine Setninfout_cll(outchan)
integer outchan .
```

Again, this allows for a redirection of the output to the standard channel `outchan`=6. To retrieve the current output channel, the subroutine

```
subroutine Getninfout_cll(outchan)
integer outchan
```

can be called. By default the output is limited to $n_{\rm inf}^{\rm max} = 1000$ messages, but the user can modify this predefined limit employing

```
subroutine SetMaxInfOut_cll(nout)
integer nout .
```

By default $n_{\rm inf}^{\rm max}$ as well as the respective counter are reset in case of a reinitialization of COLLIER, unless this reset is suppressed by passing `noreset=.true.` as additional argument to `Init_cll`. To which extent informative output is provided, can be regulated by a call of the subroutine

```
subroutine SetInfOutLev_cll(outlev)
integer outlev
```

with a suitable `integer` argument `outlev`=0, 1, 2. Initialization of COLLIER with an empty string as folder name, implies a presetting of `outlev`=0 (disabled output), while in all other cases the presetting `outlev`=2 (maximum output) is selected. In the latter case, any change of an internal parameter is recorded in the output file which can lead to excessive output for instance if some parameters (such as the UV scale $\mu^2_{\rm UV}$) are modified repeatedly (e.g. for each phase-space point). Hence, in addition an intermediate output level

`outlev`=1 is offered, tracing only more special activities which are considered to occur less frequently.

As soon as the mode in which COLLIER operates is switched for the first time to `mode`=3, the file `CheckOut.cll` is created in the common output folder. The channel automatically assigned to the output file can be changed to a different number, and the current channel number can be retrieved, calling the subroutines

```
subroutine Setncheckout_cll(outchan) ,
subroutine Getncheckout_cll(outchan)
integer outchan ,
```

respectively. In `mode`=3, integrals are calculated both with the `COLI` and the `DD` branch of the library, and the file `CheckOut.cll` collects input and results for those integrals with a relative discrepancy of more than η_{check} (see Section 5.4.2 for more details on the parameter η_{check}). Also derivatives of 2-point functions are compared between `COLI` and `DD` and reported in case of relative deviations above η_{check}. Output is limited to the first $n_{\text{check}}^{N,\text{max}}$ problematic N-point integrals and the first $n_{\text{check}}^{B',\text{max}}$ derivatives. The limit $n_{\text{check}}^{N,\text{max}}$ can be individually chosen for each N via the subroutine

```
subroutine SetMaxCheck_cll(npoints,N)
integer npoints,N ,
```

where the inputs `npoints` and `N` represent $n_{\text{check}}^{N,\text{max}}$ and N, respectively. The limit $n_{\text{check}}^{B',\text{max}}$ for the derivatives of 2-point functions can be chosen in an analogous manner employing the subroutine

```
subroutine SetMaxCheckDB_cll(npoints)
integer npoints .
```

It is also possible to set all $n_{\text{check}}^{1,\text{max}}, \ldots, n_{\text{check}}^{N_{\text{max}},\text{max}}$ with a single subroutine call, transmitting the `integer` array $\{n_{\text{check}}^{1,\text{max}}, \ldots, n_{\text{check}}^{N_{\text{max}},\text{max}}\}$ as a single argument `npointarray` to

```
subroutine SetMaxCheck_cll(npointarray)
integer npointarray (N_max) .
```

Note that the value $N_{\rm max}$ is fixed from the call of `Init_cll` of the last (re-)initialization of COLLIER (see Section 5.2). Initial values are $n_{\rm check}^{1,\rm max} = \ldots = n_{\rm check}^{N_{\rm max},\rm max} = n_{\rm check}^{B',\rm max} = 50$, and the call of `Init_cll` leads to a reinitialization of the limits and the respective counters unless the optional argument `noreset` is present and `.true.`. We further remark that the counters for the output messages in `Checkout.cll` can also be reset to zero by hand calling

```
subroutine InitCheckCnt_cll
```

for the N-point integrals, and

```
subroutine InitCheckCntDB_cll
```

for the 2-point derivatives.

Finally, the user can request additional information about integrals for which the estimated precision does not reach the threshold $\eta_{\rm crit}$ (see Section 5.4.2 for more details on the parameter $\eta_{\rm crit}$). This feature has to be launched explicitly via

```
call InitMonitoring_cll .
```

After activation, input and results for integrals that fall short in precision are recorded to the file `CritPointsOut.cll`. As for the other files, a free output channel is automatically attributed. The channel number can be set by hand and read out with the help of the subroutines

```
subroutine Setncritpointsout_cll(outchan) ,
subroutine Getncritpointsout_cll(outchan)
integer outchan .
```

Output is limited to the first $n_{\rm crit}^{N,\rm max} = 50$ problematic N-point integrals and the first $n_{\rm crit}^{B',\rm max} = 50$ derivatives. The limits $n_{\rm crit}^{N,\rm max}$ and $n_{\rm crit}^{B',\rm max}$ can be changed making use of the subroutines

```
subroutine SetMaxCritPoints_cll(npoints,N) ,
subroutine SetMaxCritPoints_cll(npointarray) ,
subroutine SetMaxCritPointsDB_cll(npoints)
integer npoints,N
integer npointarray(N_max) ,
```

which work in a completely analogous manner as the subroutines `SetMaxCheck_cll` and `SetMaxCheckDB_cll` described above. Initialization or reinitialization of COLLIER does not change $n_{\rm crit}^{N,\rm max}$, $n_{\rm crit}^{B',\rm max}$ and the respective counters. However, a call of `InitMonitoring_cll` resets $n_{\rm crit}^{1,\rm max} = \ldots = n_{\rm crit}^{N_{\rm max},\rm max} = n_{\rm crit}^{B',\rm max} = 50$, and the respective counters to zero. A reset of only the counters can be enforced by hand calling

```
subroutine InitPointsCnt_cll .
```

5.7. Sample programs

With the commands

```
"make demo"
"make democache"
```

in the directory `COLLIER-`v`/build` two sample programs can be installed that can be run executing

```
"./demo"
"./democache"
```

in the folder `COLLIER-`v`/demos`.

The program `demo` is dedicated to the calculation of single tensor integrals. During the run, the user is asked to specify the mode and to choose among various examples X of N-point integrals the one he likes to compute. The result of the calculation is written to a file `demo_`N`point_example`X`.dat`, which directs the user to the passage within `demo.f90` where the source code for the respective integral call can be found. In many cases, various calls of the same integral are shown featuring the different options for communicating the arguments to the subroutine. The variables used in the examples are defined at the beginning of the file `demo.f90`, followed by a short sequence of code common to all examples where COLLIER is initialized. It contains various commented lines which can be activated removing the exclamation mark in front and which demonstrate how global parameters of COLLIER can be modified.

The program `democache` exemplifies the usage of the cache. For a set of 1000 phase-space points a series of 8 tensor integral computations is performed several times. The toy Monte Carlo is carried out subsequently in four different subsets: first using the `COLI` branch, with and without cache, then using the `DD` branch, with and without cache. The source code is located in the file `democache.f90`.

6. Conclusions

The fortran-based library COLLIER numerically evaluates one-loop scalar and tensor integrals in perturbative relativistic quantum field theories for scattering processes with no a-priori restriction on the particle multiplicities. The particular strengths of COLLIER comprise the use of dedicated techniques to automatically optimize numerical stability in delicate phase-space regions, the support of complex internal masses for unstable particles, and the optional use of dimensional or mass regularization to treat infrared divergences. Moreover, COLLIER allows for powerful checks on the correctness and numerical stability of the results, since it is a merger of two independent integral libraries, `COLI` and `DD`.

COLLIER can be used both within traditional Feynman-diagrammatic and modern unitarity-based calculations, delivering all relevant scalar and tensor one-loop integrals on demand. The library already represents an essential building block in the automated one-loop amplitude generators OPENLOOPS and RECOLA and is now ready to be employed by other generators as well.

7. Acknowledgements

We thank B. Biedermann, F. Cascioli, R. Feger, V. Hirschi, J.N. Lang, J. Lindert, P. Maierhöfer, H. Patel, M. Pellen, S. Pozzorini, A. Scharf and S. Uccirati for performing various checks of the code. We are further grateful to J.N. Lang for providing the CMAKE makefile for the library. This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG) under reference number DE 623/2-1. The work of L.H. was supported by the grants FPA2013-46570-C2-1-P and 2014-SGR-104, and partially by the Spanish MINECO under the project MDM-2014-0369 of ICCUB (Unidad de Excelencia "María de Maeztu").

Appendix A. Sets of momentum invariants $\mathcal{P}_N$ for $N = 1, \ldots, 7$

The N-point tensor integrals depend on the complete set of momentum invariants $\mathcal{P}_N$ that can be formed from the momenta p_i entering the propagator denominators in (2). Our convention for the order of elements within the sets $\mathcal{P}_N$ is given in (12) and (13). For convenience, we list here explicitly the $\mathcal{P}_N$ for $N = 2, \ldots, 7$:

$\mathcal{P}_2 = \{p_1^2\}$,

$$\mathcal{P}_3 = \left\{p_1^2,\, (p_2-p_1)^2,\, p_2^2\right\},$$

$$\mathcal{P}_4 = \left\{p_1^2,\, (p_2-p_1)^2,\, (p_3-p_2)^2,\, p_3^2,\, p_2^2,\, (p_3-p_1)^2\right\},$$

$$\begin{aligned}\mathcal{P}_5 = \big\{&p_1^2,\, (p_2-p_1)^2,\, (p_3-p_2)^2,\, (p_4-p_3)^2,\, p_4^2,\\ &p_2^2,\, (p_3-p_1)^2,\, (p_4-p_2)^2,\, p_3^2,\, (p_1-p_4)^2\big\},\end{aligned}$$

$$\begin{aligned}\mathcal{P}_6 = \big\{&p_1^2,\, (p_2-p_1)^2,\, (p_3-p_2)^2,\, (p_4-p_3)^2,\, (p_5-p_4)^2,\, p_5^2,\\ &p_2^2,\, (p_3-p_1)^2,\, (p_4-p_2)^2,\, (p_5-p_3)^2,\, p_4^2,\, (p_1-p_5)^2,\\ &p_3^2,\, (p_4-p_1)^2,\, (p_5-p_2)^2\big\},\end{aligned}$$

$$\begin{aligned}\mathcal{P}_7 = \big\{&p_1^2,\, (p_2-p_1)^2,\, (p_3-p_2)^2,\, (p_4-p_3)^2,\, (p_5-p_4)^2,\, (p_6-p_5)^2,\, p_6^2,\\ &p_2^2,\, (p_3-p_1)^2,\, (p_4-p_2)^2,\, (p_5-p_3)^2,\, (p_6-p_4)^2,\, p_5^2,\, (p_1-p_6)^2,\\ &p_3^2,\, (p_4-p_1)^2,\, (p_5-p_2)^2,\, (p_6-p_3)^2,\, p_4^2,\, (p_1-p_5)^2,\, (p_2-p_6)^2\big\}.\end{aligned} \tag{A.1}$$

References

[1] J. Campbell, et al., Working Group Report: Quantum Chromodynamics (2013). arXiv:1310.5189.

[2] J. Butterworth, et al., Les Houches 2013: Physics at TeV Colliders, Standard Model Working Group Report (2014). arXiv:1405.1067.

[3] Z. Bern, et al., The NLO multileg working group, Summary report (2008). arXiv:0803.0494.

[4] J. Andersen, et al., The SM and NLO Multileg Working Group, Summary report (2010). arXiv:1003.1241.

[5] J. Alcaraz Maestre, et al., The SM and NLO Multileg and SM MC Working Groups: Summary Report (2012). arXiv:1203.6803.

[6] C. Berger, et al., An automated implementation of on-shell methods for one-loop amplitudes, Phys.Rev. D78 (2008) 036003. arXiv:0803.4180, doi:10.1103/PhysRevD.78.036003.

[7] S. Badger, B. Biedermann, P. Uwer, NGluon: a package to calculate one-loop multi-gluon amplitudes, Comput.Phys.Commun. 182 (2011) 1674–1692. arXiv:1011.2900, doi:10.1016/j.cpc.2011.04.008.

[8] G. Bevilacqua, et al., HELAC-NLO, Comput.Phys.Commun. 184 (2013) 986–997. arXiv:1110.1499, doi:10.1016/j.cpc.2012.10.033.

[9] G. Cullen, et al., Automated one-loop calculations with GoSam, Eur.Phys.J. C72 (2012) 1889. arXiv:1111.2034, doi:10.1140/epjc/s10052-012-1889-1.

[10] J. Alwall, et al., The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations, JHEP 07 (2014) 079. arXiv:1405.0301, doi:10.1007/JHEP07(2014)079.

[11] G. Ossola, C. G. Papadopoulos, R. Pittau, CutTools: a program implementing the OPP reduction method to compute one-loop amplitudes, JHEP 0803 (2008) 042. arXiv:0711.3596, doi:10.1088/1126-6708/2008/03/042.

[12] A. van Hameren, C. Papadopoulos, R. Pittau, Automated one-loop calculations: A proof of concept, JHEP 0909 (2009) 106. arXiv:0903.4665, doi:10.1088/1126-6708/2009/09/106.

[13] P. Mastrolia, G. Ossola, T. Reiter, F. Tramontano, Scattering amplitudes from unitarity-based reduction algorithm at the integrand-level, JHEP 1008 (2010) 080. arXiv:1006.0710, doi:10.1007/JHEP08(2010)080.

[14] V. Hirschi, et al., Automation of one-loop QCD corrections, JHEP 1105 (2011) 044. arXiv:1103.0621, doi:10.1007/JHEP05(2011)044.

[15] F. Cascioli, P. Maierhöfer, S. Pozzorini, Scattering amplitudes with Open Loops, Phys.Rev.Lett. 108 (2012) 111601. arXiv:1111.5206, doi:10.1103/PhysRevLett.108.111601.

[16] S. Actis, A. Denner, L. Hofer, A. Scharf, S. Uccirati, Recursive generation of one-loop amplitudes in the Standard Model, JHEP 1304 (2013) 037. arXiv:1211.6316, doi:10.1007/JHEP04(2013)037.

[17] S. Frixione, V. Hirschi, D. Pagani, H. S. Shao, M. Zaro, Weak corrections to Higgs hadroproduction in association with a top-quark pair, JHEP 09 (2014) 065. arXiv:1407.0823, doi:10.1007/JHEP09(2014)065.

[18] S. Frixione, V. Hirschi, D. Pagani, H. S. Shao, M. Zaro, Electroweak and QCD corrections to top-pair hadroproduction in association with heavy bosons, JHEP 06 (2015) 184. arXiv:1504.03446, doi:10.1007/JHEP06(2015)184.

[19] S. Kallweit, J. M. Lindert, P. Maierhöfer, S. Pozzorini, M. Schönherr, NLO electroweak automation and precise predictions for W+multijet production at the LHC, JHEP 04 (2015) 012. arXiv:1412.5157, doi:10.1007/JHEP04(2015)012.

[20] S. Kallweit, J. M. Lindert, S. Pozzorini, M. Schönherr, P. Maierhöfer, NLO QCD+EW predictions for V+jets including off-shell vector-boson decays and multijet merging (2015). arXiv:1511.08692.

[21] T. Hahn, Generating Feynman diagrams and amplitudes with FeynArts 3, Comput.Phys.Commun. 140 (2001) 418–431. arXiv:hep-ph/0012260, doi:10.1016/S0010-4655(01)00290-9.

[22] R. Mertig, M. Böhm, A. Denner, FEYN CALC: Computer algebraic calculation of Feynman amplitudes, Comput. Phys. Commun. 64 (1991) 345–359. doi:10.1016/0010-4655(91)90130-D.

[23] V. Shtabovenko, R. Mertig, F. Orellana, New Developments in FeynCalc 9.0 (2016). arXiv:1601.01167.

[24] T. Hahn, M. Perez-Victoria, Automatized one-loop calculations in four-dimensions and D-dimensions, Comput.Phys.Commun. 118 (1999) 153–165. arXiv:hep-ph/9807565, doi:10.1016/S0010-4655(98)00173-8.

[25] S. Agrawal, T. Hahn, E. Mirabella, FormCalc 7.5, PoS LL2012 (2012) 046. arXiv:1210.2628.

[26] B. Chokoufe Nejad, T. Hahn, J.-N. Lang, E. Mirabella, FormCalc 8: Better Algebra and Vectorization, J.Phys.Conf.Ser. 523 (2014) 012050. arXiv:1310.0274, doi:10.1088/1742-6596/523/1/012050.

[27] Z. Bern, L. J. Dixon, D. C. Dunbar, D. A. Kosower, One-loop n-point gauge theory amplitudes, unitarity and collinear limits, Nucl.Phys. B425 (1994) 217–260. arXiv:hep-ph/9403226, doi:10.1016/0550-3213(94)90179-1.

[28] Z. Bern, L. J. Dixon, D. C. Dunbar, D. A. Kosower, Fusing gauge theory tree amplitudes into loop amplitudes, Nucl.Phys. B435 (1995) 59–101. arXiv:hep-ph/9409265, doi:10.1016/0550-3213(94)00488-Z.

[29] R. Britto, F. Cachazo, B. Feng, Generalized unitarity and one-loop amplitudes in N=4 super-Yang-Mills, Nucl.Phys. B725 (2005) 275–305. arXiv:hep-th/0412103, doi:10.1016/j.nuclphysb.2005.07.014.

[30] G. Ossola, C. G. Papadopoulos, R. Pittau, Reducing full one-loop amplitudes to scalar integrals at the integrand level, Nucl. Phys. B763 (2007) 147–169. arXiv:hep-ph/0609007.

[31] R. K. Ellis, W. Giele, Z. Kunszt, A numerical unitarity formalism for evaluating one-loop amplitudes, JHEP 0803 (2008) 003. arXiv:0708.2398, doi:10.1088/1126-6708/2008/03/003.

[32] W. T. Giele, Z. Kunszt, K. Melnikov, Full one-loop amplitudes from tree amplitudes, JHEP 0804 (2008) 049. arXiv:0801.2237, doi:10.1088/1126-6708/2008/04/049.

[33] R. K. Ellis, W. T. Giele, Z. Kunszt, K. Melnikov, Masses, fermions and generalized D-dimensional unitarity, Nucl.Phys. B822 (2009) 270–282. arXiv:0806.3467, doi:10.1016/j.nuclphysb.2009.07.023.

[34] A. van Hameren, Multi-gluon one-loop amplitudes using tensor integrals, JHEP 0907 (2009) 088. arXiv:0905.1005, doi:10.1088/1126-6708/2009/07/088.

[35] L. M. Brown, R. P. Feynman, Radiative corrections to Compton scattering, Phys. Rev. 85 (1952) 231–244. doi:10.1103/PhysRev.85.231.

[36] D. Melrose, Reduction of Feynman diagrams, Nuovo Cim. 40 (1965) 181–213. doi:10.1007/BF02832919.

[37] G. Passarino, M. Veltman, One-loop corrections for e^+e^- annihilation into $\mu^+\mu^-$ in the Weinberg Model, Nucl.Phys. B160 (1979) 151. doi:10.1016/0550-3213(79)90234-7.

[38] W. van Neerven, J. Vermaseren, Large loop integrals, Phys.Lett. B137 (1984) 241. doi:10.1016/0370-2693(84)90237-5.

[39] G. van Oldenborgh, J. Vermaseren, New algorithms for one-loop integrals, Z.Phys. C46 (1990) 425–438. doi:10.1007/BF01621031.

[40] Z. Bern, L. J. Dixon, D. A. Kosower, Dimensionally regulated one-loop integrals, Phys.Lett. B302 (1993) 299–308. arXiv:hep-ph/9212308, doi:10.1016/0370-2693(93)90400-C.

[41] A. Denner, Techniques for calculation of electroweak radiative corrections at the one-loop level and results for W physics at LEP-200, Fortsch.Phys. 41 (1993) 307–420. arXiv:0709.1075, doi:10.1002/prop.2190410402.

[42] T. Binoth, J. Guillet, G. Heinrich, Reduction formalism for dimensionally regulated one-loop N-point integrals, Nucl.Phys. B572 (2000) 361–386. arXiv:hep-ph/9911342, doi:10.1016/S0550-3213(00)00040-7.

[43] A. Denner, S. Dittmaier, Reduction of one-loop tensor 5-point integrals, Nucl. Phys. B658 (2003) 175–202. arXiv:hep-ph/0212259, doi:10.1016/S0550-3213(03)00184-6.

[44] W. Beenakker, et al., NLO QCD corrections to t anti-t H production in hadron collisions, Nucl. Phys. B653 (2003) 151–203. arXiv:hep-ph/0211352, doi:10.1016/S0550-3213(03)00044-0.

[45] S. Dittmaier, Separation of soft and collinear singularities from one-loop N-point integrals, Nucl. Phys. B675 (2003) 447–466. arXiv:hep-ph/0308246.

[46] G. Duplancic, B. Nizic, Reduction method for dimensionally regulated one-loop N-point Feynman integrals, Eur.Phys.J. C35 (2004) 105–118. arXiv:hep-ph/0303184, doi:10.1140/epjc/s2004-01723-7.

[47] W. Giele, E. N. Glover, A calculational formalism for one-loop integrals, JHEP 0404 (2004) 029. arXiv:hep-ph/0402152, doi:10.1088/1126-6708/2004/04/029.

[48] W. Giele, E. W. N. Glover, G. Zanderighi, Numerical evaluation of one-loop diagrams near exceptional momentum configurations, Nucl. Phys. Proc. Suppl. 135 (2004) 275–279. arXiv:hep-ph/0407016.

[49] T. Binoth, J. P. Guillet, G. Heinrich, E. Pilon, C. Schubert, An algebraic / numerical formalism for one-loop multi-leg amplitudes, JHEP 10 (2005) 015. arXiv:hep-ph/0504267.

[50] A. Denner, S. Dittmaier, Reduction schemes for one-loop tensor integrals, Nucl. Phys. B734 (2006) 62–115. arXiv:hep-ph/0509141, doi:10.1016/j.nuclphysb.2005.11.007.

[51] J. Fleischer, T. Riemann, A Complete algebraic reduction of one-loop tensor Feynman integrals, Phys.Rev. D83 (2011) 073004. arXiv:1009.4436, doi:10.1103/PhysRevD.83.073004.

[52] G. 't Hooft, M. Veltman, Scalar one-loop integrals, Nucl.Phys. B153 (1979) 365–401. doi:10.1016/0550-3213(79)90605-9.

[53] W. Beenakker, A. Denner, Infrared divergent scalar box integrals with applications in the Electroweak Standard Model, Nucl. Phys. B338 (1990) 349–370. doi:10.1016/0550-3213(90)90636-R.

[54] Z. Bern, L. J. Dixon, D. A. Kosower, Dimensionally regulated pentagon integrals, Nucl.Phys. B412 (1994) 751–816. arXiv:hep-ph/9306240, doi:10.1016/0550-3213(94)90398-0.

[55] A. Denner, U. Nierste, R. Scharf, A compact expression for the scalar one-loop four-point function, Nucl. Phys. B367 (1991) 637–656. doi:10.1016/0550-3213(91)90011-L.

[56] A. Denner, S. Dittmaier, M. Roth, D. Wackeroth, Predictions for all processes $e^+e^- \to 4$ fermions $+\gamma$, Nucl.Phys. B560 (1999) 33–65. arXiv:hep-ph/9904472, doi:10.1016/S0550-3213(99)00437-X.

[57] A. Denner, S. Dittmaier, M. Roth, L. Wieders, Electroweak corrections to charged-current $e^+e^- \to 4$ fermion processes: Technical details and further results, Nucl.Phys. B724 (2005) 247–294, erratum-ibid. **B854** (2012) 504–507. arXiv:hep-ph/0505042, doi:10.1016/j.nuclphysb.2005.06.033,10.1016/j.nuclphysb.2011.09.001.

[58] D. T. Nhung, L. D. Ninh, D0C : A code to calculate scalar one-loop four-point integrals with complex masses, Comput.Phys.Commun. 180 (2009) 2258–2267. arXiv:0902.0325, doi:10.1016/j.cpc.2009.07.012.

[59] A. Denner, S. Dittmaier, Scalar one-loop 4-point integrals, Nucl.Phys. B844 (2011) 199–242. arXiv:1005.2076, doi:10.1016/j.nuclphysb.2010.11.002.

[60] G. van Oldenborgh, FF: A Package to evaluate one-loop Feynman diagrams, Comput.Phys.Commun. 66 (1991) 1–15. doi:10.1016/0010-4655(91)90002-3.

[61] R. K. Ellis, G. Zanderighi, Scalar one-loop integrals for QCD, JHEP 0802 (2008) 002. arXiv:0712.1851, doi:10.1088/1126-6708/2008/02/002.

[62] A. van Hameren, OneLOop: For the evaluation of one-loop scalar functions, Comput.Phys.Commun. 182 (2011) 2427–2438. arXiv:1007.4716, doi:10.1016/j.cpc.2011.06.011.

[63] G. Cullen, et al., Golem95C: A library for one-loop integrals with complex masses, Comput.Phys.Commun. 182 (2011) 2276–2284. arXiv:1101.5595, doi:10.1016/j.cpc.2011.05.015.

[64] H. H. Patel, Package-X: A Mathematica package for the analytic calculation of one-loop integrals, Comput. Phys. Commun. 197 (2015) 276–290. arXiv:1503.01469, doi:10.1016/j.cpc.2015.08.017.

[65] A. Denner, S. Dittmaier, M. Roth, L. H. Wieders, Complete electroweak $\mathcal{O}(\alpha)$ corrections to charged-current $e^+e^- \to 4$ fermion processes, Phys. Lett. B612 (2005) 223–232. arXiv:hep-ph/0502063.

[66] A. Bredenstein, A. Denner, S. Dittmaier, M. M. Weber, Precise predictions for the Higgs-boson decay $H \to WW/ZZ \to 4$ leptons, Phys. Rev. D74 (2006) 013004. arXiv:hep-ph/0604011.

[67] A. Bredenstein, A. Denner, S. Dittmaier, M. M. Weber, Radiative corrections to the semileptonic and hadronic Higgs-boson decays $H \to WW/ZZ \to 4$ fermions, JHEP 02 (2007) 080. arXiv:hep-ph/0611234.

[68] A. Bredenstein, A. Denner, S. Dittmaier, S. Pozzorini, NLO QCD corrections to $t\bar{t}b\bar{b}$ production at the LHC: 1. Quark-antiquark annihilation, JHEP 0808 (2008) 108. arXiv:0807.1248, doi:10.1088/1126-6708/2008/08/108.

[69] A. Bredenstein, A. Denner, S. Dittmaier, S. Pozzorini, NLO QCD corrections to pp $\to$ $t\bar{t}b\bar{b} + X$ at the LHC, Phys.Rev.Lett. 103 (2009) 012002. arXiv:0905.0110, doi:10.1103/PhysRevLett.103.012002.

[70] A. Bredenstein, A. Denner, S. Dittmaier, S. Pozzorini, NLO QCD Corrections to $t\bar{t}b\bar{b}$ Production at the LHC: 2. full hadronic results, JHEP 1003 (2010) 021. arXiv:1001.4006, doi:10.1007/JHEP03(2010)021.

[71] F. Cascioli, P. Maierhöfer, N. Moretti, S. Pozzorini, F. Siegert, NLO matching for $t\bar{t}b\bar{b}$ production with massive b-quarks, Phys. Lett. B734 (2014) 210–214. arXiv:1309.5912, doi:10.1016/j.physletb.2014.05.040.

[72] A. Denner, S. Dittmaier, S. Kallweit, S. Pozzorini, NLO QCD corrections to WWbb production at hadron colliders, Phys. Rev. Lett. 106 (2011) 052001. arXiv:1012.3975, doi:10.1103/PhysRevLett.106.052001.

[73] A. Denner, S. Dittmaier, S. Kallweit, S. Pozzorini, NLO QCD corrections to off-shell top-antitop production with leptonic decays at hadron colliders, JHEP 1210 (2012) 110. arXiv:1207.5018, doi:10.1007/JHEP10(2012)110.

[74] F. Cascioli, S. Kallweit, P. Maierhöfer, S. Pozzorini, A unified NLO description of top-pair and associated Wt production, Eur.Phys.J. C74 (2014) 2783. arXiv:1312.0546, doi:10.1140/epjc/s10052-014-2783-9.

[75] A. Denner, L. Hofer, A. Scharf, S. Uccirati, Electroweak corrections to lepton-pair production in association with two hard jets at the LHC, JHEP 01 (2015) 094. arXiv:1411.0916, doi:10.1007/JHEP01(2015)094.

[76] M. Billoni, S. Dittmaier, B. Jäger, C. Speckner, Next-to-leading-order electroweak corrections to pp $\to$ W^+W^- $\to$ 4 leptons at the LHC in double-pole approximation, JHEP 1312 (2013) 043. arXiv:1310.1564, doi:10.1007/JHEP12(2013)043.

[77] S. Höche, et al., Next-to-leading-order QCD predictions for top-quark pair production with up to two jets merged with a parton shower (2015). arXiv:1402.6293, doi:10.1016/j.physletb.2015.06.060.

[78] A. Denner, R. Feger, NLO QCD corrections to off-shell top-antitop production with leptonic decays in association with a Higgs boson at the LHC, JHEP 11 (2015) 209. arXiv:1506.07448, doi:10.1007/JHEP11(2015)209.

[79] B. Biedermann, A. Denner, S. Dittmaier, L. Hofer, B. Jäger, Electroweak corrections to pp $\to \mu^+\mu^-\mathrm{e}^+\mathrm{e}^- + X$ at the LHC – a Higgs background study (2016). arXiv:1601.07787.

[80] M. Grazzini, S. Kallweit, D. Rathlev, A. Torre, Zγ production at hadron colliders in NNLO QCD, Phys. Lett. B731 (2014) 204–207. arXiv:1309.7000, doi:10.1016/j.physletb.2014.02.037.

[81] F. Cascioli, et al., ZZ production at hadron colliders in NNLO QCD, Phys. Lett. B735 (2014) 311–313. arXiv:1405.2219, doi:10.1016/j.physletb.2014.06.056.

[82] T. Gehrmann, et al., $\mathrm{W}^+\mathrm{W}^-$ Production at Hadron Colliders in Next to Next to Leading Order QCD, Phys. Rev. Lett. 113 (21) (2014) 212001. arXiv:1408.5243, doi:10.1103/PhysRevLett.113.212001.