MITP/21-002, MSUHEP-20-016

MultivariateApart: Generalized Partial Fractions

Matthias Heller[a], Andreas von Manteuffel[b]

[a] *PRISMA$^+$ Cluster of Excellence, Johannes-Gutenberg Universität, 55099 Mainz, Deutschland*
[b] *Department of Physics and Astronomy, Michigan State University, East Lansing, Michigan 48824, USA*

Abstract

We present a package to perform partial fraction decompositions of multivariate rational functions. The algorithm allows to systematically avoid spurious denominator factors and is capable of producing unique results also when being applied to terms of a sum separately. The package is designed to work in Mathematica, but also provides interfaces to the Form and Singular computer algebra systems.

Email addresses: maheller@students.uni-mainz.de (Matthias Heller), vmante@msu.edu (Andreas von Manteuffel)

PROGRAM SUMMARY

Program Title: `MultivariateApart`
CPC Library link to program files: (to be added by Technical Editor)
Developer's repository link: `https://gitlab.msu.edu/vmante/multivariateapart`
Code Ocean capsule: (to be added by Technical Editor)
Licensing provisions: GPLv3
Programming language: Mathematica (Wolfram Language)
Nature of problem: Partial fraction decomposition is widely used in particle physics to bring rational functions into a unique form. Treating the multivariate case by applying the univariate method iteratively risks the introduction of spurious singularities. Here, we formulate an algorithm that handles non-linear multivariate denominators, avoids the introduction of spurious denominators, and aims at providing good performance for typical applications in particle physics. It can be used to obtain unique results also when decomposing individual terms of a sum separately, independent of the details of the input form. A variant of the approach allows to reach a Leĭnartas' decomposition as the output form.
Solution method: We reformulate the problem of calculating a partial fraction decomposition into the problem of reducing a polynomial with respect to a specific ideal. The polynomial reduction is based on a Gröbner basis calculation and guarantees a unique result for the partial fraction decomposition. We provide a complete implementation as a `Mathematica` package. Optionally, the included interfaces to `Form` and `Singular` can be used to speed up the computation.
Additional comments including restrictions and unusual features: Our algorithm does not introduce new singularities that were not presented in the input. If, however, the input contains spurious singularities, our package can be used in such a way that the cancellation of these singularities is not guaranteed. By default, the main MultivariateApart function of our package avoids this problem by first putting terms in a sum over a common denominator, and cancelling factors in the result.

Contents

1 Introduction **3**

2 Motivation **4**
2.1 Spurious poles in iterated partial fractions 4
2.2 Features of Leĭnartas' decomposition 5
2.3 A wish-list for a "good" partial fractioning algorithm 7

3 Multivariate partial fractions with polynomial reduction **8**
3.1 Reduction algorithm 8
3.2 Monomial ordering 9
3.3 Example for the algorithm 11
3.4 Efficient reduction of factorized inputs 12
3.5 A comment on rational reconstructions 13

4 The package **15**
4.1 Installation and basic usage 15
4.2 Advanced usage pattern 16
4.3 Interface with Singular 18
4.4 Interface with Form 19

5 Applications **19**
5.1 One-loop amplitudes for $\gamma^*\gamma^* \to l^+l^-$ with finite lepton mass . . 19
5.2 Two-loop amplitudes for $gg \to ZZ$ with a closed top-quark loop . 20
5.3 Two-loop amplitudes for massless five-particle scattering 22

6 Conclusion and outlook **24**

Appendices

A Polynomial reductions **25**

B Leĭnartas' requirements and polynomial reductions **27**

1. Introduction

Employing partial fraction decompositions to bring analytic expressions into a unique form has a long history in particle physics. Already one of the first computer algebra systems (CAS), Veltman's program `Schoonschip` [1], that was developed in 1963 to calculate radiative corrections, had a function to partial fraction products of two or three rational functions [2]. With `Schoonship`'s successor `Form` [3, 4], partial fraction decompositions of rational functions became widely established in the particle physics community. While the standard partial fraction decomposition is a method for rational functions of a single variable, it is often applied iteratively to treat multivariate functions. However, this generalization has two drawbacks: first this method generally introduces new

denominator factors, i.e. spurious singularities, and second, it is often slower than reduction schemes in which spurious singularities are avoided from the beginning. A method to separate linearly dependent denominators without introducing spurious poles was presented in Ref. [5] for the special case of multivariate but linear denominator factors. A representation for the general multivariate case which avoids spurious denominators is known as Leĭnartas' decomposition [6, 7, 8, 9, 10]. It separates denominators which don't share common zeros or are algebraically dependent.

Here, we discuss a method to compute a partial fraction decomposition via polynomial reductions. Our general approach coincides with that of Refs. [11, 12], but we present new insights concerning the choice of a good monomial ordering and the resulting output forms. In particular, we propose a new monomial ordering, which still guarantees the separation of denominator zeros but allows for deviations from Leĭnartas' form to allow for lower denominator degrees. We also discuss performance optimizations and other practical issues, such as the local elimination of specific denominators, if they are known to be spurious globally. We implement our algorithms in a `Mathematica` package which is publicly available. It can be used to partial fraction rational functions directly in `Mathematica`, but also allows to generate local replacement rules that can be incorporated in `Form`.

The outline of the paper is as follows. In Section 2 we motivate our method. We demonstrate the problem with iterated univariate partial fractions and discuss ambiguities when applying the "classical" Leĭnartas decomposition to individual terms of a sum. We spell out an explicit wish-list for an ideal partial fraction algorithm and discuss which items are addressed by available implementations and the one described in this article, respectively. In Section 3 we introduce our algorithms in detail, give examples and show how it can be used efficiently in complicated cases. Furthermore, we present some considerations concerning the reconstruction of rational functions from finite field samples, which can be useful to speed-up tasks like linear system solving. In Section 4 we give an introduction to our `Mathematica` package. We describe the relevant functions and give examples for their usage. In Section 5 we list "real world" applications of our methods to the calculation of Feynman amplitudes, often resulting in significantly reduced sizes of the mathematical expressions. In Section 6 we conclude and provide an outlook. Appendix A gives a brief introduction to polynomial reductions. Appendix B discusses the relation between Leĭnartas' decomposition and our polynomial reduction method.

2. Motivation

2.1. Spurious poles in iterated partial fractions

We consider a univariate rational function $r(x) = n(x)/d(x)$, where n and d are polynomials in x and the coefficient field K could be e.g. the set of rational numbers. The denominator $d(x)$ can be factored into irreducible factors $d_i(x)$ such that $d(x) = \prod_i d_i^{\alpha_i}(x)$, where the polynomials d_i can not be written as a

product of two non-constant polynomials and $\alpha_i \in \mathbb{N}$. The *univariate partial fraction decomposition* of $r(x)$ is given by

$$r(x) = \sum_i \sum_{j \leq \alpha_i} \frac{n_i(x)}{d_i^j(x)}, \tag{1}$$

where in each term the degree of the numerator is smaller than the degree of the denominator. As an example, consider the partial fraction decomposition

$$\frac{x}{(x-1)(x+1)^2} = -\frac{1}{4(x+1)} + \frac{1}{2(x+1)^2} + \frac{1}{4(x-1)}. \tag{2}$$

In the case of multivariate rational functions $r(x, y, \ldots)$ the partial fractioning can be performed iteratively in each variable. One performs a partial fraction decomposition with respect to the first variable x, treating the other variables as constants during this step. The resulting coefficients p_i are now rational functions of the remaining variables, and one can perform a partial fraction decomposition of the $p_i(y, \ldots)$ with respect to the next variable y. One iterates until the coefficients are actual numbers. As an example, consider the function $r(x, y) = 1/((x - f(y))(x - g(y)))$, where $f(y)$ and $g(y)$ are two different polynomials of y. A partial fractioning with respect to x gives

$$\frac{1}{(x-f(y))(x-g(y))} = \frac{1}{(f(y)-g(y))}\frac{1}{(x-f(y))} - \frac{1}{(f(y)-g(y))}\frac{1}{(x-g(y))}. \tag{3}$$

The partial fractioning in x thus introduces a denominator $f(y) - g(y)$, which may have zeros at regular points of the original rational function $r(x, y)$. For example, consider the special case

$$\frac{1}{(x+y)(x-y)} = \frac{1}{2y}\frac{1}{(x-y)} - \frac{1}{2y}\frac{1}{(x+y)}, \tag{4}$$

where the subsequent partial fractioning of the x-independent coefficients in y is trivial. Although the original expression is manifestly regular at $y = 0$, the individual terms of the partial fractioned form are not—they introduce spurious poles $1/y$. This obscures the interpretation of the singularity structure of the expression and can lead to loss of precision in its numerical evaluation close to $y = 0$. Note that a different spurious pole would have appeared in this example if we had chosen to first partial fraction in y and then in x.

We conclude that iterated partial fractioning will in general introduce spurious poles and is therefore not an ideal approach to the multivariate case.

2.2. Features of Leĭnartas' decomposition

Leĭnartas' decomposition is an approach to multivariate partial fractioning, which avoids the introduction of spurious singularities. In this Section, we review this decomposition and comment on possible ambiguities arising in practical calculations. Polynomial reductions and the usage of Gröbner bases will play an important role here and in the following; in case the reader is not familiar with these concepts we recommend to read Appendix A first.

Definition 1. *Leĭnartas' decomposition* [6, 7] of a rational function r of the variables $x_1, \ldots, x_n$ with coefficients in the field K is a decomposition of the form

$$r(x_1, \ldots) = \sum_{\mathcal{S}} \frac{n_{\mathcal{S}}(x_1, \ldots)}{\prod_{i \in \mathcal{S}} d_i^{\alpha_i}(x_1, \ldots)}, \tag{5}$$

where, for each term of the decomposition, $\mathcal{S}$ is an index set, such that all denominators d_i with $i \in \mathcal{S}$

(i) have common zeros in $\overline{K}^n$, and

(ii) are algebraically independent.

Here, $\overline{K}$ denotes the algebraic closure of K, e.g. the algebraic numbers for $K = \mathbb{Q}$ or $\mathbb{C}$ for $K = \mathbb{R}$.

A set of polynomials $\{d_1, \ldots, d_m\}$ is called *algebraically dependent* if there exists a non-zero polynomial A in m variables such that $A(d_1, \ldots, d_m) = 0$, see e.g. [13, 14]. A is called the *annihilator* of the ideal generated by the polynomials $\{d_1, \ldots, d_m\}$. The annihilator can be obtained by calculating the Gröbner basis of the ideal $\{y_1 - d_1(x_1, \ldots, x_n), \ldots, y_m - d_m(x_1, \ldots, x_n)\}$, if one chooses a monomial ordering in which $y_i \prec x_j \; \forall i, j$. A set of polynomials is called *algebraically independent* if it is not algebraically dependent. Since for n variables at most n polynomials are algebraically independent, there can be at most n different denominator factors for each term in (5).

Algorithm 1. *Leĭnartas' decomposition* of a rational function can be reached in two reduction steps [7]:

1. Use Hilbert's Nullstellensatz to decompose the denominator of r into several terms such that each term fulfills (i).
2. Calculate the annihilator for each term and use it to decompose each denominator to reach (ii).

In contrast to iterated partial fractioning, these decomposition steps do not introduce new singularities. Appendix B gives more details for this algorithm. A decomposition that fulfills requirements (i) and (ii) is not unique. In order to resolve this ambiguity, Refs. [9, 10] require additionally for each term (5), that the numerator $p_{\mathcal{S}}$ is reduced with respect to the ideal generated by the denominators $\{d_i | i \in \mathcal{S}\}$. We note that the algorithm of [10] assumes the input expression to be a single rational function $r = n/d$ with polynomials n and d and starts with a factorization of d.

Further ambiguities can arise due to spurious denominators which are not automatically removed. The algorithm in [10] for example does not guarantee a unique result, if n and d are not coprime. However, common factors between numerator and denominator can always be eliminated by first performing a greatest-common-divisor computation.

In practical applications one also encounters sums of rational functions, and it can be useful to decompose individual terms of the sum separately instead of

combining them over a common denominator first. With the methods spelled out so far, uniqueness is not guaranteed in this approach, even in absence of any spurious denominator. Let us illustrate this phenomenon by considering an example where

$$r(x,y) = \frac{2x-y}{y(x+y)(x-y)}. \tag{6}$$

Then a Leĭnartas decomposition of r is given by

$$r(x,y) = \frac{2}{y(x+y)} + \frac{1}{(x+y)(x-y)}. \tag{7}$$

Another Leĭnartas decomposition is given by

$$r(x,y) = \frac{2}{y(x-y)} - \frac{3}{(x+y)(x-y)}. \tag{8}$$

If we consider the terms in Eqs. (7) and (8) separately, each of them is in Leĭnartas' decomposed form and the numerator is reduced with respect to the denominator, but the resulting representation is different for the respective sums. In order resolve this ambiguity, some kind of "global" information needs to be incorporated into the decomposition of "local" terms. We will discuss a solution to this problem below, which does not necessarily require to bring the rational function into a common-denominator form first.

2.3. A wish-list for a "good" partial fractioning algorithm

Let us assume someone gives us a rational expression in any form, expanded, over a common denominator, or partially mixed. Then an "ideal" partial fraction decomposition would have the following properties

(i) it should give a unique answer,

(ii) it should not introduce spurious denominator factors,

(iii) it should commute with summation,

(iv) it should eliminate spurious denominators if they are present in the input.

Note that requirement (iii) can be crucial, if one aims at employing such a partial fraction algorithm in a system like `Form`. In such a case one usually starts with fully expanded expressions, uses local replacement rules, and then wants to obtain a unique answer, such that cancellations can take place.

Unfortunately, there is no known solution to fulfil all of these points simultaneously. In the next Section, we consider an algorithm that fulfills requirements (i), (ii) and (iii), and, in cases in which one knows which denominators are spurious in the final answer, also point (iv).

3. Multivariate partial fractions with polynomial reduction

3.1. Reduction algorithm

Let $\{d_1, \ldots, d_m\}$ be the irreducible denominators of a rational function or a sum of rational functions with $d_i \in K[x_1, \ldots, x_n]$. Consider the ideal

$$I = \langle q_1 d_1(x_1, \ldots) - 1, \ldots, q_m d_m(x_1, \ldots) - 1 \rangle \,, \tag{9}$$

where $I \subset K[q_1, \ldots, q_m, x_1, \ldots, x_n]$ and q_i label inverse denominators. The main idea here is that setting the generators of the ideal $q_i d_i(x_1, \ldots) - 1$ to zero corresponds to the relation $q_i = 1/d_i(x_1, \ldots)$. If we rewrite a rational function r as a polynomial in the variables $q_1, \ldots, x_1, \ldots$ and reduce it with respect to the ideal I, we do not introduce new denominator factors and obtain a unique representation. Furthermore, whenever we encounter a product $q_i d_i$ it will be reduced to 1. By choosing a suitable monomial ordering, one can control further features of the reduced form as will be discussed in more detail in Section 3.2 and Appendix B. Here, we only note that for any monomial ordering which sorts first for the q_i and then for the x_i, denominators with disjoint zeros will be separated and it is justified to call the reduction a partial fraction decomposition. Using such a monomial ordering, we thus achieve a multivariate partial fraction decomposition with the following two steps: 1. calculate the Gröbner basis of the ideal I and 2. reduce the rational function with respect to this Gröbner basis. This reduction yields a unique remainder, which is the partial fractioned form of r. A complete algorithm to bring a rational function into a unique partial fractioned form can be formulated as follows (see also [11, 12]).

Algorithm 2. *Multivariate partial fraction decomposition.* A rational function $r(x_1, \ldots) \in K[x_1, \ldots]$ can be decomposed into partial fractions using the following steps.

1. Bring the rational function into the form $n(x_1, \ldots)/d(x_1, \ldots)$ and cancel common factors in n and d such that they are coprime.
2. Factorize d over K. Let's call the irreducible factors of the denominator $d_i(x_1, \ldots)$ for $i = 1, \ldots, m$.
3. For each denominator factor $d_i(x_1, \ldots)$ introduce a new indeterminant q_i which represents the inverse of $d_i(x_1, \ldots)$, i.e. $q_i = 1/d_i(x_1, \ldots)$. Express all denominators in the problem in terms of the q_i such that the rational function becomes a polynomial $p \in K[q_1, \ldots, x_1, \ldots]$.
4. Calculate the Gröbner basis of the ideal I generated by $\{q_1 d_1(x_i) - 1, \ldots, q_m d_m(x_i) - 1\}$ using a monomial ordering which sorts first for the q_i and then for the x_i.
5. Find the fully reduced form of the polynomial p with respect to the Gröbner basis.
6. Replace back $q_i \to 1/d_i$.

Step 1 eliminates spurious poles and ensures a unique final form. When operating on large sums, it may be useful to skip this step and operate on the

individual terms of the sum separately. In this situation, one can still arrive at a unique output for the sum when considering the decomposition of each term separately, but using the global set of denominators in the sum. For instance, for the example of Section 2.2, Algorithm 2 finds a unique form independent of the specific initial form of r, see Eqs. (7) and (8), once all possible denominators in the problem are specified.

In our algorithm, the choice of the monomial ordering uniquely determines which denominators are preferred in the partial fractioned form, resolving an ambiguity present in the standard Leĭnartas decomposition as given e.g. in Ref. [7]. The elimination of denominators which are known to be spurious can then be achieved by a suitable choice of the monomial ordering; this will be discussed in the next Section. We stress that spurious denominators may be detected by computationally less demanding methods than the greatest-common-divisor computation suggested in Step 1, for example using the sampling techniques discussed in Section 3.5.

Step 6 is optional, of course. Leaving the denominators in abbreviated form can allow for more compact representations and as a preparation for an efficient numerical implementation.

3.2. Monomial ordering

The choice of monomial ordering has a crucial impact on the properties of the output of Algorithm 2 and the performance of its computation. We show in Appendix B that the output form will satisfy Leĭnartas' requirement (i) and thus separate denominators with disjoint zeros as long as the monomial ordering orders all q_i before the x_i, while Leĭnartas' requirement (ii) may or may not be satisfied depending on further details.

The calculation of the Gröbner basis can be very challenging for practical applications of Algorithm 2, depending heavily on the choice of monomial ordering. Here we propose a specific monomial ordering which aims to provide good computational performance. We tested this ordering in various calculations of scattering amplitudes, and we were able to compute the Gröbner basis and partial fraction the required rational functions in all cases that we considered.

To motivate our choice of monomial ordering, let us go back to Eq. (3). Note that the prefactors in front of the denominators $x - f(y)$ and $x - g(y)$ do not depend on x. In the case where now $f(y)-g(y)$ is itself a valid denominator, this identity would therefore actually be a valid replacement rule and corresponds to a polynomial in the ideal I in Eq. (9). To ensure that this polynomial leads to the reduction (3), one can choose a monomial ordering, in which the inverse denominator variables q_1 and q_2 corresponding to $d_1 = x - f(y)$ and $d_2 = x - g(y)$, respectively, are "greater" than the inverse denominator variable q_3 corresponding to $d_3 = f(y)-g(y)$. This motivates the use of a block ordering, in which we group all denominators depending on x and y and all denominators depending on only x or only y.

Algorithm 3. *Monomial block ordering.* A suitable monomial ordering for the ring $K[q_1, \ldots, q_m, x_1, \ldots, x_n]$ in Algorithm 2 can be constructed as follows.

1. Group the denominators $d_1, \ldots, d_m$ by their dependence on all variables $x_1, \ldots, x_n$, such that denominators which depend on the same set of variables form a group; each group will correspond to a block in the monomial ordering.

2. Sort the groups according to the number of variables they depend on; a group with denominators depending on fewer variables is considered "smaller" than a group with denominators depending on more variables.

3. In each group, sort the denominators according to their total degree.

4. Replace the denominators d_i by the corresponding inverse denominator variables q_i and add a last group containing the variables $x_1, \ldots, x_n$.

5. Let the sequence of groups of variables define the blocks of a monomial ordering. Within each block, use the "standard" degree reverse lexicographic ordering (degrevlex).

Let us consider at an example with the denominators

$$d_1 = x^2 + y,\ d_2 = x - y,\ d_3 = x + 1,\ d_4 = x^2 - 3,\ d_5 = y + 1,\ d_6 = y\,. \quad (10)$$

The ideal I is generated by the polynomials

$$\{q_1(x^2+y)-1, q_2(x-y)-1, q_3(x+1)-1, q_4(x^2-3)-1, q_5(y+1)-1, q_6 y-1\}. \quad (11)$$

In steps 1-3, we identify the three groups of denominators

$$\{\{d_1 = x^2 + y,\ d_2 = x - y\}, \{d_4 = x^2 - 3,\ d_3 = x + 1\}, \{d_5 = y + 1,\ d_6 = y\}\}, \quad (12)$$

which gives in step 4

$$\{\{q_1, q_2\}, \{q_4, q_3\}, \{q_5, q_6\}, \{x, y\}\}. \quad (13)$$

This defines a monomial ordering with 4 blocks, see step 5, which would be used to calculate the Gröbner basis and the subsequent polynomial reductions.

Our proposal for the monomial ordering aims to reduce the computational effort of the decomposition and to prefer low degrees of the denominator and numerator polynomials. In our experiments, it allowed for a significantly faster calculation of the Gröbner basis than a global degree ordering or a lexicographical ordering. We would like to point out that our choice of monomial order will in general only guarantee Leĭnartas' requirement (i) but not (ii). A lexicographical ordering could achieve a Leĭnartas decomposition fulfilling both requirements, but potentially at the price of significantly increased polynomial degrees. We give details and an example in Appendix B.

3.3. Example for the algorithm

We consider again the rational expression in Eq. (6),

$$r = \frac{2x - y}{(x - y)y(x + y)}, \tag{14}$$

and apply our partial fraction algorithm to it. We identify 3 irreducible denominators and label their inverses according to

$$q_1 = \frac{1}{x - y}, \quad q_2 = \frac{1}{y}, \quad q_3 = \frac{1}{x + y}, \tag{15}$$

such that we can write r as a polynomial,

$$r = (2x - y)q_1 q_2 q_3 \,. \tag{16}$$

Corresponding to the identification in Eq. (15), we consider the ideal

$$I = \langle q_1(x - y) - 1,\ q_2 y - 1,\ q_3(x + y) - 1 \rangle \,. \tag{17}$$

With the monomial ordering defined by

$$\{\{q_3, q_1\},\ \{q_2\},\ \{x, y\}\}, \tag{18}$$

we obtain a Gröbner basis of I as

$$G = \{-1 + q_2 y, -1 + q_1 x - q_1 y, -1 + q_3 x + q_3 y, -q_1 q_2 + 2q_1 q_3 + q_2 q_3\}. \tag{19}$$

Reducing the polynomial representation of r in Eq. (16) with respect to G yields the reduced form

$$r = \frac{1}{2} q_1 q_2 + \frac{3}{2} q_2 q_3 \tag{20}$$

$$= \frac{1}{2(x - y)y} + \frac{3}{2y(x + y)}, \tag{21}$$

which is our partial fractioned result. Let us now assume we start with the equivalent expression from Eq. (7), i.e.

$$r = \frac{2}{y(x + y)} + \frac{1}{(x - y)(x + y)} = 2q_2 q_3 + q_1 q_2. \tag{22}$$

Reducing this expression with respect to G indeed yields exactly the same result as before, given by Eq. (20). Therefore, in this example, our algorithm recognizes that both input forms of r are equivalent, and gives a unique result. If we however start in another form, in which spurious denominators occur, one has to be careful.

Let us consider a different representation of the same rational expression r in (14), this time containing the additional spurious denominator x:

$$r = \frac{2}{y(x + y)} + \frac{1}{2x(x - y)} + \frac{1}{2x(x + y)}. \tag{23}$$

In this case we would identify four denominators and consider the ideal

$$I = \langle q_1(x-y)-1,\ q_2 y-1,\ q_3(x+y)-1,\ q_4 x-1\rangle\,. \tag{24}$$

Our method produces in this case the block ordering

$$\{\{q_3, q_1\},\ \{q_2\},\ \{q_4\},\ \{x, y\}\}. \tag{25}$$

Calculating the corresponding Gröbner basis and reducing with respect to that basis yields

$$r = \frac{1}{2} q_1 q_4 + 2 q_2 q_4 - \frac{3}{2} q_3 q_4\,. \tag{26}$$

We see that the spurious inverse denominator q_4 does not drop out, since we use a monomial ordering in which $q_4 \prec q_1, q_2, q_3$. However, if we would have chosen a different monomial ordering, in which q_4 is greater than all other q_i, the spurious denominator would drop out after reduction. In this way, one can locally eliminate denominators which are known to be spurious or add additional denominators to the problem without altering previous reductions. These features are also supported by our package, see Section 4.

3.4. Efficient reduction of factorized inputs

If the input expression is in a form with a single common denominator, i.e.

$$r = q_1^{\alpha_1} \cdots q_m^{\alpha_m} \times \mathcal{N}, \tag{27}$$

where $\mathcal{N}$ is a possibly lengthy numerator, it may be optimal from a performance perspective to just fully expand the polynomial prior to reduction. In this case, we propose to the following guided scheme.

Algorithm 4. *Iterated reductions.* The polynomial reduction of r in Eq. (27) with respect to I can be performed as follows.

1. Set $p = \mathcal{N}$ and $Q = \{q_1^{\alpha_1}, \ldots, q_m^{\alpha_m}\}$.
2. Identify the "smallest" inverse denominator factor q_i in Q according to the monomial ordering defined in Section 3.2. Set $p \leftarrow p \times q_i^{\alpha_i}$ and remove $q_i^{\alpha_i}$ from Q.
3. Replace p by its reduced form w.r.t. the Gröbner basis of the ideal I.
4. If Q is non-empty, goto step 2. Otherwise, stop and return p.

The reason why this reduction scheme can be much faster than a "naive" direct reduction is, that the decomposed form of the common denominator $q_1^{\alpha_1}, \ldots, q_m^{\alpha_m}$ alone can result in huge expression, which only shortens once the numerator is taken into account. The iterative reduction of one denominator at a time avoids this intermediate expression swell, because it avoids the clustering of many denominators at one reduction step.

In the `Mathematica` functions of our package, the iterated scheme above can be selected as an option. Our `Form` implementation of the polynomial reductions uses this scheme by default. We observed an additional speed-up in our implementation in `Mathematica` by partitioning the whole expression into smaller pieces and reducing these individually in step 2. An example with a comparison of timings for different sizes of these partitions can be found in Section 5.3.

3.5. A comment on rational reconstructions

Symbolic manipulations of large rational expressions can be computationally expensive. Finite field sampling and rational reconstruction techniques allow to prevent intermediate expression swell and have become quite standard in high energy physics calculations today [15, 16, 17, 18, 19, 20]. The basic idea is to set the indeterminates to numerical values in prime fields, perform all complicated manipulations with machine-sized integers instead of rational functions, and to reconstruct the final rational function of interest from many such samples. In this Section, we would like to discuss the usage of information about the denominator structure for an improved reconstruction of rational functions. A major goal is to reduce the number of samples needed to reconstruct the rational functions, thus speeding up the computation. Furthermore, in applications involving linear relations between Feynman integrals, the partial fractioned forms of the coefficients are often particularly simple and it would be a significant advantage to directly reconstruct the partial fractioned form.

We first start with an observation concerning the prediction of denominator factors in linear relations between Feynman integrals, assuming the rational functions appearing in the coefficients are to be reconstructed from numerical samples. Linear relations between Feynman integrals can be generated systematically, notably from integration-by-parts (IBP) identities [21, 22], and they typically involve only a small set of denominator factors in their row echelon form. Moreover, choosing appropriate basis or master integrals can help to avoid spurious denominators in the calculation. In practice, the analysis of the denominator factors may be performed by considering only a small set of linear relations for a specific sector (set of distinct propagators) and setting subsector integrals (with fewer distinct propagators) to zero [23, 24]. This results in a list of possible denominator factors to be expected for a specific rational function. This knowledge helps to predict the denominator structure of the multivariate rational function e.g. by univariate reconstructions, see also [25].

Here, we propose a new method, which allows to straight-forwardly guess denominator factors and their powers for a rational function, based on a list of candidate denominator factors and a small number of numerical probes (samples) of the rational function to reconstruct. Once the denominator is known or at least partial information about it, the full rational function can be reconstructed from (substantially) fewer samples. Let us assume we analyzed our rational functions and expect a list of irreducible denominator factors $d_1, \ldots, d_m$ in the result. We assume that the coefficient to be reconstructed is of the form

$$r = \frac{\mathcal{N}}{d_1^{\alpha_1} \cdots d_m^{\alpha_m}}, \tag{28}$$

where $\mathcal{N}$, d_1, ..., d_m are polynomials in $\mathbb{Q}[x_1, \ldots, x_n]$, and $\alpha_1, \ldots \in \mathbb{N}_0$.

Recipe 1. *Denominator guess.* For a rational function, which is expected to be of the form (28) with known denominator factors $d_1(x_1, \ldots), \ldots$, the following construction provides a guess for the full denominator.

1. Find integer samples for all $x_1, \ldots, x_n$ such that all $d_1, \ldots, d_m$ have a distinct, largest prime factor, which we denote by p_i, and each of these largest prime factors has multiplicity one. This requirement can be weakened but the stated form simplifies the following analysis.
2. Evaluate the rational function for the numerical values of the $x_1, \ldots, x_n$. In practice, this may mean e.g. linear system solving for several prime fields, Chinese remaindering and rational reconstruction. The result of this step is a number $r_{\text{num}} \in \mathbb{Q}$, which can be written in the form $r_{\text{num}} = n_{\text{num}}/d_{\text{num}}$ with $n_{\text{num}}, d_{\text{num}} \in \mathbb{Z}$.
3. Perform a prime factor decomposition of the integer number d_{num}. Due to Eq. (28), each factor $d_i^{\alpha_i}$ must contribute $p_i^{\alpha_i}$ to the prime factors in d_{num}. On the other hand, if we find a factor $p_i^{\alpha_i}$, we can take it as an indication that it was generated by $d_i^{\alpha_i}$ in the denominator. In this way, we can make a guess for the denominator just by counting the multiplicities of the p_i in the factorization of d_{num}.

In step 3 our guess may be off due the presence of a new denominator factor which was not in the candidate list or a factor p_i appearing in the coefficients of the numerator polynomial $\mathcal{N}$. One could imagine to repeat the above construction for several samples of the indeterminates and combine this information to validate or correct the guess. In our experiments we have just picked somewhat larger integers and were in fact not even faced with these problems for the cases that we checked.

The above method lets us therefore identify (parts of) the denominator of the expression we want to reconstruct. A robust way to use this guess for the full reconstruction of a rational function is to multiply all samples with this denominator and reconstruct the resulting expression as if it was a rational function. If the guess was accurate, only the numerator needs to be reconstructed, thereby reducing the number of required samples. We emphasize that this method will allow for a successful reconstruction of the result even if the guess was incomplete or inaccurate. After reconstruction one can perform the partial fractioning as discussed in Section 3.

We successfully applied this method to integration-by-parts reductions of Feynman integrals for problems with up to three loops. Considering the reconstruction of just a single variable, we typically observe a reduction of the number of samples by a factor of about 2 due to the guessed denominators. For a reduction problem with five variables, the iterated reconstruction of all variables resulted in an overall reduction of the number of samples by about a factor of 10.

As a next step, one can ask if it is possible to directly reconstruct the result in partial fractioned form. Indeed, assuming one knows all irreducible denominator factors of an expression beforehand, our method in Section 3 actually allows to systematically construct a basis of all monomials that can be expected in the final partial fractioned result, using a bound on the monomial degree in the construction. This, in principle, allows to match against an ansatz and reconstruct the partial fractioned form directly. We observed in our experiments

that depending on the exact strategy the ansatz in partial fractioned form may require more samples to fix the free coefficients than the reconstruction in the common denominator representation along the lines described above. Furthermore, an incomplete list of denominator factors would prevent a successful fit of a partial fractioned ansatz as well as an insufficient degree of the ansatz. In contrast, the method for the reconstruction in common-denominator form presented above is robust with respect to incomplete denominator information.

4. The package

4.1. Installation and basic usage

We implement the described algorithm in a `Mathematica` package, that can be downloaded as follows:

```
git clone https://gitlab.msu.edu/vmante/multivariateapart.git
```

The package consists of just a single file, `MultivariateApart.wl`, which could also be downloaded individually. We recommend to place it in a standard Mathematica packages directory, e.g. `~/.Mathematica/Applications`, such that it can be loaded with

```
In[1]:= Needs["MultivariateApart`"]
```

from any directory. Alternatively, it can always be loaded by specifying the full path to the file. The main function, which implements the algorithm described in Section 3 can be called by `MultivariateApart`. Using this command on the example of Section 3.3 yields:

```
In[2]:= MultivariateApart[(2x-y)/(y(x+y)(x-y))]

Out[2]= 1/(2(x-y)y) + 3/(2y(x+y))
```

The function provides a unique representation and may be all that is needed in many cases.

It can be convenient to keep the inverse factors abbreviated for further processing, which can be achieved by using `MultivariateAbbreviatedApart` instead:

```
In[3]:= MultivariateAbbreviatedApart[(2x-y)/(y(x+y)(x-y))] // Normal

Out[3]= {(q1 q2)/2 + (3 q2 q3)/2, {q1 → 1/(x - y), q2 → 1/y,
          q3 → 1/(x + y)}}
```

Here, `Normal` was used to convert the dispatch table in the second element of the returned list into normal form. One can pass a list of previously identified denominator factors as a second argument to `MultivariateAbbreviatedApart`:

```
In[4]:= MultivariateAbbreviatedApart[(2x-y)/(y(x+y)(x-y)), {x-y,y,x+y}]
```

This allows one to work with a global list of abbreviations across several expressions.

The package provides further functions and options, which may be useful for more granular control, performance tuning and interfacing with other computer algebra systems. We will describe some of this functionality in the following. Please see the output of `? MultivariateApart`* for a full list of available functions and options.

4.2. Advanced usage pattern

The "all-in-one" functions described so far internally call a sequence of other functions, which can also be used directly for more advanced use cases. With default options, the first step is a call to `Together` to eliminate spurious denominators. This behavior as well as other details can be configured through the options of the function. To eliminate spurious denominators without the computationally expensive call to `Together`, one can find the list of non-spurious denominators as described in Section 3.5 and adjust the monomial ordering accordingly. For our example, of course, this step is irrelevant since there are no spurious denominators. Next, `AbbreviateDenominators` is called, which finds all denominators in an expression and replaces them by a predefined symbol (the default is q_i).

```
In[5]:= {expr, dens, qis} = AbbreviateDenominators[(2x-y)/(y(x+y)(x-y))]

Out[5]= {q1 q2 q3 (2 x - y), {x - y, y, x + y}, {q1, q2, q3}}
```

The output is a list, whose first entry is the expression in terms of the denominator symbols, the second entry a list of all irreducible denominator factors and the last entry a list of their abbreviations. Optionally, `AbbreviateDenominators` accepts a list of denominator factors as a second argument to support working with a global list of abbreviations. Next, `ApartOrder` is used to define the monomial order:

```
In[6]:= ord = ApartOrder[dens, qis]

Out[6]= {{q3, q1}, {q2}, {x, y}}
```

The returned list represents the monomial order that will be used for the Gröbner basis computation and the polynomial reductions, see Section 3.2. Here, the list is given in a package-specific format, which can be converted to `Mathematica`'s matrix order format as used for the option `MonomialOrder` of the functions `GroebnerBasis` and `PolynomialReduce`. This conversion is performed by the function `DegRevLexWeights`. For our example, we have

```
In[7]:= DegRevLexWeights[ord]

Out[7]= {{1, 1, 0, 0, 0}, {0, -1, 0, 0, 0}, {0, 0, 1, 0, 0},
         {0, 0, 0, 1, 1}, {0, 0, 0, 0, -1}}
```

The Gröbner basis is obtained with

```
In[8]:= gb = ApartBasis[dens, qis, ord]

Out[8]= {-1 + q2 y, -1 + q1 x - q1 y, -1 + q3 x + q3 y,
         -q1 q2 + 2 q1 q3 + q2 q3}
```

The actual reduction or partial fractioning is performed with

```
In[9]:= ApartReduce[expr, gb, ord]
```

$$\text{Out[9]=}\quad \frac{q1\ q2}{2} + \frac{3\ q2\ q3}{2}$$

This example also shows the alternative way of using our package. Instead of using `MultivariateApart` on a single expression, we can think about the situation where we know all denominators of a set of expressions beforehand, and then want to bring a give expression into a unique form. Let us assume we have an additional denominator x in the problem. We can then first calculate the Gröbner base including this additional denominator and then reduce with respect to that base. As we have seen already in Section 3.3 there is no guarantee that an expression is free of spurious singularities in this case:

```
In[10]:= {dens, qis} = {{x - y, y, x + y, x}, {q1, q2, q3, q4}};
         ord = ApartOrder[dens, qis]
```

```
Out[10]= {{q3, q1}, {q2}, {q4}, {x, y}}
```

Note that the algorithm considers $q_4 = 1/x$ to be "simple" and will try to eliminate the other q_i in favor of q_4.

```
In[11]:= gb = ApartBasis[dens, qis, ord];
         ApartReduce[expr, gb, ord]
```

$$\text{Out[11]=}\quad \frac{q1\ q4}{2} + 2\ q2\ q4 - \frac{3\ q3\ q4}{2}$$

Here, the pole $q_4 = 1/x$ is actually spurious.

If we knew already beforehand that q_4 is spurious, we could have used a monomial order such that q_4 drops out (one could think of a situation, in which one knows that some denominators should not occur in the final answer). In this example, this can be done by changing the list `ord` in the following way: we just reorder the blocks in `ord` by moving the block with q_4 to the left most position (i.e. that block is greater than all other):

```
In[12]:= ord = {{q4}, {q3, q1}, {q2}, {x, y}};
         gb = ApartBasis[dens, qis, ord];
         ApartReduce[expr, gb, ord]
```

$$\text{Out[12]=}\quad \frac{q1\ q2}{2} + \frac{3\ q2\ q3}{2}$$

As expected, the result is free of the spurious pole q_4 and we find the same answer, as using `MultivariateApart` directly on the whole expression. Let us demonstrate the important fact that, once a monomial order is chosen, the final answer is essentially insensitive to the exact input form, and one can equally well operate on individual terms of a sum:

```
In[13]:= terms = ApartReduce[q1 q4 / 2, gb, ord] +
          ApartReduce[2 q2 q4, gb, ord] +
          ApartReduce[-3 q3 q4 / 2, gb, ord]
```

$$\text{Out[13]=}\quad 2\ q2\ q4 + \frac{1}{2}(q1\ q2 - q2\ q4) + \frac{3}{2}(q2\ q3 - q2\ q4)$$

Note how in this case the partial fractioned form of the individual terms actually exhibit the pole q_4, but they cancel in the sum after a call to `Expand`,

```
In[14]:= Expand[terms]

Out[14]= q1 q2/2 + 3 q2 q3/2
```

which is the same result as before.

4.3. Interface with Singular

Although our package can be used entirely in `Mathematica`, for complicated examples with many denominators and/or high polynomial degrees of the denominators, the Gröbner basis calculation in `ApartBasis` may become computationally challenging. It is generally difficult to predict which algorithm works best to compute a Gröbner basis for a given set of generators. For the applications in Sections 5.1 and 5.2, we found the `slimgb()` command in `Singular` [26] to perform significantly better than `Mathematica`'s `GroebnerBasis` command used by `ApartBasis`. We suggest to first try to use the default `Mathematica` implementation, and, if the computation takes too long or fails, try to use `Singular`.

Our package provides the function `WriteSingularBasisInput` to generate an input file for the calculation of the Gröbner basis. To describe the usage, we return to the previous example:

```
In[15]:= {expr, dens, qis} = AbbreviateDenominators[(2x-y)/(y(x+y)(x-y))]

Out[15]= {q1 q2 q3 (2 x - y), {x - y, y, x + y}, {q1, q2, q3}}

In[16]:= ord = ApartOrder[dens, qis]

Out[16]= {{q3, q1}, {q2}, {x, y}}
```

We can now prepare the Gröbner basis calculation with the command

```
In[17]:= WriteSingularBasisInput[dens, qis, ord, "dir"]
```

Here, `"dir"` denotes the name of the directory to which input and output files are written. The specified directory will be created if it does not exist. All commands for `Singular` are stored in the input file `apartbasisin.sing`, which can be executed with the command `Singular apartbasisin.sing`. The output of this `Singular` run is the file `apartbasisout.m`, which contains the Gröbner basis. We can load it in `Mathematica` with

```
In[18]:= Get["apartbasisout.m"]

Out[18]= {-1 + q2 y, -1 + q1 x - q1 y, -1 + q3 x + q3 y,
          -q1 q2 + 2 q1 q3 + q2 q3}
```

This is the same result as in the previous Section, where we used `ApartBasis`.

We note that `Singular` provides support for polynomial reductions and can therefore be used instead of our function `ApartReduce`. For mostly expanded inputs, we found the direct reduction in `Singular` to perform very well. For factorized inputs, our iterated reduction scheme in Section 2.1 may improve the run times significantly. We consider to add support for iterated reductions with `Singular` in a future version of our package.

4.4. Interface with Form

We emphasize that the polynomial reduction step of our algorithm allows a straight forward implementation in symbolic manipulation programs such as `Form`, in which one uses `identify` statements to "locally" replace parts of expressions, that is, one considers one term of a potentially large sum at a time. Having calculated the Gröbner basis, these `identify` statements can be derived by considering the leading term of each generator and generating rules that replace these leading terms. We have successfully used our methods in this way and added the function `WriteFormReduceInput` in our package. It can be invoked by

```
In[19]:= WriteFormReduceInput[gb, ord, "dir"]
```

Here, `gb` is the Gröbner base from which the replacement rules are derived, `ord` is the monomial order that was used and `"dir"` is the directory where the `Form` procedure shall be written to. The command generates a `Form` input file called `"apartreduce.h"`, containing a procedure that implements the reduction scheme described in Section 3.4, and two auxiliary files, which have to be in the same directory. For usage in `Form`, the file has to be included in the beginning of a `Form` script with `"#include apartreduce.h"` and the reduction algorithm can be invoked by `"#call apartreduce(expr)"`, where `expr` is a local `Form` expression. Note that `"apartreduce.h"` contains the definition of the denominator symbols in a specific order, which is needed for an efficient reduction. The user should therefore *not define these symbols* at another point in his or her `Form` scripts.

It is also possible to use `Form` as a back end for the computation of polynomial reductions in `MultivariateApart`, `MultivariateAbbreviatedApart`, and `ApartReduce` by specifying the option `UseFormProgram -> True`. For example, the command

```
In[20]:= ApartReduce[expr, gb, ord, UseFormProgram->True]
```

calls the external program `Form` to perform the actual computation. Here, the `Form` executable is assumed to be in the user's path, and the directory `$ApartTemporaryDirectory` will be used for all temporary files.

5. Applications

The methods described in this paper have been successfully applied to several calculations of scattering amplitudes in particle physics. In this Section, we provide some details for these applications; please see also Section 3.5 for remarks on IBP reductions of Feynman integrals.

5.1. One-loop amplitudes for $\gamma^*\gamma^* \to l^+l^-$ with finite lepton mass

One of us calculated all next-to-leading-order helicity amplitudes for the process

$$\gamma^*(p_1) + \gamma^*(p_2) \to l^+(p_3) + l^-(p_4), \tag{29}$$

where the quantities in parenthesis denote the four-momenta of the particles. The calculation [27, 28] included the full dependence on the lepton mass using the ideas of the present work. Defining the kinematic quantities

$$p_1^2 = t_1, \quad p_2^2 = t_2, \quad p_3^2 = p_4^2 = m^2, \quad (p_1 + p_2)^2 = s, \quad (p_1 - p_3)^2 = t, \quad (30)$$

and setting $m^2 \equiv 1$, one encounters a total of 15 denominators in the IBP reduction of the Feynman integrals contributing to the amplitudes. In order to simplify the coefficients in these reduction identities, we use our package to order them as specified in Section 3.2. We find the following ordering of the denominators

$$\begin{aligned}
&\{\{s - 2\,s\,t + s^2\,t + s\,t^2 - s\,t_1 - s\,t\,t_1 + t_1^2 - s\,t_2 - s\,t\,t_2 - 2\,t_1\,t_2 + s\,t_1\,t_2 + t_2^2,\\
&\ 1 - 2\,s + s^2 - 2\,t + 2\,s\,t + t^2 - 4\,t_1 + 2\,t_2 - 2\,s\,t_2 - 2\,t\,t_2 + t_2^2,\\
&\ 1 - 2\,s + s^2 - 2\,t + 2\,s\,t + t^2 + 2\,t_1 - 2\,s\,t_1 - 2\,t\,t_1 + t_1^2 - 4\,t_2,\\
&\ 2 - s - t + t_1 + t_2, 1 - s - t + t_1 + t_2\}, \{s^2 - 2\,s\,t_1 + t_1^2 - 2\,s\,t_2 - 2\,t_1\,t_2 + t_2^2,\\
&\ s - t_1 - t_2\}, \{1 - 2\,t + t^2 - 2\,t_2 - 2\,t\,t_2 + t_2^2\}, \{1 - 2\,t + t^2 - 2\,t_1 - 2\,t\,t_1 + t_1^2\},\\
&\ \{t, -1 + t\}, \{s, 4 - s\}, \{t_2\}, \{t_1\}\}. \qquad (31)
\end{aligned}$$

Using the block ordering we were able calculate the Gröbner basis for the ideal formed from these denominators using `slimgb` in `Singular` in a few minutes.

It is worth noting, that three of these denominators are spurious and result as an artefact of the basis of Lorentz invariant building blocks, that are used to build the amplitudes [29]. Namely, the three denominators that should drop out in the end are

$$t_1,\ t_2,\ s - t_1 - t_2. \qquad (32)$$

Changing the monomial order slightly, such that these denominators form an extra block which is "greater" than all other denominators, we are still able to calculate the Gröbner basis of this ideal. We furthermore find that in the final answer these denominators indeed cancel out.

Comparing the size of the results in factorized form and partial fractioned form, we find a reduction by factor 7 on average. However, some individual amplitudes are reduced by a factor of 20 to 30.

5.2. Two-loop amplitudes for $gg \to ZZ$ with a closed top-quark loop

The algorithm presented in this work has also been applied in Ref. [30] to the calculation of the two-loop scattering amplitudes for the process

$$g(p_1) + g(p_2) \to Z(p_3) + Z(p_4), \qquad (33)$$

specifically, the corrections involving a closed top-quark loop. In that calculation,

$$p_1^2 = p_2^2 = 0, \quad p_3^2 = p_4^2 = m_z^2, \quad (p_1 + p_2)^2 = s, \quad (p_1 - p_3)^2 = t, \qquad (34)$$

was used for the kinematics and the masses of the top quark and the Z boson, m_t and m_z, have been fixed by $m_t = 1$ and $(m_z/m_t)^2 = 5/18$. Using IBP reductions

it is possible to choose a basis of master integrals such that the dependence on d factorizes from denominators involving the kinematic invariants s and t. Here, d denotes the number of space-time dimensions used to regularize ultraviolet and infrared divergences of the Feynman integrals. The coefficients of these IBP reductions contain the 9 d-dependent denominators

$$\{5d+52,\ \ 3d-10,\ \ 3d-8,\ \ 2d-7,\ \ d-5,\ \ d-4,\ \ d-3,\ \ d-2,\ \ d\} \quad (35)$$

and the 48 d-independent denominators

$$\{\{2125764t^6+8503056st^5-5904900t^5+12754584s^2t^4-35901792st^4 \\ +2001105t^4+8503056s^3t^3-72275976s^2t^3+25850340st^3+3863700t^3 \\ +2125764s^4t^2-60466176s^3t^2+75149694s^2t^2-19260180st^2-797850t^2 \\ -18187092s^4t+80752788s^3t-73614420s^2t+17309700st-468000t \\ +29452329s^4-50490540s^3+25891650s^2-3676500s+105625,\ \ 26244t^6 \\ +52488st^5-14580t^5+26244s^2t^4-224532st^4-56295t^4-209952s^2t^3 \\ +116640st^3+32400t^3+419904s^2t^2-249480st^2+27900t^2+64800st \\ -18000t+2500,\ \ 524880t^4+524880st^3-583200t^3+1889568s^2t^2 \\ +1953720st^2+243000t^2+3149280s^2t-1206900st-45000t+145800s^2 \\ +173250s+3125,\ \ 104976t^4+314928st^3-536544t^3+209952s^2t^2 \\ -1405512st^2+398520t^2-839808s^2t+599400st-106200t-64800s+9625, \\ 104976t^4+104976st^3+303264t^3-104976s^2t^2+29160st^2-301320t^2 \\ -104976s^3t-244944s^2t+68040st+88200t+29160s^3-32400s^2-23400s \\ -8375,\ \ 13122st^3+37908t^3+26244s^2t^2+64881st^2-56700t^2+13122s^3t \\ +26973s^2t-52650st+22725t+2025s^2+1800s-2750,\ \ 104976s^2t^2 \\ -898128st^2+1454436t^2+209952s^3t-1073088s^2t+64800st-808020t \\ +104976s^4-174960s^3-144180s^2+51300s+112225,\ \ 104976s^2t^2 \\ -898128st^2+1454436t^2-839808s^2t+3841992st-808020t+1679616s^2 \\ -997920s+112225,\ \ 324s^2t^2-2592st^2+5184t^2+648s^3t-2772s^2t \\ +1440st-2880t+324s^4-180s^3-695s^2-200s+400,\ \ 324s^2t^2-2592st^2 \\ +5184t^2-2772s^2t+11808st-2880t+4489s^2-3080s+400,\ \ 81st^2-324t^2 \\ +162s^2t-414st+180t+81s^3-90s^2+25s-25,\ \ 81st^2-324t^2-324st \\ +180t-25,\ \ 3015t^2+3015st-1675t+2916s^2-6030s+3350,\ \ 324t^2 \\ +648st-180t+25,\ \ 324t^2+558st-180t+324s^2-155s+25,\ \ 324t^2 \\ +324st-180t-1296s+25,\ \ 324t^2+324st-180t+25,\ \ 324t^2+90st-180t \\ +90s^2-25s+25,\ \ 324t^2-180t-324s^2+180s+25,\ \ 162t^2+162st-135t \\ +25,\ \ 18t^2+18st-5t+5s,\ \ 558st-90t+558s^2-580s+25,\ \ 558st-90t$$

$$+180s+25,\ \ 324st-90t+324s^2-270s+25,\ \ 324st-90t+25,\ \ 279t$$
$$+279s-200,\ \ 117t+117s-155,\ \ 18t+36s-5,\ \ 18t+18s-5,\ \ 18t+9s$$
$$-5,\ \ 18t-18s-5,\ \ 9t+9s+31,\ \ 9t+9s-5,\ \ 6t+6s-5\},\{324s^2-245s$$
$$+90,\ \ 324s-335,\ \ 18s-5,\ \ 9s+134,\ \ 9s-5,\ \ 9s-10,\ \ s-4,\ \ s\},\{31t+5,$$
$$18t+5,\ \ 18t-5,\ \ 13t+10,\ \ t-4,\ \ t\}\}. \tag{36}$$

During the course of the calculation, different choices for the master integrals were used, and the original choices did in fact introduce additional denominators, which mixed the dependence on d and the kinematic invariants. After changing to an improved set of master integrals, the methods presented in this paper allowed to systematically eliminate those denominators in a term-by-term approach and to very substantially reduce the size of the expressions. In order to reduce the computational complexity, the partial fraction decomposition was performed for expressions expanded around $d = 4$, allowing to effectively replace d by 4 in the denominator analysis. The actual polynomial reduction was performed directly in `Singular` in this case.

The final result for this amplitude was rather sizable. However, the partial fractioned form of the rational coefficients of the master integrals allowed to conveniently generate fast code for their numerical evaluation. Starting from a form with symbols for the inverse denominator factors, also powers of inverse denominator factors and of basic invariants were abbreviated. In this way, each coefficient becomes a sum of simple products of a few factors each, where the basic factors (the powers) can be precomputed numerically. This allows to straight-forwardly generate compiler friendly code, where each term is added to a summation variable numerically.

5.3. Two-loop amplitudes for massless five-particle scattering

For a comparison with the algorithm presented in Ref. [10], we applied our algorithm to a benchmark example considered in that work: the integration-by-parts matrix `IBPmatrix_dlog_basis.txt` provided in Ref. [31] for the reduction of two-loop five-point functions in terms of a dlog basis. The kinematic variables are chosen as

$$c_2 = s_{23}/s_{12}, \quad c_3 = s_{34}/s_{12}, \quad c_4 = s_{45}/s_{12}, \quad c_5 = s_{15}/s_{12},$$

where $s_{12} = 1$ and the s_{ij} are Lorentz invariant functions of the external momenta. Furthermore, ϵ is the parameter of dimensional regularization. In the matrix of rational functions, we encounter a total of 24 denominators[1],

$$\{-1+c_3, c_3, c_2, 1+c_2, c_2+c_3, c_2-c_4, 1+c_2-c_4, -1+c_4, c_4, -1+c_3+c_4,$$
$$c_3+c_4, c_2-c_5, -1+c_3-c_5, c_2+c_3-c_5, c_2-c_4-c_5, 1+c_2-c_4-c_5,$$

[1]Note that the reduced matrix given in Ref. [10] has 32 denominators because several appear twice modulo a sign change.

Option	Runtime
`Iterate->False` (default)	9457 sec
`Iterate->True PartitionSize->`∞	9254 sec
`Iterate->True, PartitionSize->5000`	635 sec
`Iterate->True, PartitionSize->2000`	422 sec
`Iterate->True, PartitionSize->1000`	358 sec
`Iterate->True, PartitionSize->100`	489 sec
`UseFormProgram->True`	60 sec

Table 1: Timing of `MultivariateApart` applied to a rational function for different options.

$$
\begin{aligned}
&-1+c_3+c_4-c_5, c_5, c_2^2-2c_2^2c_3+c_2^2c_3^2+2c_2c_3c_4-2c_2c_3^2c_4+c_3^2c_4^2-2c_2c_5\\
&+2c_2c_3c_5+2c_2c_4c_5+2c_3c_4c_5+2c_2c_3c_4c_5-2c_3c_4^2c_5+c_5^2-2c_4c_5^2+c_4^2c_5^2,\\
&-1+\epsilon,-1+2\epsilon,-1+3\epsilon,-1+4\epsilon,1+4\epsilon\}. \qquad (37)
\end{aligned}
$$

First, we apply `MultivariateApart` to the rational function appearing as the 16^{th} entry of the first row of the matrix and study the impact of different options on the runtime. In this example, the runtime to compute the Gröbner basis is less than a second and therefore negligible with respect to the polynomial reduction. Here and in the following, all timings refer to wall time measurements using a single core of the desktop CPU `Intel(R) Core(TM) i7-6700`. As can be seen in Table 1, different options can alter the performance significantly despite producing the same output form. In the Mathematica-only approach, enabling the iterated reduction scheme described in Section 3.4 can help significantly to reduce the runtime, provided one selects a suitable finite partition size in addition. We don't see a general argument why the partitioning should have the observed effect in all cases; we are tempted to speculate that finite internal buffer sizes in `Mathematica` could be the reason here. Enabling the flag to use `Form` instead of `Mathematica` for the polynomial reductions results in the smallest runtime by quite a margin in this example. We note that the `Form` implementation also uses iterated reduction approach of Section 3.4.

Next, we consider the partial fraction decomposition of all entries of the benchmark matrix `IBPmatrix_dlog_basis.txt` in Ref. [31]. We ran our algorithm in two different setups, once by performing the partial fractioning of each term independently with a Gröbner basis containing only denominators that appear in that term, and once with a global Gröbner basis containing all 24 denominators. Note that these two approaches result in well defined but different output forms. In both cases, we enable the option of our package to reduce the polynomials with `Form`, which was essential to achieve the best timing. In Table 2 we list the resulting timings of these runs and compare features of the resulting output forms to those obtained with the algorithm of [10].

The local Gröbner basis approach is most comparable to the algorithm used in Ref. [10]. With this setup, our algorithm finished the partial fraction decomposition of all elements of the benchmark matrix in roughly 5.5 hours on a single core of our desktop computer. For the approach in which one calculates

Algorithm	Runtime	File size	Max. mon. deg.	Max. term length
Ref. [10]		25.1 MB	20	3564
Global GB	863 min	22.6 MB	12	3109
Local GB	356 min	21.3 MB	12	3048

Table 2: Partial fraction decomposition of the matrix `IBPmatrix_dlog_basis.txt` in Ref. [31]. Results are shown for the algorithm of Ref. [10] and for our algorithm in the local and global reduction approach. Timings were obtained using `Form` for the polynomial reductions.

one global Gröbner basis for all 24 denominators at once, our algorithm finds an optimized monomial ordering with which the Gröbner basis can be calculated within one minute in `Mathematica`. In this approach the reduction of all matrix elements took roughly 14.4 hours on our desktop computer, which is longer than in the local Gröbner basis approach. We did not attempt to make a serious comparison of performance to the implementation of [10], but the timing for this example seems to be of a similar order of magnitude compared to the runs of our package when using `Form` for the polynomial reductions (Ref. [10] reported 27.9 hours of runtime).

The resulting output forms are of similar overall size in all cases, see Table 2. Considering the rational functions as polynomials in $\mathbb{Q}[q_1, \ldots, q_{24}, c_2, \ldots, c_5, \epsilon]$, we determine the total degrees of the output forms in the different approaches. In all cases, we consider fully expanded output forms and write them to disk with Mathematica. As is shown in Table 2, our method leads to lower degrees than the decomposition of Ref. [10], as expected due to our choice of monomial ordering, see Section 3.2. For the comparison, it is noteworthy that our package does not treat non-linear denominators in a special way, they are fully taken into account in the decomposition. Using a global Gröbner basis for the reduction yields a slightly larger result but has the big advantage that all terms are not only in a unique representation locally, but also globally. Therefore sums of different coefficients which would appear in the calculation of the amplitude can be computed without any problems term by term, e.g. using `Form`.

6. Conclusion and outlook

In this paper we introduced a method for the partial fraction decomposition of multivariate rational functions through polynomial reductions. The method does not introduce spurious singularities and allows for a straightforward implementation in `Form` using local replacement rules. We implemented our algorithms in a `Mathematica` package that is publicly available at `https://gitlab.msu.edu/vmante/multivariateapart`.

Our approach involves the construction of a special monomial ordering, which is optimized for computational performance and leads to results with comparably low degrees. We showed for different "real life" examples how the algorithm helped to calculate Feynman amplitudes, in the presence of many kinematic variables, many denominators, or high polynomial degrees of the denominators. In several of our examples, the calculation of the Gröbner basis is

highly non-trivial and became possible through our method to define the monomial ordering. The partial fractioned sum of terms lends itself to a straightforward and compiler friendly implementation in numerical codes.

We discussed options for the reconstruction of rational functions from prime field samples. Here, we proposed a new idea to reconstruct the denominator of a rational function based on knowledge about possible denominator factors and a small number of numerical samples. This method has been successfully tested in the context of linear relations between Feynman integrals and reduced the number of samples required for the reconstruction of the full rational functions significantly. Our partial fraction decomposition provides a systematic approach to the direct reconstruction rational function in this representation. Future research might provide an optimized strategy to further reduce the number of samples required to reconstruct complicated rational functions in this approach.

A possible future application of our methods to particle physics is the direct parametric integration of Feynman integrals [32, 33]. In this context it is important to recognize potential cancellations between different terms and to avoid the introduction of spurious (non-linear) denominators, since they can prevent a successful integration. The algorithms discussed in this paper provide a systematic approach to this problem. Further, particle physics calculations based on loop-tree duality [34, 35] or intersection theory [36] can benefit from our algorithms. Since our methods apply to general rational functions, they may be useful also for other areas of research unrelated to particle physics.

Acknowledgements

We would like to thank Marco Besier, Bakul Agarwal and Federico Buccioni for helpful discussions and valuable feedback on applications of our package. MH was supported in part by the German Research Foundation (DFG), through the Collaborative Research Center, Project ID 204404729, SFB 1044, and the Cluster of Excellence PRISMA$^+$, Project ID 39083149, EXC 2118/1. AvM was supported in part by the National Science Foundation under Grants No. 1719863 and 2013859.

A. Polynomial reductions

In this Appendix, we review some basic notions that can be found in any introductory textbook on the topic, see e.g. [13, 14].

Let us consider polynomials in the variables $x_1, \ldots, x_n$ with coefficients in the field K (e.g. the rational numbers). These polynomials form a *polynomial ring* $R = \mathbf{K}[x_1, \ldots, x_n]$.

A *monomial ordering* is a total order $\prec$ on the set of monomials

$$M \equiv \{x^\alpha \equiv x_1^{\alpha_1} \cdots x_n^{\alpha_n} | \alpha_i \in \mathbf{Z}_{\geq 0}\} \tag{A.1}$$

such that

(i) for all monomials x^α, x^β and x^γ: $x^\alpha \leq x^\beta \Rightarrow x^{\alpha+\gamma} \leq x^{\beta+\gamma}$,

(ii) $1 \leq x^\alpha \quad \forall \alpha$.

In many applications, in which the specific monomial ordering does not matter, one chooses the so-called *degree reverse lexicographic order* (degrevlex), which first compares the total degree of two monomials, and then, if the degrees are the same, uses a lexicographic comparison and reverses the outcome of the latter.

There is a more general way of defining a monomial ordering, which will be relevant to us. A monomial ordering is called a *block ordering*, if one can group the variables into two sets $\{x_1, \ldots, x_m\}$ and $\{y_1, \ldots, y_n\}$, such that

$$x^\alpha \prec y^\beta \quad \forall\, \alpha, \beta \text{ with } y^\beta \neq 1 \tag{A.2}$$

i.e. all monomials formed from variables from the first block are "smaller" than monomials formed from variables from the second block.

We call the "greatest" monomial of a polynomial p with respect to a monomial order the *leading monomial* of p. A term is a monomial multiplied with a coefficient, which is a non-zero rational number for our purposes. The *leading term* $\mathrm{LT}(p)$ of polynomial p is the term corresponding to the leading monomial.

Having defined a monomial ordering puts us in a position to discuss polynomial reductions. Given two polynomials p_1 and p_2 together with a monomial ordering $\prec$, we call p_1 reducible modulo p_2 if for some term t of p_1 there is a term u such that $t = u \cdot \mathrm{LT}(p_2)$. Then we say, p_1 *reduces to* p_1' *modulo* p_2, where

$$p_1' = p_1 - u \cdot p_2 \tag{A.3}$$

The effect of the polynomial reduction is that a term in p_1 is replaced by a linear combination of "smaller" terms.

It is useful to introduce another concept. A set of polynomials $g_1, \ldots, g_m \in R$, the *generators*, define an *ideal* $I \subset R$ as the set of all their linear combinations, where the coefficients are polynomials themselves, $I = \{\sum_i f_i g_i \text{ with } f_i \in R\}$. We also use the notation $I = \langle g_1, \ldots, g_m \rangle$. The choice of generators is not unique, and it is a non-trivial task to decide whether a polynomial is a member of a given ideal. It is clear that, if a polynomial reduces to zero modulo the generators of an ideal that the polynomial is a member of that ideal. The inverse, however, is not true and the remainder of a polynomial reduction will in general depend on the individual reduction steps.

It is possible to find a set of generators for an ideal, called a *Gröbner basis*, which allows one to find a unique remainder for the reduction of any polynomial modulo the generators. In this case, a reduction to zero occurs if and only if the polynomial is in the ideal. A Gröbner basis can be calculated algorithmically by adjoining specific differences of generators. The Gröbner basis depends on the choice of the monomial ordering and is unique if reduced with respect to itself.

B. Leĭnartas' requirements and polynomial reductions

In this Section we revisit the two central decomposition steps in Algorithm 1 leading to Leĭnartas' decomposition, see Section 2.2 and consider under which circumstances they are reproduced by the polynomial reductions in Algorithm 2, see Section 3. As we will show, Algorithm 2 may or may not produce a Leĭnartas' decomposition, depending on the monomial ordering. If the monomial ordering sorts first for the q_i and then for the x_i, requirement (i) will be filfilled, that is, the denominator zeros will be separated. If the q_i are sorted lexicographically, also requirement (ii), the algebraic independence of different denominator factors, will be guaranteed [12]. For degree based orderings of the q_i, requirement (ii) is violated in general. This means in particular, that our monomial block ordering proposed in Section 3.2 guarantees only requirement (i) and not (ii), and will therefore not lead to a Leĭnartas decomposition. We emphasize, that an even more general choice of monomial ordering will ensure neither of the two requirements, but still allows for a unique output form.

First, we consider the requirement (i), i.e. that the denominators of each term shall have common zeros in $\overline{K}^n$. Let us assume we encounter a term which is not fully decomposed yet and has denominators $\{d_1(x_1,\ldots),\ldots,d_m(x_1,\ldots)\}$. By Hilbert's Nullstellensatz, a finite set of polynomials $\{d_1^{\alpha_1},\ldots,d_m^{\alpha_m}\}$ has no common zeros in $\overline{K}^n$ if and only if there exist polynomials $h_1,\ldots,h_m$, such that

$$1 = \sum_i h_i(x_1,\ldots)\, d_i^{\alpha_i}(x_1,\ldots). \tag{B.1}$$

Dividing this equation by $d_1^{\alpha_1}\cdots d_m^{\alpha_m}$ gives the decomposition [7]

$$\frac{1}{d_1^{\alpha_1}\cdots d_m^{\alpha_m}} = \sum_i \frac{h_i(x_1,\ldots)}{d_1^{\alpha_1}\cdots \hat{d}_i^{\,\alpha_i}\cdots d_m^{\alpha_m}}. \tag{B.2}$$

where $\hat{d}_i^{\,\alpha_i}$ means leaving $d_i^{\alpha_i}$ out in the product.

This decomposition step in Algorithm 1 has a direct analogue in terms of a polynomial reduction in step 5 of Algorithm 2 for a suitable monomial ordering. Multiplying Eq. (B.1) by $q_1^{\alpha_1}\cdots q_m^{\alpha_1}$ and replacing $q_i d_i(x_1,\ldots) = 1$ gives

$$q_1^{\alpha_1}\cdots q_m^{\alpha_m} - \sum_i h_i(x_1,\ldots)\, q_1^{\alpha_1}\cdots \hat{q}_i^{\,\alpha_i}\cdots q_m^{\alpha_m} = 0, \tag{B.3}$$

where $\hat{q}_i^{\,\alpha_i}$ means leaving $q_i^{\alpha_i}$ out in the product. It is clear that the polynomial on the lhs of Eq. (B.3) is a member of the ideal $I = \langle 1 - q_1 d_1(x_1,\ldots),\ldots\rangle$. Furthermore, $q_1^{\alpha_1}\cdots q_m^{\alpha_m}$ is the leading monomial for any monomial ordering which sorts first for all of the $q_1,\ldots,q_m$ and then for the $x_1,\ldots,x_n$. In Algorithm 2, that leading monomial will be thus be replaced by the subleading terms on the lhs of Eq. (B.3), resulting in terms with fewer factors of q_i, that is, simpler denominators. A fully reduced term will therefore not have all factors $q_1\cdots q_n$. This proves the above statements. We conclude that, for our choice of monomial ordering, the polynomial reduction in Algorithm 2 guarantees requirement (i)

of Leĭnartas' decomposition, that is, the separation of independent denominator zeros.

Second, we consider requirement (ii), that is, the algebraic independence of the denominator factors. Let us assume that our decomposition contains a term with denominator $d_1^{\alpha_1} \cdots d_m^{\alpha_m}$, where the factors $\{d_1, \ldots, d_m\}$ are algebraically dependent. One can show that then also the polynomials $\{d_1^{\alpha_1}, \ldots, d_m^{\alpha_m}\}$ are algebraically dependent and there exists a non-zero annihilating polynomial $p(y_1, \ldots, y_m)$ with

$$p(d_1^{\alpha_1}, \ldots, d_m^{\alpha_m}) = 0 \tag{B.4}$$

after inserting the explicit expressions the $d_i(x_1, \ldots)$. In Algorithm 1, one can for example choose one of the monomials of p with smallest degree and substitute the remainder, i.e. one has

$$c_\beta (d^\alpha)^\beta = -\sum_{\gamma \in S} c_\gamma (d^\alpha)^\gamma, \tag{B.5}$$

where $d = (d_1, \ldots, d_m)$, α, β and γ are multi-indices in $\mathbb{N}^m$, with $\sum_i \beta_i \leq \sum_i \gamma_i$ for all β in the set of multi-indices S. Dividing by $c_\beta d^{\beta+1}$ gives the decomposition step [7]

$$\frac{1}{d_1^{\alpha_1} \cdots d_m^{\alpha_m}} = -\sum_{\gamma \in S} \frac{c_\gamma}{c_\beta} \prod_{i=1}^{m} \frac{d_i^{\alpha_i \gamma_i}}{d_i^{\alpha_i(\beta_i+1)}}. \tag{B.6}$$

Since $\sum_i \beta_i \leq \sum_i \gamma_i$ and $\beta \neq \gamma$, for each γ there exists an i such that $\beta_i + 1 \leq \gamma_i$ and the factor $d_i^{\beta_i+1}$ is removed from the denominator. Therefore, each term on the rhs of (B.6) depends on at least one denominator factor d_i less than the lhs.

The described decomposition step in Algorithm 1 may or may not have an analogue in terms of a polynomial reduction in step 5 of Algorithm 2, depending on the monomial ordering. In general, for a degree based ordering of the q_i variables, either globally or in blocks like proposed in this paper, a similar decomposition will *not* occur. In (B.6), the denominator degree may actually be higher on the rhs than on the lhs, and it may therefore not correspond to a polynomial reduction for such an ordering. A lexicographic ordering, on the other hand, *will* separate algebraically dependent denominators. To show this, we pick in Eq. (B.4) the unique exponent β' of p such that $(q^\alpha)^{\beta'}$ is minimal with respect to our monomial ordering. Multiplication with $(q^\alpha)^{\beta'+1}/(c_{\beta'})$ gives

$$q_1^{\alpha_1} \cdots q_m^{\alpha_m} + \sum_{\gamma \in S} \frac{c_\gamma}{c_{\beta'}} \prod_{i=1}^{m} d_i^{\alpha_i \gamma_i} q_i^{\alpha_i(\beta_i'+1)} = 0. \tag{B.7}$$

Replacing all products $q_i d_i(x_1, \ldots) = 1$ gives

$$q_1^{\alpha_1} \cdots q_m^{\alpha_m} + \sum_{\gamma \in S} \frac{c_\gamma}{c_{\beta'}} \prod_{i=1}^{m} d_i^{\max(\alpha_i(\gamma_i - \beta_i' - 1), 0)} q_i^{\max(\alpha_i(\beta_i' + 1 - \gamma_i), 0)} = 0. \tag{B.8}$$

Assuming a lexicographic ordering of the q_i with $q_1 \succ q_2 \succ \ldots$ implies that for each term γ there exists a $j = 1, \ldots, m$ such that $\beta'_i = \gamma_i$ for all $i = 1, \ldots, j$, $\beta'_j \prec \gamma_j$, the powers of q_i coincide with that of the first term for $i = 1, \ldots, j-1$, but q_j is removed. This means that the first term in (B.8) is indeed the leading term. Since furthermore the polynomial on the lhs of (B.8) is a member of the ideal $I = \langle q_1 d_1(x_1, \ldots) - 1, \ldots \rangle$, the first term in (B.8) would be reduced in step 5 of Algorithm 2 for this ordering. We conclude that, for the block monomial ordering presented in this article, the polynomial reduction in Algorithm 2 does not guarantee requirement (ii) of Leĭnartas' decomposition, that is, algebraically independent denominators. In contrast, a lexicographical ordering guarantees this requirement.

We would like to illustrate the impact of the monomial ordering on the decomposition with the following example. We consider the irreducible denominator factors

$$d_1 = x^3 + y^4, \quad d_2 = x + y^2, \quad d_3 = x^2 + y, \tag{B.9}$$

which share a common zero. We note that three denominator factors in two variables must be algebraically dependent on general grounds. Let us consider the ideal

$$I = \langle q_1 d_1(x, y) - 1, q_2 d_2(x, y) - 1, q_3 d_3(x, y) - 1 \rangle . \tag{B.10}$$

The monomial block ordering proposed in this article gives

$$\{\{q_1, q_2, q_3\}, \{x, y\}\}. \tag{B.11}$$

Calculating the Gröbner basis of I with respect to this ordering, we see that the polynomial representation of $1/(d_1 d_2 d_3)$,

$$q_1 q_2 q_3, \tag{B.12}$$

is fully reduced already, despite the denominators d_1, d_2 and d_3 being algebraically dependent.

Next, let us consider a lexicographic ordering of the q_i according to

$$\{\{q_1\}, \{q_2\}, \{q_3\}, \{x, y\}\}. \tag{B.13}$$

In this case, the Gröbner basis computation of I reveals that $q_1 q_2 q_3$ is reducible. Alternatively, we can derive a reduction identity of type (B.8) as described above. Indeed, it is not difficult to calculate the annihilator

$$\begin{aligned} p(y_1, y_2, y_3) = {} & y_1^4 - 4y_1^3 y_2^2 - 4y_1^3 y_3 + 3y_1^3 + 6y_1^2 y_2^4 + 4y_1^2 y_2^2 y_3 - y_1^2 y_2^2 + 8y_1^2 y_2 y_3^2 \\ & - 6y_1^2 y_2 y_3 - 8y_1^2 y_2 - 2y_1^2 y_3^3 + 6y_1^2 y_3^2 - 2y_1^2 y_3 + 3y_1^2 - 4y_1 y_2^6 + 4y_1 y_2^4 y_3 \\ & - 7y_1 y_2^4 - 16y_1 y_2^3 y_3^2 + 12y_1 y_2^3 y_3 - 8y_1 y_2^3 + 4y_1 y_2^2 y_3^3 + 4y_1 y_2^2 y_3^2 + 36y_1 y_2^2 y_3 \\ & - 16y_1 y_2 y_3^3 - 18y_1 y_2 y_3^2 + 2y_1 y_2 y_3 - 5y_1 y_2 + 4y_1 y_3^4 + 2y_1 y_3^3 - y_1 y_3^2 + 2y_1 y_3 \\ & + y_2^8 - 4y_2^6 y_3 + 5y_2^6 + 8y_2^5 y_3^2 - 6y_2^5 y_3 - 2y_2^4 y_3^3 + 6y_2^4 y_3^2 - 2y_2^4 y_3 + 5y_2^4 \\ & - 16y_2^3 y_3^3 - 14y_2^3 y_3^2 - 2y_2^3 y_3 + 20y_2^2 y_3^4 + 6y_2^2 y_3^3 - 8y_2 y_3^5 + 5y_2 y_3^4 + y_3^6 - 2y_3^5 \end{aligned} \tag{B.14}$$

with $p(d_1(x,y), d_2(x,y), d_3(x,y)) = 0$ and identify $q_i d_i = 1$ modulo I to obtain

$$
\begin{aligned}
&q_1q_2q_3 - \tfrac{1}{2}q_1q_2 - \tfrac{1}{2}(d_2^7 + 5d_2^5 + 5d_2^3)q_1q_3^6 + (2d_2^5 + 3d_2^4 + d_2^3 + d_2^2)q_1q_3^5 \\
&- (4d_2^4 + 3d_2^3 - 7d_2^2)q_1q_3^4 + (d_2^3 + 8d_2^2 - 3d_2)q_1q_3^3 - \tfrac{1}{2}(20d_2 + 5)q_1q_3^2 + 4q_1q_3 \\
&- \tfrac{1}{2}(d_1^3 + 3d_1^2 + 3d_1)q_2q_3^6 + (2d_1^2 + d_1 - 1)q_2q_3^5 - \tfrac{1}{2}(6d_1 - 1)q_2q_3^4 + (d_1 - 1)q_2q_3^3 \\
&- 2q_2q_3^2 + \tfrac{1}{2}(4d_1^2d_2 - 6d_1d_2^3 + d_1d_2 + 8d_1 + 4d_2^5 + 7d_2^3 + 8d_2^2 + 5)q_3^6 \\
&- (2d_1d_2 - 3d_1 + 2d_2^3 + 6d_2^2 + 18d_2 + 1)q_3^5 - (4d_1 - 8d_2^2 + 2d_2 - 9)q_3^4 \\
&- (2d_2 - 8)q_3^3 = 0,
\end{aligned}
\tag{B.15}
$$

where d_1, d_2, d_3 are meant to be replaced by their definitions (B.9). As we see, this gives a reduction identity for (B.12) since the first term is indeed the leading term in this ordering, such that the algebraically dependent denominators are decomposed. However, we also see that this decomposition leads to a significant increase in the degrees of the polynomials.

References

[1] H. Strubbe, *Manual for Schoonschip: A CDC 6000 / 7000 program for symbolic evaluation of algebraic expressions (program made by M. Veltman)*, *Comput. Phys. Commun.* **8** (1974) 1.

[2] E. Remiddi, "*Computer algebra, past, present and future.*" Talk given at Tini 80 Fest, 2011, `https://www.nikhef.nl/~t45/Tini80Fest/Remiddi.pdf`.

[3] J. A. M. Vermaseren, *New features of FORM*, `math-ph/0010025`.

[4] B. Ruijl, T. Ueda and J. Vermaseren, *FORM version 4.2*, `1707.06453`.

[5] F. Feng, `Apart`*: A Generalized Mathematica Apart Function*, *Comput. Phys. Commun.* **183** (2012) 2158 [`1204.2314`].

[6] E. K. Leĭnartas, *Factorization of rational functions of several variables into partial fractions*, *Soviet Math. (Iz. VUZ)* **22** (1978) 35.

[7] A. Raichev, *Leĭnartas' partial fraction decomposition*, `1206.4740`.

[8] C. Meyer, *Transforming differential equations of multi-loop Feynman integrals into canonical form*, *JHEP* **04** (2017) 006 [`1611.01087`].

[9] C. Meyer, *Algorithmic transformation of multi-loop master integrals to a canonical basis with CANONICA*, *Comput. Phys. Commun.* **222** (2018) 295 [`1705.06252`].

[10] J. Boehm, M. Wittmann, Z. Wu, Y. Xu and Y. Zhang, *IBP reduction coefficients made simple*, *JHEP* **12** (2020) 054 [`2008.13194`].

[11] A. Pak, *The Toolbox of modern multi-loop calculations: novel analytic and semi-analytic techniques*, *J. Phys. Conf. Ser.* **368** (2012) 012049 [1111.0868].

[12] S. Abreu, J. Dormans, F. Febres Cordero, H. Ita, B. Page and V. Sotnikov, *Analytic Form of the Planar Two-Loop Five-Parton Scattering Amplitudes in QCD*, *JHEP* **05** (2019) 084 [1904.00945].

[13] D. A. Cox, J. Little and D. O'Shea, *Ideals, Varieties, and Algorithms*. Springer, Cham, 2015, 10.1007/978-3-319-16721-3.

[14] M. Kreuzer and L. Robbiano, *Computational commutative algebra. 1*. Springer, Berlin, Heidelberg, 2000, 10.1007/978-3-540-70628-1.

[15] A. von Manteuffel and R. M. Schabinger, *A novel approach to integration by parts reduction*, *Phys. Lett.* **B744** (2015) 101 [1406.4513].

[16] A. von Manteuffel and R. M. Schabinger, *Quark and gluon form factors to four-loop order in QCD: the N_f^3 contributions*, *Phys. Rev. D* **95** (2017) 034030 [1611.00795].

[17] T. Peraro, *Scattering amplitudes over finite fields and multivariate functional reconstruction*, *JHEP* **12** (2016) 030 [1608.01902].

[18] T. Peraro, *FiniteFlow: multivariate functional reconstruction using finite fields and dataflow graphs*, *JHEP* **07** (2019) 031 [1905.08019].

[19] J. Klappert and F. Lange, *Reconstructing rational functions with FireFly*, *Comput. Phys. Commun.* **247** (2020) 106951 [1904.00009].

[20] A. V. Smirnov and F. S. Chuharev, *FIRE6: Feynman Integral REduction with Modular Arithmetic*, *Comput. Phys. Commun.* **247** (2020) 106877 [1901.07808].

[21] F. Tkachov, *A Theorem on Analytical Calculability of Four Loop Renormalization Group Functions*, *Phys.Lett.* **B100** (1981) 65.

[22] K. Chetyrkin and F. Tkachov, *Integration by Parts: The Algorithm to Calculate beta Functions in 4 Loops*, *Nucl.Phys.* **B192** (1981) 159.

[23] A. Smirnov and V. Smirnov, *How to choose master integrals*, *Nucl. Phys. B* **960** (2020) 115213 [2002.08042].

[24] J. Usovitsch, *Factorization of denominators in integration-by-parts reductions*, 2002.08173.

[25] S. Abreu, J. Dormans, F. Febres Cordero, H. Ita and B. Page, *Analytic Form of Planar Two-Loop Five-Gluon Scattering Amplitudes in QCD*, *Phys. Rev. Lett.* **122** (2019) 082002 [1812.04586].

[26] G. P. W. Decker, G.-M. Greuel and H. Schönemann, *Singular 4-1-3 — A computer algebra system for polynomial computations*, `https://www.singular.uni-kl.de/`, 2020.

[27] M. Heller, O. Tomalak, M. Vanderhaeghen and S. Wu, *Leading Order Corrections to the Bethe-Heitler Process in the $\gamma p \to l^+ l^- p$ Reaction*, *Phys. Rev. D* **100** (2019) 076013 [1906.02706].

[28] M. Heller, N. Keil and M. Vanderhaeghen, *Leading-order QED radiative corrections to timelike Compton scattering on the proton*, 2012.09565.

[29] R. Tarrach, *Invariant Amplitudes for Virtual Compton Scattering Off Polarized Nucleons Free from Kinematical Singularities, Zeros and Constraints*, *Nuovo Cim. A* **28** (1975) 409.

[30] B. Agarwal, S. P. Jones and A. von Manteuffel, *Two-loop helicity amplitudes for $gg \to ZZ$ with full top-quark mass effects*, 2011.15113.

[31] D. Bendle, J. Böhm, W. Decker, A. Georgoudis, F.-J. Pfreundt, M. Rahn et al., *Integration-by-parts reductions of Feynman integrals using Singular and GPI-Space*, *JHEP* **02** (2020) 079 [1908.04301].

[32] F. Brown, *The Massless higher-loop two-point function*, *Commun. Math. Phys.* **287** (2009) 925 [0804.1660].

[33] E. Panzer, *Algorithms for the symbolic integration of hyperlogarithms with applications to Feynman integrals*, *Comput. Phys. Commun.* **188** (2015) 148 [1403.3385].

[34] S. Catani, T. Gleisberg, F. Krauss, G. Rodrigo and J.-C. Winter, *From loops to trees by-passing Feynman's theorem*, *JHEP* **09** (2008) 065 [0804.3170].

[35] W. J. Torres Bobadilla, *Loop-tree duality from vertices and edges*, *JHEP* **04** (2021) 183 [2102.05048].

[36] P. Mastrolia and S. Mizera, *Feynman Integrals and Intersection Theory*, *JHEP* **02** (2019) 139 [1810.03818].