

Scattering amplitudes over finite fields and multivariate functional reconstruction

Tiziano Peraro

*Higgs Centre for Theoretical Physics,
School of Physics and Astronomy, The University of Edinburgh,
James Clerk Maxwell Building, Peter Guthrie Tait Road, Edinburgh EH9 3FD, U.K.*

E-mail: tiziano.peraro@ed.ac.uk

ABSTRACT: Several problems in computer algebra can be efficiently solved by reducing them to calculations over finite fields. In this paper, we describe an algorithm for the reconstruction of multivariate polynomials and rational functions from their evaluation over finite fields. Calculations over finite fields can in turn be efficiently performed using machine-size integers in statically-typed languages. We then discuss the application of the algorithm to several techniques related to the computation of scattering amplitudes, such as the four- and six-dimensional spinor-helicity formalism, tree-level recursion relations, and multi-loop integrand reduction via generalized unitarity. The method has good efficiency and scales well with the number of variables and the complexity of the problem. As an example combining these techniques, we present the calculation of full analytic expressions for the two-loop five-point on-shell integrands of the maximal cuts of the planar penta-box and the non-planar double-pentagon topologies in Yang-Mills theory, for a complete set of independent helicity configurations.

KEYWORDS: Scattering Amplitudes, Perturbative QCD

ARXIV EPRINT: [1608.01902](https://arxiv.org/abs/1608.01902)

Contents

1	Introduction	2
2	Basic concepts, definitions and notation	5
2.1	Finite fields	5
2.2	Polynomials and rational functions	6
2.3	Rational reconstruction from finite fields	7
3	Functional reconstruction	8
3.1	The black-box interpolation problem	9
3.2	Univariate polynomials	10
3.3	Univariate rational functions	12
3.4	Multivariate polynomials	14
3.5	Multivariate rational functions	14
3.6	Examples	17
4	Spinor-helicity and tree-level techniques	20
4.1	Four-dimensional momenta and spinors	20
4.2	Berends-Giele recursion	23
5	Multi-loop integrand reduction and generalized unitarity	25
5.1	Two-loop five-point planar penta-box	28
5.2	Two-loop five-point non-planar double-pentagon	30
6	Implementation	32
7	Conclusions and outlook	35
A	Basic finite-field algorithms	36
A.1	Multiplicative inverse and rational reconstruction	36
A.2	Chinese remainder theorem	38
B	Six-dimensional momenta and spinors	39
C	Two-loop unitarity cuts from Berends-Giele currents	40

1 Introduction

Scattering amplitudes in Quantum Field Theory (QFT) are an essential ingredient for understanding the interactions among fundamental particles that we observe in nature. The development of techniques and algorithms for their calculation is therefore of crucial importance for comparing observations with theoretical predictions. In particular, the high accuracy expected from experimental data which is being collected at the Large Hadron Collider (LHC), as well as the high centre-of-mass energy of the interactions produced by this machine, require accurate predictions for processes with high-multiplicity final states. This has motivated, in recent years, many studies of the structure of scattering amplitudes in QFT, especially in gauge theories, which often led to the development of new efficient methods for their evaluation.

Despite the remarkable recent progress in the calculation of high-multiplicity tree-level and one-loop amplitudes, and the numerical automation of the latter in several codes and frameworks [1–19], at two and higher loops these calculations are still essentially restricted to $2 \rightarrow 2$ processes. This is due to the significant increase in complexity of the two-loop problem with respect to the one-loop case. More in detail, the most common strategy for the calculation of loop amplitudes is to rewrite them as a linear combination of loop integrals. While the coefficients of this linear combination are rational functions of kinematic invariants, the loop integrals can instead be computed in terms of special functions. While for low-multiplicity processes the most demanding task is arguably the calculation of the integrals, for high-multiplicity processes the computation of the coefficients can often have comparable or higher complexity (notice e.g. that a complete planar basis of massless five-point two-loop integrals is known [20, 21], but five-point QCD amplitudes are still unavailable for a generic helicity configuration). While one-loop amplitudes are often computed numerically, two-loop amplitudes are more often computed analytically.

There are several reasons to prefer an analytic approach to a numerical one. Analytic expressions often yield a faster and more stable numerical evaluation than purely numerical algorithms. Moreover, analytic results allow to perform various kinds of studies and manipulations, such as the analysis of the behaviour of amplitudes in interesting kinematic limits (e.g. infrared and high-energy limits). Analytic calculations also allow to have better control over the results and possibly infer general properties which might be useful for the development of new analytic or numerical algorithms.

A well known bottleneck of analytic calculations in high-energy physics is however the large size of intermediate expressions, which can often be orders of magnitude larger than the final results. This is to be expected, since physical results often enjoy properties which are not shared by each intermediate step of the calculation. Moreover, intermediate steps are often described by a larger number of variables (such as the loop components) which do not appear in the final result. The problem can be mitigated by the use of computer algebra systems such as FORM [22], which specializes in handling large expressions, or by using techniques such as generalized unitarity [23, 24], where intermediate steps of the calculation are gauge invariant and hence the complexity of their expressions is reduced with respect to a diagrammatic approach.

In this paper we assess the possibility of side-stepping the issue of large intermediate expressions, by reconstructing analytic results from their numerical evaluation, where each intermediate step is trivially a number or a set of numbers (this will be better defined in the next paragraph). A polynomial or a rational function can be reconstructed, with very high probability, from its numerical evaluation at several values of its arguments. In particular we focus on the functional reconstruction of multivariate polynomials and rational functions with applications to calculations in high-energy physics.

The first question to address is which kind of numerical evaluation is suited for a functional reconstruction. An obvious choice would be a floating-point calculation, but these are affected by numerical inaccuracies which would add an additional layer of complexity to the functional reconstruction algorithms. Exact calculations might instead be performed over the field of rational numbers. However, numerical calculations with rational numbers are affected by a similar problem to the one of analytic calculations. Indeed, while a large intermediate expression would be translated into a rational number, in general the number of digits of the numerator and the denominator of this number would be very high. This requires extensive use of computationally expensive arbitrary-precision arithmetic, which can significantly slow down the calculation. A common and successful approach in computer algebra is the use of finite fields, which have a finite number of elements and can be represented by machine-size integers, offering the possibility of performing fast but exact calculations in statically-typed languages such as C and C++. Their main drawback is that, switching from the rational field to a finite field, some information is lost and must be recovered by repeating a functional reconstruction over several finite fields. This strategy is however much more efficient than a calculation over the rational field.

The usage of finite fields in computer algebra is actually quite common. Many computer algebra systems use finite fields under the hood for solving problems such as polynomial factorization and Greatest Common Divisor (GCD). The application of finite fields in high-energy physics has been introduced in ref. [25], in the context of Integration-By-Parts (IBPs). However, to the best of our knowledge, an application of finite-field techniques to a realistic problem in high-energy physics involving the reconstruction of multivariate rational functions is not present in the literature and is presented here for the first time. In particular, the main missing ingredient which we illustrate in this paper is the application of a functional reconstruction algorithm capable of handling relatively complex results which depend on many variables, such as those appearing in typical high-multiplicity multi-loop calculations.

The paper is roughly divided in two parts. In the first part we describe dense¹ functional reconstruction algorithms for univariate and multivariate polynomials and rational functions. These algorithms reconstruct polynomials and rational functions from their repeated numerical evaluation over finite fields (although in principle they can actually be used over any field) and they are independent of the specific algorithm used for their evalu-

¹A *dense* reconstruction algorithm for polynomials and rational functions, unlike a *sparse* reconstruction algorithm, seeks to be efficient in the general case where the result has many non-vanishing terms, rather than in the special case where it only has a small number of non-vanishing terms compared with its total degree.

ation. In particular, for univariate polynomials we use the well known Newton’s polynomial representation. For univariate rational functions, we use Thiele’s interpolation formula.² For multivariate polynomials, we use a recursive version of Newton’s formula. For multivariate rational functions we were not able to find a dense reconstruction algorithm suited for our needs in the literature. However we found that the technique proposed in ref. [26] for sparse rational functions can be easily adapted to the dense case, by combining it with the other techniques we mentioned for univariate rational functions and multivariate polynomials. The resulting algorithm is capable of efficiently reconstructing functions with many non-vanishing terms and depending on several variables, as we will show in the examples. Using these methods, the analytic calculation of any polynomial or rational function can be turned into the problem of providing an efficient numerical evaluation of the same function over finite fields.

The second part of the paper concerns the application of the mentioned reconstruction algorithms to techniques relevant for the calculation of scattering amplitudes. It should be stressed that any algorithm which can be implemented via a sequence a rational elementary operations (addition, subtraction, multiplication and division) is suited for the usage of these reconstruction techniques, which therefore have a very broad spectrum of applications. In particular, widely used methods such as tensor reduction and IBPs obviously fall into this category. In this paper we however focus on multi-loop integrand reduction via generalized unitarity [23, 24, 27–46], since the algorithm is suited for high-multiplicity processes and, as stated, writing scattering amplitudes as linear combinations of integrals (which may be further processed by IBPs at a later stage) is currently one of the main bottlenecks of high-multiplicity multi-loop calculations. These techniques have indeed been used in recent five- and six-point calculations of two-loop amplitudes in non-supersymmetric Yang-Mills theory [37, 47, 48]. In order to provide the building blocks needed by generalized unitarity, we also discuss in some detail a finite-field implementation of the spinor-helicity formalism in four [49, 50] and six dimensions [51–53], as well as the calculation of tree-level amplitudes over finite fields via Berends-Giele recursion [50]. In particular, the six-dimensional spinor-helicity formalism is used to provide a higher-dimensional embedding of loop momenta and spinors, which will thus have an explicit finite-field numerical representation.

As explicit examples combining all these techniques, we present the calculation of full analytic expressions for the two-loop five-point on-shell integrands of the maximal cuts of the planar penta-box and the non-planar double-pentagon topologies in Yang-Mills theory, for a complete set of independent helicity configurations. In particular, in the all-plus case we find agreement with the results available in the literature, while for the other helicity configurations the result is new and we make it publicly available for comparisons with future calculations.

All the algorithms discussed in the paper have been implemented in a C++ library which can produce the mentioned analytic results from finite-field evaluations using 64-bit integers, without relying on any external computer algebra system.

²The formula is named after the mathematician Thorvald Nicolai Thiele (1838–1910).

The paper is structured as follows. In section 2 we set the notation and review some notions about finite fields which we will use in the rest of the paper. While all these notions are well known, they are rarely used in high-energy physics and hence they are reviewed in some detail for the convenience of the reader. In section 3 we describe the functional reconstruction algorithms mentioned above. In section 4 we discuss the implementation of spinor-helicity and tree-level techniques over finite fields. These are mostly meant as a stepping stone for the discussion of integrand reduction and generalized unitarity, which is illustrated in section 5, where we also provide the two-loop five-point examples mentioned above. In section 6 we give some details about our proof-of-concept C++ implementation of the algorithms illustrated in this paper, which might be useful for other implementations. In section 7 we finally draw our conclusions and briefly discuss further possible applications of these techniques. In appendix A we recall some well known theorems and algorithms involving modular arithmetic and finite fields, highlighting the role they play in the reconstruction algorithms illustrated in this paper. More details about our usage of the six-dimensional spinor-helicity formalism over finite fields are given in appendix B. In appendix C we discuss an efficient method for generating two-loop unitarity cuts from Berends-Giele currents, which is a generalization of the one-loop algorithm used by the public code NJET [18], and whose two-loop extension is not present elsewhere in the literature.

2 Basic concepts, definitions and notation

In this section we set the notation and review some well known concepts about finite fields which will be used later in the paper.

2.1 Finite fields

Finite fields are fields containing a finite number of elements. For the purposes of this paper, we will only consider fields of integers modulo p , denoted by \mathbb{Z}_p , where p is a prime number.

In particular, we identify \mathbb{Z}_n , with n a positive integer (not necessarily prime), with the set of non-negative integers³ smaller than n ,

$$\mathbb{Z}_n = \{0, \dots, n-1\}. \quad (2.1)$$

Elementary arithmetic operations in \mathbb{Z}_n , such as addition, subtraction and multiplication, are defined using modular arithmetic, namely by performing the corresponding operation in \mathbb{Z} and taking the remainder of the integer division of the result modulo n . Given an element $a \in \mathbb{Z}_n$ with $a \neq 0$, if a and n are co-prime we can define the inverse a^{-1} of a in \mathbb{Z}_n with respect to multiplication, i.e. an element $b \in \mathbb{Z}_n$ such that

$$a^{-1} \bmod n \equiv b \quad \Leftrightarrow \quad (ab) \bmod n = 1. \quad (2.2)$$

One can indeed show that such a number b exists (and is unique in \mathbb{Z}_n) if and only if a and n are co-prime. If $n = p$ is a prime number, the existence of an inverse is therefore

³One could alternatively define \mathbb{Z}_n as a set of equivalence classes in \mathbb{Z} , but we find the definition given in this section more pragmatic and useful for the purposes of this paper.

guaranteed for every non-vanishing element of \mathbb{Z}_p . This implies that \mathbb{Z}_p is a field and any rational operation on its elements is well defined. The multiplicative inverse can be computed using the extended Euclidean algorithm, as explained in appendix A.1.

The existence of a multiplicative inverse implies that we can define a map between rational numbers and elements of a finite field \mathbb{Z}_p . In particular, given a rational number $q = a/b \in \mathbb{Q}$, we define

$$q \bmod p \equiv (a \times (b^{-1} \bmod p)) \bmod p. \tag{2.3}$$

This map is obviously not invertible (since \mathbb{Q} is infinite and \mathbb{Z}_p is finite), however it turns out one can reconstruct q , with very high probability, from its images in several finite fields \mathbb{Z}_{p_i} where $\{p_i\}$ is a set of prime numbers, as explained in section 2.3 and appendix A. This will enable us to reconstruct rational functions with rational coefficients from their values over finite fields. It is worth observing that we can similarly map q in \mathbb{Z}_n , with n not prime, as long as n and the denominator b are co-prime.

2.2 Polynomials and rational functions

In this paper we use a multi-index notation. Given a sequence of n variables $\mathbf{z} = (z_1, \dots, z_n)$, and the n -dimensional multi-index $\alpha = (\alpha_1, \dots, \alpha_n)$ with integers $\alpha_i \geq 0$, a monomial \mathbf{z}^α is defined as

$$\mathbf{z}^\alpha \equiv \prod_{i=1}^n z_i^{\alpha_i}. \tag{2.4}$$

The total degree of a monomial is denoted by $|a|$,

$$|a| = \sum_i \alpha_i. \tag{2.5}$$

With \mathbb{F} a generic field, we use the following (standard) definitions:

- $\mathbb{F}[\mathbf{z}]$ is the ring of polynomials in the variables \mathbf{z} with coefficients in \mathbb{F} . Any polynomial function $f \in \mathbb{F}[\mathbf{z}]$ can be uniquely identified by a set of multi-indexes $\{\alpha\}$ and coefficients $c_\alpha \in \mathbb{F}$ as

$$f(\mathbf{z}) = \sum_{\alpha} c_{\alpha} \mathbf{z}^{\alpha}. \tag{2.6}$$

- $\mathbb{F}(\mathbf{z})$ is the field of rational functions in the variables \mathbf{z} with coefficients in \mathbb{F} . Functions $f \in \mathbb{F}(\mathbf{z})$ can be expressed as a ratio of two polynomials $p, q \in \mathbb{F}[\mathbf{z}]$ as

$$f(\mathbf{z}) = \frac{p(\mathbf{z})}{q(\mathbf{z})} = \frac{\sum_{\alpha} n_{\alpha} \mathbf{z}^{\alpha}}{\sum_{\beta} d_{\beta} \mathbf{z}^{\beta}}, \tag{2.7}$$

where $n_{\alpha}, d_{\beta} \in \mathbb{F}$, while $\{\alpha\}$ and $\{\beta\}$ are sets of multi-indexes. Unlike the polynomial representation given in eq. (2.6), the representation of a rational function is not unique, even if we assume p and q to have no common polynomial factors. However, after GCD simplification, the only possible ambiguity is an overall constant normalization of the numerator and the denominator. In order to have a unique representation, useful e.g. when comparing functions obtained in different finite fields,

we use the convention of defining the coefficient of the lowest degree term of the denominator (with respect to the chosen monomial order) to be equal to one.

The functional reconstruction methods described in this paper are based on multiple evaluations of the function f to be reconstructed, which in turn correspond to assigning to each variable a value in the field \mathbb{F} . For the univariate case we denote these values by $y_i \in \mathbb{F}$ while multivariate we denote them by $\mathbf{y}_i \in \mathbb{F}^n$, where i is a label distinguishing different evaluation points.

In this paper the field \mathbb{F} will either be the rational field \mathbb{Q} or a finite field \mathbb{Z}_p . More in detail, our goal is the calculation of polynomial functions in $\mathbb{Q}[\mathbf{z}]$ and rational functions in $\mathbb{Q}(\mathbf{z})$. We will do so by performing a functional interpolation of the same functions in $\mathbb{Z}_p[\mathbf{z}]$ or $\mathbb{Z}_p(\mathbf{z})$ respectively, for several primes p if needed, and then use these to reconstruct the results over the rational field.

2.3 Rational reconstruction from finite fields

In the next sections we will describe an algorithm for efficiently reconstructing polynomials and rational functions over finite fields. As apparent from their representation in eq. (2.6) and (2.7), these functions can be identified by a sequence of monomials and their respective coefficients. The final step of the reconstruction algorithm therefore consists in promoting these coefficients from elements of a finite field \mathbb{Z}_p into a proper rational number. As we have seen in section (2.1), one can map a rational number q into an element of the set \mathbb{Z}_n of integers modulo n , as long as n and the (reduced) denominator of q are co-prime. Although this map is not invertible, it turns out one can use a variation of the extended Euclidean algorithm [54] (see also appendix A.1) in order to make a guess for q from its image $q \bmod n \in \mathbb{Z}_n$. This method is known as *rational reconstruction*. The guess will, in general, be correct when the numerator and the denominator of q are much smaller than n (heuristically one finds that the threshold is around \sqrt{n}). However, because of our requirement of working with machine-size integers (see section 6 for more details), we cannot always choose the prime p defining the field \mathbb{Z}_p to be significantly larger than the numerator and the denominator of any rational number which can be expected to appear in the results.

The solution to the problem comes from the Chinese remainder theorem, which allows to uniquely reconstruct an element in \mathbb{Z}_n , with $n = n_1 \cdots n_k$ and the n_i pairwise co-prime, from its images in \mathbb{Z}_{n_i} for $i = 1, \dots, k$, as explained in appendix A.2. This implies that, given a set of prime numbers p_i , we can perform the functional reconstruction over \mathbb{Z}_{p_i} and then combine the results to find the image of the rational coefficients in $\mathbb{Z}_{p_1 \cdots p_k}$. By performing the rational reconstruction over $\mathbb{Z}_{p_1 \cdots p_k}$, the result will thus be correct when the product of the selected primes is large enough.⁴ This allows us to use machine-size integers for a fast functional reconstruction in \mathbb{Z}_{p_i} , while the use of multi-precision arithmetic (computationally much more expensive) is restricted to this rational reconstruction

⁴A minor subtlety arises when the denominator of a rational number is a multiple of one of the primes p_i . We observe that this is very unlikely to happen if the p_i are of $\mathcal{O}(10^6)$ as in our implementation. Besides, in this case the functional reconstruction would fail and thus one can simply discard the prime and proceed with a different one.

step in $\mathbb{Z}_{p_1 \dots p_k}$, which takes a very small fraction of computing time compared to the one spent for the functional reconstruction over prime fields.

More in detail, given a sequence of primes p_1, \dots, p_k , we adopt the following algorithm for the full reconstruction of a rational function in $f \in \mathbb{Q}(\mathbf{z})$ (a completely analogous one can obviously be used for a polynomial):

1. Reconstruct the function f in $\mathbb{Z}_{p_1}(\mathbf{z})$ and store the result.
2. Use the rational reconstruction algorithm to promote the stored result to a (new) guess $g \in \mathbb{Q}(\mathbf{z})$.
3. Consider a new prime p_{i+1} , where i is the number of primes which have already been used so far. Evaluate the guess g over the new field $\mathbb{Z}_{p_{i+1}}$ for several values of \mathbf{z} . If the result for $g(\mathbf{z})$ agrees with the evaluation of the function $f(\mathbf{z})$, accept the guess g as the correct answer, i.e. assert $g(\mathbf{z}) = f(\mathbf{z})$, and successfully terminate the algorithm. Otherwise proceed to the next point.
4. Reconstruct the function in $\mathbb{Z}_{p_{i+1}}(\mathbf{z})$ and combine it with the stored result in $\mathbb{Z}_{p_1 \dots p_i}(\mathbf{z})$ using the Chinese remainder theorem in order to obtain the correct result in $\mathbb{Z}_{p_1 \dots p_{i+1}}(\mathbf{z})$. The latter thus replaces the previously stored result. Repeat from point 2.

The algorithm terminates when the comparison in point 3 is successful. For the examples presented in this paper, we typically only need to perform the functional reconstruction over one or two prime fields.

We observe that the techniques reviewed in this section (which, as stated, are well known) allow reconstructing a multivariate rational function with very high probability. In practice, exceptional cases are very artificial and irrelevant for realistic applications. It should also be noted that the final result can be extensively checked against the evaluation of the function f on even more values of the prime p and the variables \mathbf{z} .

In the rest of this paper, we will discuss a functional reconstruction algorithm which is suited for complex theoretical calculations in high-energy physics, and its application to techniques related to the computation of tree-level and multi-loop scattering amplitudes in QFT.

3 Functional reconstruction

In this section we describe a dense functional reconstruction algorithm for polynomials and rational functions whose performance scales well with the complexity of the result. For the sake of generality we make no further assumption about the functions to be reconstructed. Unless explicitly stated otherwise, the techniques illustrated in this section can be applied to functions over any field, although in practice we are mostly interested to their application over finite fields.

3.1 The black-box interpolation problem

Given a function f of n variables $\mathbf{z} = (z_1, \dots, z_n)$ over a field \mathbb{F} , the so-called *black-box interpolation problem* is the problem of reconstructing the function f from the results of its evaluation for several values of \mathbf{z} . In other words, one can think about the function as a numerical procedure of the form

$$\mathbf{z} \longrightarrow \boxed{f} \longrightarrow f(\mathbf{z}), \tag{3.1}$$

while the functional reconstruction method has no knowledge about the algorithm used for the calculation of f .

In this paper we are interested in reconstructing polynomials and rational functions over the rational field. Our setup is a modified version of the black-box interpolation problem, where the function is evaluated modulo a prime number p , and can schematically be represented as

$$(\mathbf{z}, p) \longrightarrow \boxed{f} \longrightarrow f(\mathbf{z}) \bmod p. \tag{3.2}$$

We recall that the rational reconstruction technique reviewed in section 2.3 reduces the problem of a functional reconstruction over \mathbb{Q} to the problem of a functional reconstruction over prime fields \mathbb{Z}_p for generic p . This makes the setups in eq. (3.1) and eq. (3.2) effectively equivalent for our purposes. Hence, in the remainder of this paper, we will focus on functional reconstruction techniques over finite fields \mathbb{Z}_p , where p is an arbitrary prime. As stated, these are actually valid in any field \mathbb{F} and the use of finite fields is meant to provide a fast but exact numerical evaluation of the black-box function f to be reconstructed. Therefore, in this section we will simply denote the generic field by \mathbb{F} .

The advantage of turning an analytic calculation into a black-box interpolation is that it reduces the problem of computing a function f into the one of providing a fast numerical evaluation for it. Since the reconstruction is independent of the algorithm used for the evaluation of f , it has a very broad spectrum of applications. Numerical calculations can in turn avoid issues such as large intermediate expressions, which affect many computations in high-energy physics. With this approach, the number of evaluations needed for the reconstruction of a function scales linearly with the number of terms of the result itself and is independent of the complexity of intermediate expressions which may appear using fully analytic techniques.

We remind the reader that, because we are dealing with polynomials and rational functions, which can be represented as in eq. (2.6) and eq. (2.7) respectively, the goal of a functional reconstruction algorithm is to identify the monomials appearing in their definition and the corresponding coefficients as elements of the field \mathbb{F} .

Functional reconstruction algorithms roughly fall into two categories: *dense* and *sparse* algorithms. As suggested by their name, sparse reconstruction algorithms attempt to be more efficient (with respect to number of function evaluations needed) in the case where the number of non-vanishing terms is small compared to the one expected from their total degree. In this paper we focus, as mentioned, on dense reconstruction

algorithms, seeking good efficiency in the most general case where many non-vanishing terms are present, rather than being optimal in the special cases where the function to be reconstructed is relatively simple.

In order to better motivate the discussion which follows, it is useful to consider first a straightforward system-solving strategy for the functional reconstruction, and point out why it is not suitable for our purposes. Given a set of values $\mathbf{y}_i \in \mathbb{F}^n$, using eq. (2.6) and eq. (2.7), one can build systems of equations of the form

$$\sum_{\alpha} c_{\alpha} \mathbf{y}_i^{\alpha} - f(\mathbf{y}_i) = 0 \tag{3.3}$$

and

$$\sum_{\alpha} n_{\alpha} \mathbf{y}_i^{\alpha} - \sum_{\beta} d_{\beta} \mathbf{y}_i^{\beta} f(\mathbf{y}_i) = 0, \tag{3.4}$$

to be solved for the coefficients $\{c_{\alpha}\}$ and $\{n_{\alpha}, d_{\beta}\}$ for polynomials and rational functions respectively, using basic techniques such as Gauss elimination. Notice, however, that one cannot know a priori which monomials will appear in the result. Moreover, in section 2.2 we defined the coefficients of our canonical representation of rational functions such that the coefficient of the lowest degree term in the denominator is equal to one, in order to solve the ambiguity in their representation. Unfortunately we have no knowledge about the mentioned term before having performed the reconstruction. This is however a minor issue which will be solved as discussed in section 3.5 using techniques proposed in ref. [26]. The same techniques will allow us to assess the total degree of the numerator and the denominator of a multivariate rational function, or the one of a polynomial, using a relatively small number of evaluations. Hence, a viable solution to the functional reconstruction problem consists in listing the full set of N monomial terms compatible with the total degree of the function involved, sampling the function with (at least) N values for the set of variables \mathbf{z} , and solving the resulting $N \times N$ system of equations for the coefficients. While this method is straightforward and efficient for simple functions depending on only one or two variables, it has however a bad scaling behaviour when increasing the number of variables or the total degree of the result. This can be understood simply by recalling that solving an $N \times N$ dense system of linear equations is an $\mathcal{O}(N^3)$ operation, and the multivariate problems in which we are interested in can have several thousands (or even hundreds of thousands) of potentially non-vanishing terms (notice e.g. that the most general polynomial in n variables and total degree R has $N = \binom{R+n}{R}$ terms).

The reconstruction algorithms we are going to describe have instead a much better scaling with the complexity of the result, and the time spent for the functional reconstruction itself is typically much smaller than the time required for evaluating the function to be reconstructed. In the following, we start by describing well established algorithms for the univariate case and later use them as ingredients for the multivariate one.

3.2 Univariate polynomials

For univariate polynomial functions $f = f(z)$ we adopt a well known reconstruction method based on Newton's polynomial representation. Given a sequence of distinct values

$y_0, \dots, y_R \in \mathbb{F}$, a univariate polynomial $f \in \mathbb{F}[z]$ of total degree R can be written as [55]

$$\begin{aligned}
 f(z) &= \sum_{r=0}^R a_r \prod_{i=0}^{r-1} (z - y_i) \\
 &= a_0 + (z - y_0) \left(a_1 + (z - y_1) \left(a_2 + (z - y_2) (\dots + (z - y_{r-1}) a_r) \right) \right). \quad (3.5)
 \end{aligned}$$

The coefficients a_r can be computed recursively by evaluating the function f at the values y_i as

$$\begin{aligned}
 a_0 &= f(y_0) \\
 a_1 &= \frac{f(y_1) - a_0}{y_1 - y_0} \\
 a_2 &= \left((f(y_2) - a_0) \frac{1}{y_2 - y_0} - a_1 \right) \frac{1}{y_2 - y_1} \\
 &\dots = \dots \\
 a_r &= \left(\left((f(y_r) - a_0) \frac{1}{y_r - y_0} - a_1 \right) \frac{1}{y_r - y_1} - \dots - a_{r-1} \right) \frac{1}{y_r - y_{r-1}}. \quad (3.6)
 \end{aligned}$$

An important feature of the method is that each coefficient a_r only depends on the evaluation of f at the points y_i with $i \leq r$. This implies that new evaluation points cannot change the value of the previously computed coefficients. This is ideal for the case where the total rank R of the polynomial f is not known a priori, and it is our main reason for preferring this method over alternatives. In practice, we apply the algorithm recursively until we find a set of consecutive coefficients a_r which evaluate to zero. This is the termination criterion of the algorithm (notice that, even when the canonical form of the polynomials has several vanishing entries, in general the entries of its Newton representation will be non-vanishing, hence the described termination criterion is robust and an incorrect termination is extremely unlikely).

The sequence y_i is generated dynamically, taking into account that the algorithm evaluating of the function f might fail at a particular point. More in detail, we choose y_0 as an arbitrary element of the field \mathbb{F} and then we recursively define the following ones as $y_i = y_{i-1} + 1$, as long as the evaluation of f is successful. If the evaluation of f fails, we try replacing the current point y_i with $y_i + 1$, until we find a value for which the evaluation of f is possible. If too many consecutive evaluations fail, we terminate the algorithm declaring the reconstruction unsuccessful.

Even though Newton's representation is more practical for functional reconstruction purposes, after a successful reconstruction it is convenient to convert it back into the canonical representation given by eq. (2.6), which in the univariate case can be written as

$$f(z) = \sum_{r=0}^R c_r z^r. \quad (3.7)$$

It is worth observing that the conversion from the representation in eq. (3.5) to the one in eq. (3.7) only requires the following two operations

- addition of univariate polynomials,
- multiplication of a univariate polynomial by a linear univariate polynomial.

Both operations are simple enough to be efficiently implemented over finite fields \mathbb{Z}_p in statically-typed languages, without resorting to external computer algebra systems. More details on our implementation are given in section 6.

3.3 Univariate rational functions

For univariate rational functions $f \in \mathbb{F}(z)$ it is worth distinguishing two cases. The first case is applicable when the degree of the numerator and the denominator (R and R' respectively) are known and the constant term of the denominator is known to be non-vanishing. As we shall see, this is useful for the multivariate reconstruction discussed in section 3.5. In this particular case, the function admits a representation of the form

$$f(z) = \frac{\sum_{r=0}^R n_r z^r}{\sum_{r'=0}^{R'} d_{r'} z^{r'}}, \tag{3.8}$$

with $d_0 = 1$. We find the system-solving method explained at the end of section 3.1 is well suited for this univariate case, since the rank of the numerator and the denominator are unlikely to be high enough to make Gauss elimination impractically expensive, at least for the kind of problems we are interested in this paper (as we stated, the same is not true for the multivariate case, as the number of variables or the degree of the numerator and the denominator increase). Having set $d_0 = 1$, the system of equations reads

$$\sum_{r=0}^R n_r y_i^r - \sum_{r'=1}^{R'} d_{r'} y_i^{r'} f(y_i) = f(y_i), \tag{3.9}$$

and can be solved for the unknown coefficients $\{n_r, d_{r'}\}$ by evaluating the function f at least $R + R' + 1$ times. In practice we include a few more evaluation points y_i , making the system slightly over-constrained as a cross check.

We now address the more general case where the degree of the numerator and the denominator are not known. We find convenient to use a method based on Thiele's interpolation formula [55], which expresses a rational function $f \in \mathbb{F}(z)$ as a continued fraction

$$f(z) = a_0 + \frac{z - y_0}{a_1 + \frac{z - y_1}{a_2 + \frac{z - y_2}{\dots + \frac{z - y_{r-1}}{a_N}}}} = a_0 + (z - y_0) \left(a_1 + (z - y_1) \left(a_2 + (z - y_2) \left(\dots + \frac{z - y_{N-1}}{a_N} \right)^{-1} \right)^{-1} \right)^{-1}, \tag{3.10}$$

where y_0, \dots, y_N is a sequence of distinct elements of \mathbb{F} . Thiele's interpolation formula can be regarded as the analog of Newton's formula for rational functions. The second line of eq. (3.10), by comparison with the second line of eq. (3.5), makes the analogy manifest.

The coefficients a_i can be recursively computed by evaluating the function f at the values y_i ,

$$\begin{aligned}
 a_0 &= f(y_0) \\
 a_1 &= \frac{y_1 - y_0}{f(y_1) - a_0} \\
 a_2 &= \left((f(y_2) - a_0)^{-1} (y_2 - y_0) - a_1 \right)^{-1} (y_2 - y_1) \\
 \dots &= \dots \\
 a_j &= \left(\left((f(y_j) - a_0)^{-1} (y_j - y_0) - a_1 \right)^{-1} (y_j - y_1) - \dots - a_{j-1} \right)^{-1} (y_j - y_{j-1}). \quad (3.11)
 \end{aligned}$$

Similarly to Newton’s interpolation formula, each value a_j only depends on the evaluations at $z = y_i$ with $i \leq j$, which makes the algorithm convenient in the case where the total number of terms is not known, since new evaluation points do not change previously computed terms. Our strategy for generating the sequence of points y_i is also the same as the one used in the polynomial case. It is worth pointing out that Thiele’s interpolation formula, being a continued fraction, contains spurious singularities which might make the application of equations (3.11) impossible. Similarly to the case where the evaluation of f fails, if such a spurious singularity is encountered at a point y_i , we simply discard the value and replace it with $y_i + 1$. The termination criterion is the agreement between a new evaluation $f(y_i)$ with the evaluation of the rational function defined by the coefficients a_0, \dots, a_{i-1} already computed. We make the algorithm more robust by requiring agreement between the reconstructed function and several new evaluations of f .

After a successful interpolation, the result is converted into the canonical form given by eq. (3.8), except that the condition $d_0 = 1$ is replaced by $d_{\min r'} = 1$, where $z^{\min r'}$ is the lowest degree monomial with a non-vanishing coefficient in the denominator. We observe that Thiele’s formula with $N + 1$ terms represents a rational function with degree R and R' for the numerator the denominator respectively, where $R = R' = N/2$ if N is even, and $R = R' + 1 = (N + 1)/2$ if N is odd. In other words, either the degree of the numerator and the denominator of the reconstructed function are equal, or they differ by one unity at most. This implies that the highest degree coefficients n_r or $d_{r'}$, obtained by converting the result into its canonical form, might be vanishing, in which case they are discarded. For this reason, if the degrees R and R' are already known, the system-solving strategy typically requires fewer evaluations of the function, since this way one can avoid reconstructing zeros. Thiele’s interpolation is however preferred when R and R' are not known.

The conversion into a canonical form can be implemented by performing on the polynomial numerator and the denominator of the function the same kind of operations we listed for converting Newton’s polynomial representation into the canonical form (and inversion, which is simply implemented by swapping numerator and denominator). Hence, as we stated for the previous case, this does not require the usage of computer algebra systems and can be easily implemented in statically-typed languages using functions over finite fields \mathbb{Z}_p .

We stress that the result we obtain can be shown to be minimal with respect to the degrees of the numerator and the denominator, and hence no GCD simplification is needed

after the reconstruction is converted into a canonical form (and possible high-degree zero terms are discarded).

3.4 Multivariate polynomials

Given a sequence of variables $\mathbf{z} = (z_1, \dots, z_n)$ and a multivariate polynomial function $f \in \mathbb{F}[\mathbf{z}]$, the interpolation of f can be performed by recursive application of the univariate Newton's reconstruction method described above. In other words, we consider a generic multivariate polynomial $f \in \mathbb{F}[z_1, \dots, z_n]$ as a univariate polynomial in the first variable z_1 , whose coefficients are polynomials in the remaining variables (z_2, \dots, z_n) . Newton's interpolation formula (3.12) can thus be generalized to the multivariate case, by upgrading the coefficients a_i from elements of the field \mathbb{F} to elements of the polynomial ring $\mathbb{F}[z_2, \dots, z_n]$,

$$f(z_1, \dots, z_n) = \sum_{r=0}^R a_r(z_2, \dots, z_n) \prod_{i=0}^{r-1} (z_1 - y_i), \tag{3.12}$$

where, as before, the y_i are distinct elements of \mathbb{F} . The solutions for the coefficients a_r in eq. (3.6) also apply to the multivariate case, with the following substitutions

$$f(y_j) \longrightarrow f(y_j, z_2, \dots, z_n), \quad a_j \longrightarrow a_j(z_2, \dots, z_n). \tag{3.13}$$

In this case, the right hand side of each equation (3.6) for $a_r(z_2, \dots, z_n)$ thus depends on the previously computed polynomial coefficients $a_j(z_2, \dots, z_n)$ with $j < r$, and on the function f where the first variable z_1 has been set to the value $z_1 = y_r$. Therefore the reconstruction of $a_r(z_2, \dots, z_n)$ is reduced to the black-box interpolation of a polynomial function in $n - 1$ variables. In particular, the evaluation of $a_r(z_2, \dots, z_n)$ can be obtained by evaluating $f(y_r, z_2, \dots, z_n)$ and then combining it with the previously computed polynomial coefficients a_j by applying the formulas in eq. (3.6). The recursion ends with the univariate case, where we apply the univariate polynomial reconstruction algorithm we already discussed.

In order to convert a multivariate polynomial from a (recursive) Newton representation into the canonical form in eq. (2.6), one needs the following basic operations

- addition of multivariate polynomials,
- multiplication of a multivariate polynomial by a linear univariate polynomial.

Although these operations are slightly more involved than the corresponding ones for the univariate case, they still allow a rather straightforward and efficient implementation in statically-typed languages, especially in the case of polynomials over finite fields \mathbb{Z}_p . We will discuss the details of our implementation and the polynomial representation we used in section 6.

3.5 Multivariate rational functions

The reconstruction of multivariate rational functions is a considerably more complex problem than those addressed so far. We were not able to find a dense multivariate reconstruction algorithm suited for our purposes in the literature. However, we observe that

the techniques proposed in ref. [26] for sparse rational functions, can also be applied, with some minor modifications, to dense rational functions as well. In this section we describe our version of the techniques proposed in [26], and in particular their combination with the dense functional reconstruction methods discussed above. The efficiency of the resulting algorithm turns out to meet our performance goals, as we shall see in the next sections.

We consider the variables $\mathbf{z} = (z_1, \dots, z_n)$ and a rational function $f \in \mathbb{F}(\mathbf{z})$, which admits a canonical representation of the form of eq. (2.7). A first issue to be addressed is the ambiguity in the overall normalization of the numerator and denominator of the function. As we stated, we get rid of this ambiguity by requiring the canonical representation of the function to have the coefficient of the lowest degree term in the denominator equal to one. However, identifying such a term (or indeed any term in the function) before the functional reconstruction is in general not possible. As observed in ref. [26], there is however a simple case where this issue can be easily solved, i.e. when the lowest-degree non-vanishing term of the denominator is the constant term, in which case we can simply set the canonical normalization by imposing

$$d_{(0,\dots,0)} = 1. \tag{3.14}$$

It is also noted in ref. [26] that, even in the case where $d_{(0,\dots,0)} = 0$, one can always identify a shift $\mathbf{s} = (s_1, \dots, s_n) \in \mathbb{F}^n$ in the arguments of f such that the shifted function

$$f_{\mathbf{s}}(\mathbf{z}) = f(\mathbf{z} + \mathbf{s}) = f(z_1 + s_1, \dots, z_n + s_n) \tag{3.15}$$

satisfies $d_{(0,\dots,0)} \neq 0$ (i.e. is non-singular in $z = (0, \dots, 0)$) and can thus be normalized as in eq. (3.14). Hence, in these cases, the method we are going to describe can be applied to the function $f_{\mathbf{s}}$ instead. A minor subtlety is the fact that, after such a shift, the new function $f_{\mathbf{s}}$ might become considerably more complex than the original function f , and thus harder to reconstruct. This issue is actually much more relevant for the sparse reconstruction case discussed in ref. [26], but it can also affect the dense reconstruction case, especially if the variables \mathbf{z} have been carefully chosen for being suited to describe the function at hand. We will address this subtlety later and for the time being we turn to the description of the functional reconstruction of a rational function $f \in \mathbb{F}(\mathbf{z})$ whose constant term in the denominator is non-vanishing.

The method illustrated in ref. [26] consists in introducing an auxiliary variable t and defining a new function $h \in \mathbb{F}(t, \mathbf{z})$ as

$$h(t, \mathbf{z}) = f(t\mathbf{z}) = f(tz_1, \dots, tz_n). \tag{3.16}$$

Using the canonical representation of f given by eq. (2.7) and denoting the total degree of the numerator and the denominator of f by R and R' respectively, we get

$$h(t, \mathbf{z}) = \frac{\sum_{r=0}^R p_r(\mathbf{z}) t^r}{1 + \sum_{r'=1}^{R'} q_{r'}(\mathbf{z}) t^{r'}}, \tag{3.17}$$

where

$$p_r(\mathbf{z}) \equiv \sum_{|\alpha|=r} n_\alpha \mathbf{z}^\alpha, \quad q_{r'}(\mathbf{z}) \equiv \sum_{|\beta|=r'} d_\beta \mathbf{z}^\beta. \quad (3.18)$$

In other words, the function h can be regarded as a univariate rational function in the variable t , whose coefficients are multivariate homogeneous polynomials in the variables \mathbf{z} . We obviously have

$$f(\mathbf{z}) = h(1, \mathbf{z}). \quad (3.19)$$

Notice that the constant term of the denominator of f coincides with the constant term of the denominator of h seen as a univariate rational function in t , hence by normalizing the latter to 1, the same normalization is automatically applied to the former.

Our implementation of the method consists in reconstructing the homogeneous polynomials p_r and $q_{r'}$ for $0 \leq r \leq R$ and $1 \leq r' \leq R'$, using the multivariate Newton reconstruction method illustrated above. The evaluations of these polynomials at a generic point $\mathbf{z} = \mathbf{y}_i \in \mathbb{F}^n$ are in turn obtained by reconstructing

$$h_{\mathbf{y}_i}(t) \equiv h(t, \mathbf{y}_i) \quad (3.20)$$

as a univariate rational function in t and identifying $p_r(\mathbf{y}_i)$ and $q_{r'}(\mathbf{y}_i)$ with its coefficients. Notice that this univariate reconstruction is actually equivalent to the evaluation of all the polynomials p_r and $q_{r'}$ at the same time. In order not to waste evaluations, for each sampled value $\mathbf{y}_i \in \mathbb{F}^n$ we cache the corresponding reconstructed univariate function $h_{\mathbf{y}_i} \in \mathbb{F}(t)$ so that its coefficients can be re-used for the reconstruction of several polynomials. Moreover, since $p_r(\mathbf{z})$ and $q_{r'}(\mathbf{z})$ are homogeneous polynomials of degree r and r' respectively, during their reconstruction we can drop their dependence on the first variable z_1 by setting $z_1 = 1$ and consider the polynomials $p_r(1, z_2, \dots, z_n)$ and $q_{r'}(1, z_2, \dots, z_n)$ instead. The dependence on z_1 is thus restored by homogenizing the result. This simplification makes up for having introduced the auxiliary variable t .

We now briefly discuss the univariate reconstruction algorithm we use for the functions $h_{\mathbf{y}_i} \in \mathbb{F}(t)$ at each point $\mathbf{y}_i \in \mathbb{F}^n$. Since we typically cannot know a priori the degrees R and R' of the numerator and the denominator respectively, we first perform a few univariate reconstructions using Thiele's interpolation method, in order to obtain this information. As a byproduct, this is also used to check whether the constant term of the denominator vanishes, in which case, as mentioned, we proceed by specifying a different shift $\mathbf{z} \rightarrow \mathbf{z} + \mathbf{s}$ in the arguments of f (more details about this point are given below). Once the degree of the numerator and the denominator are known, and a shift \mathbf{s} such that $d_{(0, \dots, 0)} \neq 0$ is found, we switch to the system-solving algorithm described at the beginning of section 3.3, since it typically requires fewer evaluations of the function (as explained, Thiele's interpolation becomes optimal with respect to the number of evaluations needed only when the degree of the numerator and the denominator are equal or the former is one unity higher than the latter).

We finally briefly describe a convenient way to implement the aforementioned variable shift to avoid the case where $d_{(0, \dots, 0)} \neq 0$ (this point is more extensively discussed in ref. [26], to which we refer the reader for further details). For simplicity we focus on the polynomials p_r in the numerator of the function, although it should be clear that completely

analogous statements can be made for the polynomials q_r in the denominator. The key observation is that a shift in the variables $\mathbf{z} \rightarrow \mathbf{z} + \mathbf{s}$ in one of the homogeneous polynomials p_r can only affect the homogeneous polynomials $p_{r'}$ with $r' < r$. In particular the highest degree polynomial p_R is the same for the function f and the shifted function $f_{\mathbf{s}}$ defined in eq. (3.15). We then start with the reconstruction of the highest degree polynomial p_R , which is the same for f and $f_{\mathbf{s}}$. We then compute the effects of the shift on lower rank terms, which for a generic p_r are given by

$$p_r(\mathbf{z} + \mathbf{s}) - p_r(\mathbf{z}). \tag{3.21}$$

During the reconstruction of lower degree terms, each function evaluation of the polynomials defining $f_{\mathbf{s}}$ are thus corrected by the effects of this shift induced by higher rank terms. With this method we can reconstruct the function f directly (up to an overall common normalization of the numerator and the denominator, which can be fixed at the end using our definition of their canonical form), which as stated can be expected to be simpler than $f_{\mathbf{s}}$.

As a final remark, we point out that, on top of the basic polynomial operations needed for the multivariate polynomial reconstruction, in order to implement this algorithm one also needs to compute the effects of a shift $\mathbf{z} \rightarrow \mathbf{z} + \mathbf{s}$ in the variables. This is relatively straightforward if done for one variable at a time, by using

$$(z + s)^r = \sum_{j=0}^r \binom{r}{j} s^{r-j} z^j. \tag{3.22}$$

Another operation which is needed is the homogenization required to restore the dependence of the polynomials p_r and q_r on their first variable, but depending on the polynomial representation used this can be trivial, since it amounts to adjusting the first entry of the multi-indexes $\alpha = (\alpha_1, \dots, \alpha_n)$ representing the exponents of the monomials. As already mentioned, more details about our implementation are given in section 6.

This completes the description of the functional reconstruction algorithms used in this paper. In sections 4 and 5 we show how to evaluate functions relevant for calculations high-energy physics over finite fields \mathbb{Z}_p , making thus possible to use these techniques for the reconstruction of their analytic form as rational functions over the field \mathbb{Q} .

3.6 Examples

Before moving to applications in high-energy physics, we briefly discuss two simple examples of functional reconstruction, which might be useful in order to clarify the concepts introduced in this section, especially for the reconstruction of multivariate rational functions, which is considerably more involved than the other cases. We perform the reconstruction over the field \mathbb{Q} , since it is easier to follow, but any step can equivalently be performed over finite fields \mathbb{Z}_p , yielding the same result modulo p .

Function non-singular at $\mathbf{z} = (\mathbf{0}, \dots, \mathbf{0})$. We consider the following rational function of two variables

$$f(\mathbf{z}) = f(z_1, z_2) = \frac{3 + 2z_1 + 4z_2 + 7z_1^2 + 5z_1z_2 + 6z_2^2}{1 + 7z_1 + 8z_2 + 10z_1^2 + z_1z_2 + 9z_2^2}. \tag{3.23}$$

Because the function is well defined at $\mathbf{z} = (0, 0)$ there is no need to shift variables in this case. Our goal is thus to reconstruct the analytic formula of this function from a black-box providing a numerical evaluation of it. The first step consists in defining the function h in eq. (3.17), which in this case is given by

$$h(t, \mathbf{z}) = f(t z_1, t z_2) = \frac{p_0 + p_1(z_1, z_2) t^2 + p_2(z_1, z_2) t^2}{1 + q_1(z_1, z_2) t^2 + q_2(z_1, z_2) t^2}, \quad (3.24)$$

with

$$\begin{aligned} p_0(z_1, z_2) &= 3, & q_0(z_1, z_2) &= 1, \\ p_1(z_1, z_2) &= 2z_1 + 4z_2, & q_1(z_1, z_2) &= z_1 + z_2, \\ p_2(z_1, z_2) &= 7z_1^2 + 5z_1z_2 + 6z_2^2, & q_2(z_1, z_2) &= 10z_1^2 + z_1z_2 + 9z_2^2. \end{aligned} \quad (3.25)$$

We thus reconstruct the polynomials p_r and $q_{r'}$ by interpolating the function $h_{\mathbf{y}_i}(t) \equiv h(t, \mathbf{y}_i)$ for several points \mathbf{y}_i . As stated, since the polynomials p_r and $q_{r'}$ are homogeneous and have degree r and r' respectively, we can set $z_1 = 1$ and homogenize the result at the end.

We assume no knowledge about the total degree of f . Hence we first perform a reconstruction of $h_{\mathbf{y}_1}(t)$ using Thiele's interpolation formula for a specific value of \mathbf{y}_1 , say $\mathbf{y}_1 \equiv (1, 1)$, which yields

$$h_{(1,1)}(t) = \frac{3 + 6t + 18t^2}{1 + 2t + 20t^2}. \quad (3.26)$$

This gives us information about the total degree of the numerator and the denominator, which can then be confirmed by repeating the reconstruction at more values of $\mathbf{z} = \mathbf{y}_i$. Moreover, by comparison with eq. (3.24), this also yields the value of the polynomials p_r and $q_{r'}$ at $(z_1, z_2) = (1, 1)$, namely

$$\begin{aligned} p_0(1, 1) &= 3, & q_0(1, 1) &= 1, \\ p_1(1, 1) &= 6, & q_1(1, 1) &= 2, \\ p_2(1, 1) &= 18, & q_2(1, 1) &= 20, \end{aligned} \quad (3.27)$$

which can be used for their polynomial fit. We perform two more reconstructions, say at $\mathbf{y}_2 = (1, 2)$ and $\mathbf{y}_3(1, 3)$, which yield

$$\begin{aligned} h_{(1,2)}(t) &= \frac{3 + 10t + 41t^2}{1 + 3t + 48t^2} \\ h_{(1,3)}(t) &= \frac{3 + 14t + 76t^2}{1 + 4t + 94t^2}. \end{aligned} \quad (3.28)$$

These, together with the previous point $\mathbf{z} = \mathbf{y}_1$, are sufficient in order to reconstruct, using e.g. Newton's formula, the univariate polynomials

$$\begin{aligned} p_1(1, z_2) &= 2 + 4z_2, & q_1(1, z_2) &= 1 + z_2, \\ p_2(1, z_2) &= 7 + 5z_2 + 6z_2^2, & q_2(1, z_2) &= 10 + z_2 + 9z_2^2. \end{aligned} \quad (3.29)$$

After homogenization, these reproduce the ones in eq. (3.25) and thus the function $f(\mathbf{z}) = h(1, \mathbf{z})$.

Function singular at $\mathbf{z} = (\mathbf{0}, \dots, \mathbf{0})$. We now briefly discuss the case of a function which is singular at $\mathbf{z} = (0, 0)$. We consider

$$f(\mathbf{z}) = f(z_1, z_2) = \frac{3 + 2z_1 + 4z_2 + 7z_1^2 + 5z_1z_2 + 6z_2^2}{z_1 + z_2 + 10z_1^2 + z_1z_2 + 9z_2^2}, \quad (3.30)$$

which differs from the previous example only for the missing constant term in the denominator. Using Thiele's interpolation formula for $h_{(1,1)}(t)$, we now get

$$h_{(1,1)}(t) = \frac{3/2 + 3t + 9t^2}{t + 10t^2}, \quad (3.31)$$

which tells us that the constant term of the denominator is vanishing and thus we need to specify an argument shift. By trial and error, repeating the univariate functional reconstruction for several shifts, we find e.g. that $\mathbf{s} = (2, 0)$ is a good shift, i.e.

$$f_{\mathbf{s}}(\mathbf{z}) = f(z_1 + 2, z_2) \quad (3.32)$$

is non-singular at $(z_1, z_2) \equiv 0$. The function $h_{\mathbf{s}}$ is now defined as

$$h_{\mathbf{s}}(t, \mathbf{z}) = f(tz_1 + 2, tz_2) \quad (3.33)$$

and can similarly be written as in eq. (3.24). Now its shifted polynomial coefficients are

$$\begin{aligned} p_{\mathbf{s},0}(z_1, z_2) &= \frac{35}{43}, & q_{\mathbf{s},0}(z_1, z_2) &\equiv 1, \\ p_{\mathbf{s},1}(z_1, z_2) &= \frac{30}{43}z_1 + \frac{14}{43}z_2, & q_{\mathbf{s},1}(z_1, z_2) &= \frac{41}{43}z_1 + \frac{3}{43}z_2, \\ p_{\mathbf{s},2}(z_1, z_2) &= \frac{7}{43}z_1^2 + \frac{5}{43}z_1z_2 + \frac{6}{43}z_2^2, & q_{\mathbf{s},2}(z_1, z_2) &= \frac{10}{43}z_1^2 + \frac{1}{43}z_1z_2 + \frac{9}{43}z_2^2. \end{aligned} \quad (3.34)$$

These coefficients can be reconstructed similarly to the previous case, but as we explained we use a different strategy where the effects of the shift are recursively subtracted from the evaluations of each polynomial coefficient. This allows us to reconstruct the polynomial coefficients p_r and $q_{r'}$ of the original function f , up to an overall normalization of the numerator and the denominator, rather than the coefficients $p_{\mathbf{s},r}$ and $q_{\mathbf{s},r'}$ of the shifted function $f_{\mathbf{s}}$. For simplicity we focus on the reconstruction of the polynomials p_r in the numerator. We observed that the highest degree polynomial p_2 is independent of the chosen shift, and thus we can simply reconstruct it from several interpolations of $h_{\mathbf{s},y_i}(t)$ as in the non-singular case. We then compute the effects of the variable shift on the lower degree terms, which are given by

$$p_2(z_1 + 2, z_2) - p_2(z_1, z_2) = \left(\frac{28}{43}z_1 + \frac{10}{43}z_2 \right) + \left(\frac{28}{43} \right). \quad (3.35)$$

The term in the first and second parenthesis on the r.h.s. are the effects of the variable shift coming from p_2 on the coefficients p_1 and p_0 respectively. When reconstructing p_1 , we thus evaluate $p_{\mathbf{s},1}$ (for which we can re-use the interpolations of $h_{\mathbf{s}}$ we already performed for p_2) and correct its values using

$$p_1(z_1, z_2) = p_{\mathbf{s},1}(z_1, z_2) - \left(\frac{28}{43}z_1 + \frac{10}{43}z_2 \right) = \frac{2}{43}z_1 + \frac{4}{43}z_2. \quad (3.36)$$

Hence the polynomial $p_{s,1}$ is never actually reconstructed, since we correct its evaluation as in the previous equation and reconstruct p_1 instead. We thus proceed with the reconstruction of p_0 , for which we need the effects of the shift on p_1

$$p_1(z_1 + 2, z_2) - p_1(z_1, z_2) = \left(\frac{4}{43}\right), \tag{3.37}$$

which combined with the ones on p_2 give

$$p_0 = p_{s,0} - \frac{4}{43} - \frac{28}{43} = \frac{3}{45}. \tag{3.38}$$

Once again, the evaluation of $p_{s,0}$ is obtained by the interpolations of h_s we already performed for p_2 . A similar strategy can obviously be applied to the denominator. While q_2 , following the same arguments, is not affected by the variable shift (i.e. $q_2 = q_{s,2}$), the other terms corrected by the shift are

$$q_2(z_1, z_2) = \frac{z_1}{43} + \frac{z_2}{43}, \quad q_0 = 0. \tag{3.39}$$

Notice that, after correcting for the shift, the lowest degree term of the denominator, namely $z_2/43$, does not have coefficient equal to unity. In order to bring the result into our canonical form, all the coefficients are thus multiplied by its inverse (in this case 43), obtaining the result in eq. (3.30).

4 Spinor-helicity and tree-level techniques

In this section we describe how spinor-helicity techniques and tree-level calculations can be implemented over finite fields \mathbb{Z}_p , and thus used in combination with the reconstruction algorithms illustrated above. In particular, one has to ensure that both the inputs and each intermediate step of a calculation performed via these technique can be implemented through a sequence of rational operations. These techniques also serve as building blocks for multi-loop calculations via integrand-reduction and generalized unitarity, whose implementation on finite fields will be described in section 5.

More in detail, in section 4.1 we describe how to use the well known four-dimensional spinor-helicity formalism in finite-field calculations. As we shall see, for loop calculations in the context of generalized unitarity, it can also be convenient to embed loop momenta in a \mathcal{D} -dimensional space with integer $\mathcal{D} > 4$. For this purpose, we will also use the six-dimensional spinor-helicity formalism [51–53], as described in appendix B. In section 4.2 we recall how to combine these techniques for tree-level calculations by means of Berends-Giele recursion.

4.1 Four-dimensional momenta and spinors

The four-dimensional spinor-helicity formalism offers a very convenient way of expressing helicity amplitudes in gauge theories. This formalism is widely used and here we only review a few details which are relevant for our implementation.

Given a massless momentum p , its corresponding two-component spinors $|p\rangle$ and $|p]$ can be defined as the independent solutions of the Dirac equation,

$$p^\mu \sigma_\mu |p\rangle = p^\mu \sigma_\mu |p] = 0, \tag{4.1}$$

where $\sigma_\mu = (\mathbf{1}_{2 \times 2}, \sigma_i)$ with σ_i being the Pauli matrices, and they satisfy

$$|p\rangle [p| = p_\mu \sigma^\mu. \tag{4.2}$$

In particular, if p is the momentum of an outgoing particle of a process, $|p\rangle$ and $|p]$ are identified as the negative- and positive-helicity solution of the Dirac equation respectively.

We consider an arbitrary n -point process, with external momenta p_1, \dots, p_n , all taken as outgoing for simplicity. In the following we assume all external particles are massless, but our statements can be easily generalized to the massive case by performing a massless decomposition of the massive external momenta into a sum of massless vectors. For each external particle p_i we have the associated spinors $|p_i\rangle$ and $|p_i]$, also denoted by $|i\rangle$ and $|i]$ for simplicity. Scattering amplitudes are rational functions of the spinor components, as apparent from well known relations valid for external momenta

$$p^\mu = \frac{1}{2} \langle p | \sigma^\mu | p \rangle, \tag{4.3}$$

and polarization vectors

$$\epsilon_+^\mu(p, \eta) = \frac{\langle \eta | \sigma^\mu | p \rangle}{\sqrt{2} \langle \eta p \rangle}, \quad \epsilon_-^\mu(p, \eta) = \frac{\langle p | \sigma^\mu | \eta \rangle}{\sqrt{2} [p \eta]}, \tag{4.4}$$

where η is a reference vector such that the scalar product $(p \cdot \eta) \neq 0$.

The total number of these spinor components is however much larger than the number of independent variables needed to describe an amplitude. We therefore find much more convenient to adopt the strategy described in references [37, 56, 57], which consists in describing scattering amplitudes with a minimal number of variables, which in turn provide a rational parametrization of the components of the spinors.

More in detail, we recall that, under a redefinition of the spinor phases given by the little group transformation

$$|i\rangle \rightarrow t_i |i\rangle, \quad |i] \rightarrow \frac{1}{t_i} |i], \tag{4.5}$$

an n -point amplitude $\mathcal{A}(1, \dots, n)$ transforms as

$$\mathcal{A}(1, \dots, n) \rightarrow \left(\prod_{i=1}^n t_i^{-2h_i} \right) \mathcal{A}(1, \dots, n), \tag{4.6}$$

where h_i is the helicity of the i -th particle (e.g. $\pm 1/2$ for fermions and ± 1 for gluons). It is thus convenient to extract from the amplitude an overall factor $\mathcal{A}^{(\text{phase})}(1, \dots, n)$ which, under eq. (4.5), has the same transformation properties as the amplitude \mathcal{A} . While there is obviously no unique choice for $\mathcal{A}^{(\text{phase})}$, it is clear that it can be chosen based on the external helicities only and does not require any other knowledge about the amplitude \mathcal{A}

itself (see e.g. ref. [57] for a choice valid for any number of external legs, or section 5.1 for a five-point example).

The ratio $\mathcal{A}/\mathcal{A}^{(\text{phase})}$ is obviously phase-free, i.e. invariant under the transformation in eq. (4.5). Any phase-free function of the (four-dimensional) kinematic components which is invariant under the Poincaré group can depend on $3n - 10$ independent invariants. Depending on the problem at hand, these may be chosen as a subset of the Mandelstam invariants $s_{ij} = (p_i + p_j)^2$, however these are not suited for our purposes since they don't provide a rational parametrization of the spinor components. It is however not hard to build, for an arbitrary process, a set of $3n - 10$ invariants which, up to a Poincaré and little group transformation, yield a rational parametrization of the spinor components. A particularly convenient one, based on earlier work by Hodges on momentum twistors [56], is given in references [37, 57] (we will also give some examples later, when discussing explicit applications). Hence, in the following we denote by x_1, \dots, x_{3n-10} a complete set of invariants which provide a rational parametrization of the spinor components (up to a Poincaré and little group transformation). We refer to these as *momentum-twistor variables*. Twistor variables can be interpreted as a rational parametrization of the phase space and, despite having been introduced in the context of conformal theories, they can be used in any relativistic quantum field theory and indeed they played an important role in multi-leg higher-loop calculations in non-supersymmetric gauge theories presented recent years [37, 47, 48]. Since they can give a complete description of the ratio $\mathcal{A}/\mathcal{A}^{(\text{phase})}$, a generic scattering amplitude can be rewritten in terms of them as

$$\mathcal{A}(1, \dots, n) = \mathcal{A}^{(\text{phase})}(1, \dots, n) \tilde{\mathcal{A}}(x_1, \dots, x_{3n-10}). \tag{4.7}$$

Because $\tilde{\mathcal{A}}$ is a rational function of the momentum-twistor variables, it is suited to being evaluated over finite fields \mathbb{Z}_p , and therefore to the application of the functional reconstruction algorithms described earlier in this paper.

When dealing with helicity amplitudes, our starting point is therefore the definition of the spinor components in terms of momentum-twistor variables. In practice we observe that we can always choose our variables such that all but one, say x_1 , are dimensionless. For the purpose of the functional reconstruction, it is thus convenient to set $x_1 = 1$ and recover its functional dependence at the end via dimensional analysis. Hence our chosen set of independent variables is

$$\mathbf{z} = (x_2, \dots, x_{3n-10}). \tag{4.8}$$

If internal or external massive particles are involved, we should also add their masses to the list of the independent variables. From the components of the spinors, we then build the external momenta p^μ . Explicit analytic formulas for these in terms of the corresponding spinors can be easily worked out once and for all using eq. (4.3). In particular we represent any (massless or massive) momentum p by its light-cone components

$$p \equiv (p^0 + p^3, p^0 - p^3, p^1 + i p^2, p^1 - i p^2), \tag{4.9}$$

where, with respect to their usual definition, we dropped a factor $1/\sqrt{2}$, which being irrational cannot be translated into \mathbb{Z}_p .⁵ With this representation, the light-cone components of p are also rational functions of the momentum-twistor variables (factors i cancel out against those in the Pauli matrices). Similarly, we compute polarization vectors according to eq. (4.4), except that we divide them by an extra $\sqrt{2}$ (more details are given in the next paragraph). The resulting expression, in light-cone components, is also a rational function of the momentum-twistor variables.

A minor issue arises from overall factors i and $\sqrt{2}$ present in the definition of polarization vectors and in the color-ordered Feynman rules of gauge theories. These factors are however completely spurious, since $\tilde{\mathcal{A}}(x_1, \dots, x_{3n-10})$ is always a rational function of the momentum-twistor variables, provided that a factor i is also extracted in $\mathcal{A}^{(\text{phase})}$. Therefore, if ϵ^μ is an external polarization vector, V_n is a generic n -point vertex, and P is a propagator, with colour matrices normalized as $\text{tr}[T^a, T^b] = \delta^{ab}$, we get rid of these unwanted factors by making the following substitutions

$$\epsilon^\mu \rightarrow \frac{1}{\sqrt{2}} \epsilon^\mu, \quad V_n \rightarrow (-i 2\sqrt{2})^{4-n} V_n, \quad P \rightarrow iP. \quad (4.10)$$

Notice that the rescaling of V_n is consistent when combining different kinds of vertexes (e.g. $V_3 \times V_3$ scales in the same way as V_4). The correct overall factor, which is a rational number, is restored at the end with a power counting on the number of external gluons, internal vertexes and propagators.

The ingredients outlined so far are sufficient for expressing a tree-level amplitude, as well as the coefficients of a multi-loop amplitude written as a linear combination of loop integrals, as a function of the momentum-twistor variables which can be evaluated, either on \mathbb{Q} or on finite fields \mathbb{Z}_p , via a sequence of elementary rational operations. The application of functional reconstruction algorithms based on finite-field evaluations is thus possible, and it can be an efficient way of computing scattering amplitudes in gauge theories. In the following we give more details about how to combine these ingredients for tree-level calculations via recursion relations, and for multi-loop calculations via integrand reduction and generalized unitarity.

Similar concepts apply to the spinor-helicity formalism in higher numbers of dimensions which, as stated, can be useful in order to provide a higher-dimensional embedding of loop momenta. Details on the six-dimensional spinor-helicity formalism are given in appendix B.

4.2 Berends-Giele recursion

Berends-Giele recursion [50] is an efficient method for the numerical calculation of tree-level amplitudes. Even though it is typically used with floating-point arithmetics, it can similarly be applied with trivial modifications to finite fields \mathbb{Z}_p .

It is straightforward to apply the concepts outlined in section 4.1 for the four-dimensional case and in appendix B for the six-dimensional one, to Berends-Giele recursion.

⁵In principle square roots and factors i may also be included by considering more general finite fields, but since these are always overall factors which cancel out in the final result for $\tilde{\mathcal{A}}$, we do not need to introduce this additional complication into the algorithm.

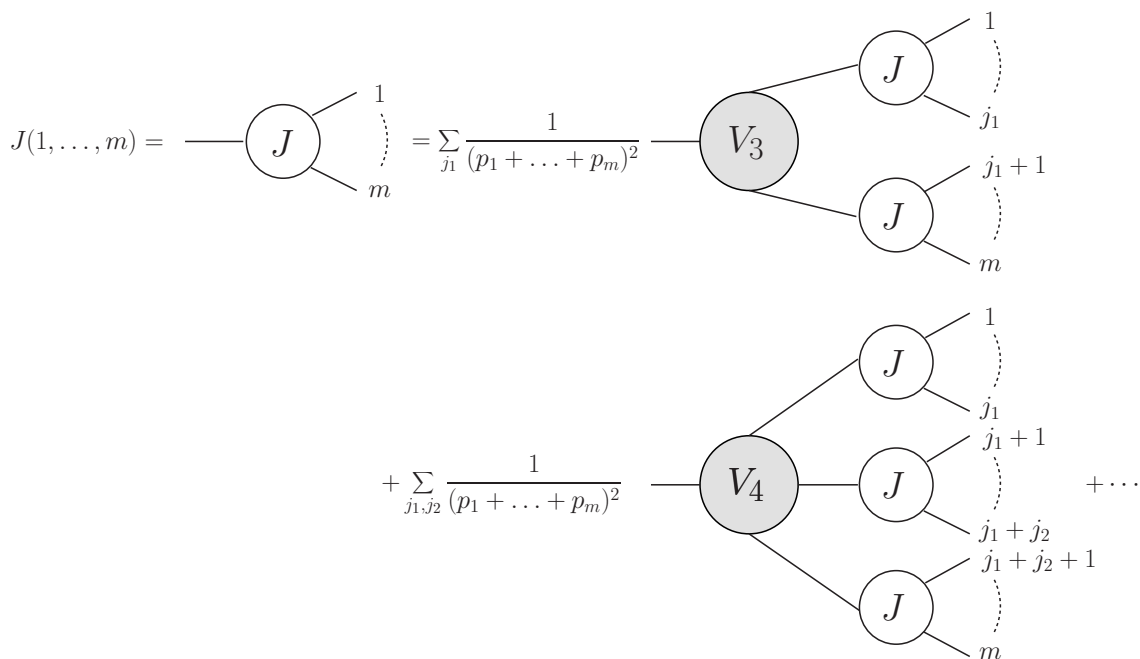


Figure 1. Schematic depiction of Berends-Giele recursion relations. The grey blobs V_k represent contractions defined by the k -point Feynman rules of the theory.

One can start defining one-point on-shell currents, which we symbolically denote by

$$J(i) = \text{---} \bigcirc \text{---} \quad i \quad , \tag{4.11}$$

as external polarization vectors ($J(i) \equiv \epsilon_{h_i}^\mu(p_i)$), spinors ($J(i) \equiv |i\rangle, |i]\rangle$) or constants ($J(i) \equiv 1$) depending on the kind of particle involved, i.e. vector bosons, fermions, and scalars respectively. Then, higher-point off-shell currents are computed by contracting two or more lower-point currents with appropriate spinor and tensor structures which can be easily worked out from the Feynman rules of the theory. For color-ordered amplitudes, the recursion for the calculation of an off-shell current $J(1, \dots, m)$ is depicted in figure 1. In the last step of the recursion for the calculation of an n -point amplitude, an on-shell current $J(1, \dots, n - 1)$ is computed via a similar recursion relation as the one in figure 1, except that there is no multiplication by the propagator factor, and then it is contracted with the on-shell one-point current defined by the n -th external leg.

Besides its efficiency, largely due to the fact that lower-point currents can be re-used for the calculation of several higher-point currents, Berends-Giele recursion also has the advantage that it can be straightforwardly implemented for any theory and that the algorithm is largely independent of the number of space-time dimensions. For these reasons Berends-Giele currents can also be efficiently used as building blocks for multi-loop integrands, as we will describe in the next section and in appendix C.

5 Multi-loop integrand reduction and generalized unitarity

In this section we discuss a finite-field implementation of a specific algorithm for the calculation of scattering amplitudes in gauge theories, namely integrand reduction via d -dimensional generalized unitarity. The algorithm uses as building blocks tree-level amplitudes and Berends-Giele currents, which in turn can be computed using the spinor-helicity formalism as discussed in section 4.

We consider ℓ -loop contributions to n -point scattering amplitudes in dimensional regularization. In particular we choose a regularization scheme such that [58]

- external momenta and polarizations are in four dimensions
- loop momenta are in d dimensions, with $d > 4$
- internal gluon states live in a d_s -dimensional space, with $d_s > d$.

The popular t'Hooft-Veltman [59] and Four-Dimensional-Helicity [58] schemes can be obtained as special cases, by setting $d_s = d$ and $d_s = 4$ respectively at the end of the calculation.

A generic contribution to a loop amplitude takes the form

$$\int_{-\infty}^{\infty} \left(\prod_{i=1}^{\ell} d^d k_i \right) \frac{\mathcal{N}(k_i)}{\prod_j D_j(k_i)}, \quad (5.1)$$

where \mathcal{N} and D are polynomials in the components of the loop momenta k_i (a rational dependence on the external kinematic variables is always understood). In particular, the denominators D_i correspond to loop propagators and have the generic quadratic form

$$D_i = \ell_i^2 - m_i^2, \quad l_i^\mu = \sum_{j=1}^{\ell} \alpha_{ij} k_j^\mu + \sum_{j=1}^n \beta_{ij} p_j^\mu \quad (\alpha_{ij}, \beta_{ij} \in \{0, \pm 1\}). \quad (5.2)$$

It is often useful to split the loop momenta k_i in a four-dimensional part $k_i^{[4]}$ and a $(d-4)$ -dimensional part $k_i^{[d-4]}$ as

$$k_i^\mu = k_i^{[4]\mu} + k_i^{[d-4]\mu}, \quad (5.3)$$

and define extra-dimensional scalar products

$$\mu_{ij} = - \left(k_i^{[d-4]} \cdot k_j^{[d-4]} \right). \quad (5.4)$$

In the regularization scheme defined above, a loop integrand can only depend on the additional $(d-4)$ -dimensional components of the loop momenta through the scalar products μ_{ij} defined above. In particular, for one-loop amplitudes we only have one extra-dimensional scalar product, i.e. μ_{11} , while at two loops we have three of them, namely μ_{11} , μ_{22} and μ_{12} . The dependence on these extra-dimensional components can be implemented by embedding the loop momenta in a \mathcal{D} -dimensional space, where \mathcal{D} is a sufficiently large integer. In particular, the choice $\mathcal{D} = 6$ is sufficient for both one- and two-loop applications. An advantage

of this higher-dimensional embedding, especially in the context of generalized unitarity, is that one can use a higher-dimensional spinor-helicity formalism for the calculation of the integrands, having thus an explicit representation of both external and internal states. This strategy has been used e.g. in references [37, 48]. Having an explicit representation of loop momenta and spinors is also obviously advantageous in numerical calculations, including computations on finite fields \mathbb{Z}_p , which make this formalism suited for our goals.

Integrand reduction methods rewrite loop integrands of the form of eq. (5.1) as a sum of irreducible contributions,

$$\frac{\mathcal{N}(k_i)}{\prod_j D_j(k_i)} = \sum_T \frac{\Delta_T(k_i)}{\prod_{j \in T} D_j(k_i)}, \tag{5.5}$$

where the sum on the r.h.s. runs over the sub-topologies T of the parent topology identified by the set of denominators D_j . The so-called on-shell numerators Δ_T , also known in this context as residues, are linear combinations of basis elements $\{\mathbf{m}_T^\alpha\}$, namely

$$\Delta_T(k_i) = \sum_\alpha c_{T,\alpha} (\mathbf{m}_T(k_i))^\alpha, \tag{5.6}$$

where α runs over an appropriate topology-dependent set of multi-indexes and \mathbf{m}_T represents a sequence of polynomials in the loop momenta. The coefficients $c_{T,\alpha}$, which only depend on the external kinematics, can be obtained by evaluating the integrand on values of the loop momenta satisfying the so-called multiple-cut conditions $\{D_j = 0\}_{j \in T}$. This corresponds to put on-shell a subset of the loop momenta. In particular, by evaluating the integrand on several solutions of the cut constraints, one obtains a linear system of equations for the coefficients $c_{T,\alpha}$ which can be solved e.g. via Gaussian elimination.

In general, there is no unique choice for the set of basis elements $\{\mathbf{m}_T^\alpha\}$, which are only constrained by the requirements of being independent of the aforementioned cut conditions and forming a complete integrand basis compatible with the rank of numerator with respect to the loop momenta, up to terms proportional to the denominators D_i of the topology T . Techniques for choosing an appropriate integrand basis have been proposed elsewhere [29–31, 48] and their discussion is outside the purposes of this paper. Here we will limit ourself to specify case-by-case the integrand basis we used in each example.

As stated, in the calculations presented in this paper, we embed the loop momenta in a \mathcal{D} -dimensional space (with $\mathcal{D} = 6$). Each loop momentum k_i is thus decomposed into a basis $\{e_{ij}\}_{j=1}^{\mathcal{D}}$,

$$k_i^\mu = \sum_{j=1}^{\mathcal{D}} y_{ij} e_{ij}^\mu. \tag{5.7}$$

For each cut, we identify a set of independent free parameters $\{\tau_k\}$ which describe the set of solutions. In particular we look for a set of variables such that the coefficients $y_{ij} = y_{ij}(\tau_k)$ of the linear combination are rational functions of the parameters τ_k , which is particularly convenient when working with finite fields.⁶ We point out that this has been shown to be

⁶Irrational solutions might also be accommodated by considering fields that are more general than \mathbb{Z}_p , at the price of making some intermediate steps of the calculation more involved.

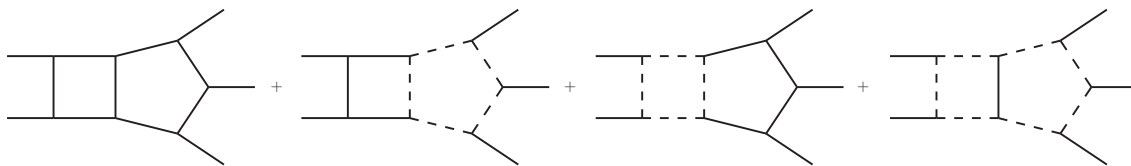


Figure 2. Sum of diagrams with gluon (solid lines) and scalar (dashed lines) loops, for the penta-box topology. Each scalar loop should be multiplied by the number of scalar flavours, which in our case is equal to $d_s - \mathcal{D}$.

possible in many two-loop examples in d dimensions [37, 38, 48], while in general it is not in the four-dimensional limit. Using \mathcal{D} -dimensional cut loop momenta defined by assigning numerical values to the free parameters τ_k , we thus evaluate both the integrand the on-shell basis \mathbf{m}_T^α , generating the linear system of equations we can solve for the coefficients $c_{T,\alpha}$ in eq. (5.6).

The evaluation of the integrand on multiple cuts might be done by means of its analytic expression, if available (e.g. via diagrammatic techniques). The method of generalized unitarity instead exploits the fact that a multi-loop integrand, when a subset of the loop momenta are put on-shell by the cut constraints, factorizes as a product of tree-level amplitudes. This factorization can be understood by the fact that the numerator of an on-shell propagator factorizes as a sum of polarization states. Using generalized unitarity, the amplitude is expressed in terms of a smaller number of contributions compared with diagrammatic techniques, and gauge invariance is guaranteed in intermediate steps of the calculation, thus avoiding cancellations of gauge-dependent terms between different diagrams. Moreover, it offers the possibility of exploiting efficient tree-level techniques for loop calculations as well. When using Berends-Giele currents, additional simplifications are possible during the evaluation of the cut integrands, along the lines of what is implemented in the one-loop public code NJET [18], whose two-loop extension is briefly discussed in appendix C.

Since we embed loop momenta in $\mathcal{D} = 6$ dimensions, we can evaluate the amplitudes relevant for each cut using the six-dimensional spinor-helicity techniques presented in ref. [51] and whose application to finite-field calculations is discussed in appendix B. We also add to the theory $d_s - \mathcal{D}$ flavours of scalars representing additional polarizations of the internal gluons which, as stated at the beginning of this section, are taken to be d_s -dimensional. In particular, each on-shell integrand at two-loops can be written as

$$\Delta_T = \Delta_T^{(\mathcal{D},0)} + (d_s - \mathcal{D}) \Delta_T^{(\mathcal{D},1)} + (d_s - \mathcal{D})^2 \Delta_T^{(\mathcal{D},2)}, \tag{5.8}$$

where $\Delta_T^{(\mathcal{D},i)}$ is a \mathcal{D} -dimensional integrand with i scalar loops. Notice that the result for Δ_T does not depend on the dimension \mathcal{D} of the chosen embedding, unlike each of the terms on the r.h.s. of the previous equation. Moreover, for pure Yang-Mills theories, the contribution $\Delta_T^{(\mathcal{D},2)}$ is non-vanishing only for one-loop squared topologies. As an example, the two-loop planar penta-box is obtained as the sum of contributions in figure 2. This is the same set-up used for the analytic calculations presented in references [37, 47, 48].

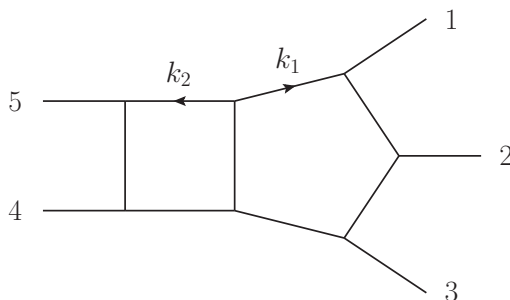


Figure 3. The two-loop planar penta-box topology.

In the following examples, we apply the functional reconstruction algorithm to the coefficients $c_{T,\alpha}^{(i)}$, with $i = 0, 1, 2$, defined by the expansion of each coefficient $c_{T,\alpha}$ of eq. (5.6) in powers of $d_s - 2$,

$$c_{T,\alpha} = c_{T,\alpha}^{(0)} + (d_s - 2) c_{T,\alpha}^{(1)} + (d_s - 2)^2 c_{T,\alpha}^{(2)}. \tag{5.9}$$

Using $d_s - 2$, rather than just d_s , is motivated by the fact that $d_s - 2$ is the number of internal gluon polarizations, and indeed this choice makes the coefficients $c_{T,\alpha}^{(i)}$ significantly simpler. In our implementation, because a numerical evaluation consists in solving for all the coefficients $c_{T,\alpha}^{(i)}$ at the same time, we cache their results and reuse them across the functional reconstruction of all the coefficients. Because the examples below involve genuine two-loop topologies (as opposed to one-loop squared topologies), in these cases we always have $c_{T,\alpha}^{(2)} = 0$. All the analytic results are publicly available, and should be useful for comparisons with future calculations.⁷

5.1 Two-loop five-point planar penta-box

We consider the two-loop penta-box topology, where the five external gluons are taken as outgoing. The loop momenta k_1 and k_2 are defined as in figure 3.

As stated, the kinematics is defined by the spinor components, which in turn are parametrized by momentum-twistor variables. For the five-point case we use the parametrization given in [37], namely

$$\begin{aligned}
 |1\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & [1] &= \begin{pmatrix} 1 \\ \frac{x_4 - x_5}{x_4} \end{pmatrix}, \\
 |2\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}, & [2] &= \begin{pmatrix} 0 \\ x_1 \end{pmatrix}, \\
 |3\rangle &= \begin{pmatrix} \frac{1}{x_1} \\ 1 \end{pmatrix}, & [3] &= \begin{pmatrix} x_1 x_4 \\ -x_1 \end{pmatrix}, \\
 |4\rangle &= \begin{pmatrix} \frac{1}{x_1} + \frac{1}{x_1 x_2} \\ 1 \end{pmatrix}, & [4] &= \begin{pmatrix} x_1(x_2 x_3 - x_3 x_4 - x_4) \\ -\frac{x_1 x_2 x_3 x_5}{x_4} \end{pmatrix}, \\
 |5\rangle &= \begin{pmatrix} \frac{1}{x_1} + \frac{1}{x_1 x_2} + \frac{1}{x_1 x_2 x_3} \\ 1 \end{pmatrix}, & [5] &= \begin{pmatrix} x_1 x_3(x_4 - x_2) \\ \frac{x_1 x_2 x_3 x_5}{x_4} \end{pmatrix}.
 \end{aligned} \tag{5.10}$$

⁷They can be obtained at the url <https://bitbucket.org/peraro/ff2lexamples>.

The variables x_i can be expressed in terms of the Mandelstam invariants s_{ij} and $\text{tr}_5(1234)$ using eq. (A.8) of ref. [37]. In particular, all the momentum-twistor variables but x_1 (we recall that $x_1 = s_{12}$) are dimensionless. This means that the dependence of the result on x_1 can be fixed by dimensional analysis and the functional reconstruction will thus use the following set of variables

$$\mathbf{z} = (x_2, x_3, x_4, x_5). \tag{5.11}$$

As discussed in section 4.1, we factor out of the result a global helicity-dependent phase $\mathcal{A}^{(\text{phase})}$. In this example we define it as

$$\begin{aligned} \mathcal{A}^{(\text{phase})}(1^+, 2^+, 3^+, 4^+, 5^+) &= i \frac{s_{12}^6}{\langle 12 \rangle \langle 23 \rangle \langle 34 \rangle \langle 45 \rangle \langle 51 \rangle} \\ \mathcal{A}^{(\text{phase})}(1^-, 2^+, 3^+, 4^+, 5^+) &= i \frac{(\langle 12 \rangle [23] \langle 31 \rangle)^2 s_{12}^3}{\langle 12 \rangle \langle 23 \rangle \langle 34 \rangle \langle 45 \rangle \langle 51 \rangle} \\ \mathcal{A}^{(\text{phase})}(1^-, 2^-, 3^+, 4^+, 5^+) &= i \frac{\langle 12 \rangle^4 s_{12}^4}{\langle 12 \rangle \langle 23 \rangle \langle 34 \rangle \langle 45 \rangle \langle 51 \rangle}, \\ \mathcal{A}^{(\text{phase})}(1^-, 2^+, 3^-, 4^+, 5^+) &= i \frac{\langle 13 \rangle^4 s_{12}^4}{\langle 12 \rangle \langle 23 \rangle \langle 34 \rangle \langle 45 \rangle \langle 51 \rangle}, \end{aligned} \tag{5.12}$$

and similar for all the ones obtained by cyclic permutations of the external legs. Helicity configurations with three or more negative helicities are not listed, since they can be obtained from the ones above and their cyclic permutations by charge conjugation.

Notice that charge conjugation corresponds to swapping the spinors $|i\rangle \leftrightarrow |i]$ both in the definition of $\mathcal{A}^{(\text{phase})}$ and in their parametrization in terms of momentum-twistor variables. In this case the definition of the x_i is modified by applying the same transformation, which amounts to the substitution $\text{tr}_5 \rightarrow -\text{tr}_5$ in eq. (A.8) of ref. [37].

Given the symmetries of the penta-box diagram, all the helicity configurations can be obtained from the following set

$$\begin{aligned} &\{(1^+, 2^+, 3^+, 4^+, 5^+), \\ &(1^-, 2^+, 3^+, 4^+, 5^+), (1^+, 2^-, 3^+, 4^+, 5^+), (1^+, 2^+, 3^+, 4^-, 5^+), \\ &(1^-, 2^-, 3^+, 4^+, 5^+), (1^-, 2^+, 3^+, 4^+, 5^-), (1^+, 2^+, 3^+, 4^-, 5^-), \\ &(1^-, 2^+, 3^-, 4^+, 5^+), (1^+, 2^-, 3^+, 4^-, 5^+), (1^-, 2^+, 3^+, 4^-, 5^+)\}. \end{aligned} \tag{5.13}$$

For each of these, the on-shell integrand is parametrized as

$$\Delta_{\text{pb}} = \mathcal{A}^{(\text{phase})} \sum_{\alpha} (s_{12})^{-|\alpha|} c_{\text{pb},\alpha} \mathbf{m}_{\text{pb}}^{\alpha}, \tag{5.14}$$

where

$$c_{\text{pb},\alpha} = c_{\text{pb},\alpha}^{(0)}(x_2, x_3, x_4, x_5) + (d_s - 2) c_{\text{pb},\alpha}^{(1)}(x_2, x_3, x_4, x_5), \tag{5.15}$$

and the $c_{\text{pb},\alpha}^{(i)}$ are computed via the functional reconstruction method previously illustrated in this paper. The integrand basis is chosen as in ref. [37], namely

$$\mathbf{m}_{\text{pb}} = (2(k_1 \cdot p_5), 2(k_2 \cdot p_2), 2(k_2 \cdot p_1), \mu_{11}, \mu_{12}, \mu_{22}), \tag{5.16}$$

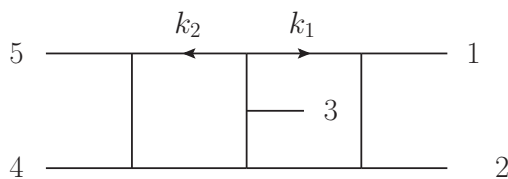


Figure 4. The two-loop non-planar double-pentagon topology.

with

$$\begin{aligned}
 \alpha \in \{ & (0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 1), (0, 0, 0, 0, 1, 0), (0, 0, 0, 1, 0, 0), (0, 0, 1, 0, 0, 0), \\
 & (0, 1, 0, 0, 0, 0), (1, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 2), (0, 0, 0, 0, 1, 1), (0, 0, 0, 1, 0, 1), \\
 & (0, 0, 1, 0, 0, 1), (0, 1, 0, 0, 0, 1), (1, 0, 0, 0, 0, 1), (0, 0, 0, 0, 2, 0), (0, 0, 0, 1, 1, 0), \\
 & (0, 0, 1, 0, 1, 0), (0, 1, 0, 0, 1, 0), (1, 0, 0, 0, 1, 0), (0, 0, 0, 2, 0, 0), (0, 0, 1, 1, 0, 0), \\
 & (0, 1, 0, 1, 0, 0), (1, 0, 0, 1, 0, 0), (0, 0, 0, 1, 0, 2), (1, 0, 0, 0, 0, 2), (0, 0, 0, 0, 2, 1), \\
 & (0, 0, 0, 1, 1, 1), (0, 0, 1, 0, 1, 1), (0, 1, 0, 0, 1, 1), (1, 0, 0, 0, 1, 1), (0, 0, 0, 2, 0, 1), \\
 & (0, 0, 1, 1, 0, 1), (0, 1, 0, 1, 0, 1), (1, 0, 0, 1, 0, 1), (0, 0, 2, 0, 0, 1), (0, 1, 1, 0, 0, 1), \\
 & (0, 0, 0, 0, 3, 0), (0, 0, 0, 1, 2, 0), (0, 0, 1, 0, 2, 0), (0, 1, 0, 0, 2, 0), (1, 0, 0, 0, 2, 0), \\
 & (0, 0, 0, 2, 1, 0), (0, 0, 1, 1, 1, 0), (0, 1, 0, 1, 1, 0), (1, 0, 0, 1, 1, 0), (0, 0, 2, 0, 1, 0), \\
 & (0, 1, 1, 0, 1, 0), (0, 0, 0, 3, 0, 0), (0, 0, 1, 2, 0, 0), (0, 1, 0, 2, 0, 0), (1, 0, 0, 2, 0, 0), \\
 & (0, 0, 2, 1, 0, 0), (0, 1, 1, 1, 0, 0), (1, 0, 0, 1, 0, 2), (1, 0, 0, 0, 2, 1), (0, 0, 1, 1, 1, 1), \\
 & (0, 1, 0, 1, 1, 1), (1, 0, 0, 1, 1, 1), (0, 0, 1, 2, 0, 1), (0, 1, 0, 2, 0, 1), (1, 0, 0, 2, 0, 1), \\
 & (0, 0, 1, 0, 3, 0), (0, 1, 0, 0, 3, 0), (1, 0, 0, 0, 3, 0), (0, 0, 1, 1, 2, 0), (0, 1, 0, 1, 2, 0), \\
 & (1, 0, 0, 1, 2, 0), (0, 0, 2, 0, 2, 0), (0, 1, 1, 0, 2, 0), (0, 0, 1, 2, 1, 0), (0, 1, 0, 2, 1, 0), \\
 & (1, 0, 0, 2, 1, 0), (0, 0, 3, 0, 1, 0), (0, 1, 2, 0, 1, 0), (0, 0, 1, 3, 0, 0), (0, 1, 0, 3, 0, 0), \\
 & (0, 0, 3, 1, 0, 0), (0, 1, 2, 1, 0, 0), (0, 0, 4, 1, 0, 0), (0, 1, 3, 1, 0, 0) \}. \tag{5.17}
 \end{aligned}$$

This basis is smooth in the four-dimensional limit $\mu_{ij} \rightarrow 0$.

The results we obtain for the all-plus topology agree with those computed in references [37, 48]. The on-shell integrands for the other helicity configurations are new. The results are summarized in table 1.

5.2 Two-loop five-point non-planar double-pentagon

We now consider the two-loop non-planar double-pentagon topology depicted in figure 4. The set-up of the calculation is completely analogous to the one for the penta-box. The kinematics and the global spinor phase are still defined using eq. (5.10) and eq. (5.12) respectively.

For this topology, we consider the following complete set of independent helicity configurations

$$\begin{aligned}
 & \{(1^+, 2^+, 3^+, 4^+, 5^+), \\
 & (1^-, 2^+, 3^+, 4^+, 5^+), (1^+, 2^+, 3^-, 4^+, 5^+),
 \end{aligned}$$

Helicity	Non-vanishing coeff.	Max. terms	Max. degree	Avg. non-zero terms
$(1^+, 2^+, 3^+, 4^+, 5^+)$	14	19	8	15.00
$(1^-, 2^+, 3^+, 4^+, 5^+)$	27	443	19	152.96
$(1^+, 2^-, 3^+, 4^+, 5^+)$	37	1977	24	674.97
$(1^+, 2^+, 3^+, 4^-, 5^+)$	61	474	18	184.05
$(1^-, 2^-, 3^+, 4^+, 5^+)$	35	1511	24	278.77
$(1^-, 2^+, 3^+, 4^+, 5^-)$	79	7027	34	1112.82
$(1^+, 2^+, 3^+, 4^-, 5^-)$	18	19	8	15.00
$(1^-, 2^+, 3^-, 4^+, 5^+)$	41	2412	22	368.41
$(1^+, 2^-, 3^+, 4^-, 5^+)$	85	18960	42	3934.96
$(1^-, 2^+, 3^+, 4^-, 5^+)$	85	10386	37	1803.52

Table 1. Summary of the results for the planar penta-box topology. For each helicity configuration we list the number of non-vanishing coefficients $c_{\text{pb},\alpha}^{(i)}$, the maximum number of terms in a coefficient, the maximum degree (of either the numerator or the denominator) of a coefficient, and the average number of terms in the non-vanishing coefficients. The number of terms in a rational function is defined as the sum of the number of monomials in its numerator and the one in its denominator. Full analytic expressions are available at the url <https://bitbucket.org/peraro/ff2lexamples>.

$$\begin{aligned} & (1^-, 2^-, 3^+, 4^+, 5^+), (1^+, 2^-, 3^-, 4^+, 5^+), \\ & (1^-, 2^+, 3^+, 4^+, 5^-), (1^-, 2^+, 3^+, 4^-, 5^+) \}. \end{aligned} \tag{5.18}$$

For this example we use a slightly different approach for the parametrization of the integrand. Rather than using an integrand basis with a smooth four-dimensional limit $\mu_{ij} \rightarrow 0$, we choose one which is only composed of scalar products between loop momenta and external momenta. While this, in general, might make some of the coefficients more complex and the integration harder, it has the advantage that the result can be directly plugged into available programs for integration by parts and other multi-loop techniques. This is also consistent with the adaptive integrand decomposition recently proposed in ref. [38], where the integrands are always expressed in terms of this kind of scalar product and the four-dimensional components of the loop which are orthogonal to the external legs are integrated out (as this is a five-point topology, it corresponds to a limiting case where no such orthogonal direction is present). More in detail, we use

$$\Delta_{\text{dp}} = \mathcal{A}^{(\text{phase})} \sum_{\alpha} (s_{12})^{-|\alpha|} c_{\text{dp},\alpha} \mathbf{m}_{\text{dp}}^{\alpha}, \tag{5.19}$$

where, as before,

$$c_{\text{dp},\alpha} = c_{\text{dp},\alpha}^{(0)}(x_2, x_3, x_4, x_5) + (d_s - 2) c_{\text{dp},\alpha}^{(1)}(x_2, x_3, x_4, x_5), \tag{5.20}$$

while the integrand basis is now given by

$$\mathbf{m}_{\text{dp}} = (2(k_1 \cdot p_5), 2(k_2 \cdot p_1), 2(k_1 \cdot p_3) - 2(k_2 \cdot p_3)), \tag{5.21}$$

Helicity	Non-vanishing coeff.	Max. terms	Max. degree	Avg. non-zero terms
$(1^+, 2^+, 3^+, 4^+, 5^+)$	104	1937	26	626.39
$(1^-, 2^+, 3^+, 4^+, 5^+)$	104	1449	27	601.43
$(1^+, 2^+, 3^-, 4^+, 5^+)$	104	1554	23	642.90
$(1^-, 2^-, 3^+, 4^+, 5^+)$	99	1751	26	739.05
$(1^+, 2^-, 3^-, 4^+, 5^+)$	104	2524	24	923.71
$(1^-, 2^+, 3^+, 4^+, 5^-)$	104	1838	27	823.00
$(1^-, 2^+, 3^+, 4^-, 5^+)$	104	1307	24	630.48

Table 2. Summary of the results for the non-planar double-pentagon topology. The entries are analogous to those in table 1. As in the previous case, full analytic expressions are available at the url <https://bitbucket.org/peraro/ff2lexamples>.

with

$$\alpha \in \{(0, 0, 0), (0, 0, 1), (0, 0, 2), (0, 0, 3), (0, 0, 4), (0, 0, 5), (0, 1, 0), (0, 1, 1), (0, 1, 2), (0, 1, 3), (0, 1, 4), (0, 2, 0), (0, 2, 1), (0, 2, 2), (0, 2, 3), (0, 3, 0), (0, 3, 1), (0, 3, 2), (0, 4, 0), (0, 4, 1), (0, 5, 0), (1, 0, 0), (1, 0, 1), (1, 0, 2), (1, 0, 3), (1, 0, 4), (1, 1, 0), (1, 1, 1), (1, 1, 2), (1, 1, 3), (1, 1, 4), (1, 2, 0), (1, 2, 1), (1, 2, 2), (1, 2, 3), (1, 3, 0), (1, 3, 1), (1, 3, 2), (1, 4, 0), (1, 4, 1), (1, 5, 0), (2, 0, 0), (2, 0, 1), (2, 0, 2), (2, 0, 3), (2, 1, 0), (2, 1, 1), (2, 1, 2), (2, 1, 3), (2, 2, 0), (2, 2, 1), (2, 2, 2), (2, 2, 3), (2, 3, 0), (2, 3, 1), (2, 3, 2), (2, 4, 0), (2, 4, 1), (2, 5, 0), (3, 0, 0), (3, 0, 1), (3, 0, 2), (3, 1, 0), (3, 1, 1), (3, 1, 2), (3, 2, 0), (3, 2, 1), (3, 2, 2), (3, 3, 0), (3, 3, 1), (3, 4, 0), (4, 0, 0), (4, 0, 1), (4, 1, 0), (4, 1, 1), (4, 2, 0), (4, 2, 1), (4, 3, 0), (5, 0, 0), (5, 1, 0), (5, 2, 0)\}. \tag{5.22}$$

The results we obtain for the all-plus topology are in agreement with those computed in ref. [47], although in that reference they were deduced from the known expressions in the planar case, while here they have been directly computed using integrand-reduction via generalized unitarity on the non-planar topology. The on-shell integrands for the other helicity configurations are computed here for the first time. The results are summarized in table 2.

6 Implementation

In this section we give some detail about our implementation of the functional reconstruction method illustrated in section 3 and its application to the techniques described in sections 4 and 5.

The programming language of our implementation is C++, and in particular the C++-11 standard of the language.⁸

The first design choice regards the definition of the finite fields \mathbb{Z}_p . Our requirement of working with machine-size integers imposes an upper limit $p < P_{\max}$ on the choice of the

⁸We make extensive use of the `std::unique_ptr` and `std::unordered_map` containers, as well as right-value and move semantics, for the implementation of our data structures, in order to optimize performance.

primes p defining the fields. We work with 64-bit unsigned integers, which can take values between $i_{\min} = 0$ and $i_{\max} = 2^{64} - 1$. The most stringent requirement is avoiding integer overflow in multiplication, i.e. the requirement that by multiplying two integers in \mathbb{Z}_p with $p < P_{\max}$, before taking modulo p of the result, we still obtain a value smaller than i_{\max} . This implies we can choose P_{\max} such that $P_{\max}^2 \leq i_{\max}$. In our implementation we find more practical to use a more conservative choice, namely $C P_{\max}^3 < i_{\max}$ with $C = 9$. This allows us to be slightly more relaxed in the implementation and take (sums of) products of three integers in \mathbb{Z}_p before taking modulo p . This approach also drastically reduces the number of $\text{mod } p$ operations which are needed. We hard-code 60 primes satisfying these criteria, in the interval $897\,473 \leq p \leq 978\,947$. With this choice, in most of the examples presented in this paper we only need one or two primes p for the rational reconstruction of the result over \mathbb{Q} (according to the discussion in section 2.3 and appendix A.1).

One important difference between native operations which involve machine integers or floating-points numbers and operations on elements of a finite field \mathbb{Z}_p concerns division. Indeed, as stated, performing a division in \mathbb{Z}_p requires the calculation of a multiplicative inverse (see appendix A.1 for an algorithm). Hence, division in \mathbb{Z}_p is a relatively expensive operation (compared to other native operations) which needs a call to a specific routine, and it is generally a good idea to minimize the number of times it is required. However, because the calculation of a multiplicative inverse is implemented by a routine, catching divisions by zero is very easy. Notice that such cases can be quite common during a functional reconstruction procedure, where many values for the variables \mathbf{z} are probed. This makes detecting (actual or spurious) singularities in the evaluation of a function relatively easy and, as stated, the reconstruction algorithms illustrated above can accommodate this possibility.

We now briefly describe our internal representation of polynomials and rational functions. For univariate Newton polynomials, defined as in eq. (3.5), we store two arrays of integers, namely the coefficients a_i and the values y_i . A completely analogous representation is used for Thiele's rational functions in eq. (3.10). For multivariate Newton polynomials depending on n variables, recursively defined as in eq. (3.12), we store the array of integers y_i and an array of pointers to the coefficients a_i , which are now Newton polynomials in $n - 1$ variables. For our canonical representation of polynomials, given in eq. (2.6), we adopt instead a sparse representation. More in detail, we store an array of pointers to a data structure representing each non-vanishing term. These terms are in turn stored as contiguous chunks of memory containing the coefficient c_α and the exponent α corresponding to each monomial. We also store the total degree $|\alpha|$ for convenience. The internal representation uses the graded lexicographic monomial order (i.e. terms are ordered by their total degree, and terms with the same total degree are ordered lexicographically). We found this representation suited for implementing the basic polynomial operations needed by our reconstruction algorithm, namely polynomial addition, multiplication by linear univariate monomials, and homogenization with respect to one variable. In particular the last two operations take advantage of the possibility of manipulating the entries of the exponents α , which is trivial using our sparse representation. Finally, rational functions are simply stored as a pair of polynomials representing the numerator and the denominator respectively.

As stated, the only technique discussed in this paper which requires multi-precision arithmetic is the rational reconstruction algorithm. In our implementation we use the popular GNU Multiple Precision Arithmetic Library (GMP). The final result over the field \mathbb{Q} is stored using a sparse representation similar to the one we use for \mathbb{Z}_p . For functions over \mathbb{Q} however we do not implement any of the polynomial algorithms mentioned above, but only the routines needed for merging results over several finite fields \mathbb{Z}_{p_i} to obtain a guess in \mathbb{Q} , as discussed in section 2.3 and appendix A.2.

It should be stressed that the results obtained with the illustrated reconstruction algorithm for rational functions is minimal with respect to the total degree of the numerator and the denominator, hence there is no need to import it into an external computer algebra system for polynomial GCD simplification (although it can obviously be used by algebra systems for other kinds of manipulations).

It is worth making a few remarks about the application of the reconstruction algorithm to the examples described in section 5. As stated, the coefficients $c_{T,\alpha}^{(i)}$ to be reconstructed for each topology T are computed by evaluating the loop integrands on the multiple cuts and thus obtaining a linear system of equations. The latter can be solved for them by means of Gauss elimination, for which we have a very straightforward implementation suited for dense systems. In order to minimize the number of integrand evaluations we need, as well as the size of the system of equations, we first evaluate each coefficient for a set of arbitrary values of the variables \mathbf{z} and the primes p defining the field \mathbb{Z}_p . These evaluations are used to collect information about which coefficients are non-vanishing for each helicity configuration. Then, when applying the functional reconstruction algorithm, we restrict the system to the non-vanishing coefficients only. With fewer unknowns, we thus require fewer evaluations of the integrand in order to find a solution. Moreover, as stated, because the solution of the system returns all the non-vanishing coefficients, we cache their value for each \mathbf{z} in a hash table. This allows to quickly look up the values of $c_{T,\alpha}^{(i)}$ if they have already been computed during the reconstruction of other coefficients.

Our current implementation can be considered a proof-of-concept and lacks many optimizations which can be estimated to improve the performance by at least one order of magnitude. However it is already suited for applications to complex problems in high-energy physics such as multi-loop high-multiplicity calculations, where it can easily outperform equivalent fully-analytic approaches. In the examples presented in the previous section, for most of the helicity configurations, the total run time required for obtaining full analytic results for all the coefficients of the on-shell integrands varies between a fraction of a second and about ten minutes, on a single core. For exceptionally complex cases (such as the one involving functions of total degree up to 42), we need to perform the reconstruction over up to three finite fields, for a total run time of about a hundred minutes.

As a final observation, we point out that the purpose of the calculations in section 5 is to illustrate a straightforward application of functional reconstruction algorithms over finite fields to modern techniques for loop calculations. In particular, we have not attempted to find an integrand basis which would yield simple results (such as the local integrands discussed in ref. [48]). We observe however that, using the techniques we described in section 3.5, one can compute the total degree of the numerator and the denominator of the

results using a relatively small number of function evaluations. This can be used in order to estimate the complexity of the total result, when looking for an integrand basis yielding simpler coefficients, without the need of performing a full functional reconstruction.

7 Conclusions and outlook

We illustrated an algorithm for the reconstruction of multivariate polynomials and rational functions from their evaluation over finite fields, which has good scaling with the number of variables and the complexity of the results. We then discussed its application to techniques for the calculation of scattering amplitudes in gauge theories, such as the spinor-helicity formalism, tree-level recursion relations, and multi-loop integrand reduction via generalized unitarity. The algorithm can compute complex analytic results, side-stepping the issue of large intermediate expressions which would arise in equivalent fully-analytic calculations. The only input required by the method is a procedure for the numerical evaluation of the function to be reconstructed over finite fields.

As an application of these techniques, we presented for the first time analytic results for the maximal cut of two five-point two-loop topologies in Yang-Mills theory, for a complete set of independent helicity configurations. The complexity of the reconstructed results highlights the suitability of the method to handle complex problems, in particular high-multiplicity two-loop calculations, which are currently of great relevance for high-energy phenomenology.

In this paper we discussed dense reconstruction algorithms. Given that some of our results contain several thousands of terms, we believe this to be the best choice to handle the general case. However, when the functions contain a relatively small number of terms compared to the one expected from the total degree of the result, dense algorithms can be outperformed by sparse algorithms. A possible approach is the usage of so-called racing algorithms, i.e. running a dense and a sparse algorithm in parallel, sharing the same function evaluations, and terminate the reconstruction when the fastest of the two is successful. If the time spent for the reconstruction is dominated by the numerical evaluation of the function, as in our case, this method can achieve optimal performance in virtually all cases.

One can observe that the simpler the result is, compared to the intermediate expressions appearing in an equivalent analytic calculation, the greater the advantage we can expect from using a black-box reconstruction method rather than following a purely analytic approach. This makes the method ideal in the case where large analytic cancellations yield simple final results. Even for somehow more complex results, such as many of the examples presented in this paper, we found that our functional reconstruction algorithm can easily outperform an equivalent analytic calculation by several orders of magnitude. We observed however that the analytic results presented here might have been significantly simpler using a better integrand basis, along the lines of what was proposed in ref. [48]. In particular, the capability of the proposed reconstruction method of quickly computing the total degree of the result can be exploited in order to estimate its complexity and thus look for a choice of variables or an integrand basis which makes the results simpler. As a future

application, the method can thus be beneficial to the extension of the concepts outlined in ref. [48] to a generic helicity configuration.

Although we focused on integrand reduction via generalized unitarity, we stress that the number of potential applications of the proposed approach is much larger. Indeed, as we observed, any method which can be numerically implemented via a sequence of elementary rational operations is suited for the functional reconstruction algorithm we illustrated. On top of the ones presented here, other possible applications are diagrammatic techniques and IBPs (indeed finite-field applications to the latter have already been proposed in ref. [25]).

The methods proposed in this paper are very general and have a very broad spectrum of potential applications. In particular, as shown by our proof-of-concept implementation, they are suited for computing complex analytic results in combination with state-of-the-art multi-loop techniques.

Acknowledgments

The author is indebted to Simon Badger for innumerable discussions and suggestions, and in particular for pointing out the usefulness of momentum-twistor variables for both analytic and numerical calculations. Thanks also go to Pierpaolo Mastrolia and Giovanni Ossola for feedback on the draft and their ongoing collaboration on integrand-reduction topics. Several analytic results presented in this paper have been numerically checked with a MATHEMATICA code developed by Simon Badger, Christian Brønnum-Hansen, and Francesco Buciumi. This work is supported by a Rutherford Grant ST/M004104/1.

A Basic finite-field algorithms

In this appendix we collect some basic algorithms relevant for the finite-field applications presented in this paper. These algorithms are well known but not broadly used in high-energy physics, hence we briefly describe them here for the convenience of the reader.

A.1 Multiplicative inverse and rational reconstruction

The calculation of a multiplicative inverse in \mathbb{Z}_n and the rational reconstruction algorithm both rely on the *extended Euclidean algorithm*. Given two integers $a, b \in \mathbb{Z}$, the extended Euclidean algorithm yields their greatest common divisor $\gcd(a, b)$ and two integers $s, t \in \mathbb{Z}$ such that

$$a s + b t = \gcd(a, b). \quad (\text{A.1})$$

It is useful to give a brief description of the algorithm, which consists in generating the sequences of integers $\{r_i\}$, $\{s_i\}$, $\{t_i\}$ and the integer quotients $\{q_i\}$ as follows

$$\begin{aligned} r_0 &= a \\ s_0 &= 1 \\ t_0 &= 0 \\ r_1 &= b \end{aligned}$$

$$\begin{aligned}
 s_1 &= 0 \\
 t_1 &= 1 \\
 \dots &= \dots \\
 q_i &= \lfloor r_{i-2}/r_{i-1} \rfloor \\
 r_i &= r_{i-2} - q_i r_{i-1} \\
 s_i &= s_{i-2} - q_i s_{i-1} \\
 t_i &= t_{i-2} - q_i t_{i-1}.
 \end{aligned} \tag{A.2}$$

Notice that, in practical implementations, only the last two entries of each sequence need to be stored. The algorithm terminates at $i = k$ when $r_k = 1$, and identifies $t = t_{k-1}$, $s = s_{k-1}$, and $\gcd(a, b) = r_{k-1}$.

Let us now turn to the calculation of a *multiplicative inverse* in \mathbb{Z}_n . As we mentioned, a non-vanishing integer $a \in \mathbb{Z}_n$ admits a multiplicative inverse if and only if a and n are co-prime, i.e. $\gcd(a, n) = 1$. In this case, inserting $b = n$ in eq. (A.1) and taking $\text{mod } n$ of both sides of the equation gives

$$a s \text{ mod } n = 1, \quad \text{if } \gcd(a, n) = 1, \tag{A.3}$$

which implies that we can identify

$$a^{-1} \text{ mod } n = s, \quad \text{if } \gcd(a, n) = 1. \tag{A.4}$$

Because every non-vanishing element of a field must have a multiplicative inverse, we can consider \mathbb{Z}_n as a field if $n = p$ is prime. Notice that the calculation of the sequence $\{t_i\}$ is not needed for the purposes of computing a multiplicative inverse.

A modified version of the extended Euclidean algorithm can also be used for guessing a rational number q from its image $a \in \mathbb{Z}_n$ [54]. Indeed, it is straightforward to show that each iteration of the Euclidean algorithm satisfies

$$a s_i + b t_i = r_i, \tag{A.5}$$

and therefore, by setting $b = n$ and taking $\text{mod } n$ of both sides,

$$r_i = a s_i \text{ mod } n \iff r_i/s_i \text{ mod } n = a. \tag{A.6}$$

Hence r_i/s_i is a possible guess for q in each iteration of the extended Euclidean algorithm. However one can see that, when n is sufficiently large, most of these guesses will have very large numerators and denominators, except for the case where

$$r_i^2, s_i^2 \lesssim n. \tag{A.7}$$

Hence the rational reconstruction algorithm can be implemented by terminating the Euclidean algorithm at the iteration $i = k$, when $r_k^2 < n$. The calculation of the sequence $\{t_i\}$ is not needed for the rational reconstruction.

This algorithm typically succeeds in reconstructing a rational number $q = a/b \in \mathbb{Q}$ from its image in \mathbb{Z}_n when its numerator and denominator satisfy

$$a^2, b^2 \lesssim n. \tag{A.8}$$

As explained in section 6, our requirement of working with machine-size integers imposes an upper bound on our choices of $n = p$ (in our implementation, $p < P_{\max}$ with $P_{\max} = \mathcal{O}(10^6)$) and thus the former relation is generally not guaranteed to hold. However, as we mentioned in section 2.3, this issue can be solved by performing the functional reconstruction on several finite fields \mathbb{Z}_{p_i} with $p_i < P_{\max}$, and combining them by means of the Chinese remainder theorem, as described below.

A.2 Chinese remainder theorem

The Chinese remainder theorem allows us to reconstruct a number $a \in \mathbb{Z}_n$ with $n = n_1 \cdots n_k$ and the n_i pairwise co-prime, from its images a_i in the sets $\mathbb{Z}_{n_1}, \dots, \mathbb{Z}_{n_k}$. Hence, by performing a calculation on several prime fields \mathbb{Z}_{p_i} , with $p_i < P_{\max}$, and combining them with the Chinese remainder theorem, one can reconstruct the image of the same result in $\mathbb{Z}_{p_1 \cdots p_k}$ and apply the rational reconstruction algorithm described above to the latter.

More in detail, given $a \in \mathbb{Z}_n$, a set of pairwise co-prime numbers n_1, \dots, n_k such that $n = n_1 \cdots n_k$, and a set of congruences

$$a_i = a \pmod{n_i}, \tag{A.9}$$

a can be uniquely determined in \mathbb{Z}_n as

$$a = \sum_i m_i a_i \pmod{n}, \tag{A.10}$$

where

$$m_i \equiv \left(\left(\frac{n}{n_i} \right)^{-1} \pmod{n_i} \right) \frac{n}{n_i}. \tag{A.11}$$

By using several primes p_1, \dots, p_k , one can thus make their product large enough for the reconstruction algorithm (described in the previous section) to succeed in $\mathbb{Z}_{p_1 \cdots p_k}$. In practice, as we explained, we perform the functional reconstruction over one prime field \mathbb{Z}_{p_i} at a time and recursively merge the result on the bigger set $\mathbb{Z}_{p_1 \cdots p_k}$. The recursion terminates when the result of the rational reconstruction is in agreement with the evaluation of the function on different prime fields. Hence, all our calculations can be performed on \mathbb{Z}_{p_i} with primes $p_i < P_{\max}$ using machine-size integers and only the application of the Chinese remainder theorem requires multi-precision arithmetic.

It is worth observing that, when applying the Chinese remainder theorem to two congruences at a time, as in our case, one has $n = n_1 n_2$ and the above formulas reduce to

$$a = (m_1 a_1 + m_2 a_2) \pmod{n_1 n_2}, \tag{A.12}$$

with

$$m_1 = (n_2^{-1} \pmod{n_1}) n_2, \quad m_2 = (n_1^{-1} \pmod{n_2}) n_1. \tag{A.13}$$

The implementation can be further simplified by noting that

$$m_2 = (1 - m_1) \pmod{n_1 n_2}. \tag{A.14}$$

B Six-dimensional momenta and spinors

We now briefly consider a six-dimensional generalization of some of the techniques outlined in section 4.1. For the purposes of this paper, six-dimensional momenta and spinors provide a higher-dimensional embedding of the loop components in one- and two-loop calculations, which is used for the applications of section 5 in the context of multi-loop generalized unitarity.

We use the following representation of a generic six-dimensional momentum p^μ , which is consistent with the light-cone representation given in eq. (4.9) for four-dimensional ones,

$$p \equiv (p^0 + p^3, p^0 - p^3, p^1 + ip^2, p^1 - ip^2, p^4 + ip^5, p^4 - ip^5). \quad (\text{B.1})$$

The six-dimensional spinor-helicity formalism has been extensively developed in ref. [51], to which we refer the reader for details. Here we only recall a few basic concepts useful for our purposes. In six dimensions, one can define four-components spinors $|p^a\rangle$ and $|p_{\dot{a}}]$ satisfying the Dirac equation

$$p^\mu \sigma_\mu^{(6)} |p^a\rangle = p^\mu \tilde{\sigma}_\mu^{(6)} |p_{\dot{a}}] = 0, \quad (\text{B.2})$$

where $\sigma_\mu^{(6)}$ and $\tilde{\sigma}_\mu^{(6)}$, for $\mu = 0, \dots, 5$, are now 4×4 matrices which can be regarded as higher-dimensional versions of the Pauli matrices (an explicit representation can be found in ref. [51]). The indexes $a, \dot{a} = 0, 1$ label the two independent solutions of the Dirac equation, namely solutions with positive and negative helicity for spinors and anti-spinors respectively. A sub-set of the six-dimensional spinor components can be identified with the ones of $|p\rangle$ and $|p]$ in the four-dimensional limit, hence making the conversion from four to six dimensions trivial. For our purposes, the most important relation satisfied by six-dimensional spinors is

$$p_\mu = -\frac{1}{4} \langle p^a | \sigma_\mu^{(6)} | p^b \rangle \epsilon_{ab}, \quad p_\mu = -\frac{1}{4} [p_{\dot{a}} | \sigma_\mu^{(6)} | p_{\dot{b}}] \epsilon^{\dot{a}\dot{b}}, \quad (\text{B.3})$$

where ϵ^{ab} and $\epsilon_{\dot{a}\dot{b}}$ are anti-symmetric Levi-Civita tensors in two dimensions with $\epsilon^{01} = -\epsilon_{01} = 1$, and a sum over repeated indexes is understood. The Levi-Civita tensors are also used for raising and lowering the indexes a and \dot{a} .

Unlike the four-dimensional case, where momenta are built from spinors (which in turn are determined by their representation in terms of momentum-twistor variables), in the six-dimensional one we are mostly interested in the opposite, i.e. building spinors $|p^a\rangle$ and $|p_{\dot{a}}]$ from the entries in eq. (B.1) of a six-dimensional momentum, such that eq.(B.3) is satisfied. This is because, in the context of generalized unitarity, we start from six-dimensional momenta corresponding to on-shell loop propagators, and from these we must build their corresponding spinors. While the solution of the problem is not unique, it is not hard to work out a suitable representation, although one has to separately consider special cases where one or more of the entries are vanishing. It should be pointed out that, once the sum over the internal helicities of a unitarity cut has been performed, spinors corresponding to loop momenta always combine as in the right-hand-sides of eq. (B.3), hence these relations (and the Dirac equation) are the only relevant ones for this purpose.

The last six-dimensional ingredient we need are polarization vectors, for cut loop propagators involving gluons. For six-dimensional gluons we have four independent polarizations, which can be identified by the labels $(a \dot{a})$ of their $SU(2) \times SU(2)$ representation

$$(a \dot{a}) \in \{(00), (11), (01), (10)\} \equiv \{(++), (--), (+-), (-+)\}. \quad (\text{B.4})$$

While $(++)$ and $(--)$ respectively correspond to the positive and negative helicity in the four-dimensional limit, the last two are instead only present in six dimensions. Given an auxiliary vector η^μ , polarization vectors can be written as [51]

$$\epsilon_{a\dot{a}}^\mu(p, \eta) = \frac{1}{\sqrt{2}(p \cdot \eta)} \langle p_a | \sigma^\mu | \eta_b \rangle \langle \eta_c | p_{\dot{a}} \rangle \epsilon^{bc}. \quad (\text{B.5})$$

As in the four-dimensional case, in our representation we divide them by an additional factor $\sqrt{2}$, such that their light-cone components become rational functions of the spinor variables. In ref. [51] it is shown that these vectors obey all the properties required by polarizations of gauge bosons, and the following relation

$$\epsilon_{a\dot{a}}^\mu(p, \eta) \epsilon^{\nu a\dot{a}}(p, \eta) = g^{\mu\nu} - \frac{1}{(p \cdot \eta)} (p^\mu \eta^\nu + p^\nu \eta^\mu). \quad (\text{B.6})$$

The latter is important in the context of generalized unitarity. Indeed, after the sum over the internal helicities, polarization vectors corresponding to cut loop propagators always appear in the combination on the l.h.s. of the equation.

Similarly to the four-dimensional case, the ingredients reviewed here allow us to work with six-dimensional spinors by performing rational operations on their components, hence they are suited for numerical evaluations over finite fields \mathbb{Z}_p and thus for the application of the functional reconstruction algorithms described in section 3.

C Two-loop unitarity cuts from Berends-Giele currents

In this appendix we briefly illustrate how to evaluate two-loop unitarity cuts efficiently using off-shell Berends-Giele currents. This is a straightforward generalization of the algorithm used by the public numerical code NJET [18] at one-loop. The algorithm is suited for numerical implementations, including evaluations over finite fields \mathbb{Z}_p .

For simplicity, we consider a multiple cut of the form depicted in figure 5, for a theory with only gluons, but it should be clear that everything can be easily extended to more general cases. In particular, figure 5 represents a unitarity cut where the momenta of the loop propagators which are put on-shell are denoted by ℓ_i . We split the loop propagators in three categories: $\{\ell_1, \dots, \ell_{j_1}\}$ are propagators depending on the loop momentum k_1 only, $\{\ell_{j_1+1}, \dots, \ell_{j_1+j_2}\}$ depend on k_2 only, and $\{\ell_{j_1+j_1+1}, \dots, \ell_{j_1+j_2+j_{12}}\}$ depend on both k_1 and k_2 . Double lines represent an arbitrary set of external legs. The cut is defined as the product of the tree-level amplitudes (represented as grey blobs in the picture) defined by the on-shell propagators, summed over the helicity states of the internal legs.

We first focus on the tree-level amplitude involving ℓ_1 and ℓ_2 . When this amplitude is computed via Berends-Giele recursion, the last step of such recursion is the contraction

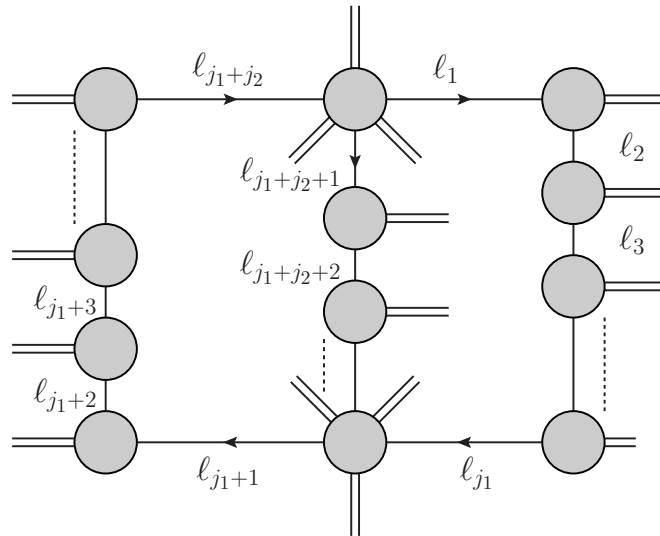


Figure 5. Schematic depiction of a unitarity cut. Grey blobs represent tree-level amplitudes and they are joined by the lines corresponding to the on-shell momenta of the cut loop propagators ℓ_i . The loop momenta are defined as $k_1 = \ell_1$, and $k_2 = -\ell_{j_1+j_2}$. Double lines represent an arbitrary number of external legs.

of the on-shell current involving ℓ_1 and the appropriate sequence of external legs with the polarization vector corresponding to the on-shell propagator ℓ_2 . This is schematically represented by the following equation

$$\mathcal{A}\left(\begin{array}{c} \parallel \\ -\ell_1^{-h_1} \text{---} \text{blob} \text{---} \ell_2^{h_2} \\ \parallel \end{array}\right) = \begin{array}{c} \parallel \\ -\ell_1^{-h_1} \text{---} \text{blob} \text{---} (J) \text{---} \\ \parallel \end{array} \cdot \epsilon_{h_2}(\ell_2). \quad (\text{C.1})$$

In the calculation of the current appearing on the r.h.s. of the previous equation, the lower-point currents depending only on the external legs only need to be computed once per phase-space point, since they are independent of the cut. We now recall that, in the product of amplitudes defined by this unitarity cut, the polarization vector $\epsilon_{h_2}(\ell_2)$ always appears in the combination $\epsilon_{h_2}^\mu(\ell_2)\epsilon_{-h_2}^\nu(-\ell_2)$. After the sum over internal helicities we have

$$\sum_h \text{sign}(h_2) \epsilon_h^\mu(\ell_2) \epsilon_{-h}^\nu(-\ell_2) = g^{\mu\nu} - \frac{1}{(\ell_2 \cdot \eta)} (\ell_2^\mu \eta^\nu + \ell_2^\nu \eta^\mu), \quad (\text{C.2})$$

as apparent e.g. from eq. (B.6). In the previous equation, η is the reference vector used to define the internal polarizations, and the second term on the r.h.s. may be dropped because of gauge invariance. This implies that, instead of computing a product of amplitudes and summing over the internal helicities, one can equivalently simply compute the Berends-Giele current on the r.h.s. of (C.1), without contracting it with the polarization vector $\epsilon_{h_2}(\ell_2)$, and use this current as an input on-shell current for the calculation of the next amplitude (in this case, the one involving ℓ_2 and ℓ_3), as if the current itself was the polarization vector associated to ℓ_2 .

The argument can obviously be iterated over all propagators ℓ_i with $i \leq j_1$, i.e. those which depend on the first loop momentum k_1 only, so that for each helicity h associated with ℓ_1 we build a current $J_{k_1,h}$ as

$$J_{k_1,h} \equiv J_{h,\ell_1 \rightarrow \ell_{j_1}} = \begin{array}{c} \parallel \\ \text{---} \circ \text{---} \ell_2 \text{---} \circ \text{---} \dots \text{---} \circ \text{---} \ell_{j_1} \text{---} \circ \text{---} J \end{array}, \quad (\text{C.3})$$

where each sum over the internal helicities of cut propagators is replaced by a contraction with $g^{\mu\nu}$. As already observed, lower-point currents depending on external legs only need to be computed once per phase-space point and can be reused on different cuts. Similarly, for each helicity h associated with ℓ_{j_1+1} in figure 5 we build a current for the propagators depending on k_2 only, as

$$J_{k_2,h} \equiv J_{h,\ell_{j_1+1} \rightarrow \ell_{j_1+j_2}} = \begin{array}{c} \parallel \\ \text{---} \circ \text{---} \ell_{k_1+2} \text{---} \circ \text{---} \dots \text{---} \circ \text{---} \ell_{j_1+j_2} \text{---} \circ \text{---} J \end{array}. \quad (\text{C.4})$$

At this point, we start building currents depending on both k_1 and k_2 , using $J_{k_1,h}$ and $J_{k_2,h}$ as the input on-shell currents of the recursion. In particular, we define the currents $J_{\text{up},k_1,k_2,h_1,h_2}$ and $J_{\text{down},k_1,k_2,h_1,h_2}$ corresponding to the upper and lower intersection of the two loops in figure 5, namely

$$J_{\text{up},h_1,h_2} \equiv \begin{array}{c} \parallel \\ \text{---} \circ \text{---} \epsilon_{h_1}(\ell_1) \\ \parallel \\ \text{---} \circ \text{---} J \\ \parallel \end{array}, \quad J_{\text{down},h_1,h_2} \equiv \begin{array}{c} \parallel \\ \text{---} \circ \text{---} J \\ \parallel \\ \text{---} \circ \text{---} \epsilon_{h_2}(\ell_{j_1+1}) \\ \parallel \end{array} \cdot J_{k_1,h_1}. \quad (\text{C.5})$$

Using $J_{\text{up},k_1,k_2,h_1,h_2}$ as input, we move down along the third sequence of propagators $\ell_{j_1+j_2+1}, \ell_{j_1+j_2+2}, \dots$ depending on both k_1 and k_2 , by means of the same algorithm which defined the currents $J_{k_1,h}$ and $J_{k_2,h}$,

$$J_{k_1,k_2,h_1,h_2} \equiv J_{\text{up},h_1,h_2} \cdot \begin{array}{c} \parallel \\ \text{---} \circ \text{---} \ell_{j_1+j_2+1} \text{---} \circ \text{---} \dots \text{---} \circ \text{---} \ell_{j_1+j_2+j_{12}} \text{---} \circ \text{---} J \end{array}. \quad (\text{C.6})$$

The value of the cut is thus

$$\sum_{h_1 h_2} \text{sign}(h_1) \text{sign}(h_2) J_{k_1,k_2,h_1,h_2} \cdot J_{\text{down},k_1,k_2,h_1,h_2}. \quad (\text{C.7})$$

Open Access. This article is distributed under the terms of the Creative Commons Attribution License ([CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/)), which permits any use, distribution and reproduction in any medium, provided the original author(s) and source are credited.

References

- [1] T. Hahn and M. Pérez-Victoria, *Automatized one loop calculations in four-dimensions and D-dimensions*, *Comput. Phys. Commun.* **118** (1999) 153 [[hep-ph/9807565](https://arxiv.org/abs/hep-ph/9807565)] [[INSPIRE](https://inspirehep.net/literature/496009)].

- [2] G. Ossola, C.G. Papadopoulos and R. Pittau, *CutTools: a program implementing the OPP reduction method to compute one-loop amplitudes*, *JHEP* **03** (2008) 042 [[arXiv:0711.3596](#)] [[INSPIRE](#)].
- [3] T. Binoth, J.P. Guillet, G. Heinrich, E. Pilon and T. Reiter, *Golem95: a numerical program to calculate one-loop tensor integrals with up to six external legs*, *Comput. Phys. Commun.* **180** (2009) 2317 [[arXiv:0810.0992](#)] [[INSPIRE](#)].
- [4] G. Cullen et al., *Golem95C: a library for one-loop integrals with complex masses*, *Comput. Phys. Commun.* **182** (2011) 2276 [[arXiv:1101.5595](#)] [[INSPIRE](#)].
- [5] J.P. Guillet, G. Heinrich and J.F. von Soden-Fraunhofen, *Tools for NLO automation: extension of the Golem95C integral library*, *Comput. Phys. Commun.* **185** (2014) 1828 [[arXiv:1312.3887](#)] [[INSPIRE](#)].
- [6] P. Mastrolia, G. Ossola, T. Reiter and F. Tramontano, *Scattering amplitudes from unitarity-based reduction algorithm at the integrand-level*, *JHEP* **08** (2010) 080 [[arXiv:1006.0710](#)] [[INSPIRE](#)].
- [7] P. Mastrolia, E. Mirabella and T. Peraro, *Integrand reduction of one-loop scattering amplitudes through Laurent series expansion*, *JHEP* **06** (2012) 095 [*Erratum* *ibid.* **11** (2012) 128] [[arXiv:1203.0291](#)] [[INSPIRE](#)].
- [8] T. Peraro, *Ninja: automated integrand reduction via Laurent expansion for one-loop amplitudes*, *Comput. Phys. Commun.* **185** (2014) 2771 [[arXiv:1403.1229](#)] [[INSPIRE](#)].
- [9] A. Denner, S. Dittmaier and L. Hofer, *Collier: a fortran-based complex one-loop library in extended regularizations*, [arXiv:1604.06792](#) [[INSPIRE](#)].
- [10] A. van Hameren, C.G. Papadopoulos and R. Pittau, *Automated one-loop calculations: a proof of concept*, *JHEP* **09** (2009) 106 [[arXiv:0903.4665](#)] [[INSPIRE](#)].
- [11] G. Bevilacqua et al., *HELAC-NLO*, *Comput. Phys. Commun.* **184** (2013) 986 [[arXiv:1110.1499](#)] [[INSPIRE](#)].
- [12] C.F. Berger et al., *An automated implementation of on-shell methods for one-loop amplitudes*, *Phys. Rev. D* **78** (2008) 036003 [[arXiv:0803.4180](#)] [[INSPIRE](#)].
- [13] V. Hirschi, R. Frederix, S. Frixione, M.V. Garzelli, F. Maltoni and R. Pittau, *Automation of one-loop QCD corrections*, *JHEP* **05** (2011) 044 [[arXiv:1103.0621](#)] [[INSPIRE](#)].
- [14] G. Cullen et al., *Automated one-loop calculations with GoSam*, *Eur. Phys. J. C* **72** (2012) 1889 [[arXiv:1111.2034](#)] [[INSPIRE](#)].
- [15] G. Cullen et al., *GoSAM-2.0: a tool for automated one-loop calculations within the Standard Model and beyond*, *Eur. Phys. J. C* **74** (2014) 3001 [[arXiv:1404.7096](#)] [[INSPIRE](#)].
- [16] F. Cascioli, P. Maierhofer and S. Pozzorini, *Scattering amplitudes with open loops*, *Phys. Rev. Lett.* **108** (2012) 111601 [[arXiv:1111.5206](#)] [[INSPIRE](#)].
- [17] S. Badger, B. Biedermann and P. Uwer, *NGluon: a package to calculate one-loop multi-gluon amplitudes*, *Comput. Phys. Commun.* **182** (2011) 1674 [[arXiv:1011.2900](#)] [[INSPIRE](#)].
- [18] S. Badger, B. Biedermann, P. Uwer and V. Yundin, *Numerical evaluation of virtual corrections to multi-jet production in massless QCD*, *Comput. Phys. Commun.* **184** (2013) 1981 [[arXiv:1209.0100](#)] [[INSPIRE](#)].
- [19] S. Actis, A. Denner, L. Hofer, J.-N. Lang, A. Scharf and S. Uccirati, *RECOLA: REcursive Computation of One-Loop Amplitudes*, [arXiv:1605.01090](#) [[INSPIRE](#)].
- [20] T. Gehrmann, J.M. Henn and N.A. Lo Presti, *Analytic form of the two-loop planar five-gluon all-plus-helicity amplitude in QCD*, *Phys. Rev. Lett.* **116** (2016) 062001 [*Erratum* *ibid.* **116** (2016) 189903] [[arXiv:1511.05409](#)] [[INSPIRE](#)].

- [21] C.G. Papadopoulos, D. Tommasini and C. Wever, *The pentabox master integrals with the simplified differential equations approach*, *JHEP* **04** (2016) 078 [[arXiv:1511.09404](#)] [[INSPIRE](#)].
- [22] J.A.M. Vermaseren, *New features of FORM*, [math-ph/0010025](#) [[INSPIRE](#)].
- [23] R. Britto, F. Cachazo and B. Feng, *Generalized unitarity and one-loop amplitudes in $N = 4$ super-Yang-Mills*, *Nucl. Phys. B* **725** (2005) 275 [[hep-th/0412103](#)] [[INSPIRE](#)].
- [24] W.T. Giele, Z. Kunszt and K. Melnikov, *Full one-loop amplitudes from tree amplitudes*, *JHEP* **04** (2008) 049 [[arXiv:0801.2237](#)] [[INSPIRE](#)].
- [25] A. von Manteuffel and R.M. Schabinger, *A novel approach to integration by parts reduction*, *Phys. Lett. B* **744** (2015) 101 [[arXiv:1406.4513](#)] [[INSPIRE](#)].
- [26] I. Kotsireas, B. Mourrain, V. Pan, A. Cuyt and W. Shin Lee, *Symbolic and numerical algorithms sparse interpolation of multivariate rational functions*, *Theor. Comput. Sci.* **412** (2011) 1445.
- [27] G. Ossola, C.G. Papadopoulos and R. Pittau, *Reducing full one-loop amplitudes to scalar integrals at the integrand level*, *Nucl. Phys. B* **763** (2007) 147 [[hep-ph/0609007](#)] [[INSPIRE](#)].
- [28] P. Mastrolia and G. Ossola, *On the integrand-reduction method for two-loop scattering amplitudes*, *JHEP* **11** (2011) 014 [[arXiv:1107.6041](#)] [[INSPIRE](#)].
- [29] S. Badger, H. Frellesvig and Y. Zhang, *Hepta-cuts of two-loop scattering amplitudes*, *JHEP* **04** (2012) 055 [[arXiv:1202.2019](#)] [[INSPIRE](#)].
- [30] Y. Zhang, *Integrand-level reduction of loop amplitudes by computational algebraic geometry methods*, *JHEP* **09** (2012) 042 [[arXiv:1205.5707](#)] [[INSPIRE](#)].
- [31] P. Mastrolia, E. Mirabella, G. Ossola and T. Peraro, *Scattering amplitudes from multivariate polynomial division*, *Phys. Lett. B* **718** (2012) 173 [[arXiv:1205.7087](#)] [[INSPIRE](#)].
- [32] S. Badger, H. Frellesvig and Y. Zhang, *An integrand reconstruction method for three-loop amplitudes*, *JHEP* **08** (2012) 065 [[arXiv:1207.2976](#)] [[INSPIRE](#)].
- [33] R.H.P. Kleiss, I. Malamos, C.G. Papadopoulos and R. Verheyen, *Counting to one: reducibility of one- and two-loop amplitudes at the integrand level*, *JHEP* **12** (2012) 038 [[arXiv:1206.4180](#)] [[INSPIRE](#)].
- [34] B. Feng and R. Huang, *The classification of two-loop integrand basis in pure four-dimension*, *JHEP* **02** (2013) 117 [[arXiv:1209.3747](#)] [[INSPIRE](#)].
- [35] P. Mastrolia, E. Mirabella, G. Ossola and T. Peraro, *Integrand-reduction for two-loop scattering amplitudes through multivariate polynomial division*, *Phys. Rev. D* **87** (2013) 085026 [[arXiv:1209.4319](#)] [[INSPIRE](#)].
- [36] P. Mastrolia, E. Mirabella, G. Ossola and T. Peraro, *Multiloop integrand reduction for dimensionally regulated amplitudes*, *Phys. Lett. B* **727** (2013) 532 [[arXiv:1307.5832](#)] [[INSPIRE](#)].
- [37] S. Badger, H. Frellesvig and Y. Zhang, *A two-loop five-gluon helicity amplitude in QCD*, *JHEP* **12** (2013) 045 [[arXiv:1310.1051](#)] [[INSPIRE](#)].
- [38] P. Mastrolia, T. Peraro and A. Primo, *Adaptive integrand decomposition in parallel and orthogonal space*, *JHEP* **08** (2016) 164 [[arXiv:1605.03157](#)] [[INSPIRE](#)].
- [39] D.A. Kosower and K.J. Larsen, *Maximal unitarity at two loops*, *Phys. Rev. D* **85** (2012) 045017 [[arXiv:1108.1180](#)] [[INSPIRE](#)].
- [40] K.J. Larsen, *Global poles of the two-loop six-point $N = 4$ SYM integrand*, *Phys. Rev. D* **86** (2012) 085032 [[arXiv:1205.0297](#)] [[INSPIRE](#)].

- [41] S. Caron-Huot and K.J. Larsen, *Uniqueness of two-loop master contours*, *JHEP* **10** (2012) 026 [[arXiv:1205.0801](#)] [[INSPIRE](#)].
- [42] H. Johansson, D.A. Kosower and K.J. Larsen, *Two-loop maximal unitarity with external masses*, *Phys. Rev. D* **87** (2013) 025030 [[arXiv:1208.1754](#)] [[INSPIRE](#)].
- [43] H. Johansson, D.A. Kosower and K.J. Larsen, *Maximal unitarity for the four-mass double box*, *Phys. Rev. D* **89** (2014) 125010 [[arXiv:1308.4632](#)] [[INSPIRE](#)].
- [44] H. Johansson, D.A. Kosower, K.J. Larsen and M. Søgaard, *Cross-order integral relations from maximal cuts*, *Phys. Rev. D* **92** (2015) 025015 [[arXiv:1503.06711](#)] [[INSPIRE](#)].
- [45] H. Ita, *Two-loop integrand decomposition into master integrals and surface terms*, [arXiv:1510.05626](#) [[INSPIRE](#)].
- [46] K.J. Larsen and Y. Zhang, *Integration-by-parts reductions from unitarity cuts and algebraic geometry*, *Phys. Rev. D* **93** (2016) 041701 [[arXiv:1511.01071](#)] [[INSPIRE](#)].
- [47] S. Badger, G. Mogull, A. Ochirov and D. O’Connell, *A complete two-loop, five-gluon helicity amplitude in Yang-Mills theory*, *JHEP* **10** (2015) 064 [[arXiv:1507.08797](#)] [[INSPIRE](#)].
- [48] S. Badger, G. Mogull and T. Peraro, *Local integrands for two-loop all-plus Yang-Mills amplitudes*, *JHEP* **08** (2016) 063 [[arXiv:1606.02244](#)] [[INSPIRE](#)].
- [49] M.L. Mangano, S.J. Parke and Z. Xu, *Duality and multi-gluon scattering*, *Nucl. Phys. B* **298** (1988) 653 [[INSPIRE](#)].
- [50] F.A. Berends and W.T. Giele, *Recursive calculations for processes with n gluons*, *Nucl. Phys. B* **306** (1988) 759 [[INSPIRE](#)].
- [51] C. Cheung and D. O’Connell, *Amplitudes and spinor-helicity in six dimensions*, *JHEP* **07** (2009) 075 [[arXiv:0902.0981](#)] [[INSPIRE](#)].
- [52] Z. Bern, J.J. Carrasco, T. Dennen, Y.-T. Huang and H. Ita, *Generalized unitarity and six-dimensional helicity*, *Phys. Rev. D* **83** (2011) 085022 [[arXiv:1010.0494](#)] [[INSPIRE](#)].
- [53] S. Davies, *One-loop QCD and Higgs to partons processes using six-dimensional helicity and generalized unitarity*, *Phys. Rev. D* **84** (2011) 094016 [[arXiv:1108.0398](#)] [[INSPIRE](#)].
- [54] P.S. Wang, *A p -adic algorithm for univariate partial fractions*, in *Proceedings of the Fourth ACM Symposium on Symbolic and Algebraic Computation, SYMSAC '81*, ACM, New York NY U.S.A. 1981, pg. 212.
- [55] M. Abramowitz and I. Stegun, *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, Dover Publications, New York NY U.S.A. (1964).
- [56] A. Hodges, *Eliminating spurious poles from gauge-theoretic amplitudes*, *JHEP* **05** (2013) 135 [[arXiv:0905.1473](#)] [[INSPIRE](#)].
- [57] S. Badger, *Automating QCD amplitudes with on-shell methods*, in *17th International workshop on Advanced Computing and Analysis Techniques in physics research (ACAT 2016)*, Valparaiso Chile January 18–22 2016 [*J. Phys. Conf. Ser.* **762** (2016) 012057] [[arXiv:1605.02172](#)] [[INSPIRE](#)].
- [58] Z. Bern, A. De Freitas, L.J. Dixon and H.L. Wong, *Supersymmetric regularization, two loop QCD amplitudes and coupling shifts*, *Phys. Rev. D* **66** (2002) 085002 [[hep-ph/0202271](#)] [[INSPIRE](#)].
- [59] G. 't Hooft and M.J.G. Veltman, *Regularization and renormalization of gauge fields*, *Nucl. Phys. B* **44** (1972) 189 [[INSPIRE](#)].