Computer Physics Communications 185 (2014) 1473–1476

Contents lists available at ScienceDirect

Computer Physics Communications

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

Finding linear dependencies in integration-by-parts equations: A Monte Carlo approach☆

Philipp Kant*

Humboldt-Universität zu Berlin, Institut für Physik, Newtonstraße 15, 12489, Berlin, Germany

ARTICLE INFO

Article history:
Received 6 November 2013
Accepted 26 January 2014
Available online 6 February 2014

Keywords:
Feynman diagram reduction
Laporta algorithm
Redundancy
Dependent systems of linear equations
Monte Carlo
Homomorphic images

ABSTRACT

The reduction of a large number of scalar integrals to a small set of master integrals via Laporta's algorithm is common practice in multi-loop calculations. It is also a major bottleneck in terms of running time and memory consumption. It involves solving a large set of linear equations where many of the equations are linearly dependent. We propose a simple algorithm that eliminates all linearly dependent equations from a given system, reducing the time and space requirements of a subsequent run of Laporta's algorithm.

Program summary

Program title: ICE—the IBP Chooser of Equations

Catalogue identifier: AESF_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AESF_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: GNU General Public License, version 3

No. of lines in distributed program, including test data, etc.: 3137

No. of bytes in distributed program, including test data, etc.: 366461

Distribution format: tar.gz

Programming language: Haskell.

Computer: any system that hosts the Haskell Platform.

Operating system: GNU/Linux, Windows, OS/X.

Classification: 4.4, 4.8, 5, 11.1.

Nature of problem: find linear dependencies in a system of linear equations with multivariate polynomial coefficients. To be used on Integration-By-Parts identities before running Laporta's Algorithm.

Solution method: map the system to a finite field and solve there, keeping track of the required equations.

Restrictions: typically less than the restrictions imposed by the requirement of being able to process the output with Laporta's Algorithm.

Unusual features: complexity increases only very mildly with the number of kinematic invariants.

Running time: depends on the individual problem. Fractions of a second to a few minutes have been observed in tests.

1. Introduction

In multi-loop calculations, one often finds that the expression for a given Feynman diagram, after tensor decomposition, is given in terms of a very large number of integrals of the form

$$I(\nu_1, \ldots, \nu_n) = \int d^d k_1 \cdots d^d k_l \frac{1}{D_1^{\nu_1} \cdots D_n^{\nu_n}}. \quad (1)$$

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Tel.: +49 3020934992.
E-mail addresses: philipp.kant@physik.hu-berlin.de, philipp.kant7@gmail.com.

http://dx.doi.org/10.1016/j.cpc.2014.01.017
0010-4655/

Here, $\nu_i \in \mathbb{Z}$ are called the *indices* of a given integral. The D_i are polynomials of total degree 2 in the loop momenta k_i and any external momenta and masses. Integrals with different indices satisfy a set of linear relations, and it is desirable to express a diagram using a minimal set of linearly independent integrals, the so-called *master integrals*.

One source of linear equations relating different integrals are the *Integration-By-Parts* (IBP) identities of [1,2]. They are a consequence of translational invariance of the integral. Additional relations are obtained from Lorentz invariance (LI) [3]. Both IBP and LI equations relate an integral with indices $\{\nu_i\}$ to integrals where some of the $\{\nu_i\}$ are shifted. The coefficients are multivariate polynomials of total degree at most 1 in scalar products of external momenta, squared masses, and the space–time dimension d.

Laporta [4] has given an algorithm that systematically solves IBP and LI identities to reduce a given set of integrals to a linearly independent set. Underlying the algorithm is the observation that allowing larger indices, the number of integrals grows slower than the number of IBP and LI identities relating these integrals with each other. At some point, the rank r of the system is sufficiently large that all integrals within a certain range of indices can be reduced to a small number of master integrals.

Laporta's algorithm proceeds by defining an order on the set of integrals that corresponds roughly to the difficulty of calculating them. In each step of the algorithm, one equation is solved for the most "difficult" integral, and the equations solved in earlier steps are inserted. Finally, all integrals are expressed through a set of "simple" master integrals. The algorithm has become a standard procedure in higher order calculations, and several public implementations [5–9] are available.

There are two inconveniences that cause Laporta's algorithm to be resource hungry. One is intermediate expression swell: starting with polynomial coefficients of low degree, the process of solving and substituting leads to equations over rational functions of high degree, and with large coefficients. Intermediate expressions are usually much larger than the final answer and can challenge the available memory and disk space. In order to mitigate the growth of coefficients and minimize the memory usage, the intermediate expressions are regularly simplified, so that the overall running time of the algorithm is dominated by multivariate gcd calculations and rational function simplification.

This build-up of large intermediate expressions is amplified by the second problem: the number of IBP and LI equations relating a given set of integrals is much larger than their rank, the number of equations that are linearly independent. Consequently, much time is spent processing redundant information, effectively calculating a lot of zeros. Eliminating the redundancy in the linear system has the potential to reduce the demands on CPU time and memory.

The problem of identifying linearly dependent equations beforehand has seen some investigation. For instance, Lee [10] gives selection criteria based on the group structure of the IBP and LI identities. We follow a different approach and propose an algorithm that detects linear dependencies in a given set of IBP and LI equations, thus reducing the time and space requirements of a subsequent run of Laporta's algorithm. Our algorithm is randomized in the Monte Carlo sense, i.e., it has deterministic running time and gives the correct answer with high probability.

2. The algorithm

We now present an algorithm that removes any redundant equations from a system of linear equations with multivariate polynomial coefficients. In the case of Laporta's algorithm, this can drastically reduce the size of the system, and thus the required CPU time and memory.

The basic idea is this: writing the system in matrix form, where each column corresponds to one integral and each row to a linear relationship between integrals, and solving by Gaussian elimination would reduce linearly dependent rows to zero during the forward elimination, allowing the identification and removal of redundant equations. But there would be no gain: determining the minimal set of equations would require the solution of the whole system in the first place.

However, the cost of Gaussian elimination can be reduced by mapping the coefficients homomorphically to a simpler domain. As long as the homomorphism does not reduce the rank of the system, one can still read off which equations are redundant. We follow the canonical choice of using $\mathbb{F}_p$, the field of integer numbers modulo a prime p. In this way, the Gauss algorithm does not suffer from intermediate expression swell, and no gcd calculations are necessary.[1]

Algorithm 1 Get a maximal linearly independent subset of a given system of linear equations over $\mathbb{Z}[x_1, \ldots, x_s]$.

Input: A, an $n \times m$ matrix over $\mathbb{Z}[x_1, \ldots, x_s]$.
Output: B, an $r \times m$ submatrix of A with linearly independent rows, where $r \leq \operatorname{rank} A$. With high probability, $r = \operatorname{rank} A$.

1: $p \leftarrow$ a large prime number
2: $A' \leftarrow A \bmod p \in \left(\mathbb{F}_p[x_1, \ldots, x_s]\right)^{n\times m}$
▷ Take the residue mod p of every coefficient of every polynomial.
3: $a_1, \ldots, a_s \leftarrow$ random points from $\mathbb{F}_p$
4: $A'' \leftarrow A'(a_1, \ldots, a_s) \in \mathbb{F}_p^{n\times m}$
▷ Evaluate every entry of A' at the point $(x_1 = a_1, \ldots, x_s = a_s) \bmod p$.
5: Perform forward Gauss elimination on A''. Before each step, perform a row permutation to get a non-zero pivot element. Let $I = \{i_1, \ldots, i_n\}$ be the resulting permutation of rows, and r the number of non-zero rows after Gaussian elimination (i.e., the rank of A'').
6: $B \leftarrow$ the matrix consisting of rows $i_1, \ldots, i_r$ of A

The resulting algorithm is depicted in Algorithm 1. The operation of taking the modulus of A in step 2 is meant to be elementwise: we take the modulus of each coefficient of each polynomial in the matrix. Likewise, the evaluation of the matrix in step 4 is meant as an evaluation (within $\mathbb{F}_p$) of every polynomial.

It should be noted that in addition to identifying a maximal linearly independent set of equations, the algorithm also identifies the master integrals: any column that does not contain a pivot element corresponds to an integral that cannot be reduced with the given set of equations. Of these, some will be integrals with large indices that could be solved with additional equations, and the others will be the master integrals.

2.1. Simple example

In order to illustrate the algorithm, we give a simple example. Consider

$$A = \begin{pmatrix} x & x+y & 1 & 0 \\ 5x & 3y & 0 & x \\ -4x & x-2y & 1 & -x \\ 0 & x & y & 3x \\ x & 2x+y & y+1 & 3x \end{pmatrix} \in \mathbb{Z}[x, y]^{5\times 4}.$$

[1] A pedagogical introduction to the technique of homomorphic images can be found, for example, in [11,12].