IFJPAN-IV-2016-xxx
arXiv:16xx.xxxxx

Fuchsia: a tool for reducing differential equations for Feynman master integrals

Version xx.xx.xx

Oleksandr Gituliar*[a] and Vitaly Magerya

[a] *Institute of Nuclear Physics, Polish Academy of Sciences, Radzikowskiego 152, 31-342, Cracow, Poland*

April 8, 2016

Abstract

We present a program Fuchsia based on the Lee algorithm [Lee15] which, given a system of first-order linear differential equations with rational function coefficients, constructs an equivalent system in the canonical form and corresponding transformation. After such a reduction is successfully done the system can be trivially solved, which makes a reduction step crucial to obtain the solution.

In principle, Fuchsia can deal with any regular systems, however it's primary task is to reduce differential equations for master integrals which arise from Feynman diagrams. It ensures that solutions contain regular singularities only due to the analyticity of S-matrix.

We discuss limitations and possible extensions of the proposed implementation.

*Corresponding author; email address: oleksandr.gituliar@ifj.edu.pl

PROGRAM SUMMARY

Program Title: `Fuchsia`

Authors: O. Gituliar and V. Magerya

Program obtainable from: `https://github.com/gituliar/fuchsia/`

Journal Reference:

Catalogue identifier:

Licensing provisions: ISC license

Programming language: Python 2.7

Operating system: Linux or any other supported by `SageMath`

RAM: Dependent upon the input data. Expect hundreds of megabytes.

Keywords: Computer algebra, Moser-reduction

Classification: 5 Computer Algebra, 11.1 High Energy Physics and Computing

External routines/libraries: SageMath (version 7.0 or higher), docopt

Nature of problem: ...

Solution method: Algorithm by R. Lee [Lee15].

Restrictions: Not all systems of differential equations can be transformed in the proposed way because some systems are irreducible and the reducibility of some others can not be determined. `Fuchsia` is designed to report if the input belongs to one of these cases.

Running time: Dependent upon the input data size and complexity. Around a minute in total for the example 8×8 matrix on a PC with a 2GHz Intel Core i5 processor.

Contents

1 Introduction

Feynman integrals calculus: importance, methods to calculate (Smirnov's book [Smi06]), differential equations method (progress [Hen15]) .

It has passed more than 50 years (?) since Richard Feynman proposed a diagrammatic approach for calculating perturbative processes in quantum field theories. Since then Feynman integrals calculus has grown to a separate branch of the mathematical physics with a big community of scientists making research in this exciting field. With no doubt we can say that none of the recent discoveries in the high-energy particle physics could happen without precise theoretical calculations, which are based in their core on the Feynman integrals calculation techniques. It is also clear that such techniques will play a key role for discoveries at the present and future high-energy colliders, hence their development and improvement is a very important task.

Recent progress in the computer industry made possible to automate calculation of Feynman integrals; problems which seemed impossible 10 years ago now are successfully solved with state-of-the-art computer algorithms and techniques. Among the most popular of them is Feynman parameters [], sector decomposition [], Mellin-Barns [], integration-by-parts [], differential equations [], and others; for a detailed review of these method see [Smi06].

Analyticity of S-matrix and Fuchsian systems of DE. Reduction to Fuchsian form as an independent problem (Moser [Mos59],Barkatou [BP09], `super_reduce` *in Maple), but mistreating* ∞*.*

Integration-by-parts reduction, canonical form (Henn) [Hen13], Lee algorithm.

The paper is organized as follows: in Section 2 we introduce notation and definitions followed by a brief review of the three key algorithms of [Lee15] and implemented in Fuchsia. In Section 3 we describe how to install and use Fuchsia from different envi-

ronments, depending on your goal and programming experience. In Section 4 we discuss limitations and possible extensions of the implemented method.

2 Overview of the method

In this section, after a few preliminaries we briefly discuss R.Lee's method for reducing a system of ODEs to canonical form.

2.1 Notation and definitions

Let us consider a system of (ODEs)

$$\frac{\mathrm{d}\bar{f}}{\mathrm{d}x} = \mathbb{A}(x,\epsilon)\,\bar{f}, \tag{1}$$

where $\bar{f}(x,\epsilon)$ is a vector of n unknown functions (e.g., master integrals). Elements of the $n \times n$ matrix $\mathbb{A}(x,\epsilon)$ are assumed to be rational functions in x of the form

$$A(x,\epsilon) = \sum_k \frac{1}{(x-x_k)^{1+p_k}} \sum_{i=0}^{\infty} A_{ik}(\epsilon)(x-x_k)^i \tag{2}$$

with *singular points* at $x = x_k$, where k runs over a final set and x_k is a complex number or ∞. (*put comment on* ∞)

Equivalent systems. We can apply a *transformation* $\mathbb{T}$ to the system (1) by changing a basis $\bar{f}$ to

$$\bar{g} = \mathbb{T}(x,\epsilon)\,\bar{f}. \tag{3}$$

That leads to the *equivalent system* of ODEs

$$\frac{\mathrm{d}\bar{g}}{\mathrm{d}x} = \mathbb{B}(x,\epsilon)\,\bar{g}, \tag{4}$$

where

$$\mathbb{B} = \mathbb{T}^{-1}\left(\mathbb{A}\mathbb{T} - \frac{\mathrm{d}\mathbb{T}}{\mathrm{d}x}\right). \tag{5}$$

We require that the transformation (3) is also in the rational form, e.g., as $\mathbb{A}$ in (2), which guarantees that the equivalent matrix $\mathbb{B}$ is of the rational form too.

Classification of singularities. For any system (1) we can define a rational number

$$m_k(\mathbb{A}) = p_k + \frac{r_k}{n} \geq 0 \tag{6}$$

as the *order* or *Poincaré rank* of $\mathbb{A}$ at $x = x_k$, where $r_k = r(A_{0k})$ is the algebraic rank of $A_{0k}(\epsilon)$, $0 \leq r_k \leq n$. If $p_k + r_k/n < 0$ we set $m_k(\mathbb{A}) = 0$.

Under the transformation (3) the order $m_k(\mathbb{A})$ may change, i.e., $m_k(\mathbb{A}) \neq m_k(\mathbb{B})$. Therefore, we introduce the *minimum order* or *generalized Poincaré rank* of the matrix $\mathbb{A}$ as

$$\mu_k(\mathbb{A}) = \min m_k(\mathbb{B}), \text{ for } \forall\, \mathbb{T}. \tag{7}$$

If we can decrease a Poincaré rank of the matrix at the point $x = x_k$, or in other words if $m_k(\mathbb{A}) > \mu_k(\mathbb{A})$, we say that the matrix is *reducible* at this point.

Considering this definition we say that the point $x = x_k$ (the change of variable $x \to 1/x$ allows to classify the point $x = \infty$ in the same way) of $\mathbb{A}$ is

- a *regular point*, for $\mu_k(\mathbb{A}) = 0$ and $m_k(\mathbb{A}) = 0$;
- an *apparent singularity* for $\mu_k(\mathbb{A}) = 0$ and $m_k(\mathbb{A}) > 0$;
- a *regular singularity*, for $0 < \mu_k(\mathbb{A}) \leq 1$;
- an *irregular singularity*, for $\mu_k(\mathbb{A}) > 1$.

We also say that matrix $\mathbb{A}$ has a *Fuchsian form* when it does not contain irregular singularities at any point $x = x_k$, where x_k is a complex number or ∞.

2.2 Reduction to canonical form

The goal of the method [Lee15], briefly described below, is to find a rational matrix $\mathbb{T}(x, \epsilon)$ which transforms a system of ordinary differential equations with rational coefficients given by the matrix $\mathbb{A}(x, \epsilon)$ to the equivalent system given by the matrix $\mathbb{M}(x, \epsilon)$ in the *canonical form* [Hen13], i.e., $\mathbb{M}(x, \epsilon) = \epsilon\, \mathbb{M}(x)$. (Here and everywhere else in the text we assume that $\epsilon \to 0$, e.g., is a dimensional regularization parameter). Once such a form is found the initial system $\mathbb{A}$ can be easily solved as a Laurent series in ϵ parameter, which is of course the ultimate goal.

The whole method can be divided into three steps [Lee15]:

1. Given a matrix $\mathbb{A}(x, \epsilon)$, find a transformation $\mathbb{T}$ such that the equivalent matrix $\mathbb{B}(x, \epsilon)$, given by eq. (5), has a Fuchsian form.
2. Given a matrix $\mathbb{A}(x, \epsilon)$ in the Fuchsian form, find a transformation $\mathbb{T}$ such that the equivalent matrix $\mathbb{B}(x, \epsilon)$ is normalized, i.e., the eigenvalues of all its residues have the form $n\epsilon$, where n is integer.
3. Given a normalized matrix $\mathbb{A}(x, \epsilon)$ in the Fuchsian form, find a transformation $\mathbb{T}$ such that the equivalent matrix $\mathbb{B}(x, \epsilon) = \epsilon\, \mathbb{B}(x)$.

Notation and definitions [Mos59]: Poincare rank, equivalent and reducible systems, transformation, Fuchsian form (regular, irregular, and apparent singularities), balance.

The *fuchsify* algorithm. At first let us make a small remark. The problem with eq. (7) is that a criterion to directly calculate $\mu_k(A)$ does not exist. For that reason we can not say whether the point $x = x_k$ is regular or irregular singularity or whether matrix can be reduced to the Fuchsian form. (However, for the n-th order ODE such a criterion exists, which is a generalization of Fuchs' theorem, see [Mos59].)

Let us mention, that in regular singular points solutions of (1) grow at most like a finite power of $|x|$, for that reason we

Describe key algorithms [Lee15]: fuchsify, normalize, factorize.

The *normalize* algorithm.

The *factorize* algorithm.

3 Using Fuchsia

3.1 Installation

3.2 Usage from the command line

An easy way to use Fuchsia is to run `fuchsia.py` shell script, which has the following invocation signature:

```
$ fuchsia.py <action> <options>
```

where

- `<action>` is one of the algorithms described in the previous section, i.e., `fuchsify`, `normalize`, `factorize`, or auxiliary action `transform`, which applies a user-defined transformation to the given matrix.
- `<options>` are action-dependent options described in the help message printed with the help of `fuchsia.py --help` command.

In the following we provide a complete help information printed by `fuchsia.py --help`:

...

3.3 Usage from SageMath

fuchsify$(\mathbb{M}, x, seed = 0)$

Reduce a system defined by matrix $\mathbb{M}$ and independent variable x to Fuchsian form. That is, make sure that the Poincare rank of all singularities of the transformed matrix $\mathbb{M}'$ are 0. Return a pair of values: the transformed matrix $\mathbb{M}'$ and the transformation $\mathbb{T}$. If the system is irreducible, raise $FuchsiaError$.

normalize$(\mathbb{M}, x, epsilon, seed = 0)$

Transform a Fuchsian system defined by matrix $\mathbb{M}$, independent variable x and infinitesimal parameter *epsilon* to a normalized form. That is, make sure that eigenvalues of all matrix residues of the transformed matrix $\mathbb{M}'$ lie in the range $[-1/2, 1/2)$ in the limit $epsilon \to 0$. Return a pair of values: the transformed matrix $\mathbb{M}'$ and the transformation $\mathbb{T}$. If such transformation can not be found, raise $FuchsiaError$.

factor_epsilon$(\mathbb{M}, x, epsilon, seed = 0)$

Transform a normalized system defined by matrix $\mathbb{M}$, independent variable x and infinitesimal parameter *epsilon* so that the transformed matrix $\mathbb{M}'$ is proportional to *epsilon*. Return a pair of values: the transformed matrix $\mathbb{M}'$ and the transformation $\mathbb{T}$. If such transformation can not be found, raise $FuchsiaError$.

simplify_by_jordanification$(\mathbb{M}, x)$

Try to simplify a system defined by matrix $\mathbb{M}$ and independent variable x by constant transformations that transform leading expansion coefficients of $\mathbb{M}$ into their Jordan forms. Return a pair of values: the simplified matrix $\mathbb{M}'$ and the transformation $\mathbb{T}$. If none of the attempted transformations reduce the complexity of $\mathbb{M}$ (as measured by **matrix_complexity**), return the original matrix and the identity transformation.

simplify_by_factorization$(\mathbb{M}, x)$

Try to simplify a system defined by matrix $\mathbb{M}$ and independent variable x by a constant transformation that extracts common factors found in $\mathbb{M}$ (if any). Return a pair of values: the simplified matrix $\mathbb{M}'$ and the transformation $\mathbb{T}$.

matrix_complexity$(\mathbb{M})$

This function is used as a measure of matrix complexity by **fuchsify** and simplification function. Currently it is defined as the length of textual representation of matrix $\mathbb{M}$.

balance$(\mathbb{P}, x_1, x_2, x)$

Return a *balance* transformation between points $x = x_1$ and $x = x_2$ using projector matrix $\mathbb{P}$.

transform$(\mathbb{M}, x, \mathbb{T})$

Transform a system defined by matrix $\mathbb{M}$ and independent variable x using transformation matrix $\mathbb{T}$. Return the transformed matrix $\mathbb{M}'$.

balance_transform$(\mathbb{M}, \mathbb{P}, x_1, x_2, x)$

Same as **transform**$(\mathbb{M}, x,$ **balance**$(\mathbb{P}, x_1, x_2, x))$, but implemented more efficiently: since the inverse of **balance**$(\mathbb{P}, x_1, x_2, x)$ is **balance**$(\mathbb{P}, x_2, x_1, x)$, this function can avoid a time-consuming matrix inversion operation that **transform** must perform.

singularities$(\mathbb{M}, x)$

Find values of x around which matrix $\mathbb{M}$ has a singularity in x. Return a dictionary with $\{x_i : p_i\}$ entries, where p_i is the Poincare rank of $\mathbb{M}$ at $x = x_i$. The set of singular points can include $+Infinity$, if $\mathbb{M}$ has a singularity at $x \to \infty$.

matrix_c0($\mathbb{M}, x, x_0, p$)
Return the 0-th coefficient of the series expansion of matrix $\mathbb{M}$ around $x = x_0$, assuming Poincare rank of $\mathbb{M}$ at that point is p. If x_0 is $+Infinity$, return the coefficient at the highest power of x.

matrix_c1($\mathbb{M}, x, x_0, p$)
Return the 1-th coefficient of the series expansion of matrix $\mathbb{M}$ around $x = x_0$, assuming Poincare rank of $\mathbb{M}$ at that point is p. If x_0 is $+Infinity$, return the coefficient at the second-to-highest power of x.

matrix_residue($\mathbb{M}, x, x_0$)
Return matrix residue of matrix $\mathbb{M}$ at $x = x_0$, assuming that Poincare rank of $\mathbb{M}$ at $x = x_0$ is 0. Return matrix residue at infinity if $x = +Infinity$.

export_matrix($file, \mathbb{M}$)
Write matrix $\mathbb{M}$ to a file-like object $file$ using MatrixMarket array format. For description of this format see [BPR96].

export_matrix_to_file($filename, \mathbb{M}$)
Write matrix $\mathbb{M}$ to a file $filename$ using MatrixMarket array format.

import_matrix($file$)
Read a symbolic matrix from a file-like object $file$, assuming it is formatted using MatrixMarket array format.

import_matrix_from_file($filename$)
Read a symbolic matrix from a file $filename$, assuming it is formatted using MatrixMarket array format.

4 Summary

Prospects: non-rational transformations (e.g. sunrise graphs).

Acknowledgment

References

[BP09] Moulay A. Barkatou and Eckhard Pflügel. On the Moser- and super-reduction algorithms of systems of linear differential equations and their complexity. *Journal of Symbolic Computation*, 44(8):1017–1036, 2009.

[BPR96] Ronald F. Boisvert, Roldan Pozo, and Karin Remington. The Matrix Market exchange formats: Initial design. Technical report, NISTIR, 5935, National Institute of Standards and Technology, Gaithersburg, MD, USA, December 1996.

[Hen13] Johannes M. Henn. Multiloop integrals in dimensional regularization made simple. *Phys. Rev. Lett.*, 110:251601, 2013, arXiv:1304.1806.

[Hen15] Johannes M. Henn. Lectures on differential equations for Feynman integrals. *J. Phys.*, A48:153001, 2015, arXiv:1412.2296.

[Lee15] Roman N. Lee. Reducing differential equations for multiloop master integrals. *JHEP*, 04:108, 2015, arXiv:1411.0911.

[Mos59] Jrgen Moser. The order of a singularity in fuchs' theory. *Mathematische Zeitschrift*, 72(1):379–398, 1959.

[Smi06] V.A. Smirnov. *Feynman Integral Calculus*. Springer, 2006.