TTP13-047
SFB/CPP-13-109

FIESTA 3: cluster-parallelizable multiloop numerical calculations in physical regions

A.V. Smirnov[a,b,*]

[a] *Scientific Research Computing Center, Moscow State University, 119992 Moscow, Russia*
[b] *Institut für Theoretische Teilchenphysik, Karlsruhe Institute of Technology, D-76128 Karlsruhe, Germany*

Abstract

The goal of this paper is to present a new major release of the program FIESTA (Feynman Integral Evaluation by a Sector decomposiTion Approach). This version presents features like cluster-parallelization, new asymptotic expansion algorithms, calculations in physical regions, new sector-decomposition strategies, as well as multiple speed, memory, and stability improvements.

Keywords: Feynman diagrams, Multiloop Feynman integrals, Dimensional regularization, Computer algebra

*Corresponding author
Email address: asmirnov80@gmail.com (A.V. Smirnov)

PROGRAM SUMMARY

Manuscript Title: FIESTA 3: cluster-parallelizable multiloop numerical calculations in physical regions
Authors: A.V. Smirnov
Program title: FIESTA 3
Licensing provisions: GPLv2
Programming language: `Wolfram Mathematica` 7.0 or higher, `c++`
Computer(s) for which the program has been designed: from a desktop PC to a supercomputer
Operating system(s) for which the program has been designed: Unix, Linux, Mac OS X
RAM required to execute with typical data: depends on the complexity of the problem
Has the code been vectorised or parallelized?: yes
Number of processors used: from 1 processor up to loading a supercomputer (tests were performed up to 1024 cores)
Supplementary material: The article, usage instructions in the program package, http://science.sander.su, https://bitbucket.org/fiestaIntegrator/fiesta/overview
Keywords: Feynman diagrams, Multiloop Feynman integrals, Dimensional regularization, Computer algebra
CPC Library Classification: 4.4 Feynman diagrams, 4.12 Other Numerical Methods, 5 Computer Algebra, 6.5 Software including Parallel Algorithms
External routines/libraries used: `Wolfram Mathematica` [1], `KyotoCabinet` [2], `Cuba` [3], `QHull` [4]
Nature of problem: The sector decomposition approach to evaluating Feynman integrals falls apart into the sector decomposition itself, where one has to minimize the number of sectors; the pole resolution and epsilon expansion; and the numerical integration of the resulting expression. Morover, in cases where the integrand is complex, one has to perform a contour deformation
Solution method: The program has a number of sector decomposition strategies. One of the most important features is the ability to perform a contour deformation, as well as the so-called preresolution in case of integrals at threshold.
Everything except the integration is performed in `Wolfram Mathematica` [1] (required version is 7.0 or higher). This part of the calculation is parallelizable with the use of shared memory. The database is stored on hard disk with the use of the `KyotoCabinet` [2] database engine.
The integration part of the algorithm can be performed on a cluster, is written in `c++` and does not need `Wolfram Mathematica`. For integration we use the Cuba

library package [3].
Restrictions: The complexity of the problem is mostly restricted by CPU time required to perform the integration and obtain a proper precision.
Running time: depends on the complexity of the problem.
References:
[1] http://www.wolfram.com/mathematica/, commercial algebraic software;
[2] http://fallabs.com/kyotocabinet/, open source;
[3] http://www.feynarts.de/cuba/, open source;
[4] http://www.qhull.org, open source.

1. Introduction

Let us consider a Feynman integral

$$\mathcal{F}(a_1,\ldots,a_n) \;=\; \int\cdots\int \frac{\mathrm{d}^d k_1\ldots\mathrm{d}^d k_h}{E_1^{a_1}\ldots E_n^{a_n}}\,, \tag{1}$$

where the denominator factors E_i are linear functions with respect to scalar products of loop momenta k_i and external momenta p_i, and dimensional regularization with $d = 4 - 2\epsilon$ is implied.

Such an integral depends on masses and kinematic invariants — scalar products of external moments. After substituting all values for kinematic invariants and masses one can evaluate the integral numerically. This can be done automatically with the so-called sector-decomposition approach, initially introduced by Binoth and Heinrich [1, 2, 3, 4, 5, 6].

This approach is based on the so-called alpha-representation of Feynman integrals:

$$\begin{aligned}&\mathcal{F}(a_1,\ldots,a_n) = (i\pi^{d/2})^l \times \\ &\frac{\Gamma(A-ld/2)}{\prod_{j=1}^n\Gamma(a_j)}\int_{x_j\geq 0} dx_i\ldots dx_n\delta\left(1-\sum_{i=1}^n x_i\right)\left(\prod_{j=1}^n x_j^{a_j-1}\right)\frac{U^{A-(l+1)d/2}}{F^{A-ld/2}},\end{aligned} \tag{2}$$

where $A = \sum_{i=1}^n a_n$, l is the number of loops and U and F are constructively defined polynomials of x_i.

There are three public programs that can perform the numerical calculation with the sector decomposition approach — `sector_decomposition` by Bogner and Weinzierl [5, 6], `SecDec` by Binoth and Heinrich [4] (later improved and made public by Borowka, Carter and Heinrich [7, 8, 9, 10, 11, 12]) and `FIESTA` [13, 14] initially created by the author of this paper with M. Tentukov and later improved together with V. Smirnov. In this paper we present a new version of `FIESTA`.

Both `FIESTA` and `FIESTA 2` had a major disadvantage when being compared with `SecDec 2` — they were not able to perform calculations in physical regions. Hence some of the advantages such as the internal code compiler designed especially for sector integrals and the multi-precision calculation could not be used by those interested in physical regions. The new version

fills this gap by utilizing the ideas initially suggested in [15, 16, 17, 18, 19] and presented as an algorithm in the second version of `SecDec` [7].

One more important new feature is the ability to use cluster parallelization. If one is going to perform multiloop calculations and aims at high-precision results, it may take a lot of time to obtain those. The new version of `FIESTA` has an improved internal structure so that the algebraic part can be performed by `Wolfram Mathematica` on a single computer and the results can be saved into a database. Afterwards the integration can be performed on a cluster with the use of `MPI`-parallelization. This approach also helps to deal with the `Mathematica` licensing policy — the integration part does not need `Mathematica` licenses anymore.

Although the sector decomposition approach is quite powerful, in some cases the integral is too complicated for direct evaluation. Even if the calculation of master integrals can be performed, the other important part, IBP reduction [20] might fail. In this case one can use the asymptotic expansion approach. The kinematic invariants are separated into groups proportional to different groups of a small parameter ρ. One is interested in the behavior of Feynman integrals when ρ tends to zero.

There are different approaches to asymptotic expansion of Feynman integrals, but we are interested in numerical algorithms. The reason is that those algorithms can be implemented as computer codes so that the asymptotic expansion can be done completely automatically.

In this paper we consider two numerical algorithms of asymptotic expansion. Both of them use sector decomposition and hence become a part of the new version of `FIESTA`.

The first algorithm uses a Mellin-Barnes integration combined with sector decomposition — it has already been encoded in FIESTA2 [14]. The disadvantage of this algorithm is that it can be applied only in situations where the kinematic invariants can be separated into two groups — large and small. Moreover this algorithm encounters some computer problems — the complexity of evaluations grows high enough to some make three-loop problems too complicated for this approach.

The second algorithm is proposed in this paper. It consists of applying the `asy` code [21, 22] in order to reveal regions [23, 24]. Then the algorithm produces contributions of regions by expanding the integrand in a certain way and applies the sector decomposition in order to calculate expansion coefficients. However in many situations the corresponding integrals happen not to be well-defined — one needs to add an extra regularization in or-

der to calculate them. This makes the sector decomposition approach more complex.

In section 2 we explain how `FIESTA` treats integrals at and above the threshold, in section 3 we describe the stages that `FIESTA` performs and the parallelization during those stages. In section 4 we present two algorithms used for numerical asymptotic expansion and in section 5 we provide installation and usage instructions.

2. Integrating at and above the threshold

Sector decomposition strategies are guaranteed to terminate while the function F from formula (1) has no monomials with negative coefficients. If due to some values of kinematic invariants F has negative terms, this approach may fail. In principle, there are three different variants:

1. Below threshold. Some terms of F are negative, but it still turns to zero only due to some variables x_i turning to zero. In this case the sector decomposition works as normal;
2. At threshold. F is never negative, but might turn to zero at some internal points. To solve this problem we use the so-called pre-resolution approach suggested in `FIESTA 2`[14] and improved in `FIESTA 3` (for details see [22]). Some transformations are performed before the sector decomposition aimed to get rid of singularities of the form $(x_i - x_j)^2$. We have multiple confirmations that this approach works well for Feynman integrals;
3. Above threshold. Here F can be of different signs in different parts of the integration domain and, therefore turns to zero at multiple points inside. The integral is a complex number, however the most complicated part is that the integration cannot be taken directly due to the zero values of F. An automatic way to deal with this problem was suggested by Binoth and Heinrich [1, 2] and implemented by Borowka and Heinrich in the second version of `SecDec` [10]. The basic idea is to transform the contour in order to shift away from the zero values of F. This makes the integration process slower but provides an automatic way to process integrals above threshold. We also implemented this strategy in `FIESTA 3` mostly following the guidelines from [10]. In order to activate this algorithm one has to set `ComplexMode=True` in `FIESTA 3`.

The contour transformation algorithm described in [10] and used after a modification in `FIESTA` consists of the following steps (each step uses `LambdaIterations` iterations):

1. The maximal values of shift functions $S_i := x_i(1-x_i)\frac{dF}{dx_i}$ are estimated (we substitute a number of random points); if a maximum is bigger than 1, the corresponding shift function is multiplied by its inverse;
2. The integration variables x_i are replaced with $x_i - \lambda \mathrm{i} S_i$, the dependence of F on the new variables is considered;
3. The code estimates the maximal possible value λ such that the cubic terms of F do not exceed the linear terms. This is close to checking that the function F does not change the sign of its complex part (as specified by the dimension-regularization prescriptions). If the resulting value is bigger that `MaxComplexShift` the latter is used;
4. The code splits the interval from 0 to the found maximal possible λ into the number of parts equal to `LambdaSplit`. For each of those values a new check is performed and a best value is chosen: we require that the sign of the complex part of F is always negative, and among those choices find such λ that the minimal absolute value of F is maximal;
5. The λ value is fixed and substituted into the deformation formula.

Those calculations are performed independently for each sector.

Alternatively, one can specify the value of `ComplexShift` to set a fixed λ and turn off this algorithm. Since the search in `Mathematica` for best possible λ is performed randomly and the speed of `Mathematica` calculations is not too fast, one cannot be absolutely sure that λ is chosen properly — the sign of the imaginary part of F might turn positive, and one obtains a wrong answer. It is even more risky if one chooses a fixed complex shift. Hence it is recommended to perform extra sign tests in the `c++` part of the program.

To do so one should run the code with the `OnlyPrepare` option so that the integrands are stored in a database. After that `Mathematica` prints a command to integrate everything, but one should first add the `-testF` argument to this command (see section 5.4). One should ignore the answer produced by the integrator and only watch whether it succeeds. It means that λ is chosen well and one can turn back to the integration.

The contour deformation does not help if F turns to 0 due to some variables being equal to 1. If one suspect that this might be the case, the `TestF1=True` setting should be used while running `FIESTA`. Then one will be informed on variables leading to singularities of this sort. Then one should

use the `BisectionVariables` setting listing the problematic variables. This makes `FIESTA` break the integration into two parts — from 0 to p and from p to 1 for some p. Then in the first part we make the variable replacement $x = yp$ and in the second: $x = 1 - y(p-1)$. Thus the singularities at $x \to 1$ are changed to $x \to 0$ and are treated by the standard sector decomposition. By default the middle point p is equal to $1/2$, but a different value can be chosen by the `BisectionPoint` setting or different values for different variables by the `BisectionPoints` list.

2.1. Example

Let us consider an example (see fig.1) — an on-shell massless K4-graph (we choose $s = 1$ and $t = -1/4$). The answer is a complex number, so the contour transformation is used.

Figure 1: K4 graph

The analytic answer for the corresponding integral was recently obtained in [25]; the coefficient at ε^2 is equal to $1024.2413 + 889.3892i$ (we cut it to 8 digits), so we can compare the numerical results with it. We performed the calculations on a computer having two Hexa-Core Intel X5675 3.07 GHz processors. The database preparation step took less than 10 minutes. The integration time and the numerical uncertainties depend on the number of sampling points and are presented in Table 1. The columns contain the number of sampling points, time needed to calculate the ε^2 part, the numerical coefficient at ε^2, the error estimate and the real error. The default setting for the number of sampling points is 50000.

We can see from table 1 that both the error estimate and the real error are decreasing proportional to the square root of the number of sampling points. The time grows linearly at large scales. The less than linear growth for smaller numbers can be explained by the fact that some small calculations might be performed with the use of processor cache and hence faster, but starting with some point the cache is not enough and the dependence becomes linear. The result is always within the error estimate, although we have to

Points	Time (sec.)	Result	Error est.(%)	Error (%)
5 000	224	$1023.64 + 889.577i$	0.0508	0.0465
50 000	3339	$1024.33 + 889.412i$	0.0183	0.0068
500 000	42940	$1024.29 + 889.387i$	0.0058	0.0035
5 000 000	426726	$1024.25 + 889.388i$	0.0018	0.0007

Table 1: Results and timings for the K4 graph

state that the error estimate is produce by the `Vegas` algorithm and is only a probability prediction but not a guarantee.

3. Cluster-parallelization and internal structure of FIESTA 3

Let us describe the internal structure of `FIESTA 3`. The input for `FIESTA` is the list of propagators, the list of internal moments, the list of indices and the requested order of epsilon.

The algorithm consists of three major stages. The **first stage** is the initial preparation and sector decomposition. The algorithm performs the following:

- Eliminating negative indices. If some of the indices are non-positive, the algorithm differentiates the integrand by the corresponding alpha-parameters according to the following rule:

$$\int_0^\infty dx \frac{x^{(a-1)}}{\Gamma(a)} f(x) = f^{(n)}(0)$$

  for non-positive integer a.

- "Pre-resolution". If we consider an integral on a threshold, the function F might have negative terms, but is non-negative totally. However it might be equal to zero somewhere in the middle in the integration domain. This might be a problem for the integration, hence we perform the so-called pre-resolution. The algorithm searches for pairs of variables that might produce negative terms due to contributions like $(x_i - ax_j)^2$, divides the infinite sector into two and makes appropriate variable replacements. The details can be found in [22].

- Sector decomposition. A number of strategies can be used including the strategy by Binoth and Heinrich, by Bogner and Weinzierl, the original

strategy from `FIESTA 1`, the Hepp and Speer sector strategies [26], as well as the most effective (by the number of sectors) but relatively slow strategy by Kaneko and Ueda [27].

This part is parallelized in `Mathematica` — different kernels work on different primary sectors.

After the first stage is over, a database is prepared. We use the kyotocabinet (`http://fallabs.com/kyotocabinet/`) engine for storing data. Now each sector can be treated independently, hence we use the `Mathematica` parallelization in order to speedup the calculations. The **second stage** consists of a number of parts. Each time the expressions are read from one database and written to the other one. Only the main `Mathematica` process accesses the databases, the subkernels only perform equivalent tasks in different sectors. There are the following parts:

- Variable substitution. The sector decomposition variable replacements are performed (the replacements rules were created at the previous stage).
- Contour transformation (only in complex mode). The contour is shifted to avoid the zeros of F.
- Pole resolution. If the integrer parts of some exponents of sector variables are non-positive, then in order to reveal singularities the integrand is treated as the first few terms of Taylor series plus the remainder.
- Expression preparation. All functions are combined and multiplied.
- Epsilon expansion. The integrands can now be expanded in epsilon up to the required degree.
- String preparation. The input for the `c++` code is prepared. Now the expressions in the database are no longer `Mathematica` expressions but strings ready to be taken by the integrator.

If the user specifies the `OnlyPrepare` mode, then at this point the algorithm stops. As a result one has a database with integration strings. Starting from this point, one can run the integration without `Mathematica`. However if `OnlyPrepare` is set to `False`, `Mathematica` runs an integrator itself and waits for results.

The integration part has the following structure. There are two possible binaries that can be used — `CIntegratePool` and `CIntegratePoolMPI`. The second one is intended for cluster usage with the `MPI`-parallelization, the first one uses threads. None of these binaries performs the integration itself. It only launches the real integrator processes, `CIntegrate` (basic variant), `CIntegrateMP` (variant with `MPFR`) or `CIntegrateMPC` (complex variant with `MPFR`). The threads version simply runs a requested number of threads, each of those forks and starts the integrator. So the main process reads from the database and distributes tasks between slaves and gathers results afterwards. If the `MPI` version is used, only one copy accesses the database as well, the tasks to other nodes are send via `MPI`.

We use the integrators from the Cuba library [28] by Hahn, however the evaluation of functions is performed in an original way. Unlike it is done in other sector decomposition programs, we do not compile the `c++` integrand by means of standard compilers. On the contrary, the expression is analyzed inside the `c++` part of `FIESTA` and is transformed into an optimized internal form allowing fast numerical function evaluation. Moreover, the expression is analyzed in order to decide, where to use double precision and where it is important to use the multi-precision evaluations (for details see [14]).

We prepared a database and then tested how the `MPI`-parallelization works for a sample massive on-shell four-loop propagator diagram (with propagators $\{-(l_1+q)^2, -(l_2+q)^2, -(l_3+q)^2, -(l_4+q)^2, -(l_4)^2+M^2, -(l_2-l_3+l_4)^2+M^2, -(l_1)^2+M^2, -(l_1-l_2)^2, -(l_4-l_3)^2, -(l_1-l_2+l_3-l_4)^2, -(l_2-l_3)^2\}$ where $M^2=q^2$), the results are shown in Table 2.

Points \ Cores	32	64	128	256	512	1024
500 000	5112	2643	1570	1381	1344	1330
5 000 000	48305	24719	13524	7833	5116	3795
50 000 000	482948	245109	133917	77951	50734	36988

Table 2: Timings of a 4-loop massive propagator

The columns of the table show the number of cores used during the parallelization, the rows — the number of sampling integration points used during the evaluation. The result is measured in seconds. This table indicates that massive parallelization is not very effective if the number points is not too high — this happens due to the database and `MPI` overhead. However this part of the evaluation has a fixed time and while the number of sampling points is increased, the efficiency of the parallelization grows well.

As for the precision obtained, it grows approximately as the square root of the number of sampling points used. Hence the `MPI`-parallelization can be a good way to improve results, however this is possible only if one has enough CPUs times hours. For example, the evaluation with 50 million sampling points with 1024 cores (the tests were performed on the "Lomonosov" supercomputer [29]; the Intel Xeon 5570 Nehalem 2.93 GHz processors connected with QDR InfiniBand, 40 Gbit/sek were used) took 100 hours resulting in a rather big number about 10 thousand CPUs times hours. Possibly it cannot grow much further because the number of jobs in this test was equal to 11161, and the number of cores should be a lot less than the number of cores in order to obtain an efficient parallelization.

A similar table was created with the `SeparateTerms` setting, when the integators are split more than one integrand per sector (resulting in 35971 `MPI` jobs instead of 11161). The results are displayed in Table 3. We can see a better dependence of integration time on the number of cores in this case.

Points \ Cores	32	64	128	256	512	1024
500 000	4684	2345	1193	614	452	451
5 000 000	41596	20720	10431	5355	2826	1579
50 000 000	399997	200004	100555	51467	27149	15195

Table 3: Timings with the `SeparateTerms` setting

These numbers are also presented in Diagram 3. As a base value we take $time(32, 50000)$ — the time in seconds needed to evaluate the integral with 32 cores and 50000 sampling points. The y-axe has $log_2(speedup)$, where $speedup = (time(n, points)/time(32, 50000)) * (50000/points)$ (the second factor is here because we measure the speedup by the number of cores). The functions $1 - 3$ on the diagram correspond to the dependences $time(n, 50000)$, $time(n, 500000)$ and $time(n, 500000)$, the functions $4 - 6$ — are the same but with the `SeparateTerms` setting. We can clearly see that with the `SeparateTerms` setting and a large enough number of sampling points the integration scales really well and suits perfectly to be run on large clusters.

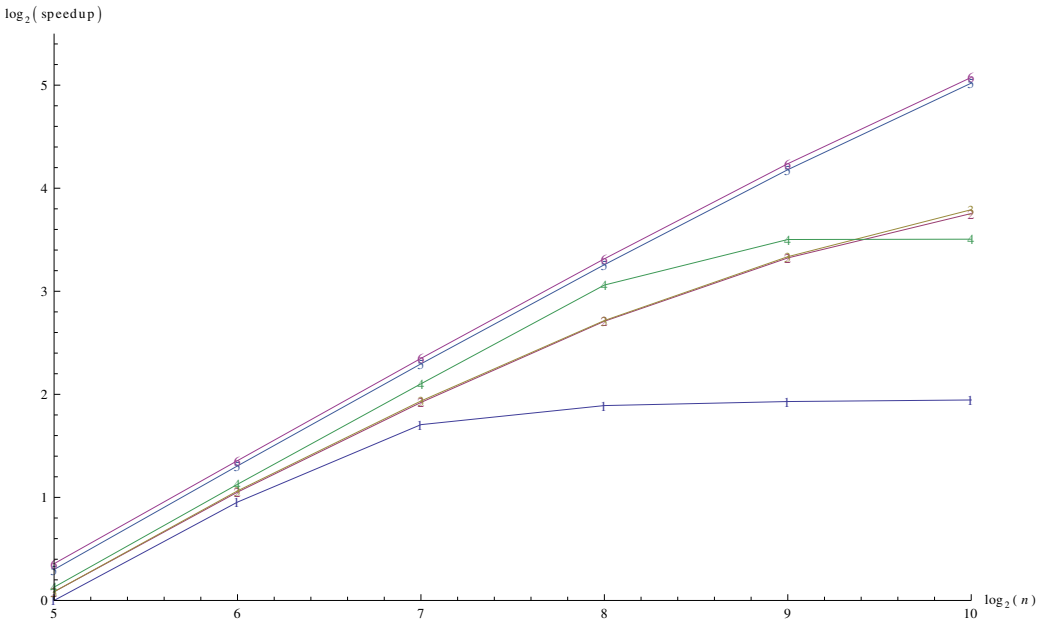

Figure 2: Graphical representation of timings

4. Numerical algorithms for asymptotic expansion of Feynman integrals

4.1. Mellin-Barnes and sector decomposition

This approach, initially suggested by Pilipp [30], is based on combining the Mellin-Barnes and the sector decomposition approaches. Let us write the integral in its alpha-representation (for explicit coefficient values see formula 2):

$$\mathcal{F}(a_1, ..., a_n) = \\ c \int_0^\infty dx_1 ... dx_n \; \delta(1 - x_1 - ... - x_n) x_1^{a_1 - 1} ... x_n^{a_n - 1} U^a F^b, \tag{3}$$

Suppose that the function U does not depend on kinematic invariants and F can be written as $W_1 + W_2$, where the terms in W_1 are much smaller that the terms in W_2. Let us introduce a small parameter ρ and replace W_1 with ρW_1 (now ρ is small and terms of W_1 are not). The parts of F now can be separated by introducing a single Mellin-Barnes integration:

$$\frac{\Gamma(a-hd/2)}{(\rho W_1+W_2)^{a-hd/2}} = \frac{1}{2\pi i}\int_{-i\infty}^{+i\infty} \mathrm{d}z\, \rho^z \frac{\Gamma(a-hd/2+z)\Gamma(-z)}{W_1^{-z} W_2^{a-hd/2+z}} \tag{4}$$

so that we obtain

$$\mathcal{F}(a_1,...,a_n) = \frac{\left(i\pi^{d/2}\right)^h}{\prod_l \Gamma(a_l)} \frac{1}{2\pi i}\int_{-i\infty}^{+i\infty} \mathrm{d}z\, \Gamma(a-hd/2+z)\Gamma(-z)\rho^z \tag{5}$$
$$\times \int_0^1 \ldots \int_0^1 \hat{\mathcal{U}}^{a-(h+1)d/2}\, W_1^z\, W_2^{-a+hd/2-z}\, \delta(1-x_1-...-x_n) x_1^{a_1-1}...x_n^{a_n-1}\ .$$

Now the integration has normal sector integrals (from 0 to 1 after taking out the δ function) and and extra integration (over variable z) from $-i\infty$ to $+i\infty$. The exponents depend on the integration variable z.

The algorithm builds sectors as if there was no extra variable z. Now it considers the variables α_i with corresponding powers depending on z. The integration contour can be closed to the right, but the first few poles have to be considered (the number of the poles depends on the coefficient at z).

Afterwards the algorithm reveals singularities in ε generated by the MB integration over z. The integral of $t_i^{b_i\varepsilon+c_i z+n_i-1}$ generates a z-dependence of the type $\Gamma\left(b_i\varepsilon+c_i z+n_i\right)$. We are concentrating on sector integrals with $c_i<0$ because they are relevant to our limit.

After revealing those singularities the algorithm can return to normal steps used in numerical sector decomposition — expansion by ε and numerical integration by a Vegas integrator.

4.2. Regions and sector decomposition

Another approach presented in this paper also uses sector decomposition but now together with the regions approach. The method of regions [31, 32] defines prescriptions to find regions, or scalings of momentum components that after the expansion provide non-zero contributions. In each region, we first Taylor expand the integrand and drop the scaling restrictions.

In [21] we presented the regions approach written in terms of alpha-representation. It looks natural to make an algorithm that considers those regions; in each region we can expand the integrand and evaluate the corresponding integrals numerically afterwards.

This idea encounters a number of technical difficulties, but what is most important, the integrals in regions might be not well defined. One needs to introduce an extra regularization parameter λ (the poles in λ get canceled out after summing over all regions). We make the indices (powers of alpha-parameters) depend linearly on λ. This does not affect the determination of regions.

After finding the regions, the algorithm expands the integrand in each of them up to the required degree. Then it starts the evaluation of expansion coefficients. Those functions no longer depend on the small parameter ρ but have an extra dependence on the small parameter λ.

Now the sector decomposition is performed. In each sector we get an integral over a unit hypercube, where the exponents depend on ε and on λ. The algorithm considers the exponents depending on λ plus some negative integer and performs a "pre-resolution" — replaces the integrand with the few terms of Taylor expansion in λ plus the remainder (such that the corresponding integral has no poles over λ). Afterwards the integrands can be expanded in λ up to order 0.

Now the algorithm proceeds with standard sector decomposition steps keeping in memory that different terms are proportional to different integer powers of λ. Finally all the coefficients of the expansion in λ and ε are evaluated numerically.

After calculating the result in all regions, the algorithm sums them up and gathers the coefficients. At this point the poles in λ should cancel, however, they can exist due to numerical uncertainties.

There are some more ideas that improve the algorithm. The expanded region integrals often are more complicated than the initial ones, so it might be problematic for the numerical integration. However, a few integrations can be taken out analytically. Quite often after the expansion one of the functions U and F no longer depend on one of the variables x_i. In this case the integration over x_i is taken out analytically.

5. Installation and usage

5.1. Installation

In order to install `FIESTA 3` one has to download the package from the following address: `http://science.sander.su/FIESTA.htm`. Then one should compile the c++ part of the code. Although `FIESTA 3` can be used without the `c++` part (by setting `UsingC=False` and `UsingQLink=False`), it is not

recommended since one will not have access to most features existing in this program.

Hence first one needs to compile FIESTA. In order to do that, the following libraries have to exist on the computer:

- KyotoCabinet — a fast database engine. It can be downloaded from http://fallabs.com/kyotocabinet/. Basically it is compiled and installed with
  ```
  ./configure
  make
  make install
  ```
  If one cannot install it system-wide, one is able to provide the paths to KyotoCabinet later. If one is aiming at a static version of FIESTA, it should be configured with -enable-static -disable-shared.

- Cuba integrator library by Thomas Hahn [28] that can be downloaded from http://www.feynarts.de/cuba/. We recommend to build it with
  ```
  ./configure
  make lib
  make install
  ```
  If one runs make instead it might require extra dependencies.

- MPFR — multiple-precision floating-point library that can be downloaded from http://www.mpfr.org/ but has a big chance to exist in the system repositories. If one is downloading it from the official web-page and is not installing it system-wide, it is recommended to configure it with a prefix and install into a local directory.

- GMP — GNU multi precision library, it is also normally installed system-wide. If not, one can download it from http://gmplib.org/, configure and make. This library is not required at the object compilation step, and for linking one might point to the .libs directory in the GMP folder.

- In case one is going to use strategies KU, KU0, KU2 or the SDExpandAsy command, then one needs to have the qhull convex hull search package installed. It might exist in the system repositories or can be downloaded from http://www.qhull.org/.

- If one is going to build an MPI version of CIntegratePool, then one should need to have one of the MPI environments to be installed on the system. Normally this is recommended if one is installing FIESTA on a cluster.

Moreover, if one is going to perform both the database preparation and the integration, it is needed to have Wolfram Mathematica 7.0 or higher installed on the computer. However, if the databases with integrands are going to be prepared elsewhere, and one is only running the integration (a typical case for clusters), Mathematica is not required.

Now one can try to build FIESTA. If the libraries mentioned earlier are installed and exist on the system paths, then one should simply cd to the directory with FIESTA and run the make command. If not, one should first edit paths.inc and add those paths with -I and -L for include and link paths correspondingly.

In rare cases one needs also to edit libs.inc to tune library linking. If one wants it static as much as possible, then libs.inc should be replaced with libs-static.inc Normally one does not need to edit the Makefile, neither the main one, nor the ones in subfolders.

The make command should build everything but the MPI version of CIntegratePool. The MPI version is build with make mpi or together with other packages with make all. There is no make install in the package.

In order to build KLink one needs to have Mathematica installed on the system. The paths should be detected automatically. If one does not want to build KLink (the database will be prepared elsewhere) and wishes to avoid compilation errors, make noklink should be used.

The build results might be verified with make test or make testall (to test also MPI). No errors should be produced. These simple tests mainly test whether the binaries are functional.

5.2. Program syntax

Let us start with basic examples. If one is loading FIESTA from Mathematica, it should either be loaded with

```
SetDirectory[<path to FIESTA>]; Get["FIESTA3.m"];
```

or

```
FIESTAPath=<path to FIESTA>; Get[FIESTAPath<>"FIESTA3.m"];.
```

Do not load FIESTA by simply specifying a full path to it, it will not work properly.

Now one can call the following commands:

`SDEvaluate[{U,F,l},indices,order]`,

where U and F are the functions from eq. (2), l is the number of loops, *indices* is the set of indices and *order* is the required order of ε-expansion.

To avoid manual construction of U and F one can use a build-in function `UF` and launch the evaluation as

`SDEvaluate[UF[loop_momenta,propagators,subst],indices,order]`,

where *subst* is a set of substitutions for external momenta, masses and other values (to remind: the code performs numerical integration so the functions U and F should not depend on anything external).

Example:

`SDEvaluate[UF[{k},{-k^2,-(k+p1)^2,-(k+p1+p2)^2,-(k+p1+p2+p4)^2},`
`{p1^2 →0,p2^2 →0,p4^2 →0, p1 p2 →-S/2,p2 p4 →-T/2,p1 p4 →(S+T)/2,`
`S→3,T→ 1}], {1,1,1,1},0]`

performs the evaluation of the massless on-shell box diagram.

In the following commands we will only provide the first version of the syntax (with `{U,F,l}`). However, in all places this triple can be replaced with the `UF` generator.

Now to expand a Feynman integral by a small parameter `tt` one should use

`SDExpand[{U,F,l},indices,order,tt,order in tt]`.

Example:

`SDExpand[UF[{k, l}, {-k^2, -(k + p1)^2, -(k + p1 + p2)^2, -l^2,`
`-(l - k)^2, -(l + p1 + p2)^2, -(l + p1 + p2 + p4)^2}, {p1^2 -> 0, p2^2`
`-> 0, p4^2 -> 0, p1*p2 -> s/2, p2*p4 -> t/2, p1*p4 -> -(s + t)/2,`
`s -> -1, t -> -tt}], {1, 1, 1, 1, 1, 1, 1} , 0, tt, 0]`

The new algorithm presented in this paper can also be used to expand this integral, however one has to provide a regularization variable with the `RegVar` setting and shift indices.

Example:

`RegVar=la;`

`SDExpandAsy[UF[{k, l}, {-k^2, -(k + p1)^2, -(k + p1 + p2)^2, -l^2,`
`-(l - k)^2, -(l + p1 + p2)^2, -(l + p1 + p2 + p4)^2}, {p1^2 -> 0, p2^2`
`-> 0, p4^2 -> 0, p1*p2 -> s/2, p2*p4 -> t/2, p1*p4 -> -(s + t)/2,`
`s -> -1, t -> -tt}], {1+la, 1, 1, 1-la, 1, 1, 1} , 0, tt, 0]`

To use the Speer sectors strategy one should use `SDEvaluateG` instead of `SDEvaluate`. The syntax is

`SDEvaluateG[graph_info,{U,F,l},indices,order]`

or

`SDExpandG[graph_info,{U,F,l},indices,order,tt,tt_order]`.

The graph information should be of the form $\{lines, external\_vertices\}$, where *lines* is a list of pairs of vertices connected by this line. The vertices should be numbered from 1 without skipping numbers. It is also very important to have the order of lines coincide with the order of propagators in the UF input. For example, for the tetrahedral input is the following.

Example:

```
SDEvaluateG[{{{1, 2}, {2, 3}, {3, 1}, {4, 2}, {4, 3}, {4, 1},
UF[{k1, k2, k3}, {-k1^2 + 1, -k2^2 + 1, -k3^2 + 1,
-(k1 - k2)^2 + 1, -(k2 - k3)^2 + 1, -(k3 - k4)^2 + 1, }, {}],
{1, 1, 1, 1, 1, 1, 1, 1}, -1]
```

There is one command that makes possible to apply sectors decomposition to integrals different from Feynman integrals:

`SDEvaluateDirect[var,function,degrees,order,deltas_optional]`.

Here `var` stands for the integration variable used in functions (for example, `x` goes for `x[1]`, `x[2]`, ...), `functions` is a list of functions and `degrees` is the list of their exponents. `order` is the requested order is epsilon, and `deltas` goes for the list of delta functions attached to the integrand. By default is is empty. For example, `{{1,3},{2,4}}` goes for the product of `Delta[x[1]+x[3]-1]` and `Delta[x[2]+x[4]-1]`.

Example:

```
SDEvaluateDirect[x, {1, x[1] x[2] x[3] + x[1] x[2] x[4] +
x[1] x[3] x[4] + x[1] x[2] x[5], x[1] x[3] + x[2] x[3] +
x[1] x[4] + x[2] x[4] + x[3] x[4] + x[1] x[5] + x[2] x[5] +
x[3] x[5]}, {1, -1 - 2 ep, -1 + 3 ep}, 0, {{1, 2, 3, 4, 5}}]
```

A similar syntax works for the expansion. In this example the integrator needs access to `Mathematica` to evaluate a `PolyGamma` function, so the path is passed with the `MathematicaBinary` argument:

`SDExpandDirect[var,function,degrees,expand_var,deltas]`.

Example:

```
MathematicaBinary="math";
SDEvaluateDirect[x, {1, x[1] x[2] x[3] + x[1] x[2] x[4] +
x[1] x[3] x[4] + x[1] x[2] x[5], x[1] x[3] + x[2] x[3] +
x[1] x[4] + x[2] x[4] + t (x[3] x[4] + x[1] x[5] +
x[2] x[5] + x[3] x[5])}, {1, -1 - 2 ep, -1 + 3 ep}, 0, t, 0,
{{1, 2, 3, 4, 5}}]
```

There is one more way to use FIESTA. An analytic Feynman integral evaluation method suggested by Lee [33, 34, 35, 36] needs to know for which values of space-time dimension d the integral can have poles. FIESTA can locate those values with the use of the following command:

SDAnalyze[{U,F,l},indices,dmin,dmax].

Here dmin and dmax are the ends of the interval where poles should be located. This syntax used only algebraic transformations, so the result is exact. However, the program might miss some pole cancellations, so some of the returned values might be "false alerts".

Example:

SDAnalyze[UF[{k},{-k^2,-(k+p$_1$)^2,-(k+p$_1$+p$_2$)^2,-(k+p$_1$+p$_2$+p$_4$)^2},
{p$_1^2$ →0,p$_2^2$ →0,p$_4^2$ →0, p$_1$ p$_2$ →-S/2,p$_2$ p$_4$ →-T/2,p$_1$ p$_4$ →(S+T)/2,
S→3,T→ 1}], {1,1,1,1},1,8]

Returned answer is {2,4} which means that the integrand has poles for d equal to 2 and 4.

5.3. Program options

FIESTA has the following options:

- DataPath: by default FIESTA stores databases in the temp subfolder. However one might wish to direct it elsewhere, especially if the folder with FIESTA is on a network disk. The database should be preferably stored on a fast local disk;

- NumberOfSubkernels: the number of subkernels that Mathematica launches. Might be set equal to the number of cores on the computer in use but should not exceed the number of licensed subkernels. This setting can speed up the integrand preparation;

- NumberOfLinks: the number of CIntegrate processes that will be launched. The name of this option is left for compatibility with old versions of FIESTA where each CIntegrate process was called by a separate MathLink connection. This setting corresponds to the parallelization during integration;

- CubaCores: the internal parallelization option of the integrators inside CIntegrate. By default the integrator uses one core, but it can be

changed with this option. This setting also corresponds to the parallelization during integration. Normally `NumberOfLinks` is more efficient, but there might be situations when increasing `CubaCores` leads to better results;

- `STRATEGY`: sector decomposition strategy. By default we use `STRATEGY_S` (our strategy), but there are also such options as `STRATEGY_B` (Bogner and Weinzierl), `STRATEGY_X` (Binoth and Heinrich), `STRATEGY_SS` (Speer sectors) and `STRATEGY_KU`, `STRATEGY_KU0`, `STRATEGY_KU2` (Kaneko and Ueda). Among the last three strategies the last variant is the full implementation of the algorithm from [27], the first two are faster but might result in more sectors;

- `QHullPath`: if one uses strategies `KU`, `KU0`, `KU2` or the new `SDExpandAsy` syntax, a correct path to the `qhull` executable should be provided. By default it is set to `qhull` assuming that the package is installed on the system, however one might provide a specific path;

- `CIntegratePath`: by default the integration pool chooses itself the integration binary, however one might provide another path;

- `UsingC`: by default this option is set to `True`. This means that `FIESTA` uses the `c++` integration. If set to `False`, it switches to `Mathematica` integration, however this is not recommended. With `UsingC` set to `False` the option `ExactIntegrationOrder` (if set) specifies the order in epsilon where `FIESTA` tries exact integration for some time. The default time is 10 seconds per sector and can be modified by the `ExactIntegrationTimeout` option;

- `UsingQLink`: by default this option is set to `True`. Switching it off will turn off database usage, however in `FIESTA 3` this is possible only together with `UsingC=False`;

- `OnlyPrepare`: by default this option is set to `False`. In this case the calculation is performed completely, otherwise a database is prepared for integration and a shell command to run `CIntegrate` without `Mathematica` is printed. This can be used if one are preparing a database on one computer and is integrating elsewhere, or if one is willing to try different integrators or precision requests;

- `SeparateTerms`: if `True`, the integrator receives integrable terms independently, not whole expressions for each sector; on one hand, this simplifies the integrands and the integrators return better precision, on the other hand the error grows after summing up the results, so normally there is no recommendation on whether to use this option or not. However, in the `MPI` mode this option might lead to a significant speedup since in this case it leads to a better parallelization;

- `ComplexMode` (`False` by default): with this setting set to `True` `FIESTA` performs a contour deformation in order to avoid poles in physical regions. The deformation size depends on a parameter, that is either set manually by giving a value to the `ComplexShift` variable or is tuned automatically in the interval from zero to `MaxComplexShift` (1 by default). Increasing the option `LambdaSplit` (4 by default) might result in better tuning but slows the preparation; same is true for the search option `LambdaIterations` that is set by default to 1000;

  The contour transformation cannot deal with cases where F turns to zero for the reason that some variables are equal to 1. In order to trace those cases, set `TestF1=True`. In order to handle those singularities, set the `BisectionVariables` equal to the list of variables such that the integration cube is divided into two parts. By default, the separation point is equal to 1/2, however this can be changed either by setting the `BisectionPoint` value, or by providing a list of `BisectionPoints`;

- `CurrentIntegrator`: (`vegasCuba` by default): the integrator used at the final stage. The options allowed in the current setup are `vegasCuba`, `suaveCuba`, `divonneCuba` and `cuhreCuba`. It is also possible to add your own integrators by modifying the `integrators.c` source file, however this is far beyond the standard usage of `FIESTA`. Related parameter (`CurrentIntegratorOptions`) presents a list of options of the currently chosen integrator (for details see [28]). By default it has no value and the actual options are printed out when one starts the evaluation (the defaults are stored within the `c++` part). The most commonly used integrator option is `maxeval`, which can be set, for example, by `CurrentIntegratorOptions = {{"maxeval","500000"}}`. The default value is `50000`. One more virtual integrator is `justEvaluate`. It simply evaluates the integrand at a given point, by default it is the point where all coordinates are equal to 1/2. Its options are `x1`, `x2` and

so on representing the integration coordinates;

- `SmallX`, `MPThreshold`, `PrecisionShift`, `MPPrecision`, `MPMinb`: the options that allow to fine-tune the MPFR subsystem. For details see the previous paper on FIESTA [14];

- `RegVar`: an option introduced for `SDExpandAsy` but usable also in other situations. Sets an extra regularization variable;

- `AnalyticIntegration`: an option used only for `SDExpandAsy`, `True` by default, tells `FIESTA` to try to take some integrations analytically after introducing regions;

- `FastASY`: (`False` by default) specifies the region search mode (used in `SDExpandAsy`). With `FastASY` set to `False` the polynomial `U` $\times$ `F` is analyzed, with `True` — the F polynomial. The `FastASY` variant might work significantly faster and will produce correct results almost all the time, but one should use it at his own risk;

- `PolesMultiplicity`: an option used only for `SDAnalyze`, `False` by default, changes the answer so that it returns not only values of `d` but also maximal pole multiplicities;

- `MathematicaBinary`: a path to the executable `Mathematica` kernel. If set, it is passed to the integration pool, so it can request `Mathematica` for values of functions it cannot evaluate itself (currently this feature is used only for `PolyGamma`);

- `BucketSize` (25 by default): an option tuning the database usage (for details see the documentation on KyotoCabinet); increasing this variable might result is faster database access, but increases the RAM usage;

- `MixSectors` (0 by default): lets `FIESTA` to sum up integrands in different sectors before integration;

- `RemoveDatabases` (`True` by default): specifies whether the database files should be removed after the integration;

- `d0` (4 by default): specifies the space-time dimension;

- `ReturnErrorWithBrackets`: (`False` by default) changes the output — with `True` the error estimates are printed as `pm[NUMBER]` instead of `pmNUMBER`;

- `FixSectors`: (`True` by default) — for the reason of fixing sector numbers and easier debugging we perform the variable substitutions stage on the main kernel. If one sets this option to false, this stage will be also made parallel, however it normally does not influence the total time much;

- `PrimarySectorCoefficients`: one might specify the list of coefficients before primary sectors. A zero means that this sector will be ignored. With this setting one can split the problem into parts and also take diagram symmetries into account;

- `NoDatabaseLock`: prevents `FIESTA` from locking the database. This may be away to avoid restrictions on some file systems but might result in corrupted databases.

5.4. CIntegratePool options

If one runs `FIESTA` with `OnlyPrepare=True`, then it prints out the command to be executed, for example,

```
bin/CIntegratePool -in /temp/db2in -out /temp/db2out
-all -direct -threads 4 -complex
```

After executing such a command and achieving a result, one might wish to rerun the integration with different options. Here we provide the list of arguments accepted by `CIntegratePool`:

- `-in`: provides the path to the database with integrands (without the .kch suffix);

- `-out`: provides the path to the database where results are stored;

- `-direct`: instructs `CIntegratePool` that it was called directly and not from `Mathematica`, so that it saves the results in the output database, does not use temporary files in order to transfer results back and prints the results to `stdout`;

- `-math`: provides the path to the `Mathematica` binary;

- `-bucket`: provides the bucket value for the output database;
- `-CIntegratePath`: provides the path to the `CIntegrate` binary. This can be either a full path (starting with `/`) or just a filename, in this case `CIntegratePool` searches for this file in the same directory. If this option is missing, it searches either for `CIntegrateMP` or `CIntegrateMPC` depending on whether the complex mode is on or off;
- `-integrator`: sets the integrator to be used. `-intpar` sets some integrator parameter, for example, `-intpar maxeval 500000`. For the list of integrators see section 5.3;
- `-MPThreshold`, `-MPPrecision`, `-PrecisionShift`, `-SmallX`, `-MPMin`: options that are fine-tuning the `MPFR` subsystem;
- `-threads`: sets the number of `CIntegrate` processes launched by the pool. This option is meaningless in the `MPI` mode;
- `-CubaCores`: sets the number of processes that the integrator starts for each integrand;
- `-test`: perform an integrator test only, `-notest`: do not perform this test, `-nopreparse`: do not perform parse check of all expressions before integration;
- `-complex`: specifies that the expression is complex. If one knows it do be real, this setting should not be used since it can slow down the integration a lot;
- `-all`: perform all integration; to the contrary, `-task` followed by a number instruct the code to evaluate only expression related to one `SDEvaluate` call. Normally, there is only one task in the database, so one would call `-task 1`, but the `SDExpandAsy` mode uses multiple tasks. `-prefix` can be used only when `-task` is set and tell the program to integrate only with given powers of ε and `RegVar`. For example `-task 1 -prefix "{-2,-{1, 2}}"` corresponds to integrals having ε order -2 and `RegVar` coefficient `RegVar * Ln [RegVar]^2`;
- `-separate_terms`: instructs the algorithm not to group expressions by sectors and to integrate each integrable term separately, might be useful if one is using massive `MPI` parallelization;

- `-testF`: instead of the integration, the code checks whether the sign of the imaginary part of F is negative. Might be used for debugging special cases in complex mode, for details see section 2;

- `-debug`: used to print all integration results.

5.5. CIntegrate options

The integration pool program, `CIntegratePool` or `CIntegratePoolMPI` distributes tasks between on of the integration programs provided with the package — `CIntegrate`, `CIntegrateMP`, `CIntegrateMPC` or even something external. However, the integration programs can be called on their own. They have no options on start, accept input from `stdin` and print it to `stdout`. Each command sent to the program is ended with a new line symbol.

The main command to be provided to the program input is `Integrate`. After that one should send the expression. It consists of a number of lines, each of them should be ended with the `;` symbol. At the end should be a line consisting of the `|` symbol. The expression lines are the following:

- The number of variables;

- The number of intermediate functions;

- A number of lines each representing an intermediate expression. If the second line is `0;`, then this part should be missing;

- The final expression.

The expressions might contain algebraic operations such as `+, -, *, /`, bracket symbols, numbers with a floating point. The integration variables should be referred as `x[1], x[2]` and so on, intermediate functions as `f[1], f[2]` and so on. Power is represented as `p[expr,exponent]`, natural logarithm as `l[expr]`. One can also use `P` for π and `G` for `EulerGamma`. `PolyGamma[arg1,arg2]` also works but one needs to provide a path to the `Mathematica` binary — the integrator cannot evaluate this function on its own.

Example:

```
1;
0;
x[0]+0.2;
```

|

If one feeds this example into the `CIntegrate` program after the `Integrate` command then it will result in integrating $x + 0.2$ from 0 to 1.

The program also accepts the following commands (most of them require an argument passed as next line):

- `Parse`: same as `Integrate`, but the expression is only parsed;
- `Exit`: quit the program;
- `CubaCores`: sets the number of cores used by the integrator, default value is 1;
- `SetMath`: provides a path to the `Mathematica` binary;
- `SetIntegrator`: sets the integrator to be used;
- `SetCurrentIntegratorParameter`: sets one of the integrator parameters, the next line should be the parameter name, the line after that — the value;
- `GetCurrentIntegratorParameters`: simply returns the list of current parameters and their values;
- `MPFR`, `Native` and `Mixed` (default variant): chooses whether the integrand should use `MPFR` everywhere, the double precision or the mixed mode. The mixed mode used the following five options to determine in which parts of the integration cube which arithmetics should be used: `SetMPPrecision`, `SetMPPrecisionShift`, `SetMPMin`, `SetMPThreshold`, `SetSmallX`;
- `Debug`: makes the code print values in all integration points;
- `TestF`: instead of the integration the code checks the sign of the imaginary part of the integrand;
- `Help`: list all those commands.

5.6. Setup for cluster calculations

In order to perform integrations on a cluster one should do the following:

1. Run `FIESTA` with `OnlyPrepare=True`.
2. Save the database file produced by the integrand preparation stage.
3. Use the command printed out by the `Mathematica` part to launch it on a cluster. One can replace the call to `CIntegratePool` with `CIntegratePoolMPI` in order to use the `MPI` version. The syntax to launch `MPI` program varies, so one will have to use instructions for the cluster in use. One can also adjust the integrator options, for example by increasing the number of sampling points.
4. The result is printed out in the `c++` log and also saved to a small output database. To see the result in the `Mathematica` form one should load `FIESTA`, provide the proper `DataPath` and run `GenerateAnswer[]`. No more options are required.

6. Conclusion

We have presented the new version of `FIESTA` — a program for automatic numerical evaluation and analytic expansion of Feynman integrals. The new version contains new sector decomposition and expansion algorithms, provides possibilities to integrate in physical regions and to perform cluster parallelization. We believe that this upgrade is an essential improvement in automatic numerical evaluations of Feynman integrals. `FIESTA` development is not over. One of the future plans is to make use of `GPU`s in order to speed up the integration.

Acknowledgements

This work was supported by DFG through SFB/TR 9. I would like to thank P. Marquard for numerous tests of the beta version of `FIESTA`, M. Tentyukov for advices on code optimization and V. Smirnov and M.Steinhauser for ongoing support as well as the careful reading of the draft of this paper.

References

[1] T. Binoth, G. Heinrich, An automatized algorithm to compute infrared divergent multi-loop integrals, Nucl. Phys. B585 (2000) 741–759.

[2] T. Binoth, G. Heinrich, Numerical evaluation of multi-loop integrals by sector decomposition, Nucl. Phys. B680 (2004) 375–388.

[3] T. Binoth, G. Heinrich, Numerical evaluation of phase space integrals by sector decomposition, Nucl. Phys. B693 (2004) 134–148.

[4] G. Heinrich, Sector Decomposition, Int. J. Mod. Phys. A23 (2008) 1457–1486.

[5] C. Bogner, S. Weinzierl, Resolution of singularities for multi-loop integrals, Comput. Phys. Commun. 178 (2008) 596–610.

[6] C. Bogner, S. Weinzierl, Blowing up Feynman integrals, Nucl. Phys. Proc. Suppl. 183 (2008) 256–261.

[7] J. Carter, G. Heinrich, SecDec: A general program for sector decomposition, Comput.Phys.Commun. 182 (2011) 1566–1581.

[8] S. Borowka, J. Carter, G. Heinrich, Numerical Evaluation of Multi-Loop Integrals for Arbitrary Kinematics with SecDec 2.0, Comput.Phys.Commun. 184 (2013) 396–408.

[9] S. Borowka, J. Carter, G. Heinrich, SecDec: A tool for numerical multi-loop calculations, J.Phys.Conf.Ser. 368 (2012) 012051.

[10] S. Borowka, G. Heinrich, Numerical evaluation of massive multi-loop integrals with SecDec, PoS LL2012 (2012) 038.

[11] S. Borowka, G. Heinrich, Massive non-planar two-loop four-point integrals with SecDec 2.1, Comput.Phys.Commun. 184 (2013) 2552–2561.

[12] S. Borowka, G. Heinrich, Numerical multi-loop calculations with SecDec (2013).

[13] A. V. Smirnov, M. N. Tentyukov, Feynman Integral Evaluation by a Sector decomposiTion Approach (FIESTA), Comput. Phys. Commun. 180 (2009) 735–746.

[14] A. V. Smirnov, V. A. Smirnov, M. Tentyukov, FIESTA 2: parallelizeable multiloop numerical calculations, Comput. Phys. Commun. 182 (2011) 790–803.

[15] D. E. Soper, Techniques for QCD calculations by numerical integration, Phys.Rev. D62 (2000) 014009.

[16] Y. Kurihara, T. Kaneko, Numerical contour integration for loop integrals, Comput.Phys.Commun. 174 (2006) 530–539.

[17] T. Binoth, J. P. Guillet, G. Heinrich, E. Pilon, C. Schubert, An Algebraic/numerical formalism for one-loop multi-leg amplitudes, JHEP 0510 (2005) 015.

[18] Z. Nagy, D. E. Soper, Numerical integration of one-loop Feynman diagrams for N-photon amplitudes, Phys.Rev. D74 (2006) 093006.

[19] C. Anastasiou, S. Beerli, A. Daleo, Evaluating multi-loop Feynman diagrams with infrared and threshold singularities numerically, JHEP 0705 (2007) 071.

[20] K. G. Chetyrkin, F. V. Tkachov, Integration by Parts: The Algorithm to Calculate beta Functions in 4 Loops, Nucl. Phys. B192 (1981) 159–204.

[21] A. Pak, A. V. Smirnov, Geometric approach to asymptotic expansion of Feynman integrals, Eur. Phys. J. C71 (2011) 1626.

[22] B. Jantzen, A. V. Smirnov, V. A. Smirnov, Expansion by regions: revealing potential and Glauber regions automatically, Eur.Phys.J. C72 (2012) 2139.

[23] M. Beneke, V. A. Smirnov, Asymptotic expansion of Feynman integrals near threshold, Nucl. Phys. B522 (1998) 321–344.

[24] V. A. Smirnov, Problems of the strategy of regions, Phys. Lett. B465 (1999) 226–234.

[25] J. M. Henn, A. V. Smirnov, V. A. Smirnov, Evaluating single-scale and/or non-planar diagrams by differential equations (2013).

[26] A. V. Smirnov, V. A. Smirnov, Hepp and Speer Sectors within Modern Strategies of Sector Decomposition, JHEP 05 (2009) 004.

[27] T. Kaneko, T. Ueda, A geometric method of sector decomposition, Comput. Phys. Commun. 181 (2010) 1352–1361.

[28] T. Hahn, CUBA: A Library for multidimensional numerical integration, Comput.Phys.Commun. 168 (2005) 78–95.

[29] V. Sadovnichy, A. Tikhonravov, V. Voevodin, V. Opanasenko, "Lomonosov": Supercomputing at Moscow State University, in: Contemporary High Performance Computing: From Petascale toward Exascale, Chapman & Hall/CRC Computational Science, Boca Raton, United States, Boca Raton, United States, 2013, pp. 283–307.

[30] V. Pilipp, Semi-numerical power expansion of Feynman integrals, JHEP 09 (2008) 135.

[31] V. A. Smirnov, Analytical result for dimensionally regularized massless on-shell double box, Phys. Lett. B460 (1999) 397–404.

[32] J. B. Tausk, Non-planar massless two-loop Feynman diagrams with four on-shell legs, Phys. Lett. B469 (1999) 225–234.

[33] R. Lee, Space-time dimensionality D as complex variable: Calculating loop integrals using dimensional recurrence relation and analytical properties with respect to D, Nucl.Phys. B830 (2010) 474–492.

[34] R. N. Lee, A. V. Smirnov, V. A. Smirnov, Analytic Results for Massless Three-Loop Form Factors, JHEP 04 (2010) 020.

[35] R. N. Lee, A. V. Smirnov, V. A. Smirnov, Master Integrals for Four-Loop Massless Propagators up to Transcendentality Weight Twelve, Nucl. Phys. B856 (2012) 95–110.

[36] R. N. Lee, V. A. Smirnov, The Dimensional Recurrence and Analyticity Method for Multicomponent Master Integrals: Using Unitarity Cuts to Construct Homogeneous Solutions, JHEP 1212 (2012) 104.