

SecDec: a toolbox for the numerical evaluation of multi-scale integrals

Stephan Jahn*

Max-Planck-Institut für Physik, München, GERMANY

E-mail: sjahn@mpp.mpg.de

We present a new version of `SECDEC`, a program for the numerical computation of parametric integrals in the context of dimensional regularization. By its modular structure, the `python` rewrite `pySECDEC` is much more customizable than earlier versions of `SECDEC`. The numerical integration is accelerated using code optimization available in `FORM`. With the new `C++` interface, `pySECDEC` can provide numerical solutions of analytically unknown integrals in user-defined code.

*13th International Symposium on Radiative Corrections (Applications of Quantum Field Theory to Phenomenology)
25-29 September, 2017
St. Gilgen, Austria*

*Speaker.

1. Introduction

Future runs of the LHC and the subsequent high-luminosity upgrade will allow for New Physics searches mostly via precision tests of the standard model throughout the next few decades. Higher order calculations are crucial for the upcoming data-taking to be meaningful. A well established procedure for multi-loop calculations is: (i) draw all contributing Feynman diagrams, (ii) insert the Feynman rules to produce algebraic expressions, (iii) identify the loop integrals to be computed, (iv) reduce these integrals to a set of master integrals, and (v) solve the master integrals.

Analytical solutions for the master integrals are desirable and recent developments made a huge class of integrals analytically tractable. However, phenomenologically relevant topologies, for example some nonplanar double-boxes with massive propagators, remain inaccessible with the currently available analytical methods.

Sector decomposition [1, 2] provides a numerical alternative when analytic solutions are out of reach with current techniques. A prominent example for the successful application of this method is the recent calculations of the full top-mass dependence in Higgs boson pair production [3, 4]. Further studies [5] indicate that picking a finite basis [6, 7, 8] makes the numerical approach competitive with analytical solutions.

In this article, we present `pySECDEC` [9], the successor of the program `SECDEC` [10, 11, 12]. Other public implementations of sector decomposition are `sector_decomposition` [13] supplemented with `CSectors` [14] and `FIESTA` [15, 16, 17, 18].

The structure of these proceedings is as follows: In Section 2 we list the key new features of `pySECDEC` compared to `SECDEC` 3. In Section 3 we illustrate the workflow from defining an integral to a sufficiently precise numerical result, including instructions how to tune numerical integration on failure. Finally, we close with concluding remarks in Section 4.

2. New features

The major rewrite in `python`, `FORM`, and `C++` is completely independent of expensive commercial programs. Rather than a text file, `pySECDEC` creates a `C++` library that can be used to dynamically evaluate loop integrals occurring in amplitude calculations.

The time needed for the numerical integration is significantly reduced by optimizing the integrand functions with `FORM` [19, 20].

Another significant speedup in the numerics could be achieved by identifying sectors that are equal up to permutations of the integration variables (Feynman parameters). Instead of integrating all equivalent sectors, only one of them is numerically integrated while the others are accounted for by multiplying the number of equivalent sectors. We implement two independent algorithms to identify equivalent sectors: The algorithm described in [21] and an algorithm based on finding graph isomorphisms with `dreadnaut` [22], see [23] for details.

The sign of the imaginary part of the 2nd Symanzik polynomial \mathcal{F} is now checked at every point in Feynman parameter space to make sure that the Feynman prescription is always fulfilled. Furthermore, we also check that the real part of the 1st Symanzik polynomial \mathcal{U} stays positive on the entire contour, which is important if \mathcal{U} has a negative exponent and in higher orders of the ε -expansion where $\log \mathcal{U}$ appears.

An improved treatment of singularities originating from $x_i = 1$ allows for the computation of integrals `SECDEC 3` could not handle, e.g. the three-point function depicted in Figure 1, which occurs in single-Z-boson production [24].

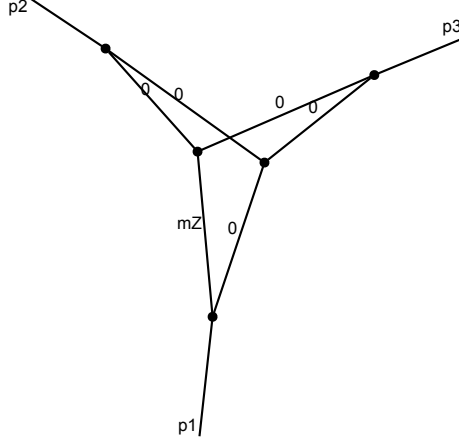


Figure 1: Three-point function with one massive propagator (m_Z) and $s = p_3^2 = m_Z^2$.

For computations beyond the traditional use cases of `SECDEC`, all individual parts of the algebraic manipulation are separately available to the user. For example, it is possible to have `pySECDEC` only perform the Feynman parametrization and further process it independently.

As part of `pySECDEC`, we provide the C++ library `secdecutil`. Class templates that are useful in a more general context than sector decomposition are implemented therein, for instance a class template that defines addition, subtraction, multiplication, and division of series expansions. Users are encouraged to use the low-level functionalities of `pySECDEC` to write their own high-level routines suitable for the problem at hand.

3. Usage

The typical usage of `pySECDEC` is to compute integrals of the form

$$I = p_0(\{a\}, \{\epsilon\}) \int_0^1 dx_1 \dots \int_0^1 dx_N \prod_{i=1}^M f_i(\{x\}, \{a\})^{b_i + \sum_k c_{ik} \epsilon_k}, \quad (3.1)$$

where p_0 is an arbitrary prefactor, the f_i are polynomials in $\{x\} = \{x_1, \dots, x_N\}$ with coefficients $\{a\}$, the b_i and c_i are numeric constants, and the ϵ_k are analytic regulators. Typically, I has regulated divergences such that $I|_{\epsilon_k=0} = \infty$. A generic (multi-)loop integral of the form

$$G = \int \prod_{l=1}^L d^D \kappa_l \frac{1}{\prod_{j=1}^N P_j^{v_j}(\{k\}, \{p\}, m_j^2)}$$

$$d^D \kappa_l = \frac{\mu^{4-D}}{i\pi^{\frac{D}{2}}} d^D k_l, \quad P_j(\{k\}, \{p\}, m_j^2) = (q_j^2 - m_j^2 + i\delta), \quad (3.2)$$

where the q_j are linear combinations of external momenta p_i and loop momenta k_l , can be brought to a parametric form as in equation (3.1),

$$G = (-1)^{N_v} \frac{\Gamma(N_v - LD/2)}{\prod_{j=1}^N \Gamma(v_j)} \int_0^\infty \prod_{j=1}^N dx_j x_j^{v_j-1} \delta(1 - \sum_{l=1}^N x_l) \frac{\mathcal{U}^{N_v - (L+1)D/2}}{\mathcal{F}^{N_v - LD/2}}, \quad (3.3)$$

where $N_v = \sum_j v_j$, by Feynman parametrization. This step is implemented in `pySECDDEC`; i.e. integrals like in equation (3.2) can directly be passed to `pySECDDEC`. `pySECDDEC` also implements Feynman parametrization of tensor integrals, which is omitted in the equations above for brevity.

The result is presented as series expansion in the analytic regulators with numerical coefficients. Obtaining a result with `pySECDDEC` generally takes three steps: (i) Define the integral, (ii) generate and build the C++ library for numerical integration, (iii) insert numerical values for the parameters $\{a\}$ (Mandelstam invariants for loop integrals) and perform the numerical integration. In the following, we guide through the most important matters on the way to a numerical result.

3.1 Definition of an integral

Integrals are communicated to `pySECDDEC` by running a python script, often referred to as "runcard", that calls `make_package` (`loop_package` for loop integrals). A minimal input to `make_package` consists of an integral name, the integration variables, the regulators, the maximum orders to expand in the regulators, and the integrand. Figure 2b shows a minimal runcard to compute

$$\int_0^1 dx \int_0^1 dy (x+y)^{-2+\varepsilon} \quad (3.4)$$

up to finite order in ε . The integrand can be given as a list of the polynomials f_i in the argument "polynomials_to_decompose". If some of the input polynomials don't give rise to divergences, e.g. numerators of loop integrals, they can be passed via the argument "other_polynomials" instead. `pySECDDEC` fully decomposes the "polynomials_to_decompose" but does not force full decomposition of the "other_polynomials". As a consequence, less sectors and therefore less integrand functions are generated. It may, however, be beneficial to decompose such polynomials anyways. The reason is that decomposed polynomials can have a lower intrinsic variance which is proportional to the error of any Monte Carlo integrator. It strongly depends on the integral at hand if decomposing these polynomials yields better results. If your integral has poles originating from the endpoint 1, for example from polynomials like $(1-x)$ or $(x-y)$, the "split" option should be set to "True".

A runcard to compute a loop integral consists of two steps. First, the loop integral is defined by either instantiating `LoopIntegralFromPropagators` or `LoopIntegralFromGraph`. These classes perform the Feynman parametrization given a list of propagators or a graph, respectively. Advanced python programmers can use these classes in their own code, for example to analytically process the Symanzik polynomials \mathcal{U} and \mathcal{F} . We refer the reader to the package documentation for a full description of Feynman parametrization in `pySECDDEC`. Second, the library for numerical integration is generated by a call to "loop_package". An example runcard how to print the Feynman prefactor $(-1)^{N_v} \Gamma(N_v - LD/2) / \prod_{j=1}^N \Gamma(v_j)$ and generate the integration library of the three-point function depicted in Figure 1 is shown in Figure 2b. Note that we set

```

1 from pySecDec import make_package
2
3 make_package(
4     name = 'easy',
5     integration_variables = ['x', 'y'],
6     regulators = ['eps'],
7
8     requested_orders = [0],
9     polynomials_to_decompose = ['(x+y)^(-2+eps)'],
10 )

```

(a)

```

1 from pySecDec.loop_integral import loop_package
2 from pySecDec.loop_integral import LoopIntegralFromPropagators
3
4 # define the loop integral
5 three_point = LoopIntegralFromPropagators(
6     loop_momenta = ['k1', 'k2'],
7     external_momenta = ['p1', 'p2', 'p3', 'p4'],
8     propagators = ['k1^2', '(k1+p1+p2)^2', '(k1-k2)^2',
9                   '(k1-k2+p1)^2-mZ^2', '(k2)^2', '(k2+p2)^2'],
10    replacement_rules = [
11        ('p1*p1', 0),
12        ('p2*p2', 0),
13        ('p3*p3', 's'),
14        ('p1*p2', 's/2'),
15        ('p2*p3', '-s/2'),
16        ('p1*p3', '-s/2'),
17        ('s', 'mZ^2')
18    ]
19 )
20
21 # print the prefactor from Feynman parametrization
22 print('prefactor from Feynman parametrization:')
23 print(three_point.Gamma_factor)
24
25 # generate the library
26 loop_package(
27     name = 'three_point',
28     additional_prefactor = '-exp(2*EulerGamma*eps)',
29     loop_integral = three_point,
30     complex_parameters = ['mZ'],
31     requested_order = 0,
32     split = True,
33 )

```

(b)

Figure 2: Example runcards for `pySecDec`. (a) Runcard to compute the integral defined in equation (3.4). (b) Runcard to compute the three-point function depicted in Figure 1.

“split” to “True” in our example which is required due to the special kinematics $s = m_Z^2$. Further note that m_Z is declared as complex parameter allowing for a nonzero width. We also define an “additional_prefactor” which in our example means we compute $-\exp(2\gamma_E \epsilon)G$ where G is the three-point function. This is a change of convention compared to `SecDec 3` where the additional prefactor was divided out rather than multiplied. The resulting “prefactor” visible in the numerics (see next section) is the product of the Feynman prefactor and the “additional_prefactor” provided by the user. By default, the “additional_prefactor” is one and therefore the prefactor in the numerics defaults to the Feynman prefactor.

3.2 Numerical integration

The following description applies for both, libraries generated with `make_package` and with `loop_package`. In fact, `loop_package` internally calls `make_package` after Feynman parametrization. Therefore, the distinction between the loop and the general branch present in `SecDec 3` no longer exists and consequently all features that make sense in both cases are available in both.

To generate the library for numerical integration, run `python` on the runcard first, then run `make` in the generated directory. For example, if you save the code of Figure 2b to a file called `generate.py`, run `python generate.py`. That generates a directory specified by the “name” argument (“easy” in our example). To build the library run `make -C easy`. The level of parallelization for building the library can be controlled from the command line via the environment variable `FORMTHREADS` (default: 2) and the `-j` option of `make`. The variable `FORMTHREADS` is

forwarded as `-w` option in every invocation of FORM; i.e. it controls how many threads FORM uses to process each single sector. The `-j` option of bash controls how many instances of FORM and the compiler are launched in parallel; i.e. how many sectors are processed at the same time. For example, to build the library using up to 4 processes, and 3 FORM threads per process run `make -j4 FORMTHREADS=3 -C easy`.

We wrap the CUBA [25, 26] library and `cquad` as implemented in the GNU scientific library (GSL) [27] for easy use with `pySECDEC`. Both packages are redistributed with `pySECDEC` and automatically available on systems where `pySECDEC` is properly installed.

```

1  from pySecDec.integral_interface import IntegralLibrary
2
3  # load c++ library
4  integral = IntegralLibrary('three_point/three_point_pylink.so')
5
6  # choose and configure integrators
7  integral.use_Divonne(epsrel=1e-5,epsabs=1e-5,maxeval=10**7,border=1e-8)
8  integral.use_CQuad(epsrel=1e-5,epsabs=1e-5) # 1D only
9
10 # integrate
11 result_without_prefactor, prefactor, full_result = integral(
12
13     # set parameter "mZ" (complex mass scheme)
14     complex_parameters=[91.19-1.25j],
15
16     # configure contour optimization (optional)
17     number_of_presamples=10**5,
18     deformation_parameters_maximum=1.0,
19     deformation_parameters_minimum=1e-5,
20
21 )
22
23 # print result
24 print('Numerical Result:')
25 print(full_result)

```

Figure 3: Example script to issue the numerical integration using the python interface.

The easiest way to issue the numerical integration is the `python` interface. The integration script shown in Figure 3 loads the necessary libraries, configures the integrators, runs the contour optimization and numerical integration, and prints the result. To run the numerical integration of your integral, write an integration script analogous to the one depicted in Figure 3. Assuming the integration script is called `integrate.py`, running `python integrate.py` will perform the integration and print the result to the screen.

The numerical accuracy goals of all integrators are controlled by the options “`epsrel`” and “`epsabs`”. In addition, CUBA integrators also implement “`maxeval`”. The integration terminates when either goal is reached. If your integral evaluates to a very small value, make sure that you decrease “`epsabs`” accordingly. The integrator `CQuad` is restricted to one dimensional integrands. A call to `integral.use_CQuad` makes `pySECDEC` call `CQuad` for every one dimensional integral while the integrator that was assigned before is used for higher dimensional integrals. Without call to `integral.use_CQuad`, the CUBA integrator is also used for one dimensional integrals.

When using a CUBA integrator, the number of threads used for the integration is controlled by the environment variable `CUBACORES`. For example, to run with 6 threads invoke the script as `CUBACORES=6 python integrate_triangle2L.py`.

The parameters left symbolic in the generation step must be set to numerical values when invoking the numerical integration. The call to `integral` takes a list of “real_paramters” and “complex_parameters”. If there are no real or complex parameters, the argument may be omitted (“real_paramters” in our example). Complex numbers (the Z-boson mass $91.19 - 1.25j$ in our example) are entered as `#±#j` in the python interface. The call that invokes the main integration also takes parameters that guide the optimization of the deformed contour. These parameters should usually be omitted; i.e. left at their default values unless the integration terminates due to a “sign_check_error”.

A “sign_check_error” means that the chosen contour in the analytic continuation to the complex plane is invalid. Since the contour deformation is by construction [28, 29, 30, 11, 31] guaranteed to be valid with sufficiently small deformation parameters, you should try to decrease “deformation_parameters_maximum”. We suggest to repeatedly lower “deformation_parameters_maximum” by an factor of ten until the integration succeeds. If your integral evaluates to a very small number, you may have to decrease the “deformation_parameters_maximum” below $1e-5$ which is the default value of “deformation_parameters_minimum”. If that happens, make sure that you also decrease “deformation_parameters_minimum” to keep it smaller than the maximum.

Note that often large cancellations giving rise to numerical instabilities occur in the integrand functions when the integration variables are close to zero. This can be problematic for adaptive integrators, in particular Divonne. It is usually necessary to set a nonzero border when using Divonne to avoid samples in unstable boundary regions. If you keep getting NaN, you should switch to Vegas and increase the default values for “nincrease” (default: 500) and “nstart” (default: 1000) by a factor of ten or more. Starting from `pySECDEC` version 1.3, you can also use the new `zero_border` option to avoid evaluation of the integrand close to the zero boundary.

4. Conclusion

We have presented `pySECDEC`, the successor of the `SECDEC 3`, a program that numerically computes parameter integrals, e.g. (multi-)loop integrals after Feynman parametrization, in the context of dimensional regularization. The open-source package can be downloaded from <http://secdec.hepforge.org/>.

A key new feature is the output of a C++ library, which is also accessible from `python`. Existing sector decomposition programs are mainly designed to provide cross-checks against analytic calculations. The different design goal makes dynamically embedding them into user-defined calculations a difficult task. However, there is foreseeable need for numerical solutions of integrals for which analytical solutions will not become available in the near future. The situation motivated the development of a program that makes both use cases easy, usage as static cross-check and as dynamic library.

`pySECDEC` is a tool that satisfies the aforementioned demands. It can dynamically provide numerical solutions of loop-integrals in user-defined codes, for example amplitude calculations. Static cross-checks can be performed with a program that evaluates the integral at a single point. It

turns out that writing such a program is as easy as writing a runcard with a suitable library interface. Nevertheless, a library opens the door to a much wider field of use cases compared to runcards.

We have demonstrated how integrals are communicated to `pySECDEC` and explained how to write `python` scripts that evaluate the integral using the `python` interface. Note that the scripts shown in this article can be embedded into arbitrary user-defined `python` code. However, the `python` interface provides only a fraction of the library's functionality. We therefore recommend to switch to the more complex but much more powerful `C++` interface for dynamical library use.

Acknowledgments

I thank my fellow `SECDEC` developer team, Sophia Borowka, Gudrun Heinrich, Stephen Jones, Matthias Kerner, Johannes Schlenk, and Tom Zirke, for helpful discussions and the successful collaboration.

References

- [1] T. Binoth and G. Heinrich, *An automatized algorithm to compute infrared divergent multi-loop integrals*, *Nucl. Phys.* **B585** (2000) 741–759, [[hep-ph/0004013](#)].
- [2] G. Heinrich, *Sector Decomposition*, *Int. J. Mod. Phys.* **A23** (2008) 1457–1486, [[0803.4177](#)].
- [3] S. Borowka, N. Greiner, G. Heinrich, S. Jones, M. Kerner, J. Schlenk et al., *Higgs Boson Pair Production in Gluon Fusion at Next-to-Leading Order with Full Top-Quark Mass Dependence*, *Phys. Rev. Lett.* **117** (2016) 012001, [[1604.06447](#)].
- [4] S. Borowka, N. Greiner, G. Heinrich, S. P. Jones, M. Kerner, J. Schlenk et al., *Full top quark mass dependence in Higgs boson pair production at NLO*, *JHEP* **10** (2016) 107, [[1608.04798](#)].
- [5] A. von Manteuffel and R. M. Schabinger, *Numerical Multi-Loop Calculations via Finite Integrals and One-Mass EW-QCD Drell-Yan Master Integrals*, *JHEP* **04** (2017) 129, [[1701.06583](#)].
- [6] A. von Manteuffel, E. Panzer and R. M. Schabinger, *On the Computation of Form Factors in Massless QCD with Finite Master Integrals*, *Phys. Rev.* **D93** (2016) 125014, [[1510.06758](#)].
- [7] E. Panzer, *On hyperlogarithms and Feynman integrals with divergences and many scales*, *JHEP* **1403** (2014) 071, [[1401.4361](#)].
- [8] A. von Manteuffel, E. Panzer and R. M. Schabinger, *A quasi-finite basis for multi-loop Feynman integrals*, *JHEP* **02** (2015) 120, [[1411.7392](#)].
- [9] S. Borowka, G. Heinrich, S. Jahn, S. P. Jones, M. Kerner, J. Schlenk et al., *pySecDec: a toolbox for the numerical evaluation of multi-scale integrals*, *Comput. Phys. Commun.* **222** (2018) 313–326, [[1703.09692](#)].
- [10] J. Carter and G. Heinrich, *SecDec: A general program for sector decomposition*, *Comput. Phys. Commun.* **182** (2011) 1566–1581, [[1011.5493](#)].
- [11] S. Borowka, J. Carter and G. Heinrich, *Numerical Evaluation of Multi-Loop Integrals for Arbitrary Kinematics with SecDec 2.0*, *Comput. Phys. Commun.* **184** (2013) 396–408, [[1204.4152](#)].
- [12] S. Borowka, G. Heinrich, S. P. Jones, M. Kerner, J. Schlenk and T. Zirke, *SecDec-3.0: numerical evaluation of multi-scale integrals beyond one loop*, *Comput. Phys. Commun.* **196** (2015) 470–491, [[1502.06595](#)].

- [13] C. Bogner and S. Weinzierl, *Resolution of singularities for multi-loop integrals*, *Comput.Phys.Commun.* **178** (2008) 596–610, [0709.4092].
- [14] J. Gluza, K. Kajda, T. Riemann and V. Yundin, *Numerical Evaluation of Tensor Feynman Integrals in Euclidean Kinematics*, *Eur.Phys.J.* **C71** (2011) 1516, [1010.1667].
- [15] A. Smirnov and M. Tentyukov, *Feynman Integral Evaluation by a Sector decomposition Approach (FIESTA)*, *Comput.Phys.Commun.* **180** (2009) 735–746, [0807.4129].
- [16] A. Smirnov, V. Smirnov and M. Tentyukov, *FIESTA 2: Parallelizable multiloop numerical calculations*, *Comput.Phys.Commun.* **182** (2011) 790–803, [0912.0158].
- [17] A. V. Smirnov, *FIESTA 3: cluster-parallelizable multiloop numerical calculations in physical regions*, *Comput.Phys.Commun.* **185** (2014) 2090–2100, [1312.3186].
- [18] A. V. Smirnov, *FIESTA4: Optimized Feynman integral calculations with GPU support*, *Comput. Phys. Commun.* **204** (2016) 189–199, [1511.03614].
- [19] J. Kuipers, T. Ueda and J. A. M. Vermaseren, *Code Optimization in FORM*, *Comput. Phys. Commun.* **189** (2015) 1–19, [1310.7007].
- [20] B. Ruijl, T. Ueda and J. Vermaseren, *FORM version 4.2*, 1707.06453.
- [21] A. Pak, *The Toolbox of modern multi-loop calculations: novel analytic and semi-analytic techniques*, *J. Phys. Conf. Ser.* **368** (2012) 012049, [1111.0868].
- [22] B. D. McKay and A. Piperno, *Practical graph isomorphism, ii*, *Journal of Symbolic Computation* **60** (2014) 94 – 112.
- [23] Jones, Stephen P., *To appear in the proceedings of the 18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017)*, *Journal of Physics: Conference Series* (2017) .
- [24] I. Dubovyk, A. Freitas, J. Gluza, T. Riemann and J. Usovitsch, *30 years, some 700 integrals, and 1 dessert, or: Electroweak two-loop corrections to the $Z\bar{b}b$ vertex*, *PoS LL2016* (2016) 075, [1610.07059].
- [25] T. Hahn, *CUBA: A library for multidimensional numerical integration*, *Comput. Phys. Commun.* **168** (2005) 78–95, [hep-ph/0404043].
- [26] T. Hahn, *Concurrent Cuba*, 1408.6373.
- [27] M. Galassi and al., *GNU Scientific Library Reference Manual - Third Edition*. Network Theory Ltd., 3rd ed., 2009.
- [28] D. E. Soper, *Techniques for QCD calculations by numerical integration*, *Phys. Rev.* **D62** (2000) 014009, [hep-ph/9910292].
- [29] T. Binoth, J. P. Guillet, G. Heinrich, E. Pilon and C. Schubert, *An Algebraic/numerical formalism for one-loop multi-leg amplitudes*, *JHEP* **10** (2005) 015, [hep-ph/0504267].
- [30] C. Anastasiou, S. Beerli and A. Daleo, *Evaluating multi-loop Feynman diagrams with infrared and threshold singularities numerically*, *JHEP* **05** (2007) 071, [hep-ph/0703282].
- [31] J. Schlenk and T. Zirke, *Calculation of Multi-Loop Integrals with SecDec-3.0*, in *Proceedings, 12th International Symposium on Radiative Corrections (Radcor 2015) and LoopFest XIV (Radiative Corrections for the LHC and Future Colliders): Los Angeles, CA, USA, June 15-19, 2015*, 2016, 1601.03982, <https://inspirehep.net/record/1415599/files/arXiv:1601.03982.pdf>.