Eur. Phys. J. C (2019) 79:256
https://doi.org/10.1140/epjc/s10052-019-6769-5

THE EUROPEAN PHYSICAL JOURNAL C

Regular Article - Theoretical Physics

BasisGen: automatic generation of operator bases

Juan Carlos Criado[a]

CAFPE, Departamento de Física Teórica y del Cosmos, Universidad de Granada, Campus de Fuentenueva, 18071 Granada, Spain

Received: 18 January 2019 / Accepted: 11 March 2019 / Published online: 21 March 2019

Abstract BasisGen is a Python package for the automatic generation of bases of operators in effective field theories. It accepts any semisimple symmetry group and fields in any of its finite dimensional irreducible representations. It takes into account integration by parts redundancy and, optionally, the use of equations of motion. The implementation is based on well-known methods to generate and decompose representations using roots and weights, which allow for fast calculations, even with large numbers of fields and high-dimensional operators. BasisGen can also be used to do some representation-theoretic operations, such as finding the weight system of an irreducible representation from its highest weight or decomposing a tensor product of representations.

1 Introduction

Effective field theory is a widely used framework for parameterizing the physics of systems whose degrees of freedom and symmetries are known. An effective Lagrangian is a linear combination of all local operators that can constructed with the fields in the theory, with the restriction that they are invariant under the action of the symmetry group. Usually, there is some other constraint that reduces the number of possibilities to a finite one, such as imposing a maximum canonical dimension. In this context, it is often convenient to obtain a complete set of independent operators, which is called a basis. BasisGen automatizes this task.

The input data needed for this calculation are the symmetry group G of the theory and the representation of G corresponding to each field. Once they are specified, one can obtain, for every monomial in the fields, the number of independent ways of forming an invariant under the action of G out of it. It must also be taken into account that total derivative terms can be added to the Lagrangian without changing the physics (except for effects of surface terms in the action). This means that some operators with derivatives can be rewritten in terms of others. Moreover, at each order in the effective Lagrangian, the addition of an operator proportional to the equations of motion does not change the S matrix up to higher order effects [1–5]. It follows that the equations of motion can be used, for example, to obtain a basis in which all the operators proportional to the functional derivative of the kinetic term have been removed [6–10]. For the Standard Model Effective Field Theory (SMEFT) (see Ref. [11] for a review), several bases and (incomplete) sets of independent operators have been computed taking all these facts into account [12–15]. Computer tools can be used to translate from one basis to another [16–19].

In the last few years, many developments have been made in the automatization of the generation of operator bases. Hilbert series methods provide an elegant way to compute invariants [20–24]. They can be directly implemented in a computer system with symbolic capabilities, as done for the SMEFT case in the auxiliary Mathematica notebook of Ref. [23]. One possible drawback of this approach, when used in computer code, is its performance, as an overhead due to the symbolic nature of the calculations might be introduced. The program DEFT [19], written in Python, uses a different approach to check and generate bases of operators for the SMEFT. The operators are not only counted but they are given explicitly, including their index contraction structure and the fields to which the derivatives are applied (see Ref. [25] for a non-automatic calculation of the explicit operators in a basis). Additionally, it can perform changes of bases. The method it implements can be generalized to theories with a symmetry group given by a product of unitary groups.

BasisGen uses yet another approach, which is valid for any semisimple symmetry group and avoids the need for symbolic calculations. The algorithms that it uses to deal with representations of semisimple Lie algebras are the classical ones, based on weight vectors. They are reviewed, for example, in Ref. [26], and implemented in several computer

[a] e-mail: jccriadoalamo@ugr.es

packages with different purposes [27–32]. To remove integration by parts redundancy, an adaptation of the method in Ref. [24] is used. `BasisGen` is $\sim$ 150 times faster than the implementation in the auxiliary notebook of ref. [23]. For example, `BasisGen` takes 3 s. to compute the 84 dimension-6 operators of the 1-generation SMEFT (in a laptop with a 2.6 GHz Intel Core i5 processor), while the notebook of Ref. [23] takes 7 min. `DEFT` also takes minutes for the calculation of a dimension-6 basis of the 1-generation SMEFT (according to Ref. [19]), although it must be taken into account that it does more work, as the concrete operators are given instead of just being counted.

For computations with effective field theories, `BasisGen` assumes 4-dimensional Lorentz invariance. In addition, an internal symmetry group must be specified. This is, in general, the product of the global symmetry group and the gauge group. Derivatives are assumed to be gauge-covariant derivatives, so that the derivative of any field has the same representation under the internal symmetry group as the field itself. The gauge field strengths to be included in a calculation should be provided by the user. The fields must belong to linear irreducible representations of both the Lorentz group and the internal symmetry group. Finally, it is required that a power counting based on canonical dimensions can be used.

In this context, `BasisGen` generates bases of invariant operators. It gives the number of independent invariants that can be formed with each possible field content for an operator. Sets of all covariant operators, with their corresponding irreducible representations (irreps), can also be computed. The basic representation-theoretic functionalities needed for these calculations are: obtaining weight systems of irreps and decomposing their tensor products. An interface for their direct use is provided.

Although `BasisGen` does not provide the explicit index contraction structure of the operators in the basis, the functionality of decomposing tensor products can be used to help in their construction. For a particular field content, one can take the tensor product of the first two fields. Then, for each irrep in the decomposition, take the tensor product with the next field. This process can be iterated, keeping track of the intermediate irreps. In the end, one can obtain all the possible ways of doing the products of the fields that give an invariant. Nevertheless, some extra information (the corresponding Clebsch–Gordan coefficients) is needed to completely determine the operator.

`BasisGen` can be installed using `pip` by doing: `pip install basisgen`. It requires `Python` version 3.5 or higher. Its code can be downloaded from the GitHub repository https://github.com/jccriado/basisgen, where some examples of usage can be found. A simple script using `BasisGen` is presented in Listing 1. It defines an effective theory with internal symmetry group $SU(2) \times U(1)$ for a complex scalar $SU(2)$-doublet field with charge 1/2. It computes a basis of operators of dimension 8 or less. The output is presented in Listing 2. Each line gives the number of independent invariant operators that can be constructed with each field content.

Listing 1 Simple EFT example script

```
from basisgen import algebra, irrep, scalar, Field, EFT

phi = Field(
    name='phi',
    lorentz_irrep=scalar,
    internal_irrep=irrep('SU2', '1'),
    charges=[1/2]
)

my_eft = EFT(algebra('SU2'), [phi, phi.conjugate])

invariants = my_eft.invariants(max_dimension=8)

print(invariants)
print("Total:", invariants.count())
```

Listing 2 Simple EFT example script's output

```
phi phi*: 1
(phi)^2 (phi*)^2: 1
(phi)^2 (phi*)^2 D^2: 2
(phi)^2 (phi*)^2 D^4: 3
(phi)^3 (phi*)^3: 1
(phi)^3 (phi*)^3 D^2: 2
(phi)^4 (phi*)^4: 1
Total: 11
```

The rest of this article is divided in two sections (apart from the conclusions). They describe `BasisGen`'s implementation (Sect. 2) and interface (Sect. 3).

2 Implementation

2.1 Basic operations with representations

In this section, the methods implemented in `BasisGen` to deal representations of semisimple Lie algebras are presented. A representation of a semisimple algebra is just a tensor product of representations of the algebra's simple ideals. Using this fact, `BasisGen` decomposes calculations with semisimple algebras into smaller ones with simple algebras. The basic operations with representations of simple algebras are: the generation of the weight system of an irrep from its highest weight and the decomposition of a reducible representation into a direct sum of irreps. They are both implemented using well-known methods (see Refs. [26–32]), which are summarized here, for completeness.

In the Dynkin basis, which we use in what follows, all weights are tuples of integers. Thus, the operations done here

involve only addition and multiplication of integer numbers. Each irrep of a simple algebra is uniquely characterized by its highest weight Λ, which is a tuple $(a_1 \cdots a_n)$ of non-negative integers. Every such tuple is the highest weight of one irrep. The complete weight system of an irrep may be obtained from its highest weight by the following procedure:

1. Set $W = \{\}$ and $W_{\text{new}} = \{\Lambda\}$.
2. Choose some $\lambda \in W_{\text{new}}$.
3. For each positive component $\lambda_i > 0$, select the ith row α of the Cartan matrix. Append to W_{new} all weights of the form $\lambda - k\alpha$, with $0 < k \leq \lambda_i$.
4. Remove λ from W_{new}. Append it to W.
5. If W_{new} is empty, terminate. Otherwise, go to step 2.

This produces the set W of all weights. The multiplicity n_λ of each weight λ can then be obtained recursively using the Freudenthal formula:

$$n_\lambda = \frac{2\sum_\alpha \sum_{k>0} n_{\lambda+k\alpha}\,(\lambda + k\alpha, \alpha)}{(\Lambda + \delta, \Lambda + \delta) - (\lambda + \delta, \lambda + \delta)}, \tag{1}$$

where $\delta = (11 \cdots 1)$ and the summation for α runs over all positive roots.

The algorithm for the decomposition of a reducible representation as a direct sum of irreps is straightforward: from the collection of weights of the representation in question, find the highest and remove from the collection all the weights in the corresponding irrep. Repeat until the collection is empty. Then, the successive highest weights that were found in the process are the highest weights of the irreps in the decomposition. A direct application of this functionality is to decompose the tensor product of irreps. Let W_1 and W_2 be the weight systems of two representations R_1 and R_2. The weight system W of $R_1 \otimes R_2$ is the collection of all $\lambda_1 + \lambda_2$ for $(\lambda_1, \lambda_2) \in W_1 \times W_2$. Once W is constructed, it can be decomposed using the general decomposition algorithm.

In some cases, the symmetric or anti-symmetric tensor power of some representation is needed. If $W = \{\lambda_i\}_{i\in\{1,\ldots,n\}}$ is the weight system of some representation R, the weight system of the symmetric tensor power $\text{Sym}^k(R)$ is the collection of weights computed as $\lambda_1 + \cdots + \lambda_k$ for every k-tuple $(\lambda_{i_1}, \ldots, \lambda_{i_k})$ where $i_1 \leq \cdots \leq i_k$. The weight system of the anti-symmetric power $\Lambda^k(R)$ is constructed in a similar way, but using all k-tuples $(\lambda_{i_1}, \ldots, \lambda_{i_k})$ with $i_1 < \cdots < i_k$ instead.

2.2 Constructing invariants in effective theories

`BasisGen` can do calculations for 4-dimensional Lorentz-invariant effective field theories whose internal symmetry group is of the form $G \times U(1)^n$, where G is semisimple. An effective theory is specified when the following data are provided:

- The semisimple Lie algebra $\mathfrak{g}$ of G.
- A collection of fields $\phi_1, \ldots, \phi_m$. Each ϕ_i must be equipped with:
  - An irrep $R^{(i)}_{\text{Lorentz}}$ of the Lorentz algebra $\mathfrak{su}_2 \oplus \mathfrak{su}_2$.
  - An irrep $R^{(i)}_{\text{internal}}$ of $\mathfrak{g}$.
  - A tuple $\left(c_1^{(i)}, \ldots, c_n^{(i)}\right)$ of charges under the $U(1)$ factors.
  - The statistics S_i. Either boson or fermion.
  - A positive real number d_i, specifying the canonical dimension of the field.

It is assumed that a power counting based on canonical dimensions of the fields, with derivatives having dimension 1, can be applied. This is used to reduce the number of possible operators to a finite one.

The main functionality of `BasisGen` is to compute the number of independent invariant operators, constructed with the fields ϕ_i and their (covariant) derivatives, and having dimension less than or equal to some fixed $d_{\max}$. To do this, first, all the possible operator field contents are found. The field content for some operator is identified by a tuple $\mathcal{C} = (e_1, \ldots, e_m)$, representing the exponents of each field in the operator: $\mathcal{O} \sim (\phi_1)^{e_1} \cdots (\phi_m)^{e_m}$. For each $\mathcal{C}$, the following (possible reducible) representation is computed:

$$\text{Rep}(\mathcal{C}) = T_1^{e_1}(R^{(1)}) \otimes \cdots \otimes T_m^{e_m}(R^{(m)}), \tag{2}$$

where $T_i^k(V)$ is the symmetric power $\text{Sym}^k(V)$ if the statistics S_i are bosonic, and the anti-symmetric power $\Lambda^k(V)$ if they are fermionic. Once $\text{Rep}(\mathcal{C})$ is obtained, it is decomposed into a direct sum of irreps. The number of independent invariant combinations of the fields in $\mathcal{C}$ is then easily obtained as the number of singlet irreps in the decomposition.

To take into account (covariant) derivatives, the same procedure is used, but now including the fields $D_\mu\phi_i$, $\{D_\mu, D_\nu\}\phi_i$, etc. Anti-symmetric combinations of derivatives are automatically discarded, as they are equivalent to field strength tensors. Optionally, the equations of motion of the fields can be applied. This means that, for each $D_{\mu_1} \cdots D_{\mu_m}\phi_i$, only the totally symmetric representation is retained (see Ref. [22]).

Let I be the set of all operators constructed with the fields and their derivatives (using equations of motion if necessary) that are invariant under the internal symmetry group (but are not necessarily scalars). To eliminate integration by parts redundancy from I, it is first split into the set of operators with zero derivatives I_0, the set of operators with one derivative I_1, etc. Then, the following procedure is applied:

1. Set $R = \{\}$.
2. Take one operator $\mathcal{O}$ from the non-empty I_n with lowest n.
3. Remove $\mathcal{O}$ from I_n and append it to R.
4. Compute the decomposition into irreps of $D_\mu\mathcal{O}$ and eliminate the corresponding operators from I_{n+1}. Compute the decomposition of $\{D_\mu, D_\nu\}\mathcal{O}$ and remove it from I_{n+2}. Continue until the maximum dimension is reached.
5. If all I_k are empty, terminate. Otherwise, go to step 2.

After this is done, a basis (in which integration by parts has been taken into account) is obtained by selecting those operators in R that are scalars. Notice that the irreps in the decomposition of the derivatives of operators are computed and removed. In particular, if no (non-zero) scalar appears in the decomposition, then the corresponding scalar operator will not be eliminated. This avoids the over-counting of integration by parts redundancy in Ref. [22] that was pointed out in Ref. [23].

3 Interface

3.1 Basic objects

The basic objects for the usage of `BasisGen` are presented here. All of them can be imported with:

```
from basisgen import (
    algebra, irrep, Field, EFT, boson, fermion,
    scalar, L_spinor, R_spinor, vector, L_tensor, R_tensor
)
```

Functions

`algebra` Creates a (semi)simple Lie algebra from one string argument. The returned object is of the class `SimpleAlgebra` or `SemisimpleAlgebra` from the module `algebra`.

Examples of arguments: `'A3'`, `'C12'`, `'F4'`, `'SU3'`, `'B2+E7'`, `'SU5 x SO6 x Sp10'`.

`irrep` Creates an irreducible representation from 2 string arguments: the first represents the algebra and the second the highest weight.[1] The returned object is of the class `representations.Irrep`.

Example: `irrep('SU4 x Sp7', '1 0 1 0 2 1')`.

[1] The highest weights for many irreps of several groups can be found, for example in Ref. [26]. In particular, notice that the highest weight of an $SU(2)$ irrep is its dimension minus one.

The weight system of a `representations.Irrep` object can be obtained by calling its `weights_view` method. Irreps with the same algebra can be multiplied to get the decomposition of their tensor product. Any two irreps can be added to give an irrep of the direct sum of their algebras.

Examples, showing the weights of the octet irrep of $SU(3)$ (which has highest weight (11)) and the decomposition of the product of a triplet (10) and an anti-triplet (01) as an octet plus a singlet:

```
>>> irrep('SU3', '1 1').weights_view()
    (1  1)
(2 -1)  (-1  2)
 (0  0)  (0  0)
(1 -2)  (-2  1)
   (-1 -1)
>>> irrep('SU3', '1 0') * irrep('SU3', '0 1')
[1 1] + [0 0]
```

Classes

`Field` Has an attribute `conjugate`, the conjugate field. The constructor arguments are presented in Table 1.

`EFT` Constructor arguments:

`internal_algebra` The semisimple Lie algebra of the internal symmetry group.

`fields` A list of `Field` objects representing the field content of the theory.

Methods:

`invariants` Returns a basis of operators, encapsulated in an `EFT.Invariants` object. These can be directly printed (implement `__str__`). They have a method `count` to calculate the total number of operators in the basis, and a method `show_by_classes`, which returns a simplified string representation of the basis, provided a dictionary whose keys are the fields and values are strings representing classes of fields.

`covariants` Returns a collection of all operators with all possible irreps, in the form of a `EFT.Covariants` instance. Its only purpose is to hold the information until it is printed (implements `__str__`).

Both receive the same arguments: `max_dimension`, the maximum dimension of the operators computed; `use_eom` (default: `True`) a boolean to specify whether the equations of motion should be used; `ignore_lower_dimension` (default: `False`), a boolean to specify whether operators

Table 1 Arguments of the `Field` constructor

Name	Description	Default
`name`	String identifier	
`lorentz_irrep`	Lorentz group irrep	
`internal_irrep`	Irrep of the internal (semisimple) symmetry group	
`charges`	Charges under an arbitrary number of $U(1)$ factors	`[]`
`statistics`	Either `boson` or `fermion`	`boson`
`dimension`	Canonical dimension of the field	`1`
`number_of_flavors`	Number of different copies of the same field	`1`

with dimension less than `max_dimension` should be included in the results; and `verbose` (default: `False`), a boolean enabling/disabling messages about the progress of the calculations.

Other

The following irreps of the Lorentz group have been defined, for ease of use: `scalar`, `L_spinor`, `R_spinor`, `vector`, `L_tensor`, `R_tensor`. `L_spinor` and `R_spinor` correspond to left and right Weyl spinors, respectively. `L_tensor` and `R_tensor` correspond to the left and right parts of an antisymmetric tensor with two indices.

The statistics of a field can be specified by using the variables `boson` and `fermion`, which are set to the values `BOSON` and `FERMION` of the enum class `Statistics` from the module `statistics`.

3.2 The `smeft` module

The `smeft` module contains the definitions of all the Standard Model fields:

- The Higgs doublet `phi` and its conjugate `phic`.
- The left and right parts `GL` and `GR` of the $SU(3)$ field strength.
- The left and right parts `WL` and `WR` of the $SU(2)$ field strength
- The left and right parts `BL` and `BR` of the $U(1)$ field strength.
- The quark doublet `Q` and its conjugate `Qc`.
- The lepton doublet `L` and its conjugate `Lc`.
- The up-type quark singlet `u` and its conjugate `uc`.
- The down-type quark singlet `d` and its conjugate `dc`.
- The electron singlet `e` and its conjugate `ec`.

The bosons are objects of the `Field` class. The fermions are functions that take the number of generations and return a `Field`. Similarly, the function `smeft` takes the number of fermion flavors and returns an `EFT` object representing the SMEFT. The algebra $\mathfrak{su}_3 \oplus \mathfrak{su}_2$ is named `sm_internal_algebra`. A dictionary named `sm_field_classes` is included, to simplify the presentation of the results by passing it as an argument to the method `show_by_classes` of an `EFT.Invariants` object.

Listing 3 SMEFT example

```
from basisgen.smeft import smeft, sm_field_classes
import sys

invariants = smeft(number_of_flavors=1).invariants(
    max_dimension=int(sys.argv[1]),
    verbose=True,
    ignore_lower_dimension=True
)

print(invariants.show_by_classes(sm_field_classes(1)))
print("Number of invariants: {}".format(invariants.count()))
```

Listing 3 contains an example script for the computation of bases of arbitrary dimension (passed as an argument to the script) for the 1-generation SMEFT. It gives 84 operators for dimension 6 (in about 3 s in a personal computer with a 2.6 GHz Intel Core i5 processor) and 993 operators for dimension 8 (in around 40 s in the same computer).

4 Conclusions

`BasisGen` computes bases of operators for effective field theories in a general setting: the internal symmetry group can be any product of a semisimple group and an arbitrary number of $U(1)$ factors. 4-dimensional Lorentz invariance is assumed to provide support for concrete applications, although adaptations to other spacetime dimensions can be easily made, due to the generality of the core functionalities.

The decision of using the equations of motion is left to the user, as it may be convenient to work with redundant bases in some cases (see Ref. [5]). It is also possible not only to compute invariants but to generate all covariant operators, classified by their irreps. This can be useful, for example, to find the representation of fields that couple linearly to an already known theory, which are often the most relevant ones for phenomenology [33–37]. An interface for doing basic

operations with representations of semisimple groups is also provided.

`BasisGen`'s speed for large numbers of fields and high-dimensional operators makes it possible to calculate bases for the SMEFT or for other effective theories for physics beyond the Standard Model, in times ranging from seconds (for the dimension-8 operators in the SMEFT) to minutes (for higher-dimensional operators or larger number of fields) in personal computers.

Acknowledgements The author would like to thank M. Pérez-Victoria for useful discussions and comments. This work has been supported by the Spanish MINECO project FPA2016-78220-C3-1-P (Fondos FEDER), the Junta de Andalucía Grant FQM101 and the Spanish MECD Grant FPU14.

Data Availability Statement This manuscript has no associated data or the data will not be deposited. [Authors' comment: This paper does not use any data that could be deposited.]

Funded by SCOAP[3].

References

1. H.D. Politzer, Nucl. Phys. B **172**, 349 (1980). https://doi.org/10.1016/0550-3213(80)90172-8
2. C. Grosse-Knetter, Phys. Rev. D **49**, 6709 (1994). https://doi.org/10.1103/PhysRevD.49.6709
3. C. Arzt, Phys. Lett. B **342**, 189 (1995). https://doi.org/10.1016/0370-2693(94)01419-D
4. J. Wudka, Int. J. Mod. Phys. A **9**, 2301 (1994). https://doi.org/10.1142/S0217751X94000959
5. J.C. Criado, M. Pérez-Victoria, JHEP **3**, 38 (2019). https://doi.org/10.1007/JHEP03(2019)038
6. H. Georgi, Nucl. Phys. B **361**, 339 (1991). https://doi.org/10.1016/0550-3213(91)90244-R
7. B. Grzadkowski, Z. Hioki, K. Ohkuma, J. Wudka, Nucl. Phys. B **689**, 108 (2004). https://doi.org/10.1016/j.nuclphysb.2004.04.006
8. P.J. Fox, Z. Ligeti, M. Papucci, G. Perez, M.D. Schwartz, Phys. Rev. D **78**, 054008 (2008). https://doi.org/10.1103/PhysRevD.78.054008
9. J.A. Aguilar-Saavedra, Nucl. Phys. B **812**, 181 (2009). https://doi.org/10.1016/j.nuclphysb.2008.12.012
10. J.A. Aguilar-Saavedra, Nucl. Phys. B **821**, 215 (2009). https://doi.org/10.1016/j.nuclphysb.2009.06.022
11. I. Brivio, M. Trott, Phys. Rept. **793**, 1 (2019). https://doi.org/10.1016/j.physrep.2018.11.002
12. K. Hagiwara, S. Ishihara, R. Szalapski, D. Zeppenfeld, Phys. Rev. D **48**, 2182 (1993). https://doi.org/10.1103/PhysRevD.48.2182
13. G.F. Giudice, C. Grojean, A. Pomarol, R. Rattazzi, JHEP **06**, 045 (2007). https://doi.org/10.1088/1126-6708/2007/06/045
14. B. Grzadkowski, M. Iskrzynski, M. Misiak, J. Rosiek, JHEP **10**, 085 (2010). https://doi.org/10.1007/JHEP10(2010)085
15. J. Elias-Miró, C. Grojean, R.S. Gupta, D. Marzocca, JHEP **05**, 019 (2014). https://doi.org/10.1007/JHEP05(2014)019
16. A. Falkowski, B. Fuks, K. Mawatari, K. Mimasu, F. Riva, V. Sanz, Eur. Phys. J. C **75**(12), 583 (2015). https://doi.org/10.1140/epjc/s10052-015-3806-x
17. J.C. Criado, Comput. Phys. Commun. **227**, 42 (2018). https://doi.org/10.1016/j.cpc.2018.02.016
18. J. Aebischer, J. Kumar, D.M. Straub, Eur. Phys. J. C **78**(12), 1026 (2018). https://doi.org/10.1140/epjc/s10052-018-6492-7
19. B. Gripaios, D. Sutherland, JHEP **1**, 128 (2019). https://doi.org/10.1007/JHEP01(2019)128
20. L. Lehman, A. Martin, Phys. Rev. D **91**, 105014 (2015). https://doi.org/10.1103/PhysRevD.91.105014
21. B. Henning, X. Lu, T. Melia, H. Murayama, Commun. Math. Phys. **347**(2), 363 (2016). https://doi.org/10.1007/s00220-015-2518-2
22. L. Lehman, A. Martin, JHEP **02**, 081 (2016). https://doi.org/10.1007/JHEP02(2016)081
23. B. Henning, X. Lu, T. Melia, H. Murayama, JHEP **08**, 016 (2017). https://doi.org/10.1007/JHEP08(2017)016
24. B. Henning, X. Lu, T. Melia, H. Murayama, JHEP **10**, 199 (2017). https://doi.org/10.1007/JHEP10(2017)199
25. C. Hays, A. Martin, V. Sanz, J. Setford, JHEP **2**, 123 (2019). https://doi.org/10.1007/JHEP02(2019)123
26. R. Slansky, Phys. Rep. **79**, 1 (1981). https://doi.org/10.1016/0370-1573(81)90092-2
27. M.A.A. van Leeuwen, A.M. Cohen, B. Lisser, Computer Algebra Nederland (Amsterdam, 1992) **(ISBN 90-74116-02-7)**
28. A. Candiello, Comput. Phys. Commun. **81**, 248 (1994). https://doi.org/10.1016/0010-4655(94)90123-6
29. T. Fischbacher (2002). arXiv:hep-th/0208218
30. C. Horst, J. Reuter, Comput. Phys. Commun. **182**, 1543 (2011). https://doi.org/10.1016/j.cpc.2011.03.025
31. A. Nazarov, Comput. Phys. Commun. **183**, 2480 (2012). https://doi.org/10.1016/j.cpc.2012.06.014
32. R. Feger, T.W. Kephart, Comput. Phys. Commun. **192**, 166 (2015). https://doi.org/10.1016/j.cpc.2014.12.023
33. F. del Aguila, M. Perez-Victoria, J. Santiago, JHEP **09**, 011 (2000). https://doi.org/10.1088/1126-6708/2000/09/011
34. F. del Aguila, J. de Blas, M. Perez-Victoria, Phys. Rev. D **78**, 013010 (2008). https://doi.org/10.1103/PhysRevD.78.013010
35. F. del Aguila, J. de Blas, M. Perez-Victoria, JHEP **09**, 033 (2010). https://doi.org/10.1007/JHEP09(2010)033
36. J. de Blas, M. Chala, M. Perez-Victoria, J. Santiago, JHEP **04**, 078 (2015). https://doi.org/10.1007/JHEP04(2015)078
37. J. de Blas, J.C. Criado, M. Perez-Victoria, J. Santiago, JHEP **03**, 109 (2018). https://doi.org/10.1007/JHEP03(2018)109