

# MatchingTools: a Python library for symbolic effective field theory calculations

Juan C. Criado<sup>a,\*</sup>

<sup>a</sup>CAFPE and Departamento de Física Teórica y del Cosmos, Universidad de Granada, E-18071 Granada, Spain

## Abstract

*MatchingTools* is a Python library for doing symbolic calculations in effective field theory. It provides the tools to construct general models by defining their field content and their interaction Lagrangian. Once a model is given, the heavy particles can be integrated out at the tree level to obtain an effective Lagrangian in which only the light particles appear. After integration, some of the terms of the resulting Lagrangian might not be independent. *MatchingTools* contains functions for transforming these terms to rewrite them in terms of any chosen set of operators.

**Keywords:** effective; tree; integration; matching; redundancies; python;

## PROGRAM SUMMARY

*Program Title:* MatchingTools

*Licensing provisions:* MIT

*Programming language:* Python (compatible with versions 2 and 3)

*Nature of problem:*

The program does two kinds of calculations: computing an effective Lagrangian for the light fields of a field theory by integrating out at the tree level the heavy fields and performing algebraic manipulations with tensors in the (effective) Lagrangian.

*Solution method:*

The tree level integration of heavy fields is done by substituting them inside the Lagrangian by a covariant derivative expansion of the solution to their equations of motion. The transformation of Lagrangians is implemented as an algorithm for finding patterns of tensor products and replacing them by sums of other products.

## 1. Introduction

When studying physical phenomena in the framework of field theory, it is often convenient to describe the low energy behavior of the system in a way that does not involve the heavy degrees of freedom. A low energy effective theory can be derived from a more fundamental one, when the latter is known. The connection between both descriptions is done by integrating out the heavy fields. The basic idea is to find the set the effective interactions of the light fields such that the corresponding low energy observables match, to the desired precision, those computed using the full theory [1, 2].

An important example arises in particle physics, when studying extensions of the Standard Model. The latest experimental results from the Large Hadron Collider (LHC) do not show any evidence of direct production of new particles (see for example [3, 4, 5, 6]). Therefore, the discovery of new physics arising

within the range of energies of the current phase of the LHC seems more and more unlikely.

In view of this perspective, it is interesting to extract features of physics at higher, currently unreachable energies by using precision measurements. This can be done using an effective theory approach. In the Standard Model Effective Field Theory (SMEFT), the Standard Model is extended to include non-renormalizable operators (see [7] and references therein). In this setting, the high energy physics is parametrized in low energy by the coefficients of the new operators. These coefficients can be constrained by the experimental data [8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. Another effective approach to extending the Standard Model is the Higgs effective theory, in which the gauge symmetry is realized non-linearly [18, 19, 20, 21].

Many of the proposed theories for physics beyond the Standard Model predict the existence of new, heavy particles [22, 23, 24, 25]. The result of integrating out these heavy fields is the collection of the corresponding coefficients of the operators of the SMEFT [26, 27, 28, 29].

The procedure of matching can be described algebraically in terms of tensor calculus manipulations involving the computation of functional derivatives and the substitution of heavy fields by other previously obtained expressions [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. The complexity of the process quickly grows with the number of heavy fields and their interactions. It is in this context where the development of a computer tool to automatize the process becomes necessary.

*MatchingTools* can perform tree-level integration of heavy fields in any given Lagrangian. It has been developed with the application to the SMEFT in mind, but it is able to work with any situation describable by a Lorentz invariant field theory in which the high energy degrees of freedom to be removed are scalars, vector-like or Majorana fermions, or vectors. By introducing the generic solution to their equations of motion, other types of fields can be treated as well. The validity of *MatchingTools* extends to any level in the expansion in inverse powers of the cut-off energy of the effective theory.

\*Corresponding author.

E-mail address: jccriadoalamo@ugr.es

The Lagrangian resulting from integration usually contains redundancies: operators that can be written in terms of others using identities of the symmetry group, integration by parts or equations of motion of the light fields [42, 43, 44, 45]. A complete set of operators that are independent under this set of transformations is called a basis. Several such bases have been described [46, 47, 48].

The transformation of the results of an integration to a chosen basis can also be done using *MatchingTools*. One should introduce the identities between tensor expressions needed to transform some operators into others, as well as the desired basis.

There are other tools for the manipulation of bases of operators, such as *Rosetta* [49]. The portion of *MatchingTools* that deals with this calculations differs from it in two main points: first, it allows not only for the transformations between sets of already independent operators, but for the transformation of any set of operators into a basis. Moreover, *MatchingTools* has the ability of doing transformations not with the operators themselves, but with parts of them, allowing for general transformations between parts of tensor expressions into others. Actually, *MatchingTools* can be used as system for tensor calculus manipulations, not necessarily in the context of an effective field theory. It provides a fast way of doing complex symbolic calculations with many fields and terms involved, which is safe against algebraic errors.

A direct application of *MatchingTools*, which has also served as an extensive check of its validity, is the integration of all possible new fields that have linear gauge-invariant renormalizable couplings to the Standard Model, keeping terms up to dimension six in the results [50].

A package that implements a similar way of dealing with the specification of models is *FeynRules* [51, 52], thought its objectives are completely distinct. One possible direction for future work with *MatchingTools* is making the connection with *FeynRules*.

Among other computer tools for calculations in the context of the SMEFT we have *DsixTools* [53] (which allows for several calculations including a case of tree level matching) and *SMEFTsim* [54] (which is able to produce theoretical predictions and constraints for the Wilson coefficients of the dimension 6 SMEFT).

*MatchingTools* is available in GitHub (<https://github.com/jccriado/matchingtools>) and in the PyPI repository (<https://pypi.python.org/pypi/matchingtools/>), so it can be installed using pip [55] as

```
pip install matchingtools
```

This article is organized as follows: section 2 describes the procedure and the formulas that the library uses for the integration of heavy particles. Sections 3, 4, 5 and 6 explain the features of *MatchingTools* and how to use it. Section 7 proposes a simple example that serves to see the library in action and as a test case. Some extra features for the applications in physics beyond the Standard Model are introduced in section 8. Section 9 is an explanation of how to integrate out new types of fields that are not included in *MatchingTools*.

## 2. Theoretical framework

### 2.1. Tree level integration

Starting with a high energy theory with action  $S[\phi, \Phi]$  depending on the light and heavy fields  $\phi$  and  $\Phi$  the effective action  $S_{eff}$  for the light fields is obtained through:

$$e^{iS_{eff}[\phi]} \propto \int \mathcal{D}\Phi e^{iS[\phi, \Phi]}, \quad (1)$$

where  $\int \mathcal{D}\Phi$  means integrating over all the configurations of the heavy fields  $\Phi$ . The configuration that contributes the most to this integral is the classical configuration  $\Phi_c$ , which extremizes the action. To leading order in  $(\Phi - \Phi_c)$ , we get

$$S_{eff}[\phi] = S[\phi, \Phi_c], \quad (2)$$

known as the tree level approximation. It is the one that we will use in this article.

### 2.2. Equations of motion and their solution

To obtain the classical configuration of the heavy fields it is necessary to solve their equations of motion. They are determined from the condition:  $\delta S / \delta \Phi = 0$ .

The variation of a local action  $S = \int d^m x \mathcal{L}$  can be written as

$$\frac{\delta S}{\delta \Phi} = \sum_n (-1)^n D_{\mu_1} D_{\mu_2} \cdots D_{\mu_n} \frac{\partial \mathcal{L}}{\partial (D_{\mu_n} \cdots D_{\mu_2} D_{\mu_1} \Phi)}, \quad (3)$$

which we have expressed in terms of a covariant derivative  $D$  for the gauge group of the low energy effective field theory. It is convenient to split the action into a quadratic and an interaction part:

$$S = \int d^m x \mathcal{L}_{quad} + S_{int}, \quad \mathcal{L}_{quad} = -\Phi^\dagger P \Phi, \quad (4)$$

where  $P$  is some differential operator.

For the bosonic fields, the solution to the equation of motion will be given by the application of the inverse of  $P$  to a functional derivative of the interaction action.  $P^{-1}$  can be expanded in each case in powers of  $D_\mu/M$ . For fermions, the solution will be given as a system of two equations. Recursive substitution of one into the other will give the solution to any order in  $D_\mu/M$ . Because we usually limit the dimension of the operators appearing in the effective Lagrangian we will only need to substitute a finite number of terms of these infinite expansions.

Several fields can be integrated out together. The solution to the equation of motion of each of them may involve the others, but they can be replaced recursively by their corresponding solutions to the equations motion to obtain solutions that only involve the light fields to the desired order.

The Lagrangian  $\mathcal{L}_{quad}$  and the solution to the equations of motion is, for the following types of fields:

- Real scalar:

$$\mathcal{L}_{quad} = -\frac{1}{2} \Phi (D^2 + M^2) \Phi, \quad (5)$$

$$\Phi_c = \sum_{n=0}^{\infty} (-1)^n \frac{D^{2n}}{M^{2n+2}} \frac{\delta S_{int}}{\delta \Phi}. \quad (6)$$

- Complex scalar:

$$\mathcal{L}_{quad} = -\Phi^\dagger(D^2 + M^2)\Phi, \quad (7)$$

$$\Phi_c = \sum_{n=0}^{\infty} (-1)^n \frac{D^{2n}}{M^{2n+2}} \frac{\delta S_{int}}{\delta \Phi^\dagger}, \quad (8)$$

$$\Phi_c^\dagger = \sum_{n=0}^{\infty} (-1)^n \frac{D^{2n}}{M^{2n+2}} \frac{\delta S_{int}}{\delta \Phi}. \quad (9)$$

- Real vector:

$$\mathcal{L}_{quad} = \frac{1}{2} V^\mu \{ \eta_{\mu\nu} (D^2 + M^2) - D_\nu D_\mu \} V^\nu, \quad (10)$$

$$V_c = -\frac{1}{M^2} \sum_{n=0}^{\infty} Q^n \frac{\delta S_{int}}{\delta V}, \quad (11)$$

where  $Q$  is a differential operator that acts on a Lorentz vector and gives a Lorentz vector as:

$$(QV)_\mu := \frac{D_\nu D_\mu - \eta_{\mu\nu} D^2}{M^2} V^\nu. \quad (12)$$

- Complex vector:

$$\mathcal{L}_{quad} = V^{\dagger\mu} \{ \eta_{\mu\nu} (D^2 + M^2) - D_\nu D_\mu \} V^\nu, \quad (13)$$

$$V_c = -\frac{1}{M^2} \sum_{n=0}^{\infty} Q^n \frac{\delta S_{int}}{\delta V^\dagger}, \quad (14)$$

$$V_c^\dagger = -\frac{1}{M^2} \sum_{n=0}^{\infty} Q^n \frac{\delta S_{int}}{\delta V}. \quad (15)$$

- Vector-like fermion (using two-component spinor notation):

$$\mathcal{L}_{quad} = iF_{L\dot{\alpha}}^\dagger \bar{\sigma}_\mu^{\dot{\alpha}\alpha} D^\mu F_{L\alpha} + iF_{R\dot{\alpha}}^\dagger \sigma_\mu^{\dot{\alpha}\alpha} D_\mu F_{R\alpha} - M(F_{L\dot{\alpha}}^\dagger F_{R\dot{\alpha}} + F_{R\dot{\alpha}}^\dagger F_{L\dot{\alpha}}), \quad (16)$$

$$(F_c)_{L\alpha} = \frac{i}{M} \sigma_\mu^{\dot{\alpha}\alpha} D_\mu F_{R\dot{\alpha}} + \frac{1}{M} \frac{\delta S_{int}}{\delta F_{R\dot{\alpha}}^\dagger}, \quad (17)$$

$$(F_c)_{R\dot{\alpha}} = \frac{i}{M} \bar{\sigma}_\mu^{\dot{\alpha}\alpha} D^\mu F_{L\alpha} + \frac{1}{M} \frac{\delta S_{int}}{\delta F_{L\alpha}^\dagger}, \quad (18)$$

$$(F_c)_{L\dot{\alpha}}^\dagger = -\frac{i}{M} \sigma_\mu^{\dot{\alpha}\alpha} D^\mu F_{R\dot{\alpha}} - \frac{1}{M} \frac{\delta S_{int}}{\delta F_{R\dot{\alpha}}}, \quad (19)$$

$$(F_c)_{R\dot{\alpha}}^\dagger = -\frac{i}{M} \bar{\sigma}_\mu^{\dot{\alpha}\alpha} D^\mu F_{L\alpha} - \frac{1}{M} \frac{\delta S_{int}}{\delta F_{L\alpha}}. \quad (20)$$

- Majorana fermion:

$$\mathcal{L}_{quad} = iF_\alpha^\dagger \bar{\sigma}_\mu^{\dot{\alpha}\alpha} D^\mu F_\alpha - \frac{1}{2} \left( \epsilon^{\alpha\beta} F_\beta F_\alpha + F_\alpha^\dagger \epsilon^{\dot{\alpha}\dot{\beta}} F_\beta^\dagger \right), \quad (21)$$

$$(F_c)_\beta = \epsilon_{\alpha\beta} \left( i\bar{\sigma}_\mu^{\dot{\alpha}\alpha} D^\mu F_\alpha^\dagger + \frac{\delta S_{int}}{\delta F_\alpha} \right), \quad (22)$$

$$(F_c)_\beta^\dagger = \epsilon_{\beta\dot{\alpha}} \left( i\bar{\sigma}_\mu^{\dot{\alpha}\alpha} D^\mu F_\alpha + \frac{\delta S_{int}}{\delta F_\alpha^\dagger} \right). \quad (23)$$

### 3. Creation of models

In this section we will describe how to create a model using the module `matchingtools.core`. It assumes that the classes and functions that are used are in the namespace. To import all the classes and functions that appear here do

```
from matchingtools.core import (
    Tensor, Operator, OperatorSum,
    TensorBuilder, FieldBuilder,
    D, Op, OpSum,
    number_op, power_op)
```

The `from ... import ...` style is recommended, as the expressions that appear when using this library tend to be long, so having the short names directly accessible is preferable.

#### 3.1. Creation of tensors and fields

In *MatchingTools*, the basic building blocks for everything are the objects of the class `Tensor`, which we simply call tensors here. Examples of tensors are fields (light and heavy), symmetry group related tensors (such as Pauli matrices) or coupling constants (including gauge couplings, Yukawa couplings and masses).

Tensors have an attribute `is_field` that is `True` if and only if they are spacetime dependent (i.e., they are fields). Fields can have derivatives applied to them. The attribute `num_of_der` counts the number of derivatives that apply to a field. Derivatives are understood here to be covariant derivatives  $D_\mu$  corresponding to the gauge group of the low energy effective theory. Each derivative applies only to one field. The Leibniz rule is used whenever a derivative of a product is encountered. Tensors can be either commuting or anti-commuting, which is distinguished by the attribute `statistics`. It can be set equal to either `boson` or `fermion`, both being variables defined in this module. Finally, all tensors have an attribute `indices`, a list of integer numbers representing their tensor indices; and an attribute `name`, an identifier. Other attributes, `content` and `exponent`, are for internal use. Names starting with the character '\$' are also reserved for internal calculations.

To create the tensors and fields of a model, the classes `TensorBuilder` and `FieldBuilder` should be used. For example, the Pauli matrices  $\sigma_{ij}^a$  could be defined as

```
sigma = TensorBuilder("sigma")
```

and then used when needed as `sigma(i1, i2, i3)` where `i1`, `i2` and `i3` are the indices. Similarly, a boson field  $\phi$  (with its conjugate  $\phi^*$ ) and a fermion  $f$  (with its separate chiralities and their conjugates) are defined as

```
phi = FieldBuilder("phi", 1, boson)
phic = FieldBuilder("phic", 1, boson)
```

```
fL = FieldBuilder("fL", 1.5, fermion)
fR = FieldBuilder("fR", 1.5, fermion)
fLc = FieldBuilder("fLc", 1.5, fermion)
fRc = FieldBuilder("fRc", 1.5, fermion)
```

The second argument of `FieldBuilder` is the dimension of the field.

### 3.2. Definition of the interaction Lagrangian

Once all the tensors are created, we are ready to define the interaction Lagrangian. It should be a sum of operators, which in turn are just products of fields. Using the functions `OpSum` and `Op`:

```
int_lag = -OpSum(Op(...), Op(...), ...)
```

The minus sign is defined for operator sums and individual operators. The function `OpSum` creates an object of the class `OperatorSum`, a container for a list of operators representing their sum. The function `Op` creates an `Operator` that contains a list of tensors and represents their product:

```
Op(tensor1(i1, i2, ...),
    tensor2(i3, i4, ...), ...)
```

Positive indices are used to express contraction. During the creation of the model, any index should be contracted with another, so we will only use here positive ones. When indices are repeated inside the same operator, the corresponding contraction is understood. For example, the product of tensors  $r_{ij} \text{Stimm} t_{njl}$  would be written as

```
Op(r(0, 1), s(3, 0, 4, 5, 4), t(5, 1, 3))
```

To introduce a covariant derivative inside an operator, the appropriate function is `D`, whose first argument is the Lorentz index of the derivative and whose second one is the tensor to which it is to be applied:

```
D(i1, tensor(i2, ...))
```

For numeric coefficients, the function `number_op` creates an operator with only one special tensor representing a number (its name is `"$number"` and has an attribute `content` with the actual number). Multiplication is defined for operators, so the operator  $iV_\mu S_a^* D_\mu S_a$  can be expressed as

```
number_op(1j) * Op(V(0), Sc(1), D(0, S(1)))
```

Tensors representing a symbolic constant exponentiated to some power can be created using the function `power_op`, that takes the base (a string) and the exponent (a number) (represented by an extra internal attribute of tensors: `exponent`) and optionally some indices and returns an operator containing only the corresponding tensor. This is useful specially for the masses of the heavy particles, which tend to appear several times with different powers in all calculations.

A summary of the tools presented in this section is shown in table 1.

### 3.3. Dealing with spinors

*MatchingTools* uses the two-component spinor formalism to treat spinor fields following the conventions in [56]. The module `matchingtools.core` defines the following tensors to work with them:

Tensors	<code>t_name = TensorBuilder("t_name")</code>
Fields	<code>f_name = FieldBuilder("f_name", dim, statistics)</code>
Lagrangian	<code>lag = -OpSum(Op(...), Op(...), ...)</code>
Operators	<code>Op(tensor1(i1, i2, ...), ...)</code>
Derivatives	<code>Op(..., D(i1, tensor(...)), ...)</code>
Num. coef.	<code>number_op(number) * Op(...)</code>
Symb. power	<code>inv_mass_sq = power_op("M", -2)</code>

Table 1: Summary of the tools for the creation of a model.

- `epsUp` and `epsDown`: the totally anti-symmetric tensors  $\epsilon^{\alpha\beta}$  and  $\epsilon_{\alpha\beta}$  with two undotted two-component spinor indices defined by  $\epsilon^{12} = -\epsilon^{21} = -\epsilon_{12} = \epsilon_{21} = 1$ .
- `epsUpDot` and `epsDownDot`: the totally anti-symmetric tensors  $\epsilon^{\dot{\alpha}\dot{\beta}}$  and  $\epsilon_{\dot{\alpha}\dot{\beta}}$  with two dotted two-component spinor indices given by  $\epsilon_{\dot{\alpha}\dot{\beta}} = (\epsilon_{\alpha\beta})^*$  and  $\epsilon^{\dot{\alpha}\dot{\beta}} = (\epsilon^{\alpha\beta})^*$ .
- `sigma4` and `sigma4bar`: the tensors  $\sigma_{\alpha\dot{\alpha}}^\mu$  and  $\bar{\sigma}_{\dot{\mu}\alpha}^\mu$  given by  $\sigma^\mu = (I_{2\times 2}, \vec{\sigma})$  and  $\bar{\sigma}_\mu = (I_{2\times 2}, -\vec{\sigma})$ , where  $\vec{\sigma}$  is the three-vector of the Pauli matrices. The first index of `sigma4` and `sigma4bar` corresponds to the Lorentz index.

## 4. Integration

This section explains how to use the classes that represent the heavy fields as well as the function `integrate`, to integrate them out. They belong to the module `matchingtools.integration`. To import them do:

```
from matchingtools.integration import (
    RealScalar, ComplexScalar,
    RealVector, ComplexVector,
    VectorLikeFermion, MajoranaFermion,
    integrate)
```

To integrate out the heavy fields from a previously defined Lagrangian we should specify which of the fields are heavy. This is done using the classes:

- `RealScalar`. Its constructor receives as arguments the name of the field and the number of indices it has.
- `ComplexScalar`. Requires a field-conjugate field pair. The arguments of the constructor are the name of the field, the name of its conjugate and its number of indices.
- `RealVector`. The arguments are the name of the field and the number of indices. The first index of the field is understood to be the Lorentz vector index.
- `ComplexVector`. The arguments are the name of the field, the name of its conjugate and the number of indices. The first index of both fields should be their corresponding Lorentz vector index.

- **VectorLikeFermion.** The first argument of the constructor is the name of the field. The second and third are the names of the left-handed and right-handed parts. The fourth and fifth are their conjugates. The last is the number of indices. The first index of the each of the four fields is taken to be their two-component spinor index.
- **MajoranaFermion.** The arguments are the name of the field and the name of its conjugate. The first index of both fields should be their two-component spinor index.

The constructors for the bosons have the optional arguments: `order` (default 2), specifying the order in  $(D/M)^2$  to which the solution to the equation of motion is to be expanded, and `max_dim` (default 4), representing the maximum allowed dimension for the operators appearing in this expansion. Both bosons and fermions receive the optional argument `has_flavor` (default True) stating whether the heavy field has a flavor index. In case it is true, the flavor index is taken to be the last one.

The heavy field classes include the quadratic terms for the kind of particle they represent, as well as the solutions to the equations of motion presented in section 2. The mass of a field `f` is represented by a tensor whose name is of the form `mass = "M" + f.name`. This tensor has one index if the heavy field has flavor and none otherwise.

Therefore, the first step for integration is defining the heavy fields:

```
heavy_f = HeavyFieldClass("field_name", ...)
```

Given an interaction Lagrangian `int_lag`, the integration is done using the function `integrate`, which takes as arguments a list of the heavy fields, the interaction Lagrangian and a maximum dimension `max_dim` for the operators of the effective theory. It returns the corresponding effective Lagrangian:

```
heavy_fields = [heavy_f_1, heavy_f_2, ...]
eff_lag = integrate(
    heavy_fields, int_lag, max_dim)
```

## 5. Transformations of the effective Lagrangian

After integration, the effective Lagrangian contains in general operators that are not independent. To rewrite it in terms of a set of independent operators some manipulations are needed, such as using identities for combinations of tensors related to the symmetry groups, integrating by parts to move derivatives from some fields to others, or using the equations of motion of the light fields.

The `matchingtools.transformations` module introduces the functions for doing this kind of manipulations and for the simplification of the Lagrangian. We will describe here the functions that are imported with

```
from matchingtools.transformations import (
    simplify, apply_rules)
```

First, the function `simplify` returns a simplified version of the Lagrangian it gets as an argument. Tensors representing a number that appear inside an operator are collected and multiplied. Tensors representing a symbolic constant exponentiated to some power are also collected to give only one tensor with the correct exponent. `simplify` also looks for Kronecker deltas (tensors with the name "kdelta" and two indices) removes them by contracting the corresponding indices.

The transformations of a Lagrangian are done using what we call here *rules*. A rule is a pair (a tuple with two elements) whose first element is an operator representing a *pattern* and whose second element is an operator sum representing a *replacement*. They are used by the function `apply_rules` to find occurrences of the pattern and replace them by the replacement. A rule is written as

```
rule = (Op(...), OpSum(Op(...), Op(...), ...))
```

The indices that appear in tensors inside the rule can be general integer numbers. Non-negative integers represent contracted indices, as explained in section 3. Negative indices are used for free indices and those in the replacement should match the corresponding ones in the pattern. For example the substitution of  $\sigma_{ij}^a \sigma_{kl}^b$  by  $2\delta_{il}\delta_{kj} - \delta_{ij}\delta_{kl}$  can be done using the rule

```
rule_fierz_SU2 = (
    Op(sigma(0, -1, -2), sigma(0, -3, -4)),
    OpSum(number_op(2) * Op(delta(-1, -4),
                                delta(-3, -2)),
          -Op(delta(-1, -2),
                delta(-3, -4))))
```

To transform the Lagrangian using integration by parts or equations of motion of the light fields the user should also specify the corresponding rules following this procedure.

The function `apply_rules` repeatedly tries to apply every rule of a list to each operator in an operator sum. If the pattern matches some part of an operator, the rule is applied and the operator sum updated. The first argument to `apply_rules` is the operator sum, the second is the list of rules and the last one is the number of iterations. It returns the resulting operator sum.

To rewrite the Lagrangian in terms of a chosen set of independent operators the procedure is: define the rules to get to the desired basis, add some rules to identify the operators and apply the function `apply_rules`.

The basis operators should be defined using `tensor_op`, a function that creates an operator with one tensor inside whose name is the argument of the function. Then write a rule to identify it. For example, for the operator  $O_{\phi D} = (\phi^\dagger D_\mu \phi)(D^\mu \phi)^\dagger \phi$  we would write

```
OphiD = tensor_op("OphiD")
rule_def_OphiD = (
    Op(phic(0), D(1, phi(0)),
        D(1, phic(0)), phi(0)),
    OpSum(OphiD))
```

If the basis operator in question has some flavor indices, `flavor_tensor_op` is to be used instead of `tensor_op`. It creates a callable object that takes the corresponding free indices as arguments. As an example, for the operator  $(O_{e\phi})_{ij} = \bar{l}_{Li}\phi e_{Rj}\phi^\dagger$  we would have:

```
Oephi = flavor_tensor_op("Oephi")
rule_def_Oephi = (
    Op(lLc(0, 1, -1), phi(1), eR(0, -2),
        phic(2), phi(2)),
    OpSum(Oephi(-1, -2)))
```

## 6. Output

The class `matchingtools.output.Writer` serves to nicely represent an effective Lagrangian. It is convenient that the final result is represented as a list of the coefficients of the operators in the basis. That is, if each of the terms of the Lagrangian contains a tensor that represents an operator of the basis, we would like to see what are the tensors that multiply each of them. This is what `Writer` does. If `eff_lag` is our final effective Lagrangian and `op_names` is a list of the names of the tensors representing the operators in the basis, do

```
eff_lag_writer = Writer(eff_lag, op_names)
```

The constructor admits an optional argument `conjugates`, a dictionary whose keys are the names of all the tensors involved in the final output and whose values are the names of their conjugates. This helps `Writer` collect pairs of conjugate products of tensors returning their real or imaginary part.

The string representation can be obtained just by using the `str` method of the class `Writer`. To write it to a text file use

```
eff_lag_writer.write_text_file(filename).
```

The method `write_latex_file` writes a LaTeX file with the representation. It receives four arguments: the name of the output file, the LaTeX representation of the tensors, the LaTeX representation of the coefficients of the basis operators and a list of the strings to be used to represent the indices. The LaTeX representations are given by dictionaries whose keys are the names of the tensors to be represented (or whose coefficient is to be represented) and whose values are the corresponding code. This code should contain placeholders for the necessary indices written as "{}" (Python's format style). To produce the characters "{" and "}" in the final code they should appear duplicated in the dictionary values.

For a better LaTeX output for the numerical coefficients, the parameter passed to `number_op` in the definitions should be either an `int` or a `fractions.Fraction`. In this context, the imaginary unit can be introduced by multiplying by the operator `core.i_op`.

## 7. An example

In this section we will be creating a simple model to show some of the features of *MatchingTools*. The model is described as follows: it has  $SU(2) \times U(1)$  gauge symmetry and contains a complex scalar doublet  $\phi$  (the Higgs) with hypercharge  $1/2$  and a real scalar triplet  $\Xi$  with zero hypercharge that couple as:

$$\mathcal{L}_{int} = -\kappa \Xi^a \phi^\dagger \sigma^a \phi - \lambda \Xi^a \Xi^a \phi^\dagger \phi, \quad (24)$$

where  $\kappa$  and  $\lambda$  are a coupling constants and  $\sigma^a$  are the Pauli matrices. We will then integrate out the heavy scalar  $\Xi$  to obtain an effective Lagrangian which we will finally write in terms of the operators

$$\begin{aligned} O_{\phi 6} &= (\phi^\dagger \phi)^3, & O_{\phi 4} &= (\phi^\dagger \phi)^2, \\ O_{\phi}^{(1)} &= \phi^\dagger \phi (D_\mu \phi)^\dagger D^\mu \phi, & O_{\phi}^{(3)} &= (\phi^\dagger D_\mu \phi) (D^\mu \phi)^\dagger \phi, \\ O_{D\phi} &= \phi^\dagger (D_\mu \phi) \phi^\dagger D^\mu \phi, & O_{D\phi}^* &= (D_\mu \phi)^\dagger \phi (D^\mu \phi)^\dagger \phi. \end{aligned} \quad (25)$$

Notice that this is not an independent set of operators, as some linear combinations of them are total derivatives. Because the purpose of this section is to present a very simple model, we will not be doing integration by parts and therefore we will not simplify the results any further.

### 7.1. Creation of the model

The required imports are

```
from matchingtools.operators import (
    TensorBuilder, FieldBuilder, Op, OpSum,
    number_op, tensor_op, boson, fermion, kdelta)

from matchingtools.integration import (
    RealScalar, integrate)

from matchingtools.transformations import (
    apply_rules)

from matchingtools.output import Writer
```

We will need three tensors, the Pauli matrices and the coupling constants:

```
sigma = TensorBuilder("sigma")
kappa = TensorBuilder("kappa")
lamb = TensorBuilder("lamb")
```

We will also use three fields: the Higgs doublet, its conjugate and the new scalar:

```
phi = FieldBuilder("phi", 1, boson)
phic = FieldBuilder("phic", 1, boson)
Xi = FieldBuilder("Xi", 1, boson)
```

Now we are ready to write the interaction Lagrangian:

```
interaction_Lagrangian = -OpSum(
    Op(kappa(), Xi(0), phic(1),
        sigma(0, 1, 2), phi(2)),
    Op(lamb(), Xi(0), Xi(0),
        phic(1), phi(1)))
```

## 7.2. Integration

To integrate out the heavy  $\Xi$  we write

```
heavy_Xi = RealScalar("Xi", 1, has_flavor=False)
effective_Lagrangian = integrate(
    [heavy_Xi], interaction_Lagrangian, 6)
```

## 7.3. Transformations of the effective Lagrangian

After the integration we get operators that contain  $(\phi^\dagger \sigma^a \phi)(\phi^\dagger \sigma^a \phi)$ . This product can be rewritten in terms of the operator  $(\phi^\dagger \phi)^2$ . To do this, we can use the  $SU(2)$  Fierz identity:

$$\sigma_{ij}^a \sigma_{kl}^a = 2\delta_{il}\delta_{kj} - \delta_{ij}\delta_{kl}. \quad (26)$$

We now know that we can define a rule to transform everything that matches the left-hand side of the equality into the expression in the right-hand side with the code

```
fierz_rule = (
    Op(sigma(0, -1, -2), sigma(0, -3, -4)),
    OpSum(number_op(2) * Op(kdelta(-1, -4),
                             kdelta(-3, -2)),
          -Op(kdelta(-1, -2),
              kdelta(-3, -4))))
```

We should now define the operators in terms of which we want to express the effective Lagrangian

```
Ophi6 = tensor_op("Ophi6")
Ophi4 = tensor_op("Ophi4")
O1phi = tensor_op("O1phi")
O3phi = tensor_op("O3phi")
ODphi = tensor_op("ODphi")
ODphic = tensor_op("ODphic")
```

and then use some rules to express them in terms of the fields and tensors that appear in the effective Lagrangian

```
definition_rules = [
    (Op(phic(0), phi(0), phic(1), phi(1),
        phic(2), phi(2)),
     OpSum(Ophi6)),
    (Op(phic(0), phi(0), phic(1), phi(1)),
     OpSum(Ophi4)),
    (Op(D(2, phic(0)), D(2, phi(0)),
        phic(1), phi(1)),
     OpSum(O1phi)),
    (Op(phic(0), D(2, phi(0)),
        D(2, phic(1)), phi(1)),
     OpSum(O3phi)),
    (Op(phic(0), D(2, phi(0)),
        phic(1), D(2, phi(1))),
     OpSum(ODphi)),
    (Op(D(2, phic(0)), phi(0),
        D(2, phic(1)), phi(1)),
     OpSum(ODphic))]
```

To apply the  $SU(2)$  Fierz identity to every operator until we get to the chosen operators, we do

```
rules = [fierz_rule] + definition_rules
max_iterations = 2
transf_eff_lag = apply_rules(
    effective_Lagrangian, rules,
    max_iterations)
```

## 7.4. Output

The class `Writer` can be used to represent the coefficients of the operators of a Lagrangian as plain text and write them to a file

```
final_coef_names = [
    "Ophi6", "Ophi4", "O1phi",
    "O3phi", "ODphi", "ODphic"]
eff_lag_writer = Writer(
    transf_eff_lag, final_coef_names)
eff_lag_writer.write_text_file(
    "simple_example_results.txt")
```

It can also write a LaTeX file with the representation of these coefficients and export it to pdf to show it directly. For this to be done, we should define how the objects that we are using are represented in LaTeX code and the symbols we want to be used as indices

```
latex_tensor_reps = {"kappa": r"\kappa",
                     "lamb": r"\lambda",
                     "MXi": r"M_{\Xi}",
                     "phi": r"\phi_{\phantom{a}}",
                     "phic": r"\phi^*_{\phantom{a}}"}

latex_op_reps = {
    "Ophi":
        r"\frac{\{\alpha_{\{\phi\}}\}\{\Lambda^2\}}{2}",
    "Ophi4":
        r"\alpha_{\{\phi\ 4\}}"}

latex_indices = ["i", "j", "k", "l"]

eff_lag_writer.write_latex(
    "simple_example", latex_tensor_reps,
    latex_op_reps, latex_indices)
```

The expected result is a `.tex` file (ready to be compiled) with the coefficients of the operators we defined.

## 8. Extras for beyond the Standard Model applications

*MatchingTools* includes a subpackage called `extras`, with some modules defining tensors and rules that are useful for the applications to physics beyond the Standard Model. These modules are `SU2`, `SU3`, `Lorentz`, `SM` and `SM_dim6_basis`. Other modules will be added in the future and will be available in the GitHub repository of the program, as well as in its updates in the pypi repository [55].

### 8.1. The $SU(2)$ module

This module defines the following tensors related to  $SU(2)$ :

- **epsSU2**: The totally antisymmetric tensor  $\epsilon_{ij}$  with two doublet indices and  $\epsilon_{12} = 1$ .
- **sigmaSU2**: The Pauli matrices  $\sigma_{ij}^a$ . The first index is the triplet index, whereas the second and third are the doublet ones.
- **CSU2** and **CSU2c**: the Clebsh-Gordan coefficients  $C_{a\beta}^I$  with the first index  $I$  being a quadruplet index, the second  $a$  a triplet index, and the third  $\beta$  a doublet index. The tensor  $C$  contracted with the corresponding three objects produces a singlet.
- **epsSU2triplets**: Totally antisymmetric tensor  $\epsilon_{abc}$  with three  $SU(2)$  triplet indices such that  $\epsilon_{123} = 1$ .
- **fSU2**: Totally antisymmetric tensor with three  $SU(2)$  triplet indices given by  $f_{abc} = \frac{i}{\sqrt{2}}\epsilon_{abc}$ .

It also implements the rules for taking expressions with  $\epsilon_{ij}\epsilon_{kl}$ ,  $\sigma_{ij}^a\sigma_{kl}^a$ ,  $C_{ap}^I\epsilon_{pm}\sigma_{ij}^aC_{bq}^{I*}\epsilon_{qn}\sigma_{kl}^b$  or contractions of anti-symmetric tensors, and rewriting them in terms of Kronecker deltas. All the rules are collected in the list `rules_SU2`. The LaTeX representation of the tensors defined is given by the dictionary `latex_SU2`.

### 8.2. The $SU(3)$ module

The  $SU(3)$  tensors defined in this module are:

- **epsSU3**: Totally antisymmetric tensor  $\epsilon_{ABC}$  with three  $SU(3)$  triplet indices such that  $\epsilon_{123} = 1$ .
- **TSU3**:  $SU(3)$  generators  $(T_A)_{BC} = \frac{1}{2}(\lambda_A)_{BC}$ , where  $\lambda_A$  are the Gell-Mann matrices. The first index is the octet index. The second and third are the anti-triplet and triplet ones.
- **fSU3**:  $SU(3)$  structure constants  $f_{ABC}$ .

The rule for transforming  $\epsilon_{ijk}\epsilon_{ilm}$  into a combination of Kronecker deltas is implemented. It is included in the one-element list `rules_SU3`. The LaTeX representation of the tensors defined is in `latex_SU3`.

### 8.3. The Lorentz module

This module includes the tensors `epsUp`, `epsUpDot`, `epsDown`, `epsDownDot`, `sigma4`, `sigma4bar` from `matchingtools.operators` and defines:

- **eps4**: Totally antisymmetric tensor  $\epsilon_{\mu\nu\rho\sigma}$  with four Lorentz vector indices where  $\epsilon_{0123} = 1$ .
- **sigmaTensor**: Lorentz tensor

$$\sigma^{\mu\nu} = \frac{i}{4} (\sigma_{\alpha\gamma}^\mu \bar{\sigma}^{\nu\beta} - \sigma_{\alpha\gamma}^\nu \bar{\sigma}^{\mu\beta}). \quad (27)$$

The list `rules_Lorentz` contains the rules for substituting  $\epsilon^{\alpha\beta}\epsilon^{\dot{\alpha}\dot{\beta}}$  by  $\frac{1}{2}\bar{\sigma}^{\mu,\dot{\alpha}\alpha}\bar{\sigma}^{\dot{\beta}\beta}_\mu$ ,  $\epsilon_{\alpha\beta}\epsilon_{\dot{\alpha}\dot{\beta}}$  by  $\frac{1}{2}\bar{\sigma}^{\mu}_{\alpha\dot{\alpha}}\bar{\sigma}^{\dot{\beta}\beta}_\mu$  and contracted  $\epsilon$  tensors by combinations of Kronecker deltas.

### 8.4. The $SM$ module

Here, the tensors corresponding to the Standard Model fields and its gauge coupling constants, Yukawa couplings and CKM matrix are defined.

The Standard Model fields are:

- **phi** and **phic**: The Higgs boson and its conjugate. One  $SU(2)$  doublet index.
- **lL** and **lLc**: The left-handed lepton doublet. Its indices are, in order: the two-component spinor index, the  $SU(2)$  doublet index and the flavor index.
- **qL** and **qLc**: The left-handed quark doublet. Its indices are: the two-component spinor index, the  $SU(3)$  triplet (or anti-triplet) index, the  $SU(2)$  doublet index and the flavor index.
- **eR** and **eRc**: The right-handed electron. Indices: two-component spinor and flavor.
- **uR** and **uRc**: The right-handed up quark. Indices: two-component spinor,  $SU(3)$  triplet (or antitriplet) and flavor.
- **dR** and **dRc**: The right-handed down quark. Indices: two-component spinor,  $SU(3)$  triplet (or antitriplet) and flavor.
- **bFS**:  $U(1)$  field strength tensor. Two Lorentz vector indices.
- **wFS**:  $SU(2)$  field strength tensor. Two Lorentz vector indices and one  $SU(2)$  triplet index.
- **gFS**:  $SU(3)$  field strength tensor. Two Lorentz vector indices and one  $SU(3)$  octet index.

The constant tensors are:

- **gb** and **gw**: The  $U(1)$  and  $SU(2)$  gauge coupling constants.
- **ye**, **yec**, **yd**, **ydc**, **yu** and **yuc**: The diagonalized Yukawa matrices for the leptons, the down quarks, the up quarks and their conjugates. They have two indices: the first one corresponds to the flavor of the doublets and the second to the flavor of the singlets.
- **V** and **Vc**: CKM matrix.

The module also includes a list of rules `eoms_SM`, defined to substitute the equations of motion, replacing derivatives of the Standard Model fields by a combination of the other fields. There is a dictionary `latex_SM` containing the LaTeX representation of the tensors that are defined.

### 8.5. The $SM_{dim_6}$ basis module

In this module, the basis for the Standard Model effective Lagrangian up to dimension six that appears in [29] is defined. The rules to identify them are given in the list `rules_basis_definition`. The LaTeX representation of their coefficients is in `latex_basis_coefs`. Modules containing other bases, such as the one in [46], will be added in the future.



## 9. Using *MatchingTools* with other types of fields

As explained above, *MatchingTools* can integrate scalars, vector-like or Majorana fermions, and vectors in Lorentz-invariant theories. For this purpose, several classes representing the heavy fields are supplied. Other kinds of fields (for instance, with non canonical kinetic terms, spin  $> 1$ , or non relativistic) can be treated as well, once the corresponding class for it is provided.

Specifically, to treat a new type of field one should define a Python class implementing the following methods:

- `equations_of_motion`. Receives an `OperatorSum` object representing an interaction Lagrangian. Returns a dictionary whose keys are strings with the names of the heavy fields involved (for example, a field and its conjugate, if it is a complex boson) and whose values are `OperatorSum` objects representing the corresponding solution to their equation of motion. These solutions can be written in terms of other heavy fields, but they should be such that iterative substitutions of their respective equations motion reaches a point where no heavy fields appear to the desired order in the dimension of the operators.
- `quadratic_terms`. Does not have any parameters. Returns the kinetic and mass terms of the corresponding heavy field.

For the definition of these methods, it is recommended to use the tools provided by the core module. Once such a class is defined, its objects can be included in the list of heavy fields to be passed to `integration.integrate` and they will be dealt with in the same way as the others.

## 10. Conclusions

We have presented *MatchingTools*, a Python library implementing symbolic tree-level integration of heavy fields for any given model. It is also able to transform the resulting Lagrangian using rules specified by the user to remove redundant operators. With this program one can safely automatize these kind of calculations, which practically eliminates the possibility of algebraic errors and drastically reduces the calculation times. Even calculations with complex Lagrangians involving  $\sim 100$  independent terms (thousands of terms in some intermediate steps) can be performed in about thirty seconds (using a 2.6GHz Intel Core i5 processor).

## Acknowledgments

The author would like to thank J. de Blas, M. Pérez-Victoria and J. Santiago for their very useful guidance, comments and corrections.

**Funding:** This work was supported by the Spanish MECD grant FPU14, the Spanish MINECO grants FPA2013-47836-C3-2-P and FPA2016-78220-C3-1-P (Fondos FEDER) and the Junta de Andalucía grant FQM101.

## References

- [1] H. Georgi, *Weak Interactions and Modern Particle Theory*, 1984.
- [2] J. F. Donoghue, E. Golowich, B. R. Holstein, *Dynamics of the standard model*, *Camb. Monogr. Part. Phys. Nucl. Phys. Cosmol.* 2 (1992) 1–540, [*Camb. Monogr. Part. Phys. Nucl. Phys. Cosmol.*35(2014)].
- [3] M. Aaboud, et al., Search for scalar leptoquarks in pp collisions at  $\sqrt{s} = 13$  TeV with the ATLAS experiment, *New J. Phys.* 18 (9) (2016) 093016. [arXiv:1605.06035](#), [doi:10.1088/1367-2630/18/9/093016](#).
- [4] M. Aaboud, et al., Search for pair production of vector-like top quarks in events with one lepton, jets, and missing transverse momentum in  $\sqrt{s} = 13$  TeV *pp* collisions with the ATLAS detector, *JHEP* 08 (2017) 052. [arXiv:1705.10751](#), [doi:10.1007/JHEP08\(2017\)052](#).
- [5] A. M. Sirunyan, et al., Searches for W bosons decaying to a top quark and a bottom quark in proton-proton collisions at 13 TeV, *JHEP* 08 (2017) 029. [arXiv:1706.04260](#), [doi:10.1007/JHEP08\(2017\)029](#).
- [6] A. M. Sirunyan, et al., Search for heavy resonances that decay into a vector boson and a Higgs boson in hadronic final states at  $\sqrt{s} = 13$  TeV, *Eur. Phys. J. C* 77 (9) (2017) 636. [arXiv:1707.01303](#), [doi:10.1140/epjc/s10052-017-5192-z](#).
- [7] I. Brivio, M. Trott, *The Standard Model as an Effective Field Theory*, [arXiv:1706.08945](#).
- [8] Z. Han, W. Skiba, Effective theory analysis of precision electroweak data, *Phys. Rev. D* 71 (2005) 075009. [arXiv:hep-ph/0412166](#), [doi:10.1103/PhysRevD.71.075009](#).
- [9] J. de Blas, M. Chala, J. Santiago, Global Constraints on Lepton-Quark Contact Interactions, *Phys. Rev. D* 88 (2013) 095011. [arXiv:1307.5068](#), [doi:10.1103/PhysRevD.88.095011](#).
- [10] J. de Blas, Electroweak limits on physics beyond the Standard Model, *EPJ Web Conf.* 60 (2013) 19008. [arXiv:1307.6173](#), [doi:10.1051/epjconf/20136019008](#).
- [11] R. S. Gupta, A. Pomarol, F. Riva, BSM Primary Effects, *Phys. Rev. D* 91 (3) (2015) 035001. [arXiv:1405.0181](#), [doi:10.1103/PhysRevD.91.035001](#).
- [12] J. Ellis, V. Sanz, T. You, The Effective Standard Model after LHC Run I, *JHEP* 03 (2015) 157. [arXiv:1410.7703](#), [doi:10.1007/JHEP03\(2015\)157](#).
- [13] A. Falkowski, F. Riva, Model-independent precision constraints on dimension-6 operators, *JHEP* 02 (2015) 039. [arXiv:1411.0669](#), [doi:10.1007/JHEP02\(2015\)039](#).
- [14] J. de Blas, M. Chala, J. Santiago, Renormalization Group Constraints on New Top Interactions from Electroweak Precision Data, *JHEP* 09 (2015) 189. [arXiv:1507.00757](#), [doi:10.1007/JHEP09\(2015\)189](#).
- [15] L. Berthier, M. Trott, Consistent constraints on the Standard Model Effective Field Theory, *JHEP* 02 (2016) 069. [arXiv:1508.05060](#), [doi:10.1007/JHEP02\(2016\)069](#).
- [16] A. Pomarol, F. Riva, Towards the Ultimate SM Fit to Close in on Higgs Physics, *JHEP* 01 (2014) 151. [arXiv:1308.2803](#), [doi:10.1007/JHEP01\(2014\)151](#).
- [17] J. de Blas, M. Ciuchini, E. Franco, S. Mishima, M. Pierini, L. Reina, L. Silvestrini, The Global Electroweak and Higgs Fits in the LHC era, 2017. [arXiv:1710.05402](#).
- [18] T. Appelquist, C. W. Bernard, Strongly Interacting Higgs Bosons, *Phys. Rev. D* 22 (1980) 200. [doi:10.1103/PhysRevD.22.200](#).
- [19] A. C. Longhitano, Heavy Higgs Bosons in the Weinberg-Salam Model, *Phys. Rev. D* 22 (1980) 1166. [doi:10.1103/PhysRevD.22.1166](#).
- [20] A. C. Longhitano, Low-Energy Impact of a Heavy Higgs Boson Sector, *Nucl. Phys. B* 188 (1981) 118–154. [doi:10.1016/0550-3213\(81\)90109-7](#).
- [21] F. Feruglio, The Chiral approach to the electroweak interactions, *Int. J. Mod. Phys. A* 8 (1993) 4937–4972. [arXiv:hep-ph/9301281](#), [doi:10.1142/S0217751X93001946](#).
- [22] H. Georgi, S. L. Glashow, Unity of All Elementary Particle Forces, *Phys. Rev. Lett.* 32 (1974) 438–441. [doi:10.1103/PhysRevLett.32.438](#).
- [23] H. Fritzsch, P. Minkowski, Unified Interactions of Leptons and Hadrons, *Annals Phys.* 93 (1975) 193–266. [doi:10.1016/0003-4916\(75\)90211-0](#).
- [24] N. Arkani-Hamed, S. Dimopoulos, G. R. Dvali, The Hierarchy problem and new dimensions at a millimeter, *Phys. Lett. B* 429 (1998) 263–272. [arXiv:hep-ph/9803315](#), [doi:10.1016/S0370-2693\(98\)00466-3](#).
- [25] L. Randall, R. Sundrum, A Large mass hierarchy from a

- small extra dimension, *Phys. Rev. Lett.* 83 (1999) 3370–3373. [arXiv:hep-ph/9905221](#), [doi:10.1103/PhysRevLett.83.3370](#).
- [26] F. del Aguila, M. Perez-Victoria, J. Santiago, Observable contributions of new exotic quarks to quark mixing, *JHEP* 09 (2000) 011. [arXiv:hep-ph/0007316](#), [doi:10.1088/1126-6708/2000/09/011](#).
- [27] F. del Aguila, J. de Blas, M. Perez-Victoria, Effects of new leptons in Electroweak Precision Data, *Phys. Rev. D* 78 (2008) 013010. [arXiv:0803.4008](#), [doi:10.1103/PhysRevD.78.013010](#).
- [28] F. del Aguila, J. de Blas, M. Perez-Victoria, Electroweak Limits on General New Vector Bosons, *JHEP* 09 (2010) 033. [arXiv:1005.3998](#), [doi:10.1007/JHEP09\(2010\)033](#).
- [29] J. de Blas, M. Chala, M. Perez-Victoria, J. Santiago, Observable Effects of General New Scalar Particles, *JHEP* 04 (2015) 078. [arXiv:1412.8480](#), [doi:10.1007/JHEP04\(2015\)078](#).
- [30] C. M. Fraser, Calculation of Higher Derivative Terms in the One Loop Effective Lagrangian, *Z. Phys. C* 28 (1985) 101. [doi:10.1007/BF01550255](#).
- [31] I. J. R. Aitchison, C. M. Fraser, Fermion Loop Contribution to Skyrmon Stability, *Phys. Lett.* 146B (1984) 63–66. [doi:10.1016/0370-2693\(84\)90644-0](#).
- [32] I. J. R. Aitchison, C. M. Fraser, Derivative Expansions of Fermion Determinants: Anomaly Induced Vertices, Goldstone-Wilczek Currents and Skyrme Terms, *Phys. Rev. D* 31 (1985) 2605. [doi:10.1103/PhysRevD.31.2605](#).
- [33] I. J. R. Aitchison, C. M. Fraser, Trouble With Boson Loops in Skyrmon Physics, *Phys. Rev. D* 32 (1985) 2190. [doi:10.1103/PhysRevD.32.2190](#).
- [34] L. H. Chan, EFFECTIVE ACTION EXPANSION IN PERTURBATION THEORY, *Phys. Rev. Lett.* 54 (1985) 1222–1225, [Erratum: *Phys. Rev. Lett.* 56,404(1986)]. [doi:10.1103/PhysRevLett.54.1222](#).
- [35] L.-H. Chan, Derivative Expansion for the One Loop Effective Actions With Internal Symmetry, *Phys. Rev. Lett.* 57 (1986) 1199. [doi:10.1103/PhysRevLett.57.1199](#).
- [36] M. K. Gaillard, The Effective One Loop Lagrangian With Derivative Couplings, *Nucl. Phys. B* 268 (1986) 669–692. [doi:10.1016/0550-3213\(86\)90264-6](#).
- [37] O. Cheyette, Derivative Expansion of the Effective Action, *Phys. Rev. Lett.* 55 (1985) 2394. [doi:10.1103/PhysRevLett.55.2394](#).
- [38] F. del Aguila, Z. Kunszt, J. Santiago, One-loop effective lagrangians after matching, *Eur. Phys. J. C* 76 (5) (2016) 244. [arXiv:1602.00126](#), [doi:10.1140/epjc/s10052-016-4081-1](#).
- [39] S. A. R. Ellis, J. Quevillon, T. You, Z. Zhang, Mixed heavy-light matching in the Universal One-Loop Effective Action, *Phys. Lett. B* 762 (2016) 166–176. [arXiv:1604.02445](#), [doi:10.1016/j.physletb.2016.09.016](#).
- [40] J. Fuentes-Martin, J. Portoles, P. Ruiz-Femenia, Integrating out heavy particles with functional methods: a simplified framework, *JHEP* 09 (2016) 156. [arXiv:1607.02142](#), [doi:10.1007/JHEP09\(2016\)156](#).
- [41] Z. Zhang, Covariant diagrams for one-loop matching, *JHEP* 05 (2017) 152. [arXiv:1610.00710](#), [doi:10.1007/JHEP05\(2017\)152](#).
- [42] R. E. Kallosh, I. V. Tyutin, The Equivalence theorem and gauge invariance in renormalizable theories, *Yad. Fiz.* 17 (1973) 190–209, [*Sov. J. Nucl. Phys.* 17,98(1973)].
- [43] H. D. Politzer, Power Corrections at Short Distances, *Nucl. Phys. B* 172 (1980) 349–382. [doi:10.1016/0550-3213\(80\)90172-8](#).
- [44] H. Georgi, On-shell effective field theory, *Nucl. Phys. B* 361 (1991) 339–350. [doi:10.1016/0550-3213\(91\)90244-R](#).
- [45] C. Arzt, Reduced effective Lagrangians, *Phys. Lett. B* 342 (1995) 189–195. [arXiv:hep-ph/9304230](#), [doi:10.1016/0370-2693\(94\)01419-D](#).
- [46] B. Grzadkowski, M. Iskrzynski, M. Misiak, J. Rosiek, Dimension-Six Terms in the Standard Model Lagrangian, *JHEP* 10 (2010) 085. [arXiv:1008.4884](#), [doi:10.1007/JHEP10\(2010\)085](#).
- [47] R. Contino, M. Ghezzi, C. Grojean, M. Muhlleitner, M. Spira, Effective Lagrangian for a light Higgs-like scalar, *JHEP* 07 (2013) 035. [arXiv:1303.3876](#), [doi:10.1007/JHEP07\(2013\)035](#).
- [48] E. Masso, An Effective Guide to Beyond the Standard Model Physics, *JHEP* 10 (2014) 128. [arXiv:1406.6376](#), [doi:10.1007/JHEP10\(2014\)128](#).
- [49] A. Falkowski, B. Fuks, K. Mawatari, K. Mimasu, F. Riva, V. Sanz, Rosetta: an operator basis translator for Standard Model effective field theory, *Eur. Phys. J. C* 75 (12) (2015) 583. [arXiv:1508.05895](#), [doi:10.1140/epjc/s10052-015-3806-x](#).
- [50] J. de Blas, J. C. Criado, M. Perez-Victoria, J. Santiago, Effective description of general extensions of the Standard Model: the complete tree-level dictionary, *JHEP* 03 (2018) 109. [arXiv:1711.10391](#), [doi:10.1007/JHEP03\(2018\)109](#).
- [51] N. D. Christensen, C. Duhr, FeynRules - Feynman rules made easy, *Comput. Phys. Commun.* 180 (2009) 1614–1641. [arXiv:0806.4194](#), [doi:10.1016/j.cpc.2009.02.018](#).
- [52] A. Alloul, N. D. Christensen, C. Degrande, C. Duhr, B. Fuks, FeynRules 2.0 - A complete toolbox for tree-level phenomenology, *Comput. Phys. Commun.* 185 (2014) 2250–2300. [arXiv:1310.1921](#), [doi:10.1016/j.cpc.2014.04.012](#).
- [53] A. Celis, J. Fuentes-Martin, A. Vicente, J. Virto, DsixTools: The Standard Model Effective Field Theory Toolkit, *Eur. Phys. J. C* 77 (6) (2017) 405. [arXiv:1704.04504](#), [doi:10.1140/epjc/s10052-017-4967-6](#).
- [54] I. Brivio, Y. Jiang, M. Trott, The SMEFTsim package, theory and tools [arXiv:1709.06492](#).
- [55] pip: The pypa recommended tool for installing python packages. URL <https://pypi.python.org/pypi/pip/>.
- [56] H. K. Dreiner, H. E. Haber, S. P. Martin, Two-component spinor techniques and Feynman rules for quantum field theory and supersymmetry, *Phys. Rept.* 494 (2010) 1–196. [arXiv:0812.1594](#), [doi:10.1016/j.physrep.2010.05.002](#).