matchingtools Documentation

Juan Carlos Criado Alamo

Mar 17, 2022

Contents:

MatchingTools is a Python library for doing symbolic calculations in effective field theory.

It provides the tools to create a Lagrangian and integrate out heavy fields at the tree level. It also includes functions for applying customizable transformations (for example, Fierz identities or simplification using equations of motion of the light fields) to the effective Lagrangian to simplify it or write it in terms of a chosen effective operator basis.

CHAPTER 1

Quickstart

1.1 Installation

To install MatchingTools using pip do:

```
pip install matchingtools
```

The source can be downloaded from the GitHub repository.

1.2 A simple example

In this section we will be creating a simple model to show some of the features of Effective. This example and more, involving more complex cases that make use of the *extras* package can be found in the examples folder at the GitHub repository of the project.

The model is described as follows: it has symmetry $SU(2) \times U(1)$ containing a complex scalar doublet ϕ (the Higgs) with hypercharge $1/2$ and a real scalar triplet Ξ with zero hypercharge that couple as:

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

where κ and λ are a coupling constants and σ^a are the Pauli matrices. We will then integrate out the heavy scalar Ξ to obtain an effective Lagrangian which we will finally write in terms of the operators

$$\begin{aligned} \mathcal{O}_{\phi 6} &= (\phi^\dagger \phi)^3, & \mathcal{O}_{\phi 4} &= (\phi^\dagger \phi)^2, \\ \mathcal{O}_\phi^{(1)} &= \phi^\dagger \phi (D_\mu \phi)^\dagger D^\mu \phi, & \mathcal{O}_\phi^{(3)} &= (\phi^\dagger D_\mu \phi)(D^\mu \phi)^\dagger \phi, \\ \mathcal{O}_{D\phi} &= \phi^\dagger (D_\mu \phi) \phi^\dagger D^\mu \phi, & \mathcal{O}_{D\phi}^* &= (D_\mu \phi)^\dagger \phi (D^\mu \phi)^\dagger \phi \end{aligned}$$

1.2.1 Creation of the model

The imports that we will need 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
```

The basic building blocks of our model are **tensors** and **fields**. For our example, 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)
```

The second argument of `FieldBuilder` represent the energy dimensions of the field, and the third corresponds its the statistics and can either be `boson` or `` `fermion ``.

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)))
```

1.2.2 Integration

Before doing the integration of the heavy fields, we must specify who they are. To integrate out the heavy Ξ we do:

```
heavy_Xi = RealScalar("Xi", 1, has_flavor=False)
```

Now it is ready to be integrated out:

```
heavy_fields = [heavy_Xi]
max_dim = 6
effective_lagrangian = integrate(
    heavy_fields, interaction_lagrangian, max_dim)
```

1.2.3 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:

$$\sigma^a_{ij}\sigma^a_{kl} = 2\delta_{il}\delta_{kj} - \delta_{ij}\delta_{kl}.$$

We can define a rule to transform everything that matches the left-hand side of the equality into the right-hand 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))))
```

Notice the use of the function `number_op`. Observe also the appearance of negative indices to represent free (not contracted) indices and how those of the replacement match the ones in the pattern.

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 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)
```

1.2.4 Output

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

```
final_coef_names = [
  "Ophi6", "Ophi4", "O1phi", "O3phi", "ODphi", "ODphic"]
eff_lag_writer = Writer(trasnf_eff_lag, final_coef_names)
eff_lag_writer.write_text_file("simple_example")
```

It can also to 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 have to be 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_{}",
                     "phic": r"\phi^*_{}"}

latex_coef_reps = {
  "Ophi6": r"\frac{{\alpha_{{\phi 6}}}}{{\Lambda^2}}",
  "Ophi4": r"\alpha_{{\phi 4}}",
  "O1phi": r"\frac{{\alpha^{{(1)}}_{{\phi}}}}{{\Lambda^2}}",
  "O3phi": r"\frac{{\alpha^{{(3)}}_{{\phi}}}}{{\Lambda^2}}",
  "ODphi": r"\frac{{\alpha_{{D\phi}}}}{{\Lambda^2}}",
  "ODphic": r"\frac{{\alpha^*_{{D\phi}}}}{{\Lambda^2}}"}

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

eff_lag_writer.write_pdf(
    "simple_example", latex_tensor_reps,
    latex_coef_reps, latex_indices)
```

Double curly brackets are used when one curly bracket should be present in the LaTeX code and simple curly brackes are used as placeholders for indices.

The expected result is a pdf file containing the coefficients for the operators we defined plus some other operators with covariant derivatives of the Higgs.

CHAPTER 2

Overview

2.1 Creation of models

Note: This section assumes that the classes and functions from `matchingtools.operators` that it uses are in the namespace. To import all the classes and functions that appear here do:

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

The construction of a model is done in two steps: the creation of the tensors and fields and the definition of the interaction Lagrangian.

The basic building block for a Lagrangian is tensor, an object of the class `Tensor`. Direct usage of the `Tensor` constructor obscures the code. There are two classes defined to make the process of creating tensors easier and cleaner, `TensorBuilder` and `FieldBuilder`.

When a field has covariant derivatives applied to it, that is represented internally in the attributes of its representative `Tensor` object. For aesthetical reasons and easeness of usage, instead of manually modifying the attributes, it's better to use the function `D` to create covariant derivatives of fields.

MatchingTools handles Lagrangians that are polynomials of the fields. They are thus a sum of terms that are products of tensors. They are represented as `OperatorSum` objects, with only one attribute: a list of its terms. Each term should be an operator, that is, a product of tensors represented by an `Operator` object with only one attribute: a list of its factors.

Instead of using the constructors for both classes, the functions `OpSum` and `Op` are available to make the definitions clearer.

Minus signs are defined in the library for operators (`Operator.__minus__()`) and operator sums (`OperatorSum.__minus__()`). Multiplication `*` is defined for operators too (as the concatenation of the tensors they contain, see `Operator.__mul__()`).

MatchingTools treats in a special way tensors whose name starts and ends with square or curly brakets. A tensor name enclosed in square brakets is understood as a (complex) number to be read from the name using `float(name[1:-1])`. The function `number_op()` is to be used to create operators with such tensors inside.

When the name of a tensor starts and ends with curly brakets it's it represents some symbolic constant that appears exponentiated. The name should be of the form `"{base^exponent}"`. Curly brakets allow for the summation of the exponents of tensors that appear in the same tensor and have the same base and indices. This is used mainly to produce more readable results. The function designed to create operators containing such tensors is `power_op()`.

2.1.1 Creation of tensors

Create a tensor as:

```
my_tensor = TensorBuilder("my_tensor")
```

and then use it inside an operator:

```
Op(..., my_tensor(ind1, ind2, ...), ...)
```

with `ind1`, `ind2`, ... being integers.

2.1.2 Creation of fields

Create a field as:

```
my_field = FieldBuilder("my_field", dimension, statistics)
```

where dimension (float) represents the energy dimensions of the field and statistics is either equal to `matchingtools.algebra.boson` or `matchingtools.algebra.fermion`. Then use it inside an operator:

```
Op(..., my_field(ind1, ind2, ...), ...)
```

with `ind1`, `ind2`, ... being integers.

2.1.3 Definition of the Lagrangian

Define the interaction Lagrangian as an operator sum:

```
int_lag = OpSum(op1, op2, ...)
```

Each argument to the function `matchingtools.operators.OpSum()` should be an operator defined as:

```
op1 = Op(tens1(ind1, ind2, ...), field1(ind3, ind4, ...), ...)
```

The arguments of the function `matchingtools.operators.Op()` are tensors (and fields). Their indices are integer numbers. Negative integers are reserved for free indices. Free indices are not meant to be used in the operators appearing in the Lagrangian, but later in the definition of their transformations.

Non-negative integers represent contracted indices. Contraction is expressed by repetition of indices.

To introduce the covariant derivative with index `ind` of a tensor `tens` inside an operator, use the function `matchingtools.operators.D()` in the following way:

```
D(ind, tens(ind1, ind2, ...))
```

If a numeric coefficient `num` is needed for some operator `op` it can be introduced as:

```
number_op(num) * op
```

A symbolic constant `s` to some power `p` can multiply an operator as:

```
power_op("s", p) * op
```

2.2 Integration

Note: This section assumes that the classes and functions from *matchingtools.integration* that it uses are in the namespace. To import all the classes and functions that appear here do:

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

To integrate some heavy fields out of a previously constructed Lagrangian the heavy fields should be specified first. The heavy fields should be objects of any of the following classes:

- *RealScalar*
- *ComplexScalar*
- *RealVector*
- *ComplexVector*
- *VectorLikeFermion*
- *MajoranaFermion*

Create a heavy field using the constructors of these classes.

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

Then collect the heavy fields in a list and use the function *integrate()* to perform the integration:

```
heavy_fields = [heavy_field_1, heavy_field_2, ...]
eff_lag = integrate(heavy_fields, int_lag, max_dim=...)
```

2.3 Transformations of the effective Lagrangian

Note: This section assumes that the functions from *matchingtools.transformations* and *matchingtools.transformations* that it uses are in the namespace. To import all the functions that appear here do:

```
from matchingtools.operators import tensor_op, flavor_tensor_op

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

An effective Lagrangian obtained from integration of heavy fields usually contains operators that aren't independent. Several transformations can be applied to them to write the Lagrangian in terms of a set of operators (a basis) that spans the space of effective operators.

These transformations are such as Fierz identities or substitutions of the equations of motion of the light particles. All of them consist of the subsitution of operators or parts of them by sums of other operators. The operations described in this section applied to effective Lagrangians or to any other kind of operators sum.

The first step to simplify an effective Lagrangian is to collect and multiply numeric coefficients and constant tensors that appear several times inside the same operator. To do this, use:

```
simplified_lag_1 = simplify(effective_lagrangian)
```

Then we can define a set of rules as a list of pairs `(pattern, replacement)` where `pattern` is an operator and `replacement` is an operator sum:

```
rules = [(Op(...), OpSum(...)), (Op(...), OpSum(...)), ...]
```

`pattern` may contain tensors with negative indices corresponding to indices that are not contracted inside `pattern`. In that case, the operators in replacement should also contain the same negative indices. When `pattern` is substituted inside an operator `op`, the indices in `op` outside `pattern` that are contracted with indices inside pattern appear as contracted with the corresponding ones in the operators of `replacement`. For example, to replace $t_{ij}r_{ik}$ by $-s_{mnk}u_{nmj}$ we would write the rule:

```
(Op(t(1, -1), r(1, -2)), OpSum(-Op(s(1, 2, -2), u(2, 1, -1))))
```

The operators of the basis should be represented by tensor with a name identifing the operator. They can be defined using `matchingtools.operators.tensor_op()` when they don't have free indices and `matchingtools.operators.flavor_tensor_op()` when they do. So we usually define:

```
Op1 = tensor_op("Op1")
Op2 = tensor_op("Op2")
...

Opf1 = flavor_tensor_op("Opf1")
Opf2 = flavor_tensor_op("Opf1")
...
```

and then specify how to identify them using rules:

```
op_def_rules = [(Op(...), OpSum(Op1)),
                (Op(...), OpSum(Op2)),
                ...
                (Op(...), OpSum(Opf1(i1, i2, ...))),
                (Op(...), OpSum(Opf2(i1, i2, ...)))
                ...]
```

Then we are ready to apply the rules using *`apply_rules()`*:

```
simplified_lag_2 = apply_rules_until(
    simplified_lag_1, rules + op_def_rule, max_iter)
```

where `max_iter` is the maximum number of applications of all the rules to each operator.

2.4 Output of the results

Note: This section assumes that the class *`matchingtools.output.Writer`* that it uses is in the namespace. To import it, do:

```
from matchingtools.output import Writer
```

It's usually convenient to organize the final results by presenting the coefficient to each operator of the effective Lagrangian. When a set of rules has been applied to the effective Lagrangian so that it is written as an `matchingtools.operators.OperatorSum` whose elements are `matchingtools.operators.Operator` objects each of which contains one tensor representing the actual operator in the basis and other tensors representing the coefficient the operator has.

To output the results in this form in a human readable format, the *`Writer`* is provided. If `op_names` is a list of the names of the tensors representing the operators in the basis and `lag` is the Lagrangian that we want to write, we do:

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

To write the results to a file in plain text, just use:

```
lag_writer.write_text_file("filename")
```

To write it in LaTeX two python dictionaries expressing how the tensors that appear in the coefficients and how the name of the coefficients for the operators should be written in LaTeX:

```
tensors_latex = {"tensor1": r"latexrep", "tensor2": ..., ...}
ops_latex = {"op1": r"latexrep", ...}
```

The values of the dictionary should be code to be written inside some LaTeX equation environment. It is recommended to use `r"..."` instead of `"..."` to easily write instructions as `\instr` instead of the form `\\instr` that would be needed for the case with just `"..."`. The placeholders for the indices should be written in python's `str.format` style `"{}"`. This implies that whenever curly braces are needed for the LaTeX code, double braces `{{...}}` should be used.

The symbols to de used to represent indices should be given also as a list of strings containing the LaTeX code representing them:

```
indices_latex = ["i", "j", ...]
```

Finally we can write the LaTeX document using:

```
lag_writer.write_latex("filename", tensors_latex, ops_latex, indices_latex)
```

Or we can instead use `Writer.show_latex()` to write it, compile it and show it all in method:

```
lag_writer.show_latex("filename", pdf_viewer, tensors_latex,
                      ops_latex, indices_latex)
```

CHAPTER 3

Modules documentation

3.1 The `operators` module

Core module with the definitions of the basic building blocks: the classes *Tensor*, *Operator*, and *OperatorSum*. Implements the Leibniz rule for derivatives and the algorithms for matching and replacing as well as functional derivatives.

Defines interfaces for the creation of tensors, fields, operators and operator sums: *TensorBuilder*, *FieldBuilder*, *Op()* and *OpSum()*; the interface for creating derivatives of single tensors: the function *D()*; and interfaces for creating special single-tensor associated to (complex) numbers and powers of constants.

Defines the Lorentz tensors *epsUp*, *epsUpDot*, *epsDown*, *epsDownDot*, *sigma4* and *sigma4bar*.

class matchingtools.core.**Tensor**(*name*, *indices*, *is_field=False*, *num_of_der=0*, *dimension=0*, *statistics=True*, *content=None*, *exponent=None*)

Basic building block for operators.

A tensor might have some derivatives applied to it. The indices correponding to the derivatives are given by the first indices in the list of indices of the Tensor.

name
identifier

Type string

indices
indices of the tensor and the derivaties applied to it

Type list of ints

is_field
specifies whether it is non-constant

Type bool

num_of_der
number of derivatives acting

Type int

dimension
energy dimensions

Type int

statistics
True for bosons and False for fermions

Type bool

content
to be used internally to carry associated data

exponent
to be used internally to simplify repetitions of a tensor

class matchingtools.core.**Operator**(*tensors*)
Container for a list of tensors with their indices contracted.

Indices repeated in different tensors mean contraction. Contracted indices should be positive. Negative indices represent free indices and should appear in order: -1, -2, ...

The methods include the basic derivation, matching and replacing operations, as well as the implementation of functional derivatives.

tensors
list of the tensors contained

Type [*Tensor*]

variation(*field_name*, *statistics*)
Take functional derivative of the spacetime integral of self:

Parameters

- **field_name** (*string*) – the name of the field with respect to which the functional derivative is taken
- **statistics** (*bool*) – statistics of the field

replace_first(*field_name*, *operator_sum*)
Replace the first ocurrence of a field.

Parameters

- **field_name** (*string*) – name of the field to be replaced
- **operator_sum** (OperatorSum) – replacement

Returns An OperatorSum resulting from replacing the first ocurrence of the field by its replacement

replace_all(*substitutions*, *max_dim*)
Replace all ocurrences of several fields.

Parameters

- **substitutions** (*[(string, OperatorSum)]*) – list of pairs with the first element of the pair being the name of a field to be replaced and the second being the replacement.
- **max_dim** (*int*) – maximum dimension of the operators in the result

Returns An OperatorSum resulting from replacing every ocurrence of the fields by their replacements

match_first(*pattern*)
Match the first ocurrence of a pattern

Parameters **pattern** (Operator) – contains the tensors and index structure to be matched

Returns

if the matching succeeds, an Operator with the first occurrence of the pattern substituted by a "generic" tensor (with a sign change if needed); None otherwise

__eq__(*other*)
Match self with other operator. All tensors and index contractions should match. No sign differences allowed. All free indices should be equal.

class matchingtools.core.**OperatorSum**(*operators=None*)
Container for lists of operators representing their sum.

The methods perform the basic operations defined for operators generalized for sums of them

operators
the operators whose sum is represented

Type [*Operator*]

variation(*field_name*, *statistics*)
Take functional derivative of the spacetime integral of self.

Parameters

- **field_name** (*string*) – the name of the field with respect to which the functional derivative is taken
- **statistics** (*bool*) – statistics of the field

replace_all(*substitutions*, *max_dim*)
Replace all ocurrences of several fields.

Parameters

- **substitutions** (*[(string, OperatorSum)]*) – list of pairs with the first element of the pair being the name of a field to be replaced and the second being the replacement.
- **max_dim** (*int*) – maximum dimension of the operators in the result

Returns An OperatorSum resulting from replacing every ocurrence of the fields by their replacements

class matchingtools.core.**TensorBuilder**(*name*)
Interface for the creation of constant tensors.

name
the name attribute of the tensors to be created

Type string

__call__(**indices*)
Creates a Tensor object with the given indices and the following attributes: `is_field = False`, `num_of_der = 0`, `dimension = 0`, `statistics = boson`

class matchingtools.core.**FieldBuilder**(*name*, *dimension*, *statistics*)
Interface for the creation of fields.

name
identifier of the fields to be created

Type string

dimension
energy dimensions of the fields to be created

Type int

statistics
statistics of the fields to be created

Type bool

__call__(**indices*)
Creates a Tensor object with the given indices and the following attributes: `is_field = True`, `num_of_der = 0`

matchingtools.core.**D**(*index*, *tensor*)
Interface for the creation of tensors with derivatives applied.

matchingtools.core.**D_op**(*index*, **tensors*)
Interface for the creation of operator sums of obtained from de application of derivatives to producs of tensors.

matchingtools.core.**Op**(**tensors*)
Interface for the creation of operators

matchingtools.core.**OpSum**(**operators*)
Interface for the creation of operator sums

matchingtools.core.**number_op**(*number*)
Create an operator correponding to a number.

matchingtools.core.**power_op**(*name*, *exponent*, *indices=None*)
Create an operator corresponding to a tensor exponentiated to some power.

matchingtools.core.**tensor_op**(*name*, *indices=None*)

matchingtools.core.**flavor_tensor_op**(*name*)
Interface for the creation of one-tensor operators with indices

matchingtools.core.**kdelta**
Kronecker delta. To be replaced by the correponding index contraction appearing instead (module transformations).

matchingtools.core.**generic**
Generic tensor to be used for intermediate steps in calculations and in the output of matching.

matchingtools.core.**epsUp**
Totally anti-symmetric tensor with two two-component spinor undotted superindices

matchingtools.core.**epsUpDot**
Totally anti-symmetric tensor with two two-component spinor dotted superindices

matchingtools.core.**epsDown**
Totally anti-symmetric tensor with two two-component spinor undotted subindices

matchingtools.core.**epsDownDot**
Totally anti-symmetric tensor with two two-component spinor dotted subindices

matchingtools.core.**sigma4bar**
Tensor with one lorentz index, one two-component spinor dotted superindex and one two-component spinor undotted superindex. Represents the four-vector of 2x2 matrices built out identity and minus the Pauli matrices.

matchingtools.core.**sigma4**
Tensor with one lorentz index, one two-component spinor undotted subindex and one two-component spinor dotted subindex. Represents the four-vector of 2x2 matrices built out identity and the Pauli matrices.

3.2 The integration module

Module for the integration of heavy fields from a high-energy lagrangian. Contains the function *integrate()* and the classes for representing the different types of heavy fields.

class matchingtools.integration.**RealScalar**(*name*, *num_of_inds*, *has_flavor=True*, *order=2*, *max_dim=4*)
Representation for heavy real scalar bosons.

The has_flavor argument to the constructor is a bool that specifies whether there are several generations of the heavy field, whereas the order argument gives a maximum order in the derivatives for the propagator.

name
name identifier of the corresponding tensor

Type string

num_of_inds
number of indices of the corresponding tensor

Type int

class matchingtools.integration.**ComplexScalar**(*name*, *c_name*, *num_of_inds*, *has_flavor=True*, *order=2*, *max_dim=4*)
Representation for heavy complex scalar bosons.

The has_flavor argument to the constructor is a bool that specifies whether there are several generations of the heavy field, whereas the order argument gives a maximum order in the derivatives for the propagator.

name
name identifier of the corresponding tensor

Type string

c_name
name identifier of the conjugate tensor

Type string

num_of_inds
number of indices of the corresponding tensor

Type int

class matchingtools.integration.**RealVector**(*name*, *num_of_inds*, *has_flavor=True*, *order=2*, *max_dim=4*)
Representation for heavy real vector bosons.

The has_flavor argument to the constructor is a bool that specifies whether there are several generations of the heavy field, whereas the order argument gives a maximum order in the derivatives for the propagator.

name
name identifier of the corresponding tensor

Type string

num_of_inds
number of indices of the corresponding tensor

Type int

class matchingtools.integration.**ComplexVector**(*name*, *c_name*, *num_of_inds*, *has_flavor=True*, *order=2*, *max_dim=4*)

Representation for heavy complex vector bosons.

The has_flavor argument to the constructor is a bool that specifies whether there are several generations of the heavy field, whereas the order argument gives a maximum order in the derivatives for the propagator.

name
name identifier of the corresponding tensor

Type string

c_name
name identifier of the conjugate tensor

Type string

num_of_inds
number of indices of the corresponding tensor

Type int

class matchingtools.integration.**VectorLikeFermion**(*name*, *L_name*, *R_name*, *Lc_name*, *Rc_name*, *num_of_inds*, *has_flavor=True*)

Representation for heavy vector-like fermions

The has_flavor argument to the constructor is a bool that specifies whether there are several generations of the heavy field.

name
name of the field

Type string

L_name
name of the left-handed part

Type string

R_name
name of the right-handed part

Type string

Lc_name
name of the conjugate of the left-handed part

Type string

Rc_name
name of the conjugate of the right-handed part

Type string

num_of_inds
number of indices of the corresponding tensor

Type int

class matchingtools.integration.**MajoranaFermion**(*name*, *c_name*, *num_of_inds*, *has_flavor=True*)

Representation for heavy Majorana fermions.

The `has_flavor` argument to the constructor is a bool that specifies whether there are several generations of the heavy field.

name
name identifier of the corresponding tensor

Type string

c_name
name identifier of the conjugate tensor

Type string

num_of_inds
number of indices of the corresponding tensor

Type int

`matchingtools.integration.`**`integrate`**(*heavy_fields*, *interaction_lagrangian*, *max_dim=6*, *verbose=True*)

Integrate out heavy fields.

Heavy field classes: `RealScalar`, `ComplexScalar`, `RealVector`, `ComplexVector`, `VectorLikeFermion` or `MajoranaFermion`.

Parameters

- **heavy_fields** (*list of heavy fields*) – to be integrated out
- **interaction_lagrangian** (`matchingtools.operators.OperatorSum`) – from which to integrate out the heavy fields
- **max_dim** (*int*) – maximum dimension of the operators in the effective lagrangian
- **verbose** (*bool*) – specifies whether to print messages signaling the start and end of the integration process.

3.3 The `transformations` module

Module dedicated to the transformation and simplification of operators and operator sums. Defines functions for collecting numeric and symbolic factors.

Implements the function *apply_rules()* for the systematic substitution of patterns inside operators in an operator sum until writing the sum it in a given basis of operators; and the function *collect()* for collecting the coefficients of each operator in a given list.

`matchingtools.transformations.`**`collect_numbers`**(*operator*)
Collect all the tensors representing numbers into a single one.

A tensor is understood to represent a number when its name is `"$number"` or `$i`.

`matchingtools.transformations.`**`collect_powers`**(*operator*)
Collect all the tensors that are equal and return the correspondin powers.

`matchingtools.transformations.`**`collect_numbers_and_powers`**(*op_sum*)
Collect the numeric factors and powers of tensors.

`matchingtools.transformations.`**`apply_rule`**(*operator*, *pattern*, *replacement*)
Replace the first occurrence of `pattern` by `replacement` in `operator`

`matchingtools.transformations.`**`apply_rules`**(*op_sum*, *rules*, *max_iterations*, *verbose=True*)
Apply all the given rules to the operator sum.

With the adecuate set of rules this function can be used to express an effective lagrangian in a specific basis of operators

Parameters

- **op_sum** (matchingtools.operator.OperatorSum) – to which the rules should be applied.
- **rules** (list of pairs (matchingtools.operators.Operator, matchingtools.operators.OperatorSum)) – The first element of each pair represents a pattern to be subtituted in each operator by the second element using *apply_rule()*.
- **max_iterations** (*int*) – maximum number of application of rules to each operator.
- **verbose** (*bool*) – specifies whether to print messages signaling the start and end of the integration process

Returns OperatorSum containing the result of the application of rules.

matchingtools.transformations.**collect_by_tensors**(*op_sum*, *tensor_names*)

Collect the coefficients of the given tensors.

Usually, these tensors represent operators of a basis in which the effective lagrangian is expressed.

Parameters

- **op_sum** (OperatorSum) – whose terms are to be collectd
- **tensor_names** (*list of strings*) – names of the tensors whose coefficients will be obtained

Returns A pair (collection, rest) where collection is a list of pairs (name, coef) where name is the name of each of the tensors and coef is an OperatorSum representing its coefficients; and where rest is an OperatorSum with the operators that didn't contain a tensor with one of the given names.

matchingtools.transformations.**collect**(*op_sum*, *tensor_names*, *verbose=True*)

Simplify the numeric and exponentiated symbolic tensors (using *collect_numbers_and_powers()*) and collect the coefficients of the given tensors (using *collect_by_tensors()*).

Parameters

- **op_sum** (OperatorSum) – whose terms are to be collected
- **tensor_names** (*list of strings*) – names of the tensors whose coefficients will be obtained
- **verbose** (*bool*) – specify whether to write messages signaling the start and end of the computation.

3.4 The output module

Module dedicated to the conversion of operator sums to plain text and latex code.

Defines the class *Writer* to represent the coefficients of some given operators in both formats.

class matchingtools.output.**Writer**(*op_sum*, *op_names*, *conjugates=None*, *verbose=True*)

Class to write an operator sum in various formats.

The coefficients of the tensors with the given names are collected and prepared to be written.

collection (list of pairs (string, list of pairs
(complex, Operator)): the first element of each pair in the main list is the name of the tensor having the second part as coefficient. The pairs in the list that is the second element represent a numeric coefficient and an Operator (product of tensors) that multiplied together and summed with the others give the complete coefficient.

rest
represents a sum of the operators that couldn't be collected with their corresponding coefficients.

Type list of pairs (complex, *Operator*)

conjugates (dictionary string
string): name of the complex conjugate corresponding to the name of each tensor. If it is not None, this is used for collecting conjugate pairs to give their real or imaginary parts.

no_parens
names of tensors that, when having a non-unit exponent are to represented in latex as tensor^exp instead of (tensor)^exp

Type list of strings

numeric
names of tensors that symbolically represent numbers. The effect of this is the latex output: they come after the "$number" tensors and before the "$i".

Type list of strings

__init__(*op_sum*, *op_names*, *conjugates=None*, *verbose=True*)

Parameters

- **op_sum** (*OperatorSum*) – to be represented
- **op_names** (*list of strings*) – the names of the tensors (represented as tensors) whose coefficients are to be collected and written

__str__()
Plain text representation

write_text_file(*filename*)

latex_code(*structures*, *op_reps*, *inds*, *no_parens=None*, *numeric=None*)
Representation as LaTeX's amsmath `align` environments

Parameters

- **structures** (*dict*) – the keys are the names of all the tensors. The corresponding values are the LaTeX formula representation, using python's `str.format` notation `"{}"` to specify the popsitions where the indices should appear.
- **structures** – the keys are the names of all the operators in the basis. The corresponding values are the LaTeX formula representation.
- **inds** (*list of strings*) – the symbols to be used to represent the indices, in the order in which they should appear.
- **no_parens** (*function*) – receives the name of a tensor and returns True if its LaTeX representation should not have parenthesis around it when exponentiated to some power.
- **numeric** (*list of strings*) – list of names of tensors that should appear with the numeric coefficients in the representation, before the rest of the tensors.

write_latex(*filename*, *structures*, *op_reps*, *inds*, *no_parens=None*, *numeric=None*)
Write a LaTeX document with the representation.

Parameters

- **filename** (*string*) – the name of the file without the extension `".tex"` in which to write
- **structures** (*dict*) – the keys are the names of all the tensors. The corresponding values are the LaTeX formula representation, using python's `str.format` notation `"{}"` to specify the positions where the indices should appear.
- **structures** – the keys are the names of all the operators in the basis. The corresponding values are the LaTeX formula representation.
- **inds** (*list of strings*) – the symbols to be used to represent the indices, in the order in which they should appear.
- **no_parens** (*function*) – receives the name of a tensor and returns True if its LaTeX representation should not have parenthesis around it when exponentiated to some power.
- **numeric** (*list of strings*) – list of names of tensors that should appear with the numeric coefficients in the representation, before the rest of the tensors.

show_pdf (*filename*, *pdf_viewer*, *structures*, *op_reps*, *inds*)

Directly show the pdf file with the results, obtained using `pdflatex` on the `.tex` file.

Parameters

- **filename** (*string*) – the name of the files without the extension `".tex"` and `.pdf` in which to write.
- **pdfviewer** (*string*) – name of the program (callable from the command-line) to show the pdf.
- **structures** (*dict*) – the keys are the names of all the tensors. The corresponding values are the LaTeX formula representation, using python's `str.format` notation `"{}"` to specify the positions where the indices should appear.
- **structures** – the keys are the names of all the operators in the basis. The corresponding values are the LaTeX formula representation.
- **inds** (*list of strings*) – the symbols to be used to represent the indices, in the order in which they should appear.v

3.5 The `extras` package

The `matchingtools.extras` package provides several modules with some useful definitions of tensors related to the Lorentz group and the groups $SU(3)$ and $SU(2)$, as well as rules for transforming some combinations of these tensors into others.

The definitions for the Standard Model tensors and fields, together with the rules derived from their equations of motion for the substitution of their covariant derivatives are provided. A basis for the Standard Model effective Lagrangian up to dimension 6 is also given.

3.5.1 The `extras.SU2` module

This module defines tensors and rules related to the group $SU(2)$.

`matchingtools.extras.SU2.`**epsSU2**

Totally antisymmetric tensor $\epsilon = i\sigma^2$ with two $SU(2)$ doublet indices such that $\epsilon_{12} = 1$.

matchingtools.extras.SU2.**sigmaSU2**
Pauli matrices $(\sigma^a)_{ij}$.

matchingtools.extras.SU2.**CSU2**
Clebsh-Gordan coefficients $C^I_{a\beta}$ with the first index I being a quadruplet index, the second a a triplet index, and the third β a doublet index.

matchingtools.extras.SU2.**CSU2c**
Conjugate of the Clebsh-Gordan coefficients $C^I_{a\beta}$.

matchingtools.extras.SU2.**epsSU2triplets**
Totally antisymmetric tensor ϵ_{abc} with three $SU(2)$ triplet indices such that $\epsilon_{123} = 1$.

matchingtools.extras.SU2.**epsSU2quadruplets**
Two-index that gives a singlet when contracted with two $SU(2)$ quadruplets.

matchingtools.extras.SU2.**fSU2**
Totally antisymmetric tensor with three $SU(2)$ triplet indices given by $f_{abc} = \frac{i}{\sqrt{2}}\epsilon_{abc}$ with $\epsilon_{123} = 1$.

matchingtools.extras.SU2.**rule_SU2_fierz**
Subtitute $\sigma^a_{ij}\sigma^a_{kl}$ by $2\delta_{il}\delta_{kj} - \delta_{ij}\delta kl$.

matchingtools.extras.SU2.**rule_SU2_product_sigmas**
Subtitute $\sigma^a_{ij}\sigma^a_{jk}$ by $3\delta_{ik}$.

matchingtools.extras.SU2.**rule_SU2_free_eps**
Subtitute $\epsilon_{ij}\epsilon_{kl}$ by $\delta_{ik}\delta_{jl} - \delta_{il}\delta_{jk}$.

matchingtools.extras.SU2.**rules_SU2_eps_cancel**
Substitute contracted ϵ tensors with the corresponding Kronecker delta.

matchingtools.extras.SU2.**rule_SU2_eps_zero**
Substitute ϵ_{ii} by zero because ϵ is antisymmetric.

matchingtools.extras.SU2.**rules_SU2_epsquadruplets_cancel**
Substitute contracted ϵ tensors with the corresponding Kronecker delta.

matchingtools.extras.SU2.**rules_SU2_C_sigma**
Substitute $C^I_{ap}\epsilon_{pm}\sigma^a_{ij}C^{I*}_{bq}\epsilon_{qn}\sigma^b_{kl}$ by the equivalent $-\frac{2}{3}\delta_{mn}\delta_{ij}\delta_{kl} + \frac{4}{3}\delta_{mn}\delta_{il}\delta_{kj} + \frac{2}{3}\delta_{ml}\delta_{in}\delta_{kj} - \frac{2}{3}\delta_{mj}\delta_{il}\delta_{kn}$.

matchingtools.extras.SU2.**rules_SU2**
All the rules defined in *matchingtools.extras.SU2* together

matchingtools.extras.SU2.**latex_SU2**
LaTeX code representation of the $SU(2)$ tensors.

3.5.2 The extras.SU3 module

This module defines tensors related to the group $SU(3)$.

matchingtools.extras.SU3.**epsSU3**
Totally antisymmetric tensor ϵ_{ABC} with three $SU(3)$ triplet indices such that $\epsilon_{123} = 1$.

matchingtools.extras.SU3.**TSU3**
$SU(3)$ generators $(T_A)_{BC}$ (half of the Gell-Mann matrices).

matchingtools.extras.SU3.**fSU3**
$SU(3)$ structure constants f_{ABC}.

3.5.3 The extras.Lorentz module

This module defines rules related to the Lorentz group.

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

matchingtools.extras.Lorentz.**sigmaTensor**
Lorentz tensor $\sigma^{\mu\nu} = \frac{i}{4}\left(\sigma^{\mu}_{\alpha\dot{\gamma}}\bar{\sigma}^{\nu\dot{\gamma}\beta} - \sigma^{\nu}_{\alpha\dot{\gamma}}\bar{\sigma}^{\mu\dot{\gamma}\beta}\right)$.

matchingtools.extras.Lorentz.**rule_Lorentz_free_epsUp**
Substitute $\epsilon^{\alpha\beta}\epsilon^{\dot{\alpha}\dot{\beta}}$ by

$$-\frac{1}{2}\bar{\sigma}^{\mu,\dot{\alpha}\alpha}\bar{\sigma}^{\dot{\beta}\beta}_{\mu}$$

matchingtools.extras.Lorentz.**rule_Lorentz_free_epsDown**
Substitute $\epsilon_{\alpha\beta}\epsilon_{\dot{\alpha}\dot{\beta}}$ by

$$-\frac{1}{2}\bar{\sigma}^{\mu}_{\alpha\dot{\alpha}}\bar{\sigma}_{\mu,\beta\dot{\beta}}$$

matchingtools.extras.Lorentz.**rules_Lorentz_eps_cancel**
Substitute contracted ϵ tensors with undotted indices by the corresponding Kronecker delta.

matchingtools.extras.Lorentz.**rules_Lorentz_epsDot_cancel**
Substitute contracted ϵ tensors with dotted indices by the corresponding Kronecker delta.

matchingtools.extras.Lorentz.**rules_Lorentz**
All the rules defined in *matchingtools.extras.Lorentz* together

matchingtools.extras.Lorentz.**latex_Lorentz**
LaTeX code representation of the Lorentz tensors.

3.5.4 The extras.SM module

This module defines the Standard Model tensors and fields and the rules for substituting their equations of motion.

matchingtools.extras.SM.**gb**
$U(1)$ coupling constant g'

matchingtools.extras.SM.**gw**
$SU(2)$ coupling constant g

matchingtools.extras.SM.**mu2phi**
Higgs quadratic coupling μ^2_ϕ

matchingtools.extras.SM.**lambdaphi**
Higgs quartic coupling λ_ϕ

matchingtools.extras.SM.**ye**
Yukawa coupling for leptons y^e_{ij}

matchingtools.extras.SM.**yec**
Conjugate of the Yukawa coupling for leptons y^{e*}_{ij}

matchingtools.extras.SM.**yd**
Yukawa coupling for down quarks y^d_{ij}

matchingtools.extras.SM.**ydc**
Conjugate of the Yukawa coupling for down quarks y^{d*}_{ij}

matchingtools.extras.SM.**yu**
Yukawa coupling for up quarks y^u_{ij}

matchingtools.extras.SM.**yuc**
Conjugate of the Yukawa coupling for up quarks y^{u*}_{ij}

matchingtools.extras.SM.**V**
CKM matrix

matchingtools.extras.SM.**Vc**
Conjugate of the CKM matrix

matchingtools.extras.SM.**deltaFlavor**
Kronecker delta for flavor indices

matchingtools.extras.SM.**phi**
Higgs doublet

matchingtools.extras.SM.**phic**
Conjugate of the Higgs doublet

matchingtools.extras.SM.**lL**
Lepton left-handed doublet

matchingtools.extras.SM.**lLc**
Conjugate of the lepton left-handed doublet

matchingtools.extras.SM.**qL**
Quark left-handed doublet

matchingtools.extras.SM.**qLc**
Conjugate of the quark left-handed doublet

matchingtools.extras.SM.**eR**
Electron right-handed singlet

matchingtools.extras.SM.**eRc**
Conjugate of the electron right-handed singlet

matchingtools.extras.SM.**dR**
Down quark right-handed doublet

matchingtools.extras.SM.**dRc**
Conjugate of the down quark right-handed doublet

matchingtools.extras.SM.**uR**
Up quark right-handed doublet

matchingtools.extras.SM.**uRc**
Conjugate of the up quark right-handed doublet

matchingtools.extras.SM.**bFS**
$U(1)$ gauge field strength $B_{\mu\nu}$

matchingtools.extras.SM.**wFS**
$SU(2)$ gauge field strength $W^a_{\mu\nu}$

matchingtools.extras.SM.**gFS**
$SU(3)$ gauge field strength $G^A_{\mu\nu}$

matchingtools.extras.SM.**eom_phi**
Rule using the Higgs doublet equation of motion. Substitute $D^2\phi$ by

$$\mu^2_\phi\phi - 2\lambda_\phi(\phi^\dagger\phi)\phi - y^{e*}_{ij}\bar{e}_{Rj}l_{Li} - y^{d*}_{ij}\bar{d}_{Rj}q_{Li} + V^*_{ki}y^u_{kj}i\sigma^2\bar{q}^T_{Li}u_{Rj}$$

matchingtools.extras.SM.**eom_phic**
Rule using the conjugate of the Higgs doublet equation of motion. Substitute $D^2\phi^\dagger$ by

$$\mu_\phi^2\phi^\dagger - 2\lambda_\phi(\phi^\dagger\phi)\phi^\dagger - y_{ij}^e\bar{l}_{Li}e_{Rj} - y_{ij}^d\bar{q}_{Li}d_{Rj} - V_{ki}y_{kj}^{u*}\bar{u}_{Rj}q_{Li}^T i\sigma^2$$

matchingtools.extras.SM.**eom_bFS**
Rule using the $U(1)$ gauge field strength equation of motion. Substitute $D_\mu B^{\mu\nu}$ by

$$-g'\sum_f Y_f\bar{f}\gamma^\nu f - (\frac{i}{2}g'\phi^\dagger D^\nu\phi + h.c.)$$

where $\sum_f$ runs over all SM fermions f and the Y_f are the hypercharges.

matchingtools.extras.SM.**eom_wFS**
Rule using the $SU(2)$ gauge field strength equation of motion. Substitute $D_\mu W^{a,\mu\nu}$ by

$$-\frac{1}{2}g\sum_F \bar{F}\sigma^a\gamma^\nu F - (\frac{i}{2}g\phi^\dagger\sigma^a D^\nu\phi + h.c.)$$

where $\sum_F$ runs over the SM fermion doublets F.

matchingtools.extras.SM.**eom_lL**
Rule using the equation of motion of the left-handed lepton doublet. Substitute $\gamma^\mu D_\mu l_{Li}$ by $-iy_{ij}^e\phi e_{Rj}$.

matchingtools.extras.SM.**eom_lLc**
Rule using the equation of motion of conjugate of the left-handed lepton doublet. Substitute $D_\mu\bar{l}_{Li}\gamma^\mu$ by $iy_{ij}^{e*}\bar{e}_{Rj}\phi^\dagger$.

matchingtools.extras.SM.**eom_qL**
Rule using the equation of motion of the left-handed quark doublet. Substitute $\gamma^\mu D_\mu q_{Li}$ by $-iy_{ij}^d\phi d_{Rj} - iV_{ji}^*y_{kj}^u\tilde{\phi}u_{Rj}$.

matchingtools.extras.SM.**eom_qLc**
Rule using the equation of motion of the conjugate of the left-handed quark doublet. Substitute $D_\mu\bar{q}_{Li}\gamma^\mu$ by $iy_{ij}^{d*}\bar{d}_{Rj}\phi^\dagger + iV_{ji}y_{kj}^{u*}\bar{u}_{Rj}\tilde{\phi}^\dagger$.

matchingtools.extras.SM.**eom_eR**
Rule using the equation of motion of the right-handed electron singlet. Substitute $\gamma^\mu D_\mu e_{Rj}$ by $-iy_{ij}^{e*}\phi^\dagger l_{Li}$.

matchingtools.extras.SM.**eom_eRc**
Rule using the equation of motion of the conjugate of the right-handed electron singlet. Substitute $D_\mu\bar{e}_{Rj}\gamma^\mu$ by $iy_{ij}^e\bar{l}_{Li}\phi$.

matchingtools.extras.SM.**eom_dR**
Rule using the equation of motion of the right-handed down quark singlet. Substitute $\gamma^\mu D_\mu d_{Rj}$ by $-iy_{ij}^{d*}\phi^\dagger q_{Li}$.

matchingtools.extras.SM.**eom_dRc**
Rule using the equation of motion of the conjugate of the right-handed down quark singlet. Substitute $D_\mu\bar{d}_{Rj}\gamma^\mu$ by $iy_{ij}^d\bar{q}_{Li}\phi$.

matchingtools.extras.SM.**eom_uR**
Rule using the equation of motion of the right-handed up quark singlet. Substitute $\gamma^\mu D_\mu u_{Rj}$ by $-iV_{ki}y_{kj}^{u*}\tilde{\phi}^\dagger q_{Li}$.

matchingtools.extras.SM.**eom_uRc**
Rule using the equation of motion of the conjugate of the right-handed up quark singlet. Substitute $D_\mu\bar{u}_{Rj}\gamma^\mu$ by $iV_{ki}^*y_{kj}^u\bar{q}_{Li}\tilde{\phi}$.

matchingtools.extras.SM.**eoms_SM**
Rules that use the equations of motion of the Standard Model fields to substitute some expressions involving their derivatives by other combinations of the fields.

matchingtools.extras.SM.**latex_SM**
LaTeX code representation of the tensors and fields of the Standard Model.

3.5.5 The extras.SM_dim_6_basis module

This module defines a basis of operators for the Standar Model effective lagrangian up to dimension 6.

The basis is the one in arXiv:1412.8480v2.

matchingtools.extras.SM_dim_6_basis.**Okinphi**
Higgs kinetic term $\mathcal{O}_{kin,\phi} = (D_\mu\phi)^\dagger D^\mu\phi$.

matchingtools.extras.SM_dim_6_basis.**Ophi4**
Higgs quartic coupling $\mathcal{O}_{\phi 4} = (\phi^\dagger\phi)^2$.

matchingtools.extras.SM_dim_6_basis.**Ophi2**
Higgs quadratic coupling $\mathcal{O}_{\phi 2} = \phi^\dagger\phi$.

matchingtools.extras.SM_dim_6_basis.**Oye**(**indices*)
Lepton Yukawa coupling $(\mathcal{O}_{y^e})_{ij} = \bar{l}_{Li}\phi e_{Rj}$.

matchingtools.extras.SM_dim_6_basis.**Oyec**(**indices*)
Conjugate lepton Yukawa coupling $(\mathcal{O}_{y^e})^*_{ij} = \bar{e}_{Rj}\phi^\dagger l_{Li}$.

matchingtools.extras.SM_dim_6_basis.**Oyd**(**indices*)
Down quark Yukawa coupling $(\mathcal{O}_{y^d})_{ij} = \bar{q}_{Li}\phi d_{Rj}$.

matchingtools.extras.SM_dim_6_basis.**Oydc**(**indices*)
Conjugate down quark Yukawa coupling $(\mathcal{O}_{y^d})^*_{ij} = \bar{d}_{Rj}\phi^\dagger q_{Li}$.

matchingtools.extras.SM_dim_6_basis.**Oyu**(**indices*)
Up quark Yukawa coupling $(\mathcal{O}_{y^u})_{ij} = \bar{q}_{Li}\tilde{\phi} u_{Rj}$.

matchingtools.extras.SM_dim_6_basis.**Oyuc**(**indices*)
Conjugate up quark Yukawa coupling $(\mathcal{O}_{y^d})^*_{ij} = \bar{d}_{Rj}\tilde{\phi}^\dagger q_{Li}$.

matchingtools.extras.SM_dim_6_basis.**O5**(**indices*)
Weinberg operator $\mathcal{O}_5 = \overline{l^c}_L\tilde{\phi}^*\tilde{\phi}^\dagger l_L$.

matchingtools.extras.SM_dim_6_basis.**O5c**(**indices*)
Conjugate Weinberg operator $\mathcal{O}_5 = \bar{l}_L\tilde{\phi}\tilde{\phi}^T l^c_L$.

matchingtools.extras.SM_dim_6_basis.**O1ll**(**indices*)
LLLL type four-fermion operator $(\mathcal{O}^{(1)}_{ll})_{ijkl} = \frac{1}{2}(\bar{l}_{Li}\gamma_\mu l_{Lj})(\bar{l}_{Lk}\gamma^\mu l_{Ll})$.

matchingtools.extras.SM_dim_6_basis.**O1qq**(**indices*)
LLLL type four-fermion operator $(\mathcal{O}^{(1)}_{qq})_{ijkl} = \frac{1}{2}(\bar{q}_{Li}\gamma_\mu q_{Lj})(\bar{q}_{Lk}\gamma^\mu q_{Ll})$.

matchingtools.extras.SM_dim_6_basis.**O8qq**(**indices*)
LLLL type four-fermion operator $(\mathcal{O}^{(8)}_{qq})_{ijkl} = \frac{1}{2}(\bar{q}_{Li}T_A\gamma_\mu q_{Lj})(\bar{q}_{Lk}T_A\gamma^\mu q_{Ll})$.

matchingtools.extras.SM_dim_6_basis.**O1lq**(**indices*)
LLLL type four-fermion operator $(\mathcal{O}^{(1)}_{lq})_{ijkl} = (\bar{l}_{Li}\gamma_\mu l_{Lj})(\bar{q}_{Lk}\gamma^\mu q_{Ll})$.

matchingtools.extras.SM_dim_6_basis.**O3lq**(**indices*)
LLLL type four-fermion operator $(\mathcal{O}^{(8)}_{qq})_{ijkl} = (\bar{l}_{Li}\sigma_a\gamma_\mu l_{Lj})(\bar{q}_{Lk}\sigma_a\gamma^\mu q_{Ll})$.

matchingtools.extras.SM_dim_6_basis.**Oee**(**indices*)
RRRR type four-fermion operator $(\mathcal{O}_{ee})_{ijkl} = \frac{1}{2}(\bar{e}_{Ri}\gamma_\mu e_{Rj})(\bar{e}_{Rk}\gamma^\mu e_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O1uu**(**indices*)
RRRR type four-fermion operator $(\mathcal{O}^{(1)}_{uu})_{ijkl} = \frac{1}{2}(\bar{u}_{Ri}\gamma_\mu u_{Rj})(\bar{u}_{Rk}\gamma^\mu u_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O1dd**(**indices*)
RRRR type four-fermion operator $(\mathcal{O}^{(1)}_{dd})_{ijkl} = \frac{1}{2}(\bar{d}_{Ri}\gamma_\mu d_{Rj})(\bar{d}_{Rk}\gamma^\mu d_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O1ud**(**indices*)
RRRR type four-fermion operator $(\mathcal{O}^{(1)}_{uu})_{ijkl} = (\bar{u}_{Ri}\gamma_\mu u_{Rj})(\bar{d}_{Rk}\gamma^\mu d_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O8ud**(**indices*)
RRRR type four-fermion operator $(\mathcal{O}^{(8)}_{uu})_{ijkl} = (\bar{u}_{Ri}T_A\gamma_\mu u_{Rj})(\bar{d}_{Rk}T_A\gamma^\mu d_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**Oeu**(**indices*)
RRRR type four-fermion operator $(\mathcal{O}_{eu})_{ijkl} = (\bar{e}_{Ri}\gamma_\mu e_{Rj})(\bar{u}_{Rk}\gamma^\mu u_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**Oed**(**indices*)
RRRR type four-fermion operator $(\mathcal{O}_{ed})_{ijkl} = (\bar{e}_{Ri}\gamma_\mu e_{Rj})(\bar{d}_{Rk}\gamma^\mu d_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**Ole**(**indices*)
LLRR type four-fermion operator $(\mathcal{O}_{le})_{ijkl} = (\bar{l}_{Li}\gamma_\mu l_{Lj})(\bar{e}_{Rk}\gamma^\mu e_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**Oqe**(**indices*)
LLRR type four-fermion operator $(\mathcal{O}_{qe})_{ijkl} = (\bar{q}_{Li}\gamma_\mu q_{Lj})(\bar{e}_{Rk}\gamma^\mu e_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O1u**(**indices*)
LLRR type four-fermion operator $(\mathcal{O}_{lu})_{ijkl} = (\bar{l}_{Li}\gamma_\mu l_{Lj})(\bar{u}_{Rk}\gamma^\mu u_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O1d**(**indices*)
LLRR type four-fermion operator $(\mathcal{O}_{ld})_{ijkl} = (\bar{l}_{Li}\gamma_\mu l_{Lj})(\bar{d}_{Rk}\gamma^\mu d_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O1qu**(**indices*)
LLRR type four-fermion operator $(\mathcal{O}^{(1)}_{qu})_{ijkl} = (\bar{q}_{Li}\gamma_\mu q_{Lj})(\bar{u}_{Rk}\gamma^\mu u_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O8qu**(**indices*)
LLRR type four-fermion operator $(\mathcal{O}^{(8)}_{qu})_{ijkl} = (\bar{q}_{Li}T_A\gamma_\mu q_{Lj})(\bar{u}_{Rk}T_A\gamma^\mu u_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O1qd**(**indices*)
LLRR type four-fermion operator $(\mathcal{O}^{(1)}_{qd})_{ijkl} = (\bar{q}_{Li}\gamma_\mu q_{Lj})(\bar{d}_{Rk}\gamma^\mu d_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**O8qd**(**indices*)
LLRR type four-fermion operator $(\mathcal{O}^{(8)}_{qd})_{ijkl} = (\bar{q}_{Li}T_A\gamma_\mu q_{Lj})(\bar{d}_{Rk}T_A\gamma^\mu d_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**Oledq**(**indices*)
LRRL type four-fermion operator $(\mathcal{O}_{leqd})_{ijkl} = (\bar{l}_{Li}e_{Rj})(\bar{d}_{Rk}q_{Ll})$.

matchingtools.extras.SM_dim_6_basis.**Oledqc**(**indices*)
LRRL type four-fermion operator $(\mathcal{O}_{leqd})^*_{ijkl} = (\bar{e}_{Rj}l_{Li})(\bar{q}_{Ll}d_{Rk})$.

matchingtools.extras.SM_dim_6_basis.**O1qud**(**indices*)
LRLR type four-fermion operator $(\mathcal{O}^{(1)}_{qud})_{ijkl} = (\bar{q}_{Li}u_{Rj})i\sigma_2(\bar{q}_{Lk}d_{Rl})^T$.

matchingtools.extras.SM_dim_6_basis.**O1qudc**(**indices*)
LRLR type four-fermion operator $(\mathcal{O}^{(1)}_{qud})^*_{ijkl} = (\bar{u}_{Rj}q_{Li})i\sigma_2(\bar{d}_{Rl}q_{Lk})^T$.

matchingtools.extras.SM_dim_6_basis.**O8qud**(**indices*)
LRLR type four-fermion operator $(\mathcal{O}^{(8)}_{qud})_{ijkl} = (\bar{q}_{Li}T_Au_{Rj})i\sigma_2(\bar{q}_{Lk}T_Ad_{Rl})^T$.

matchingtools.extras.SM_dim_6_basis.**O8qudc**(**indices*)
LRLR type four-fermion operator $(\mathcal{O}^{(8)}_{qud})^*_{ijkl} = (\bar{u}_{Rj}T_Aq_{Li})i\sigma_2(\bar{d}_{Rl}T_Aq_{Lk})^T$.

matchingtools.extras.SM_dim_6_basis.**Olequ**(**indices*)
LRLR type four-fermion operator $(\mathcal{O}_{lequ})_{ijkl} = (\bar{l}_{Li}e_{Rj})i\sigma_2(\bar{q}_{Lk}u_{Rl})^T$.

matchingtools.extras.SM_dim_6_basis.**Olequc**(**indices*)
LRLR type four-fermion operator $(\mathcal{O}_{lequ})^*_{ijkl} = (\bar{e}_{Rj}l_{Li})i\sigma_2(\bar{u}_{Rl}q_{Lk})^T$.

matchingtools.extras.SM_dim_6_basis.**Oluqe**(**indices*)
LRLR type four-fermion operator $(\mathcal{O}_{luqe})_{ijkl} = (\bar{l}_{Li}u_{Rj})i\sigma_2(\bar{q}_{Lk}e_{Rl})^T$.

matchingtools.extras.SM_dim_6_basis.**Oluqec**(**indices*)
LRLR type four-fermion operator $(\mathcal{O}_{luqe})^*_{ijkl} = (\bar{l}_{Li}u_{Rj})i\sigma_2(\bar{q}_{Lk}e_{Rl})^T$.

matchingtools.extras.SM_dim_6_basis.**Olqdu**(**indices*)
Four-fermion operator $(\mathcal{O}_{lqdu})_{ijkl} = \epsilon_{ABC}(\bar{l}_{Li}i\sigma_2 q^{c,A}_{Lj})(\bar{d}^B_{Rk}u^{c,C}_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**Olqduc**(**indices*)
Four-fermion operator $(\mathcal{O}_{lqdu})^*_{ijkl} = -\epsilon_{ABC}(\bar{q}^{c,A}_{Lj}i\sigma_2 l_{Li})(\bar{u}^{c,C}_{Rl}d^B_{Rk})$.

matchingtools.extras.SM_dim_6_basis.**Oqqeu**(**indices*)
Four-fermion operator $(\mathcal{O}_{lqdu})_{ijkl} = \epsilon_{ABC}(\bar{q}^A_{Li}i\sigma_2 q^{c,B}_{Lj})(\bar{e}_{Rk}u^{c,C}_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**Oqqeuc**(**indices*)
Four-fermion operator $(\mathcal{O}_{lqdu})^*_{ijkl} = -\epsilon_{ABC}(\bar{q}^{c,B}_{Lj}i\sigma_2 q^A_{Li})(\bar{u}^{c,C}_{Rl}e_{Rk})$.

matchingtools.extras.SM_dim_6_basis.**O1lqqq**(**indices*)
Four-fermion operator $(\mathcal{O}^{(1)}_{lqqq})_{ijkl} = \epsilon_{ABC}(\bar{l}_{Li}i\sigma_2 q^{c,A}_{Lj})(\bar{q}^B_{Lk}i\sigma_2 q^{c,C}_{Ll})$.

matchingtools.extras.SM_dim_6_basis.**O1lqqqc**(**indices*)
Four-fermion operator $(\mathcal{O}^{(1)}_{lqqq})^*_{ijkl} = \epsilon_{ABC}(\bar{q}^{c,A}_{Lj}i\sigma_2 l_i)(\bar{q}^{c,C}_{Ll}i\sigma_2 q^B_{Lk})$.

matchingtools.extras.SM_dim_6_basis.**Oudeu**(**indices*)
Four-fermion operator $(\mathcal{O}_{udeu})_{ijkl} = \epsilon_{ABC}(\bar{u}^A_{Ri}d^{c,B}_{Rj})(\bar{e}_{Rk}u^{c,C}_{Rl})$.

matchingtools.extras.SM_dim_6_basis.**Oudeuc**(**indices*)
Four-fermion operator $(\mathcal{O}_{udeu})^*_{ijkl} = \epsilon_{ABC}(\bar{d}^{c,B}_{Rj}u^A_{Ri})(\bar{u}^{c,C}_{Rl}e_{Rk})$.

matchingtools.extras.SM_dim_6_basis.**O3lqqq**(**indices*)
Four-fermion operator $(\mathcal{O}^{(3)}_{lqqq})_{ijkl} = \epsilon_{ABC}(\bar{l}_{Li}\sigma_a i\sigma_2 q^{c,A}_{Lj})(\bar{q}^B_{Lk}\sigma_a i\sigma_2 q^{c,C}_{Ll})$.

matchingtools.extras.SM_dim_6_basis.**O3lqqqc**(**indices*)
Four-fermion operator $(\mathcal{O}^{(3)}_{lqqq})^*_{ijkl} = \epsilon_{ABC}(\bar{q}^{c,A}_{Lj}isigma_2\sigma_a l_{Li})(\bar{q}^{c,C}_{Ll}i\sigma_2\sigma_a q^B_{Lk})$.

matchingtools.extras.SM_dim_6_basis.**Ophisq**
S type operator $\mathcal{O}_{\phi\Box} = \phi^\dagger\phi\Box(\phi^\dagger\phi)$.

matchingtools.extras.SM_dim_6_basis.**Ophi**
S type six Higgs interaction operator $\mathcal{O}_\phi = \frac{1}{3}(\phi^\dagger\phi)^3$.

matchingtools.extras.SM_dim_6_basis.**O1phil**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi l})_{ij} = (\phi^\dagger iD_\mu\phi)(\bar{l}_{Li}\gamma^\mu l_{Lj})$.

matchingtools.extras.SM_dim_6_basis.**O1philc**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi l})^*_{ij} = (-i(D_\mu\phi)^\dagger\phi)(\bar{l}_{Lj}\gamma^\mu l_{Li})$.

matchingtools.extras.SM_dim_6_basis.**O1phiq**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi q})_{ij} = (\phi^\dagger iD_\mu\phi)(\bar{q}_{Li}\gamma^\mu q_{Lj})$.

matchingtools.extras.SM_dim_6_basis.**O1phiqc**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi q})^*_{ij} = (-i(D_\mu\phi)^\dagger\phi)(\bar{q}_{Lj}\gamma^\mu q_{Li})$.

matchingtools.extras.SM_dim_6_basis.**O3phil**(**indices*)
SVF type operator $(\mathcal{O}^{(3)}_{\phi l})_{ij} = (\phi^\dagger iD_\mu\phi)(\bar{l}_{Li}\gamma^\mu l_{Lj})$.

matchingtools.extras.SM_dim_6_basis.**O3philc**(**indices*)
SVF type operator $(\mathcal{O}^{(3)}_{\phi l})^*_{ij} = (-i(D_\mu\phi)^\dagger\phi)(\bar{l}_{Lj}\gamma^\mu l_{Li})$.

matchingtools.extras.SM_dim_6_basis.**O3phiq**(**indices*)
SVF type operator $(\mathcal{O}^{(3)}_{\phi q})_{ij} = (\phi^\dagger iD_\mu\phi)(\bar{q}_{Li}\gamma^\mu q_{Lj})$.

matchingtools.extras.SM_dim_6_basis.**O3phiqc**(**indices*)
SVF type operator $(\mathcal{O}^{(3)}_{\phi q})^*_{ij} = (-i(D_\mu\phi)^\dagger\phi)(\bar{q}_{Lj}\gamma^\mu q_{Li})$.

matchingtools.extras.SM_dim_6_basis.**O1phie**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi e})_{ij} = (\phi^\dagger iD_\mu\phi)(\bar{e}_{Ri}\gamma^\mu e_{Rj})$.

matchingtools.extras.SM_dim_6_basis.**O1phiec**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi e})^*_{ij} = (-i(D_\mu\phi)^\dagger\phi)(\bar{e}_{Rj}\gamma^\mu e_{Ri})$.

matchingtools.extras.SM_dim_6_basis.**O1phid**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi d})_{ij} = (\phi^\dagger iD_\mu\phi)(\bar{d}_{Ri}\gamma^\mu d_{Rj})$.

matchingtools.extras.SM_dim_6_basis.**O1phidc**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi d})^*_{ij} = (-i(D_\mu\phi)^\dagger\phi)(\bar{d}_{Rj}\gamma^\mu d_{Ri})$.

matchingtools.extras.SM_dim_6_basis.**O1phiu**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi u})_{ij} = (\phi^\dagger iD_\mu\phi)(\bar{u}_{Ri}\gamma^\mu u_{Rj})$.

matchingtools.extras.SM_dim_6_basis.**O1phiuc**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi u})^*_{ij} = (-i(D_\mu\phi)^\dagger\phi)(\bar{u}_{Rj}\gamma^\mu u_{Ri})$.

matchingtools.extras.SM_dim_6_basis.**Ophiud**(**indices*)
SVF type operator $(\mathcal{O}^{(1)}_{\phi ud})_{ij} = -(\tilde{\phi}^\dagger iD_\mu\phi)(\bar{u}_{Ri}\gamma^\mu d_{Rj})$.

matchingtools.extras.SM_dim_6_basis.**Ophiudc**(**indices*)
SVF type operator $(\mathcal{O}_{\phi ud})^*_{ij} = (i(D_\mu\phi)^\dagger\tilde{\phi})(\bar{u}_{Rj}\gamma^\mu d_{Ri})$.

matchingtools.extras.SM_dim_6_basis.**OeB**(**indices*)
STF type operator $(\mathcal{O}_{eB})_{ij} = (\bar{l}_{Li}\sigma^{\mu\nu}e_{Rj})\phi B_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OeBc**(**indices*)
STF type operator $(\mathcal{O}_{eB})^*_{ij} = \phi^\dagger(\bar{e}_{Rj}\sigma^{\mu\nu}l_{Li})B_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OeW**(**indices*)
STF type operator $(\mathcal{O}_{eW})_{ij} = (\bar{l}_{Li}\sigma^{\mu\nu}e_{Rj})\sigma^a\phi W^a_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OeWc**(**indices*)
STF type operator $(\mathcal{O}_{eW})^*_{ij} = \phi^\dagger\sigma^a(\bar{e}_{Rj}\sigma^{\mu\nu}l_{Li})W^a_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OuB**(**indices*)
STF type operator $(\mathcal{O}_{uB})_{ij} = (\bar{q}_{Li}\sigma^{\mu\nu}u_{Rj})\tilde{\phi}B_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OuBc**(**indices*)
STF type operator $(\mathcal{O}_{uB})^*_{ij} = \tilde{\phi}^\dagger(\bar{u}_{Rj}\sigma^{\mu\nu}q_{Li})B_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OuW**(**indices*)
STF type operator $(\mathcal{O}_{uW})_{ij} = (\bar{q}_{Li}\sigma^{\mu\nu}u_{Rj})\sigma^a\tilde{\phi}W^a_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OuWc**(**indices*)
STF type operator $(\mathcal{O}_{uW})^*_{ij} = \tilde{\phi}\sigma^a(\bar{u}_{Rj}\sigma^{\mu\nu}q_{Li})W^a_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OdB**(**indices*)
STF type operator $(\mathcal{O}_{dB})_{ij} = (\bar{q}_{Li}\sigma^{\mu\nu}d_{Rj})\phi B_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OdBc**(**indices*)
STF type operator $(\mathcal{O}_{dB})^*_{ij} = \phi^\dagger(\bar{d}_{Rj}\sigma^{\mu\nu}q_{Li})B_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OdW**(**indices*)
STF type operator $(\mathcal{O}_{dW})_{ij} = (\bar{q}_{Li}\sigma^{\mu\nu}d_{Rj})\sigma^a\phi W^a_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OdWc**(**indices*)
STF type operator $(\mathcal{O}_{dW})^*_{ij} = \phi^\dagger\sigma^a(\bar{d}_{Rj}\sigma^{\mu\nu}q_{Li})W^a_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OuG**(**indices*)
STF type operator $(\mathcal{O}_{uG})_{ij} = (\bar{q}_{Li}\sigma^{\mu\nu}T_A u_{Rj})\tilde{\phi}G^A_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OuGc**(**indices*)
STF type operator $(\mathcal{O}_{uG})^*_{ij} = \tilde{\phi}^\dagger(\bar{u}_{Rj}\sigma^{\mu\nu}T_A q_{Li})G^A_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OdG**(**indices*)
STF type operator $(\mathcal{O}_{dG})_{ij} = (\bar{q}_{Li}\sigma^{\mu\nu}T_A d_{Rj})\phi G^A_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OdGc**(**indices*)
STF type operator $(\mathcal{O}_{dG})^*_{ij} = \phi^\dagger(\bar{d}_{Rj}\sigma^{\mu\nu}T_A q_{Li})G^A_{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**Oephi**(**indices*)
SF type operator $(\mathcal{O}_{e\phi})_{ij} = (\phi^\dagger\phi)(\bar{l}_{Li}\phi e_{Rj})$.

matchingtools.extras.SM_dim_6_basis.**Odphi**(**indices*)
SF type operator $(\mathcal{O}_{d\phi})_{ij} = (\phi^\dagger\phi)(\bar{q}_{Li}\phi d_{Rj})$.

matchingtools.extras.SM_dim_6_basis.**Ouphi**(**indices*)
SF type operator $(\mathcal{O}_{u\phi})_{ij} = (\phi^\dagger\phi)(\bar{q}_{Li}\tilde{\phi}u_{Rj})$.

matchingtools.extras.SM_dim_6_basis.**Oephic**(**indices*)
SF type operator $(\mathcal{O}_{e\phi})^*_{ij} = (\phi^\dagger\phi)(\bar{e}_{Rj}\phi^\dagger l_{Li})$.

matchingtools.extras.SM_dim_6_basis.**Odphic**(**indices*)
SF type operator $(\mathcal{O}_{d\phi})^*_{ij} = (\phi^\dagger\phi)(\bar{d}_{Rj}\phi^\dagger q_{Li})$.

matchingtools.extras.SM_dim_6_basis.**Ouphic**(**indices*)
SF type operator $(\mathcal{O}_{u\phi})^*_{ij} = (\phi^\dagger\phi)(\bar{u}_{Rj}\tilde{\phi}^\dagger q_{Li})$.

matchingtools.extras.SM_dim_6_basis.**OphiD**
Oblique operator $\mathcal{O}_{\phi D} = (\phi^\dagger D_\mu\phi)((D^\mu\phi)^\dagger\phi)$.

matchingtools.extras.SM_dim_6_basis.**OphiB**
Oblique operator $\mathcal{O}_{\phi B} = \phi^\dagger\phi B_{\mu\nu}B^{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OphiBTilde**
Oblique operator $\mathcal{O}_{\phi\tilde{B}} = \phi^\dagger\phi\tilde{B}_{\mu\nu}B^{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OphiW**
Oblique operator $\mathcal{O}_{\phi W} = \phi^\dagger\phi W^a_{\mu\nu}W^{a,\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OphiWTilde**
Oblique operator $\mathcal{O}_{\phi\tilde{W}} = \phi^\dagger\phi\tilde{W}^a_{\mu\nu}W^{a,\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OWB**
Oblique operator $\mathcal{O}_{WB} = \phi^\dagger\sigma^a\phi W^a_{\mu\nu}B^{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OWBTilde**
Oblique operator $\mathcal{O}_{\tilde{W}B} = \phi^\dagger\sigma^a\phi\tilde{W}^a_{\mu\nu}B^{\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OphiG**
Oblique operator $\mathcal{O}_{\phi W} = \phi^\dagger\phi G^A_{\mu\nu}G^{A,\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OphiGTilde**
Oblique operator $\mathcal{O}_{\phi\tilde{W}} = \phi^\dagger\phi\tilde{G}^A_{\mu\nu}G^{A,\mu\nu}$.

matchingtools.extras.SM_dim_6_basis.**OW**
Gauge type operator $\mathcal{O}_W = \varepsilon_{abc}W^{a,\nu}_\mu W^{b,\rho}_\nu W^{c,\mu}_\rho$.

matchingtools.extras.SM_dim_6_basis.**OWTilde**
Gauge type operator $\mathcal{O}_{\tilde{W}} = \varepsilon_{abc}\tilde{W}^{a,\nu}_\mu W^{b,\rho}_\nu W^{c,\mu}_\rho$.

matchingtools.extras.SM_dim_6_basis.**OG**
Gauge type operator $\mathcal{O}_G = f_{ABC} G_\mu^{A,\nu} G_\nu^{B,\rho} G_\rho^{C,\mu}$.

matchingtools.extras.SM_dim_6_basis.**OGTilde**
Gauge type operator $\mathcal{O}_{\tilde{G}} = f_{ABC} \tilde{G}_\mu^{A,\nu} G_\nu^{B,\rho} G_\rho^{C,\mu}$.

matchingtools.extras.SM_dim_6_basis.**rules_basis_definitions**
Rules defining the operators in the basis in terms of Standard Model fields.

matchingtools.extras.SM_dim_6_basis.**latex_basis_coefs**
LaTeX representation of the coefficients of the operators in the basis.

Python Module Index

m

Index

Symbols

A

B

C

D

E

F

G

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y