PyR@TE - Manual

Draft

current version: 1.0.3

F. Lyonnet[1 a]

[a] *Laboratoire de Physique Subatomique et de Cosmologie, UJF Grenoble 1, CNRS/IN2P3, INPG, 53 Avenue des Martyrs, F-38026 Grenoble, France*

Abstract

Although the two-loop renormalization group equations for a general gauge field theory have been known for quite some time, deriving them for specific models has often been difficult in practice. This is mainly due to the fact that, albeit straightforward, the involved calculations are quite long, tedious and prone to error. The present manual describes the collection of `Python` routines that we dubbed PyR@TE which is an acronym for "Python Renormalization group equations At Two-loop for Everyone". In PyR@TE, once the user specifies the gauge group and the particle content of the model, the routines automatically generate the full two-loop renormalization group equations for all (dimensionless and dimensionful) parameters. The results can optionally be exported to LaTeX and `Mathematica`, or stored in a `Python` data structure for further processing by other programs. For ease of use, we have implemented an interactive mode for PyR@TE in form of an `IPython Notebook`. The different options and possibilities of PyR@TE are explained at length in this manual which is then complementary to [1].

[1] florian.lyonnet@lpsc.in2p3.fr

1. Introduction

PyR@TE aims at generating the two-loop renormalization group equations (RGEs) for a general gauge theory, given in terms of its gauge groups, particle content and scalar potential. In developing PyR@TE, all known typos in the original series of papers by Machacek and Vaughn have been taken into account[2], and the code has been validated against several known results in the literature (see [1] for the details). Also, independently of the `Python` program, `Mathematica` routines [4] have been developed and cross-checked against the PyR@TE, so that we feel confident to have eliminated most sources of possible errors that might affect the correctness of the RGEs.

This manual describes the structure of the code as well as the different functionalities of PyR@TE, from very simple task to the more elaborate ones. Special attention is dedicated to the interactive mode (based on IPython notebook) since it was not really covered in the publication [1].

This manual is organized as follows. We explain in Section 2 the installation of PyR@TE and walked the user through the basic steps to run PyR@TE in Section 3. Then we present in turn all the settings and options that allow the user to control PyR@TE, Section 4, to implement his own model, Section 5, to generate the different output, Section 7, to solve the RGEs produced by PyR@TE numerically using either `Mathematica` or `Python`. Finally, we detail the interactive mode Section 9 draw the attention of the user on different pittfalls and give our conclusions.

2. Download and Installation

PyR@TE is free software under the copyleft of the GNU General Public License and can be downloaded from the following web page:

http://pyrate.hepforge.org

To install PyR@TE, simply open a shell and type:

```
1 cd $HOME
2 wget http://pyrate.hepforge.org/downloads/pyrate-1.0.0.tar.gz
3 tar xfvz pyrate-1.0.0.tar.gz
4 cd pyrate-1.0.0/
```

For definiteness, we will assume here and in the following that you want to install PyR@TE in your home directory (cf. line 1 in the listing above). Otherwise, simply replace "`$HOME`" by a directory of your choice. At the time of writing the present article, PyR@TE is available in the version 1.0.0, and later you may need to replace this by a more recent version number[3] (cf. line 2). Unpacking the tar ball (line 3) will then create a subdirectory that contains PyR@TE. We will describe how to run the program in **??**.

PyR@TE has the following minimal software requirements:

- `Python` $\geq$ 2.7.1[4] [5]
- `NumPy` $\geq$ 1.7.1 [6] and `SciPy` 0.12.0 [7]

[2] See Ref. [2] and the appendix of Ref. [3].

[3] All versions of PyR@TE will be available in the "Downloads" section of our web page [1].

[4] PyR@TE was developed with `Python` 2.7.1 but should work with more recent versions with the exception of `Python` 3 for which it has not been tested.

- SymPy ≥ 0.7.2 [8]
- IPython ≥ 0.12 [9]
- PyYAML ≥ 3.10 [10]

Most of these packages ship with any standard Linux distribution and are by default pre-installed on your system, but in case they are not, you can easily install them. All but one are available in the standard repositories and can be installed by the respective package manager of your system, e.g. "`yum install SymPy`" for a Fedora-based distribution and "`apt-get install python-sympy`" for a Debian-based one. For PyYAML, you have to visit its web page [10] and follow the installation instructions.

If SymPy 0.7.2 is not available for your system in the repositories (or not in the correct version[5]), you can easily install it by downloading the source code from its web page [8]:

```
1 wget https://github.com/sympy/sympy/archive/sympy-0.7.2.tar.gz
2 tar xfvz sympy-0.7.2
3 mv sympy-sympy-0.7.2/sympy $HOME/pyrate-1.0.0/
```

After unpacking the tarball (line 2), move the subdirectory "SymPy" to where PyR@TE is installed (line 3). In the next section, we will explain in detail how to run PyR@TE.

3. A Quick Start

We will first describe how to run PyR@TE from the command line and later explain in some detail the interactive mode in Section 9. Throughout this section, we will use the SM to illustrate how to use PyR@TE, since it is the theory people are most familiar with. Also, for the SM the output of PyR@TE can easily be compared to the literature.

3.1. First Steps

To run PyR@TE, open a shell, change to the directory where it is installed and enter:

```
python pyR@TE.py -m models/SM.model
```

The option "`-m`" (or "`--Model`") is used to read in a model file, in this case the SM. For now, we defer the discussion of how to create a model file to Section 5 and proceed directly with the calculation of the RGEs. Because the calculations can be quite time-consuming, PyR@TE does not calculate them by default. Rather, the user has complete freedom over the parts of the calculation he needs. For instance, to calculate the RGEs for the gauge, Yukawa or quartic couplings, one would add the options "`--Gauge-Couplings`", "`--Yukawas`", "`--Quartic-Couplings`", respectively, or alternatively "`-gc`", "`-yu`" or "`-qc`":

```
python pyR@TE.py -m ./models/SM.model -gc -yu -qc
```

After PyR@TE terminates and the shell prompt reappears, the results of the calculation will be available in the newly created subdirectory "`$HOME/pyrate-1.0.0/results`". Specifically, the LaTeX file "`RGEsOutput.tex`" contains the RGEs and a summary of the settings and of the model for which the calculation was done. We will discuss other forms of output later in Section 7. Before we go into those details, we would first like to give an exhaustive list of the options used to control PyR@TE.

[5] If SymPy 0.7.3 is available on your system, you can patch it so that it works with PyR@TE. You can find detailed instructions on how to do this on our web page [1].

4. Settings and options

PyR@TE and the type of output it generates are controlled by various options that we have summarized in Tab. 1. Alternatively, one can also obtain the complete list of options by typing

```
python pyR@TE.py --help
```

at the shell prompt. Most options are self-explanatory, and we will therefore not go into any details at this point. In later sections, we will illustrate their use by providing examples.

As the number of options increases, it is more convenient to save all settings in a file which can then be passed to PyR@TE instead of appending a long string of options[6]:

```
python pyR@TE.py -f SMsets.settings
```

The input file "`SMsets.settings`" is written in YAML [11] which is a human-readable format for storing information that can also be easily accessed by a computer. The lines in this file have the following generic structure:

```
keyword: value
```

Here, "`keyword`" is a keyword predefined in PyR@TE, and "`value`" is either a path, a filename or a Boolean, i.e. "`True`" or "`False`". For example, a typical "`SMsets.settings`" file could look like this:

Listing 1: SMsets.settings

```
1  # YAML 1.1
2  ---
3  Model: ./models/SM.model
4  Gauge-Couplings: True
5  Quartic-Couplings: True
6  Yukawas: True
7  ScalarMass: False
8  Two-Loop: False
9  verbose: True
```

Note that (i) strings need not be delimited by quotes, (ii) you can only use spaces as whitespaces, i.e. tabulators are not allowed, and (iii) the space after ":" is mandatory. For the keys that can be used in the settings file, we refer the reader again to Tab. 1.

5. Implementing your own model

The previous sections described how to run PyR@TE to calculate the RGEs for a given model. In this section we will explain how to create your own model file that you can use with PyR@TE. As before, we will use the SM as an epitome to explain the format of the model file. In Appendix B, we give several examples of model files for various extensions of the SM. These examples and many more are also available in the "`models`" subdirectory that ships with PyR@TE.

The three ingredients needed to define a model file are the gauge group, the particle content and the scalar potential. The general form of the model file is similar to that of the settings file already described in Section 4. Consider the following model file given in 2. The first line indicates that this is a YAML file. Lines 3-5 indicate the author of the model, the filename and the date when it was created.

[6] Note that we provide no default settings file and that you have to create your own one e.g. by copying the lines given in 1.

Table 1: List of all options that can be used to control PyR@TE. Note that the last two are only available in versions 1.1.x

Option	Keyword | Default	Description
`--Settings/-f`	`-` | `-`	Specify the name of a *.settings* file.
`--Model/-m`	`Model` | `-`	Specify the name of a . *model* file.
`--verbose/-v`	`verbose` | `False`	Set verbose mode.
`--VerboseLevel/-vL`	`VerboseLevel` | `Critical`	Set the verbose level: *Info, Debug, Critical*
`--Gauge-Couplings/-gc`	`Gauge-Couplings` | `False`	Calculate the gauge couplings RGEs.
`--Quartic-Couplings/-qc`	`Quartic-Couplings` | `False`	Calculate the quartic couplings RGEs.
`--Yukawas/-yu`	`Yukawas` | `False`	Calculate the Yukawa RGEs.
`--ScalarMass/-sm`	`ScalarMass` | `False`	Calculate the scalar mass RGEs.
`--FermionMass/-fm`	`FermionMass` | `False`	Calculate the fermion mass RGEs.
`--Trilinear/-tr`	`Trilinear` | `False`	Calculate the trilinear term RGEs.
`--All-Contributions/-a`	`all-Contributions` | `False`	Calculate all the RGEs.
`--Two-Loop/-tl`	`Two-Loop` | `False`	Calculate at two-loop order.
`--Weyl/-w`	`Weyl` | `True`	The particles are Weyl spinors.
`--LogFile/-lg`	`LogFile` | `True`	Produce a log file.
`--LogLevel/-lv`	`LogLevel` | `Info`	Set the log level: *Info, Debug, Critical*
`--LatexFile/-tex`	`LatexFile` | `RGEsOutput.tex`	Set the name of the LaTeX output file.
`--LatexOutput/-texOut`	`LatexOutput` | `True`	Produce a LaTeX output file.
`--Results/-res`	`Results` | `./results`	Set the directory of the results
`--Pickle/-pkl`	`Pickle` | `False`	Produce a pickle output file.
`--PickleFile/-pf`	`PickleFile` | `RGEsOutput.pickle`	Set the name of the pickle output file.
`--TotxtMathematica/-tm`	`ToM` | `False`	Produce an output to Mathematica.
`--TotxtMathFile/-tmf`	`ToMF` | `RGEsOutput.txt`	Set the name of the Mathematica output file.
`--Export/-e`	`Export` | `False`	Produce the numerical output.
`--Export-File/-ef`	`ExportFile` | `BetaFunction.py`	File in which the beta functions are written.
`--Skip/-sk`	`Skip` | ”	Set the various terms that can be neglected during the calculation.
`--Only/-onl`	`Only` | {}	Set a dictionary of terms you want to calculate.

Listing 2: models/SM.model

```
# YAML 1.1
---
Author: Florian Lyonnet
Date: 11.04.2013
Name: SM
Groups: {'U1': U1, 'SU2L': SU2, 'SU3c': SU3}

##############################
#Fermions assumed weyl spinors
##############################
Fermions: {
   Qbar: {Gen: 3, Qnb:{ U1: -1/6, SU2L: -2, SU3c: -3}},
   Lbar: {Gen: 3, Qnb:{ U1: 1/2, SU2L: -2, SU3c: 1}},
   uR: {Gen: 3, Qnb:{ U1: 4/6, SU2L: 1, SU3c: [1,0]}},
   dR: {Gen: 3, Qnb:{ U1: -1/3, SU2L: 1, SU3c: 3}},
   eR: {Gen: 3, Qnb:{ U1: -1, SU2L: 1, SU3c: 1}}
}

##############################
#Real Scalars, none in the SM
```

```
21  ############################
22
23  RealScalars: {
24  }
25
26  ####################################################
27  #Complex Scalars : give names for the real components
28  ####################################################
29
30  CplxScalars: {
31    H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 1/2, SU2L: 2, SU3c: 1}},
32    H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: -1/2, SU2L: -2, SU3c: 1}}
33  }
34
35  Potential: {
36
37  ####################################
38  # All particles must be defined above !
39  ####################################
40
41  Yukawas:{
42    'Y_{u}': {Fields: [Qbar,uR,H*], Norm: 1},
43    'Y_{d}': {Fields: [Qbar,dR,H], Norm: 1},
44    'Y_{e}': {Fields: [Lbar,eR,H], Norm: 1}
45   },
46  QuarticTerms: {
47   \Lambda_1 : {Fields : [H,H*,H,H*], Norm : 1/2}
48   },
49  ScalarMasses: {
50   \mu_1 : {Fields : [H*,H], Norm : 1}
51   }
52  }
```

On line 6 you find the definition of the gauge group labelled by the keyword "`Groups`". The gauge group is a product of simple Lie algebras and any number of U(1) factors (note, however, that we have not implemented kinetic mixing between the U(1) factors). In turn, each simple Lie algebra or U(1) is associated with a user-defined name (e.g. "`SU3c`" on line 6), and a predefined PyR@TE keyword that specifies the Lie algebra as a mathematical object (cf. SU(3) on line 6). So far, we have implemented SU(N) for $N = 2, \ldots, 6$ and U(1), and in Appendix A we present a list of irreducible representations (irreps) that are currently recognized by PyR@TE. Note that this list will be extended in future versions of PyR@TE.

Next, we discuss how to add particles to our model (lines 11-33 in 2). We distinguish between "`Fermions`", "`RealScalars`" and "`CplxScalars`". Each particle is defined by giving it a name and then listing all its quantum numbers, cf. e.g. line 12 in 2:

```
Qbar: {Gen: 3, Qnb:{ U1: -1/6, SU2L: -2, SU3c: -3}}
```

Here, "`Gen`" is a predefined keyword denoting the number of generations, but the names for the gauge group factors correspond to those specified by the user on line 6. The number of generations for a given particle can in principle be kept general, but then PyR@TE will not be able to perform some basic simplifications and the result may look more complicated. The gauge quantum numbers (specified via the keyword "`Qnb`") can either be specified by the dimension of the corresponding irrep[7], or their Dynkin labels (see definition of "`uR`" on line 14). This is possible for all simple gauge groups, but for SU(2) we have to use a slightly more complicated notation, since we need to distinguish between a given representation and its complex

[7]For simple gauge group factors we use a minus sign to distinguish between a representation and its complex conjugate one. For a U(1) factor the quantum number corresponds to the usual U(1) charge in some physics normalization.

conjugate[8]: "$(n-1,)$" will correspond to the n-dimensional representation, and "$(n-1,\text{True})$" to its complex conjugate. Note that internally all the quantum numbers are translated to Dynkin labels, so if the dimension of a given irrep does not define it uniquely, the user has to use the Dynkin labels. A table with all the irreps that can be used in PyR@TE is given in Appendix A.

Let us now turn to discussing how to add scalars (lines 23-33 in 2). Real scalars are declared by using the keyword "`RealScalars`" and then specifying their gauge quantum numbers (see below for examples including real scalars). For complex scalars (keyword "`CplxScalars`"), one has to name the real degrees of freedom using the keyword "`RealFields`" because each complex field will be split into real components during the calculation. The user can choose a convenient normalization for the complex scalar using the keyword "`Norm`". Also note that you have to declare H^* explicitly (see line 32).

We mention in passing that in order to simplify the notation we have introduced a short-hand syntax. The preceding declarations (lines 1-33 in 2) can also be rewritten in the form given in 3.

Listing 3: Short-hand syntax for the SM model file

```
1  Groups: [U1,SU2,SU3]
2  Fermions: {
3     Qbar: [3, -1/3, -2,-3],
4     Lbar: [3, 1,-2,1],
5     uR: [3,4/3,1, [1,0]],
6     dR: [3,-2/3, 1,3],
7     eR: [3,-2,1,1]
8  }
9  RealScalars: {
10 }
11 CplxScalars: {
12   H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : [1, 2,  1]},
13   H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : [-1,-2,1]}
14 }
```

We now come to the potential which is introduced by the keyword "`Potential`" (lines 35-51 in 2) and has five parts, each preceded by its own keyword: Yukawa interactions ("`Yukawas`"), quartic terms ("`QuarticTerms`"), scalar masses ("`ScalarMasses`"), trilinear interactions ("`TrilinearTerms`"), and fermion masses ("`FermionMasses`"). Each term in one of the five parts is represented by a coupling constant (e.g. "`mu_1`" on line 50), a number of fields ("`[H*,H]`") and a numerical factor ("`Norm : 1`"). Note that for the coupling constant we can use LaTeX notation[9] which will then be used for the output.

6. Addition in versions 1.1.x

The last two options of the table ref Tab. 1 are only available since versions 1.1.x. The fisrt one allows the user to skip some of the terms directly at the calculation level. The different pieces that can be skipped are defined according to [2], e.g. a viable entry would be

```
skip: ['CAabcd','CL2abcd']
```

would skip equations Eq. (43) and (39) of [2]. Note that the label are identical to the ones defined in the article with a "C" added in front and with the subtitution "Lambda→L". All the existing label as well as their precise definition can be seen in "Source/Core/RGEsDefinition.py". The second option, "Only" allows the

[8] In SU(2) any representation is equivalent to its complex conjugate one, but for contracting the SU(2) indices this change of basis matters.

[9] In this case, quotation marks must be used so that the string is recognized as a latex expression.

user to calculate only some of the constants defined in the Lagrangian while keeping the dependance on the others. E.g. in a model in which three quartic terms are defined, “lambda1, lambda2, lambda3”, the user might be interested only in the RGE of “lambda1” while keeping the terms involving “lambda2, lambda3” in the calculation. This is accomplished via the option “Only”.

```
Only: {'QuarticTerms': ['\lambda2','\lambda3']}
```

The user must pass the whole argument as a string if using the 'Only' option via the command line, e.g.

```
python pyR@TE -m models/2HDM.model --Only ''{'QuarticTerms': ['\lambda3']}''
```

Appart from these two new options we also introduced the possibility to define sums of terms. For instance, declaring the term $\lambda_1(H^\dagger H H^\dagger \phi - \phi^\dagger H H^\dagger H)$ in PyR@TE one would need to enter

```
lambda_1: {Fields: [[H*,H,H*,Phi],[Phi*,H,H*,H]],Norm: [1,-1]}
```

in which the norm could also be a single number, e.g. $1/2$ in case where both terms have the same normalization. This allows for a much more efficient disentangling of the various RGEs that mix with each others.

7. Output

In this section we explain in some more detail the various formats in which PyR@TE can generate output.

LaTeX. With the option "`--Latex-Output`", PyR@TE generates a LaTeX file whose name can be set by "`--LatexFile`" followed by a filename. This is the most convenient way to obtain the RGEs in a human-readable format. The file will be saved in the directory specified by the option "`--Results`", or, more conveniently, set in a settings file (cf. 1 on page 4).

Pickle. As the name suggests, Pickle is used to efficiently store Python data structures (in our case the partial or full results of our calculations) for later use. It is particularly useful when combined with the interactive mode to be described in Section 9. We refer the reader to Tab. 1 for a short description of the options "`--Pickle`" and "`--PickleFile`".

Mathematica. To export results to Mathematica, PyR@TE can produce a text file with lines that can be directly copy-pasted into a Mathematica notebook. This option is controlled by the switches "`--TotxtMathematica`" and "`--TotxtMathFile`" (see Tab. 1 for more details).

8. Numerical evaluation

The RGEs generated by PyR@TE can be directly solved and visualized in one of two ways: Either from within `Python` or in a Mathematica notebook. We will describe in turn both approaches.

8.1. *Solving within Mathematica*

The option "`--TotxtMathematica`" also produces a file[10] that ends on "`_numerics.m`" that contains the equations as well as the information required by `Mathematica` to solve the RGEs. The package "`RunPyRate_RGEs.m`" which is included in the directory "`Source/Output`" prepares the equations for `Mathematica` and uses its internal routines to solve the system. For instance, one would enter the following lines in `Mathematica`:

```
1 PATH = "$HOME/pyrate-1.0.0/";
2 Get[PATH <> "results/RGEsOutput.txt_numeric.m"];
3 Get[PATH <> "/Source/Output/RunPyRate_RGEs.m"];
4 IncludeOffDiagonal=True;
5 AllParameters
```

Line 1 tells `Mathematica` where PyR@TE is installed. Line 2 points to the file where the results are stored, and line 3 loads the package to solve the RGEs. The switch "`IncludeOffDiagonal=True`" instructs `Mathematica` to include the full matrix structure of the parameters in solving the RGEs and not to neglect off-diagonal entries. By contrast, "`IncludeOffDiagonal=False`" will neglect the off-diagonal terms. Note that the variable "`AllParameters`" contains all the different RGEs that can be solved i.e. all the ones for which the user can set initial values and explore the results. After the initialization, a routine called "`RunRGEs`" is available that takes as input the starting and ending points of the interval over which the RGEs are to be integrated as well as the initial values of the parameters. Passing a last argument with values "`True/False`" will tell `Mathematica` to ignore or not the two loop contributions in the RGEs if they are available[11].

```
running=RunRGEs[3, 16, {g1->0.36, gSU2L->0.65, gSU3c->1.08},False];
```

The first and second inputs are the logarithms of the scales where the running starts and ends, respectively. If Landau poles appear, `Mathematica` will terminate before reaching the end point. The third input is the initialization of the parameters that have non-zero values at the starting scale. For instance, to run the gauge couplings in the SM from 1 TeV to 10^{16} GeV and to plot the result, simply enter:

```
Plot[{g1[x],gSU2L[x],gSU3c[x]} /.running[[1]],{x,3,16}];
```

This example is also included in the file "`Example.nb`" inside the PyR@TE directory.

8.2. *Using* `Python`

Now we explain how to run the RGEs from within `Python`. With the options "`--Export`" and "`--Export-File`"[12] PyR@TE creates two files: The first one contains the results of the calculation in a form that is amenable to numerical analysis (i.e. `NumPy` objects). The second one, named `SolveRGEs.py`, is a `Python` script that solves[13] the RGEs stored in the first file and contains instructions on how to plot the results with Matplotlib. Note that the user is responsible for setting the interval over which the RGEs will be integrated (start and end points) and also the initial values of the parameters. In a schematic way there are three steps to be done to solve the RGEs within our framework:

[10] If the filename is not set by "`--TotxtMathFile`", the default name "`RGEsOutput.txt_numerics.m`" will be chosen.
[11] This is only possible since version 1.0.3
[12] If this option is skipped, the file will be named "`BetaFunction.py`" by default.
[13] We use python.scipy.integrate [7] to numerically solve ordinary differential equations.

(i) Create an instance of the RGEclass, our own class to represent the RGEs of a given model. This is done by calling the constructor of the class which takes three inputs. The name of the function encoding the beta functions (declared inside BetaFunction.py), number of equations contained in the system as well as a list of the labels to identify the different equations.

```
myrges = RGEclass(beta_function_toymodel,3,labels=['g1','g2','g3'])
```

(ii) Set the `Y0` attributes of the instance just created (myrges in this example) to the initial values.

```
myrges.Y0 = [0.36,0.65,1.08]
```

(iii) Call the method `solve_rges` of the RGEclass instance to solve the RGEs. This method takes in input the initial scale, the final scale, the step of iteration[14] as well as an optional dictionary `assumptions` that we will describe below. Note the scale value must be given in $\log_{10}$ as in the `Mathematica` case.

```
myrges.solve_rges(3,16,0.1,assumptions={})
```

Once these three steps have been done, the user can access the results via the `Sol` attributes of the instance which is a dictionary e.g.

```
myrges.Sol['g1']#contains the values calculated for the 'g1' equation
```

Note that the results are stored according to the labels specified when creating the instance. The `assumptions` dictionary allows the user to customize the solving of the RGEs. By default there are two switches implemented which are `'two-loop': True/False` and `'diag': True/False`. The first one will turn-off all the two-loop contributions when set to `False` and the second one will turn-off the off-diagonal term when set to `False`. However, one can look into the "`BetaFunction.py`" file and implement one's own switches as desired. The file generated when calling PyR@TE with the option "`--Export`" is ready to use and carries all the steps described here for your specific model. The only input from the user is the initial values that have to be set. Comments that will guide the eye of the user to perform the necessary modifications are present in this file.

In the next section we will introduce a user interface à la `Mathematica`, called an `IPython Notebook`, in which we can perform all the tasks described so far in an interactive way.

9. The interactive mode

9.1. starting IPython Notebook

A very convenient and user friendly way of using PyR@TE is to combine our code with an `IPython Notebook` [9]. The first thing to do is to start it by typing in the PyR@TE directory[15]:

```
cd $HOME/pyrate-1.0.0/
ipython notebook
```

The `IPython Notebook` will then start in your default browser, and you will see all the available notebooks that are located in the PyR@TE directory. You can now start executing one of these notebooks by simply clicking on the link. An easy way to get started is to look at one of the tutorials available online and/or to go through the InteractivePyRaTE.ipynb file located in the PyR@TE directory.

[14] i.e. $(t_f - t_i)/t_s$ fixes the number of points calculated, where t_i is the initial scale, t_f the final scale and t_s the step.

[15] The `IPython Notebook` is included in recent installations of `ipython`. If you are using an older version, you can download it from its web page [9] or use the command "`pip install ipython`" (recommended) which should also take care of possible dependencies. If not pre-installed, the package manager "`pip`" can be installed by hand or using "`easy_install pip`".

9.2. Running PyR@TE from within a notebook

An IPython Notebook, that we will simply call notebook in the following, is a web interface inside which you can execute any `Python` command. Once you open a new notebook you have access to cells (as in `Mathematica`) in which you execute any python command of your will as well as external python scripts. Therefore, to run PyR@TE from within the notebook simply enters in one of the cell :

```
%run pyR@TE.py -m models/SM.model -a -v
```

and then press `shift+Enter` to execute the cell. You will see the same output as when executing this command from the shell appear on the webpage. The main advantage of the notebook is that once the execution is finished, all the variables defined during the run are accessibles. Therefore, one has access to the results of the calculation and can study them directly inside the notebook. Note that the equations are stored in the `RGEs` list and can be access like this :

```
#first result depending on the model it can be the gauge couplings or something else
RGEs[0]
```

All the algebraic equations are rendered in latex directly[16] inside the notebook and a right click on one of the equations gives you the possibility to get the latex code directly. In order to make full use of this framework we developped a Toolbox that implements various functions to do common tasks including the export of results, simplifications of equations and comparison of models. All the functions in the Toolbox are now described in the next section.

9.3. The Toolbox

To load the Toolbox from a notebook, make sure that the path `'./Source/Output'` is available (automatic if you have run PyR@TE in the same notebook otherwise you may need to run `import sys sys.path.append('./Source/Output')`). Then import the Toolbox via the regular python command, i.e.

```
from Toolbox import *
```

We now describe all the functions present in the Toolbox and examplify them when necessary.

```
ExportToLatex(FileName,Expression,AModel)
ExportToMathematica(FileName,Expression,AModel)
ExportToPickle(FileName,Expression,AModel,description='')
ExportBetaFunction(FileName,AModel)
```

(i) `FileName` is the name of the output file

(ii) `Expression` is the list containing the results i.e. RGEs

(iii) `AModel` is the model instance of the ModelClass i.e. model

[16] to start the latex printing the user might need to type in the command `init_printing(using_latex=True)`

(iv) `description` is a string which contains some additional description of the results being stored in the pickle file

These can only be called after a run of PyR@TE inside the same notebook since the variables `RGEs` and `model` are defined during the run. Therefore these functions will always be called with the values `model` and `RGEs` for the values `Expression` and `AModel` respectively. They execute the same actions as the corresponding run options : `--LatexOutput`,`--TotxtMathematica`, `--Pickle` and `--Export`.

```
getoneloop(Expression)
gettwoloop(Expression)
```

Where `Expression` in the first two functions can be either a list, a dictionary or an algebraic expression. These functions, get rid of the factor $1/4\pi$,$1/(4\pi)^2$ respectively.

```
loadmodel(FileName)
```

where `FileName` is a pickle file containing a model i.e. a calculation that has been saved via the function `ExportToPickle` or during a previous run (e.g. a calculation done with the option `--Pickle`). Hence someonde can load multiple results in the same notebook and study them.

```
CompareModels(Model1,Model2,subrules=[],display=False):
```

Where `Model1` and `Model2` are either two strings pointing to a pickle file containing a model. or two models loaded in the notebook via the `loadmodel` function. This function calculates the differences for each terms present in both models and display them if `display` is set to `True` else it just returns the result of the comparison as a list where each entry correspond to a different beta function. Finally, the `subrules` argument can be a list of tuple where each tuple contains a replacement rule i.e. of the form `(old symbol, new symbol)` that will be apply to each beta function difference.

```
settozero(Expression,ListofSymols)
```

where `Expression` is an algebraic expression and `ListofSymbols` is a list of symbols[17] that must be set to zero. This function set all the symbols in `ListofSymbols` to zero as well as the traces and matrix multiplications consistently.

10. Pitfalls

There are some subtleties in the implementation of a model to which we would like to draw the user's attention. We start with commenting on the restrictions concerning the input format. To ascertain that `Python` interprets all parts of the model file correctly, the user must make sure that:

(a) only spaces and line breaks can be used as whitespaces, i.e. tabulators are not allowed,

(b) there is a space after each colon,

(c) each element in any input file except for the last one should be separated by a comma.

[17] the symbols must be proper Sympy symbols declared e.g. `Yu=Symbol('Y_u')`

In addition, beware that no operation on the fields is recognized, i.e. for complex conjugated fields one needs to introduce a new symbol. For complex scalars, the real degrees of freedom have to be defined together with the required normalization. The Yukawa matrices are assumed to be symmetric in generation space. Therefore, if e.g. some Yukawa terms are antisymmetric, PyR@TE will return zero.

Finally, all indices are contracted automatically by PyR@TE. For this purpose a database with the most common Clebsch-Gordan coefficients (CGCs) has been created, see Appendix A. This database uses the following conventions:

Normalization. We assume a set of n fields ϕ_i with dimensions D_i under an SU(N) gauge group. We will denote the CGC that gives the contraction of indices to an invariant combination as C, i.e.

$$C_{i_1 i_2 \ldots i_n} \phi_{i_1} \phi_{i_2} \ldots \phi_{i_n}. \tag{1}$$

does not transform under SU(N). Here, the i_x, $x = 1, \ldots, n$ are the charge indices with respect to the gauge group. In contrast to `Susyno` which has been used to create the database of CGCs we use a different normalization. Our convention is that

$$\sum_{i_1=1}^{D_1} \sum_{i_2=1}^{D_2} \cdots \sum_{i_n=1}^{D_n} |C_{i_1 i_2 \ldots i_n}|^2 = \max(D_i). \tag{2}$$

With this normalization we reproduce for instance the standard CGCs for all bilinear terms, but also those for SU(2)$_L$ triplets with non-zero hypercharge and color sextets. However, we do not distinguish between SU(2)$_L$ triplets with and without hypercharge and use the same CGCs for both of them. Therefore, our convention for triplets without hypercharge is different to the standard one by a factor $1/\sqrt{2}$.

Conjugate irreps. In general, conjugate irreps are either defined by the corresponding Dynkin indices or by a negative dimension. However, we would like to stress that

(a) the **2** under SU(2) is related to its conjugate representation $\mathbf{2^*}$. Nevertheless, it is possible to use `"-2"` to represent $\mathbf{2^*}$ which is then treated as a doublet with an additional $i\sigma_2$. For instance, the tensor product $\mathbf{2^*} \otimes \mathbf{2}$ is contracted with the Kronecker δ_{ij} whereas $\mathbf{2} \otimes \mathbf{2}$ by the anti-symmetric tensor ϵ_{ij}. This shows up e.g. in the case of the SM Yukawa couplings Y_d and Y_u.

(b) For self-conjugate representations like the adjoint ones, there are two ways to obtain a gauge singlet. To distinguish these two cases it is possible to use `-A` as dimension of the adjoint of SU(N). The convention is then that bilinear terms of the form $\boldsymbol{A}^* \otimes \boldsymbol{A}$ are always contracted with a Kronecker δ_{ij}, while for $\boldsymbol{A} \otimes \boldsymbol{A}$ the CGCs as calculated by `Susyno` are used. For instance, $\mathbf{3} \otimes \mathbf{3}$ in SU(2) is contracted by a matrix of the form

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{3}$$

while for $\mathbf{3}^* \otimes \mathbf{3}$ the three-dimensional identity matrix is used.

11. Conclusions

To the present date, the automated generation of two-loop renormalization group equations was available only for supersymmetric models. With PyR@TE, we attempted to close this gap that automatically generates

for a general gauge theory the two-loop RGEs for all dimensionless and dimensionful parameters. PyR@TE is easy to use: Once the user specifies in an intuitive format the gauge group and the particle content of a given model, the RGEs are generated, and, for ease of inspection, directly exported to LaTeX. Also, the results can be exported to `Mathematica` where the RGEs can be numerically solved and plotted. Furthermore, we have developed an interactive mode in form of an `IPython Notebook` that mimics much of the functionality of `Mathematica`.

Since the calculations that lead to the RGEs are if not difficult so at least involved, we paid special attention to validating our results. To that end, we not only compared the RGEs generated by PyR@TE with complete or partial results that are available in the literature, but also developed `Mathematica` routines that will be part of an upcoming version of `SARAH` 4. With `SARAH` 4 we find complete agreement, whereas we have some disagreement with the literature, as elaborated on in the previous section.

In future we plan to extend the functionality of PyR@TE to (i) include kinetic mixing, (ii) include partial 3-loop contributions to the RGEs, (iii) extend the library of gauge groups and irreps, (iv) support generation indices for scalars, and (v) run the VEVs (including their gauge dependence) [12].

We believe that PyR@TE can make an important contribution to exploring physics beyond the Standard Model. We have developed the code in the spirit that calculational or technical details should not stop us exploring new scenarios and that one should make sensible use of computer-aided calculations. We hope that the high-energy physics community will find PyR@TE useful and we encourage interested readers to send us constructive feedback which will be helpful to further improve future versions.

Appendix A. List of available irreducible representations

We list below all the gauge groups with their respective irreps available in PyR@TE.

Table A.2: List of all the irreps available in PyR@TE. Note that the True argument for SU(2) represents the conjugate representation.

Gauge Group	Irreps: dimension	Gauge Group	Irreps: dimension
SU(2)	(0,) : 1 (1,) : 2 (1,True) : 2 (2,) : 3 (2,True) : 3 (3,) : 4 (3,True) : 4	SU(4)	(0,0,0) : 1 (0,0,1) : 4 (0,0,2) : 10 (0,1,0) : 6 (1,0,0) : 4 (1,0,1) : 15 (2,0,0) : 10
SU(3)	(0,0) : 1 (0,1) : 3 (0,2) : 6 (0,3) : 10 (1,0) : 3 (1,1) : 8 (2,0) : 6 (3,0) : 10	SU(5)	(0,0,0,0) : 1 (0,0,0,1) : 5 (0,0,0,2) : 15 (0,0,1,0) : 10 (0,1,0,0) : 10 (1,0,0,0) : 5 (1,0,0,1) : 24 (2,0,0,0) : 15
SU(6)	(0,0,0,0,0) : 1 (0,0,0,0,1) : 6 (0,0,0,0,2) : 21 (0,0,0,1,0) : 15 (0,0,1,0,0) : 20 (0,1,0,0,0) : 15 (1,0,0,0,0) : 6 (1,0,0,0,1) : 35 (2,0,0,0,0) : 21	$U(1)_Y$	

Appendix B. Sample Model Files

Listing 4: models/SM_BiD.model

```
# YAML 1.1
---
Author: Florian Lyonnet
Date: 9.08.2013
Name: SMBiD
Groups: {'U1': U1, 'SU2L': SU2, 'SU2R': SU2}

##############################
#Fermions assumed weyl spinors
##############################
Fermions: {
   QL: {Gen: 3, Qnb:{ U1: 1/6, SU2L: 2, SU2R: 1}},
   QR: {Gen: 3, Qnb:{ U1: -1/6, SU2L: 1, SU2R: 2}},
   LL: {Gen: 3, Qnb:{ U1: -1/2, SU2L: 2, SU2R: 1}},
   LR: {Gen: 3, Qnb:{ U1: 1/2, SU2L: 1, SU2R: 2}},
}

#############
#Real Scalars
#############

RealScalars: {
}

#####################################################
#Complex Scalars : give names for the real components
#####################################################

CplxScalars: {
  H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 0, SU2L: 2, SU2R: 2}},
  H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 0, SU2L: -2, SU2R: -2}}
}

Potential: {

#######################################
# All particles must be defined above !
#######################################

Yukawas:{
 'Y_{q}': {Fields: [H,QL,QR], Norm: 1},
 'Y_{l}': {Fields: [H,LL,LR], Norm: 1}
 },
QuarticTerms: {
'\lambda_{1}' : {Fields : [H,H*,H,H*], Norm : 1/2}
 },
ScalarMasses: {
'\mu_{1}' : {Fields : [H*,H], Norm : 1}
 }
}
```

Listing 5: models/SMCplexDoubletScalar.model

```
# YAML 1.1
# #This is the A.4 Model of 1203.5106
---
Author: Florian Lyonnet
Date: 26.07.2013
Name: SMCplexDoubletScalar
Groups: {'U1': U1, 'SU2L': SU2, 'SU3c': SU3}

##############################
#Fermions assumed weyl spinors
##############################
Fermions: {
   Qbar: {Gen: 3, Qnb:{ U1: -1/6, SU2L: -2, SU3c: -3}},
   Lbar: {Gen: 3, Qnb:{ U1: 1/2, SU2L: -2, SU3c: 1}},
   uR: {Gen: 3, Qnb:{ U1: 2/3, SU2L: 1, SU3c: [1,0]}},
   dR: {Gen: 3, Qnb:{ U1: -1/3, SU2L: 1, SU3c: 3}},
   eR: {Gen: 3, Qnb:{ U1: -1, SU2L: 1, SU3c: 1}},
}

#############
#Real Scalars
#############

RealScalars: {
}

#####################################################
#Complex Scalars : give names for the real components
#####################################################

CplxScalars: {
  H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 1/2, SU2L: 2, SU3c: 1}},
  H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: -1/2, SU2L: -2, SU3c: 1}},
  D: {RealFields: [PiD,I*SigmaD], Norm: 1/Sqrt(2), Qnb : {U1: 1/2, SU2L: 2, SU3c: 1}},
  D*: {RealFields: [PiD,-I*SigmaD], Norm: 1/Sqrt(2), Qnb : {U1: -1/2, SU2L: -2, SU3c: 1}},
}

Potential: {

#####################################
# All particles must be defined above !
#####################################

Yukawas:{
  'Y_{u}': {Fields: [Qbar,uR,H*], Norm: 1},
  'Y_{d}': {Fields: [Qbar,dR,H], Norm: 1},
  'Y_{e}': {Fields: [Lbar,eR,H], Norm: 1}
 },
QuarticTerms: {
'\lambda_{1}' : {Fields : [H,H*,H,H*], Norm : 1/2},
'\lambda_{D}' : {Fields: [D,D*,D,D*], Norm : 1/2},
'\kappa_{D}' : {Fields: [D,D*,H,H*], Norm : 1/2},
'\Pkappa_{D}' : {Fields: [D,H*,H,D*], Norm : 1/2}
 },
ScalarMasses: {
'\mu_{H}' : {Fields : [H,H*], Norm : 1},
'\mu_{D}' : {Fields : [D,D*], Norm : 1}
 }
}
```

Listing 6: models/SM.model

```
# YAML 1.1
---
Author: Florian Lyonnet
Date: 11.06.2013
Name: SM
Groups: {'U1': U1, 'SU2L': SU2, 'SU3c': SU3}

##############################
#Fermions assumed weyl spinors
##############################
Fermions: {
   Qbar: {Gen: 3, Qnb:{ U1: -1/6, SU2L: -2, SU3c: -3}},
   Lbar: {Gen: 3, Qnb:{ U1: 1/2, SU2L: -2, SU3c: 1}},
   uR: {Gen: 3, Qnb:{ U1: 2/3, SU2L: 1, SU3c: [1,0]}},
   dR: {Gen: 3, Qnb:{ U1: -1/3, SU2L: 1, SU3c: 3}},
   eR: {Gen: 3, Qnb:{ U1: -1, SU2L: 1, SU3c: 1}}
}

#############
#Real Scalars
#############

RealScalars: {
}

#####################################################
#Complex Scalars : give names for the real components
#####################################################

CplxScalars: {
  H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 1/2, SU2L: 2, SU3c: 1}},
  H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: -1/2, SU2L: -2, SU3c: 1}}
}

Potential: {

#######################################
# All particles must be defined above !
#######################################

Yukawas:{
  'Y_{u}': {Fields: [Qbar,uR,H*], Norm: 1},
  'Y_{d}': {Fields: [Qbar,dR,H], Norm: 1},
  'Y_{e}': {Fields: [Lbar,eR,H], Norm: 1}
 },
QuarticTerms: {
'\Lambda_{1}' : {Fields : [H,H*,H,H*], Norm : 1/2}
 },
ScalarMasses: {
'\mu_{1}' : {Fields : [H*,H], Norm : 1}
 }
}
```

Listing 7: models/ScalarSinglet.model

```
# YAML 1.1
---
Author: Florian Lyonnet
Date: 22.07.2013
Name: ScalarSinglet
Groups: {'U1': U1, 'SU2L': SU2, 'SU3c': SU3}

############################
#Fermions assumed weyl spinors
############################
Fermions: {
   Qbar: {Gen: 2, Qnb:{ U1: -1/6, SU2L: -2, SU3c: -3}},
   Lbar: {Gen: 2, Qnb:{ U1: 1/2, SU2L: -2, SU3c: 1}},
   uR: {Gen: 3, Qnb:{ U1: 2/3, SU2L: 1, SU3c: [1,0]}},
   dR: {Gen: 3, Qnb:{ U1: -1/3, SU2L: 1, SU3c: 3}},
   eR: {Gen: 3, Qnb:{ U1: -1, SU2L: 1, SU3c: 1}}
}

############
#Real Scalars
#############

RealScalars: {
  si : {U1: 0, SU2L: 1, SU3c: 1}
}

#####################################################
#Complex Scalars : give names for the real components
#####################################################

CplxScalars: {
  H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 1/2, SU2L: 2, SU3c: 1}},
  H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: -1/2, SU2L: -2, SU3c: 1}}
}

Potential: {

#######################################
# All particles must be defined above !
#######################################
 Yukawas:{
  'Y_{u}': {Fields: [Qbar,uR,H*], Norm: 1},
  'Y_{d}': {Fields: [Qbar,dR,H], Norm: 1},
  'Y_{e}': {Fields: [Lbar,eR,H], Norm: 1}
 },

QuarticTerms: {
'\lambda_{1}' : {Fields : [H,H*,H,H*], Norm: 1/2},
 '\lambda_{s}' : {Fields : [si,si,si,si], Norm: 1/2},
 '\kappa_{s}' : {Fields : [H,H*,si,si], Norm: 1/2}
 },

ScalarMasses: {
 '\mu_{1}' : {Fields: [H,H*], Norm: 1},
 '\mu_{s}' : {Fields: [si,si], Norm: 1/2}
 }
}
```

Listing 8: models/SMSingletDoublet.model

```
# YAML 1.1
---
Author: Florian Lyonnet
Date: 30.07.2013
Name: SMSingletDoublet
Groups: {'U1': U1, 'SU2L': SU2, 'SU3c': SU3}

############################
#Fermions assumed weyl spinors
############################
Fermions: {
   Qbar: {Gen: 3, Qnb:{ U1: -1/6, SU2L: -2, SU3c: -3}},
   Lbar: {Gen: 3, Qnb:{ U1: 1/2, SU2L: -2, SU3c: 1}},
   uR: {Gen: 3, Qnb:{ U1: 2/3, SU2L: 1, SU3c: [1,0]}},
   dR: {Gen: 3, Qnb:{ U1: -1/3, SU2L: 1, SU3c: 3}},
   eR: {Gen: 3, Qnb:{ U1: -1, SU2L: 1, SU3c: 1}},
   D: {Gen : 1, Qnb:{ U1:  -1/2, SU2L: 2, SU3c: 1}},
   Dc: {Gen: 1, Qnb:{ U1: 1/2, SU2L: 2,SU3c: 1}},
   S: {Gen: 1, Qnb:{ U1: 0, SU2L: 1, SU3c: 1}}
}

############
#Real Scalars
############

RealScalars: {
}

#####################################################
#Complex Scalars : give names for the real components
#####################################################

CplxScalars: {
  H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 1/2, SU2L: 2, SU3c: 1}},
  H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: -1/2, SU2L: -2, SU3c: 1}}
}

Potential: {

#######################################
# All particles must be defined above !
#######################################

Yukawas:{
  'Y_{u}': {Fields: [Qbar,uR,H*], Norm: 1},
  'Y_{d}': {Fields: [Qbar,dR,H], Norm: 1},
  'Y_{e}': {Fields: [Lbar,eR,H], Norm: 1},
  'g_{d}': {Fields: [H,S,D], Norm: 1/Sqrt(2)},
  'g_{u}': {Fields: [H*,S,Dc], Norm: 1/Sqrt(2)}
 },
QuarticTerms: {
 '\lambda_1' : {Fields : [H,H*,H,H*], Norm : 1/2}
 },
ScalarMasses: {
 '\mu_1' : {Fields : [H*,H], Norm : 1},
 },
FermionMasses:{
 '\mD': {Fields: [D,Dc], Norm: 1},
 '\mS': {Fields: [S,S], Norm: 1/2}
}
}
```

Listing 9: models/SMCplxTriplet.model

```
# YAML 1.1
---
Author: Florian Lyonnet
Date: 9.07.2013
Name: SMCplxTriplet
Groups: {'U1': U1, 'SU2L': SU2, 'SU3c': SU3}

############################
#Fermions assumed weyl spinors
############################
Fermions: {
   Qbar: {Gen: 3, Qnb:{ U1: -1/6, SU2L: -2, SU3c: -3}},
   L: {Gen: 3, Qnb:{ U1: -1/2, SU2L: 2, SU3c: 1}},
   uR: {Gen: 3, Qnb:{ U1: 2/3, SU2L: 1, SU3c: 3}},
   dR: {Gen: 3, Qnb:{ U1: -1/3, SU2L: 1, SU3c: 3}},
   eR: {Gen: 3, Qnb:{ U1: -1, SU2L: 1, SU3c: 1}},
   PsiL: {Gen: 1, Qnb: {U1: -1/2, SU2L: 2, SU3c: 1}},
   PsiRbar: {Gen: 1, Qnb: {U1: 1/2, SU2L: -2, SU3c: 1}}
}
############
#Real Scalars
############

RealScalars: {
}

#####################################################
#Complex Scalars : give names for the real components
#####################################################
CplxScalars: {
  H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 1/2, SU2L: 2, SU3c: 1}},
  H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: -1/2, SU2L: -2, SU3c: 1}},
  T : {RealFields: [T1,I*T2], Norm: 1/Sqrt(2), Qnb: {U1: 1, SU2L : 3, SU3c: 1}},
  T* : {RealFields: [T1,-I*T2], Norm: 1/Sqrt(2), Qnb: {U1: -1, SU2L : 3, SU3c: 1}}
}

Potential: {
#######################################
# All particles must be defined above !
#######################################
##############
#The doublet vector like and the yukawa corresponding to the triplet are not included yet
Yukawas:{
  'Y_{u}': {Fields: [H*,Qbar,uR], Norm: 1},
  'f_{L}': {Fields: [T,L,L], Norm: 1/Sqrt(2)},
        'f_{\psi}': {Fields: [T, PsiL,PsiL], Norm: 1/Sqrt(2)}
 },
QuarticTerms: {
'\lambda_{1}' : {Fields : [H,H*,H,H*], Norm : 1/2},
'\lambda_{T}' : {Fields: [T,T*,T,T*], Norm: 1/2},
'\kappa_{T}': {Fields: [T,T*,H,H*], Norm: 1}
 },
ScalarMasses: {
'\mu_{1}' : {Fields : [H,H*], Norm : 1},
 mT : {Fields: [T,T*], Norm: 1/2},
 },
TrilinearTerms: {
 fH : {Fields: [T*,H,H], Norm: 1/Sqrt(2)},
 },
FermionMasses : {
 mD : {Fields: [PsiL,PsiRbar], Norm: 1, latex: \m_D},
 }
}
```

Listing 10: models/SMTripletDoublet.model

```
# YAML 1.1
---
Author: Florian Lyonnet
Date: 30.07.2013
Name: SMTripletDoublet
Groups: {'U1': U1, 'SU2L': SU2, 'SU3c': SU3}

############################
#Fermions assumed weyl spinors
############################
Fermions: {
   Qbar: {Gen: 3, Qnb:{ U1: -1/6, SU2L: -2, SU3c: -3}},
   Lbar: {Gen: 3, Qnb:{ U1: 1/2, SU2L: -2, SU3c: 1}},
   uR: {Gen: 3, Qnb:{ U1: 2/3, SU2L: 1, SU3c: [1,0]}},
   dR: {Gen: 3, Qnb:{ U1: -1/3, SU2L: 1, SU3c: 3}},
   eR: {Gen: 3, Qnb:{ U1: -1, SU2L: 1, SU3c: 1}},
   D: {Gen : 1, Qnb:{ U1:  -1/2, SU2L: 2, SU3c: 1}},
   Dc: {Gen: 1, Qnb:{ U1: 1/2, SU2L: 2,SU3c: 1}},
   T: {Gen: 1, Qnb: { U1: 0, SU2L: 3, SU3c: 1}}
}

############
#Real Scalars
############

RealScalars: {
}

#####################################################
#Complex Scalars : give names for the real components
#####################################################

CplxScalars: {
  H: {RealFields: [Pi,I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: 1/2, SU2L: 2, SU3c: 1}},
  H*: {RealFields: [Pi,-I*Sigma], Norm: 1/Sqrt(2), Qnb : {U1: -1/2, SU2L: -2, SU3c: 1}}
}

Potential: {

#######################################
# All particles must be defined above !
#######################################

Yukawas:{
  'Y_{u}': {Fields: [Qbar,uR,H*], Norm: 1},
  'Y_{d}': {Fields: [Qbar,dR,H], Norm: 1},
  'Y_{e}': {Fields: [Lbar,eR,H], Norm: 1},
  'g_{d}': {Fields: [T,D,H], Norm: -1},
  'g_{u}': {Fields: [T,Dc,H*] ,Norm: 1}
 },
QuarticTerms: {
 '\lambda_1' : {Fields : [H,H*,H,H*], Norm : 1/2},
 },
ScalarMasses: {
 '\mu_1' : {Fields : [H,H*], Norm : 1},
 },
FermionMasses: {
 'm_{T}' : {Fields: [T,T], Norm: 1/2},
 'm_{D}' : {Fields: [D,Dc], Norm: 1}
}
}
```

[1] F. Lyonnet, I. Schienbein, F. Staub, and A. Wingerter, "The PyR@TE web page," 2013. http://pyrate.hepforge.org.
[2] M. X. Luo and Y. Xiao, "Two-loop renormalization group equations in the standard model," *Phys. Rev. Lett.* **90** (2003) 011601, hep-ph/0207271.
[3] A. Wingerter, "Implications of the Stability and Triviality Bounds on the Standard Model with Three and Four Chiral Generations," *Phys.Rev.* **D84** (2011) 095012, 1109.5140.
[4] F. Staub, "SARAH 4: A tool for – not only SUSY – model builders," *in prep.* (2013).
[5] G. van Rossum and J. de Boer, "Interactively testing remote servers using the python programming language," *CWI Quarterly* **4** (1991) 283–303.
[6] P. F. Dubois, K. Hinsen, and J. Hugunin, "Numerical python," *Computers in Physics* **10** (May/June, 1996).
[7] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python." http://www.scipy.org.
[8] SymPy Development Team, "Sympy: Python library for symbolic mathematics." http://www.sympy.org.
[9] F. Pérez and B. E. Granger, "IPython: a System for Interactive Scientific Computing," *Comput. Sci. Eng.* **9** (May, 2007) 21–29.
[10] K. Simonov, "Pyyaml." http://pyyaml.org/wiki/PyYAML.
[11] C. Evans, "YAML Ain't Markup Language." http://www.yaml.org, 2001.
[12] M. Sperling, D. Stöckinger, and A. Voigt, "Renormalization of vacuum expectation values in spontaneously broken gauge theories," *JHEP* **1307** (2013) 132, 1305.1548.