RGESolver

Generated by Doxygen 1.9.6

Chapter 1

<tt>RGESolver</tt>

A C++ library to perform renormalization group evolution of SMEFT coefficients numerically. A faster, approximate solution that neglects the scale dependence of the anomalous dimension matrix is also available. The general flavour case at dimension-six level is considered. Operators that violate lepton and/or baryon number conservation are not considered.
The documentation for this library can be found here.
RGESolver is a free software under the copyright of the GNU General Public License.
If you use RGESolver please cite https://arxiv.org/abs/2210.06838.

1.1 Dependencies

- GSL : The GNU Scientific Library (GSL) is a C library for numerical computations. More details can be found in the GSL website.
- BOOST : BOOST is a set of libraries for the C++ programming language. RGESolver requires only the BOOST headers, not the full libraries, thus a header-only installation is sufficient. More details can be found in the BOOST website.
- C++11 : A compiler that supports at least C++11 standard is required.

1.2 Installation

The installation of RGESolver requires the availability of CMake in the system (version 3.1 or greater). A description of CMake and the instructions for its installation can be found in the CMakewebsite. Clone the repository with

```
git clone https://github.com/silvest/RGESolver --recursive
```

The installation can be performed writhing the following lines in a terminal session (in the RGESolver directory):

```
mkdir build && cd build
cmake .. <options>
cmake --build .
cmake --install .
```

Note that depending on the setting of installation prefix (see below) the user might need root privileges to be able to install RGESolver (thus cmake --install . should be replaced with sudo cmake --install .)

1.2.1 Command line options for the installation

- -DLOCAL_INSTALL:BOOL=<ON or OFF> : to install RGESolver in the directory build/install (default: OFF).
- -DCMAKE_INSTALL_PREFIX:PATH=<RGESolver installation directory> : the directory in which RGESolver will be installed (default: /usr/local). This variable cannot be modified when -DLOCAL INSTALL ALL=ON is set.
- -DDEBUG_MODE:BOOL=<ON or OFF> : to enable the debug mode (default: OFF).
- -DBOOST_INCLUDE_DIR:PATH=<include path>/boost/ : CMakechecks for BOOST headers availability in the system and fails if they are not installed. Thus, if BOOST is not installed in the predefined search path, the user can specify where it is with this option. The path must end with the boost/directory which contains the headers.
- -DGSL_CONFIG_DIR:PATH=<gsl-config directory> : RGESolver uses gsl-config to get the GSL parameters. If this is not in the predefined search path, the user can specify it with this option.

1.3 Usage

The rgesolver-config script is available in the <CMAKE_INSTALL_PREFIX>/bin directory (default: /usr/local), which can be invoked with the following options:

- --cflags: to obtain the include path needed for compilation against RGESolver.
- --libs: to obtain the flags needed for linking against RGESolver.

If the path <CMAKE_INSTALL_PREFIX>/bin is not in the predefined search path, the compilation will (most likely) fail. if the user wants to use the compilation command above, it is suggested to add <CMAKE_INSTALL↩ _PREFIX>/bin to the $PATH variable. Alternatively, the script can be invoked from a terminal session in <CMAKE_INSTALL_PREFIX>/bin to visualize the paths to the library and to the headers.

After the installation, the example program ExampleEvolution.cpp (available in the Examples directory) can be compiled with the command

```
g++ -o app ExampleEvolution.cpp `rgesolver-config --cflags` `rgesolver-config --libs`
```

1.4 Uninstall

The user can uninstall the library typing in a terminal session in the build directory:

```
cmake --build . --target uninstall
```

Also in this case, depending on the setting of installation prefix, the user might need root privileges to be able to uninstall RGESolver.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 RGESolver Class Reference

A class that performs renormalization group evolution in the context of the SMEFT.

`#include <RGESolver.h>`

Public Member Functions

- RGESolver ()
  The default constructor.
- ~**RGESolver** ()
  The default destructor.

Parameters related to the numeric integration.

- double **epsrel** ()
  Getter for the relative error used in the numerical integration.
- double **epsabs** ()
  Getter for the absolute error used in the numerical integration.
- void **Setepsrel** (double epsrel)
  Setter for the relative error used in the numerical integration (default value = 0.005)
- void **Setepsabs** (double epsabs)
  Setter for the absolute error used in the numerical integration (default value = e-13)

Evolution

- void Evolve (std::string method, double muI, double muF)
  Performs the RGE evolution.
- void EvolveToBasis (std::string method, double muI, double muF, std::string basis)
  Performs the RGE evolution and the back rotation on the coefficients with flavour indices.
- void GenerateSMInitialConditions (double mu, std::string basis, std::string method)
  Generates the initial conditions for Standard Model's parameters (gauge couplings, Yukawa coupling, quartic coupling and Higgs' boson mass) at the scale `mu` (in GeV), using one-loop pure SM beta functions. Default low-energy input is used.
- void GenerateSMInitialConditions (double muIn, double muFin, std::string basis, std::string method, double g1in, double g2in, double g3in, double lambdain, double mh2in, double Muin[3], double Mdin[3], double Mein[3], double s12in, double s13in, double s23in, double deltain)

Generates the initial conditions for Standard Model's parameters (gauge couplings, Yukawa coupling, quartic coupling and Higgs' boson mass) at the scale mu *(in GeV), using one-loop pure SM beta functions. User-defined low energy input is used.*

- void EvolveSMOnly (std::string method, double muI, double muF)

  Same as Evolve, but only for the SM parameters. The user should use this method instead of Evolve when interested in pure SM running. Using this function is the same of using Evolve with all the SMEFT coefficients set to 0, but it is faster since it does compute only the evolution for the SM parameters.

Input/output

- double GetCKMAngle (std::string name)

  Getter function for the CKM matrix angles $\theta_{12}, \theta_{13}, \theta_{23}$.
- double GetCKMRealPart (int i, int j)

  Getter function for the CKM matrix (real part)
- double GetCKMImagPart (int i, int j)

  Getter function for the CKM matrix (imaginary part)
- double GetCKMPhase ()

  Getter function for the CKM matrix phase δ.
- void SetCoefficient (std::string name, double val)

  Setter function for scalar/0F parameters (no flavour indices).
- void SetCoefficient (std::string name, double val, int i, int j)

  Setter function for 2F parameters (2 flavour indices).
- void SetCoefficient (std::string name, double val, int i, int j, int k, int l)

  Setter function for 4F parameters (4 flavour indices).
- double GetCoefficient (std::string name)

  Getter function for scalar/0F parameters (no flavour indices).
- double GetCoefficient (std::string name, int i, int j)

  Getter function for 2F parameters (2 flavour indices).
- double GetCoefficient (std::string name, int i, int j, int k, int l)

  Getter function for 4F parameters (4 flavour indices). one of the inserted indices is outside the [0:2] range, an error message is printed and the value 0 is returned.
- void **Reset** ()

  Resets all the SMEFT coefficients to 0 and the SM parameters to their default value. ϵ_{abs} *and* ϵ_{rel} *are reset to their default value (in the UP basis).*
- void SaveOutputFile (std::string filename, std::string format)

  Saves the current values of parameters in a file.

4.1.1 Detailed Description

A class that performs renormalization group evolution in the context of the SMEFT.

The class solves the Renormalization Group Equations (RGEs) numerically. A faster, approximate solution that neglects the scale dependence of the anomalous dimension matrix is also available. Only operators up to dimension six that preserve lepton and baryon numbers are considered. The operator basis is the Warsaw basis, defined in https://arxiv.org/abs/1008.4884. RGESolver splits real and imaginary part of each complex parameter.

The numerical integration is performed with an adaptive step-size routine (the explicit embedded Runge-Kutta-Fehlberg method), using the tools in the GNU Scientific Library. See https://www.gnu.org/software/gsl/doc/html/ode-initval.html for all the details.
The accuracy level of the numerical integration can be tuned selecting the parameters ϵ_{rel} and ϵ_{abs} using the dedicated setter functions.

All the SMEFT coefficients are set using the SetCoefficient methods and accessed with the GetCoefficient methods. There exist three different signatures for each method, depending on the number of flavour indices of the parameter (0,2,4).
These two routines must be used also for the SM parameters $g_1, g_2, g_3, \lambda, m_h^2, \mathrm{Re}(\mathcal{Y}_u), \mathrm{Im}(\mathcal{Y}_u), \mathrm{Re}(\mathcal{Y}_d), \mathrm{Im}(\mathcal{Y}_d),$

$\mathrm{Re}(\mathcal{Y}_e), \mathrm{Im}(\mathcal{Y}_e)$ (we follow https://arxiv.org/abs/1308.2627 for what concerns the conventions in the Higgs' sector).
The routines GetCKMAngle, GetCKMPhase, GetCKMRealPart, GetCKMImagPart should be used when interested in the CKM parameters or elements. The usage of this method is recommended after methods such GenerateSMInitialConditions or EvolveToBasis that choose a specific flavour basis ("UP" or "DOWN"), in which cases the CKM matrix is updated. A complete list of the keys that must be used to correctly invoke setter/getter methods are given in tables 4.1, 4.2, 4.3 and 4.4.
A summary of the operators symmetry classes is given in table 4.5.

We follow http://www.utfit.org/UTfit/Formalism for what concerns the conventions for the CKM matrix.

Author

S. Di Noi, L. Silvestrini.

Copyright

GNU General Public License

Table 4.1 Standard Model parameters. The labels in the left column must be used with the GetCoefficient/SetCoefficient methods, the ones in the right column must be used with GetCKMAngle methods.

Parameter	Name
g_1	g1
g_2	g2
g_3	g3
λ	lambda
$m_h^2\ [\mathrm{GeV}^2]$	mh2
$\mathrm{Re}(\mathcal{Y}_u)$	YuR
$\mathrm{Im}(\mathcal{Y}_u)$	YuI
$\mathrm{Re}(\mathcal{Y}_d)$	YdR
$\mathrm{Im}(\mathcal{Y}_d)$	YdI
$\mathrm{Re}(\mathcal{Y}_e)$	YeR
$\mathrm{Im}(\mathcal{Y}_e)$	YeI

Parameter	Name
$\sin(\theta_{12})$	s12
$\sin(\theta_{13})$	s13
$\sin(\theta_{23})$	s23

Table 4.2 Scalar (and real) SMEFT operators.

Classes 1-3 Coefficient	Name
C_G	CG
$C_{\tilde{G}}$	CGtilde
C_W	CW
$C_{\tilde{W}}$	CWtilde
C_H	CH
$C_{H\Box}$	CHbox
C_{HD}	CHD

Class 4 Coefficient	Name
C_{HG}	CHG
$C_{H\tilde{G}}$	CHGtilde
C_{HW}	CHW
$C_{H\tilde{W}}$	CHWtilde
C_{HB}	CHB
$C_{H\tilde{B}}$	CHBtilde
C_{HWB}	CHWB
$C_{H\tilde{W}B}$	CHWtildeB

Table 4.3 2F SMEFT operators.

Class 5 Coefficient	Name	Symmetry
$\mathrm{Re}(C_{eH})$	CeHR	WC1
$\mathrm{Im}(C_{eH})$	CeHI	WC1
$\mathrm{Re}(C_{uH})$	CuHR	WC1
$\mathrm{Im}(C_{uH})$	CuHI	WC1
$\mathrm{Re}(C_{dH})$	CdHR	WC1
$\mathrm{Im}(C_{dH})$	CdHI	WC1

Class 6 Coefficient	Name	Symmetry
$\mathrm{Re}(C_{eW})$	CeWR	WC1
$\mathrm{Im}(C_{eW})$	CeWI	WC1
$\mathrm{Re}(C_{eB})$	CeBR	WC1
$\mathrm{Im}(C_{eB})$	CeBI	WC1
$\mathrm{Re}(C_{uG})$	CuGR	WC1
$\mathrm{Im}(C_{uG})$	CuGI	WC1
$\mathrm{Re}(C_{uW})$	CuWR	WC1
$\mathrm{Im}(C_{uW})$	CuWI	WC1
$\mathrm{Re}(C_{uB})$	CuBR	WC1
$\mathrm{Im}(C_{uB})$	CuBI	WC1
$\mathrm{Re}(C_{dG})$	CdGR	WC1
$\mathrm{Im}(C_{dG})$	CdGI	WC1
$\mathrm{Re}(C_{dW})$	CdWR	WC1
$\mathrm{Im}(C_{dW})$	CdWI	WC1
$\mathrm{Re}(C_{dB})$	CdBR	WC1
$\mathrm{Im}(C_{dB})$	CdBI	WC1

Class 7 Coefficient	Name	Symmetry
$\mathrm{Re}(C_{Hl1})$	CHl1R	WC2R
$\mathrm{Im}(C_{Hl1})$	CHl1I	WC2I
$\mathrm{Re}(C_{Hl3})$	CHl3R	WC2R
$\mathrm{Im}(C_{Hl3})$	CHl3I	WC2I
$\mathrm{Re}(C_{He})$	CHeR	WC2R
$\mathrm{Im}(C_{He})$	CHeI	WC2I
$\mathrm{Re}(C_{Hq1})$	CHq1R	WC2R
$\mathrm{Im}(C_{Hq1})$	CHq1I	WC2I
$\mathrm{Re}(C_{Hq3})$	CHq3R	WC2R
$\mathrm{Im}(C_{Hq3})$	CHq3I	WC2I
$\mathrm{Re}(C_{Hu})$	CHuR	WC2R
$\mathrm{Im}(C_{Hu})$	CHuI	WC2I
$\mathrm{Re}(C_{Hd})$	CHdR	WC2R
$\mathrm{Im}(C_{Hd})$	CHdI	WC2I
$\mathrm{Re}(C_{Hud})$	CHudR	WC1
$\mathrm{Im}(C_{Hud})$	CHudI	WC1

Table 4.4 4F SMEFT Operators.

Class 8 $(\bar{L}L)(\bar{L}L)$		
Coefficient	**Name**	**Symmetry**
$\mathrm{Re}(C_{ll})$	`CllR`	WC6R
$\mathrm{Im}(C_{ll})$	`CllI`	WC6I
$\mathrm{Re}(C_{qq1})$	`Cqq1R`	WC6R
$\mathrm{Im}(C_{qq1})$	`Cqq1I`	WC6I
$\mathrm{Re}(C_{qq3})$	`Cqq3R`	WC6R
$\mathrm{Im}(C_{qq3})$	`Cqq3I`	WC6I
$\mathrm{Re}(C_{lq1})$	`Clq1R`	WC7R
$\mathrm{Im}(C_{lq1})$	`Clq1I`	WC7I
$\mathrm{Re}(C_{lq3})$	`Clq3R`	WC7R
$\mathrm{Im}(C_{lq3})$	`Clq3I`	WC7I

Class 8 $(\bar{L}R)(\bar{L}R)$		
Coefficient	**Name**	**Symmetry**
$\mathrm{Re}(C_{quqd1})$	`Cquqd1R`	WC5
$\mathrm{Im}(C_{quqd1})$	`Cquqd1I`	WC5
$\mathrm{Re}(C_{quqd8})$	`Cquqd8R`	WC5
$\mathrm{Im}(C_{quqd8})$	`Cquqs8I`	WC5
$\mathrm{Re}(C_{lequ1})$	`Clequ1R`	WC5
$\mathrm{Im}(C_{lequ1})$	`Clequ1I`	WC5
$\mathrm{Re}(C_{lequ3})$	`Clequ3R`	WC5
$\mathrm{Im}(C_{lequ3})$	`Clequ3I`	WC5

Class 8 $(\bar{R}R)(\bar{R}R)$		
Coefficient	**Name**	**Symmetry**
$\mathrm{Re}(C_{ee})$	`CeeR`	WC8R
$\mathrm{Im}(C_{ee})$	`CeeI`	WC8I
$\mathrm{Re}(C_{uu})$	`CuuR`	WC6R
$\mathrm{Im}(C_{uu})$	`CuuI`	WC6I
$\mathrm{Re}(C_{dd})$	`CddR`	WC6R
$\mathrm{Im}(C_{dd})$	`CddI`	WC6I
$\mathrm{Re}(C_{eu})$	`CeuR`	WC7R
$\mathrm{Im}(C_{eu})$	`CeuI`	WC7I
$\mathrm{Re}(C_{ed})$	`CedR`	WC7R
$\mathrm{Im}(C_{ed})$	`CedI`	WC7I
$\mathrm{Re}(C_{ud1})$	`Cud1R`	WC7R
$\mathrm{Im}(C_{ud1})$	`Cud1I`	WC7I
$\mathrm{Re}(C_{ud8})$	`Cud8R`	WC7R
$\mathrm{Im}(C_{ud8})$	`Cud8I`	WC7I

Class 8 $(\bar{L}R)(\bar{R}L)$		
Coefficient	**Name**	**Symmetry**
$\mathrm{Re}(C_{ledq})$	`CledqR`	WC5
$\mathrm{Im}(C_{ledq})$	`CledqI`	WC5

Class 8 $(\bar{L}L)(\bar{R}R)$		
Coefficient	**Name**	**Symmetry**
$\mathrm{Re}(C_{le})$	`CleR`	WC7R
$\mathrm{Im}(C_{le})$	`CleI`	WC7I
$\mathrm{Re}(C_{lu})$	`CluR`	WC7R
$\mathrm{Im}(C_{lu})$	`CluI`	WC7I
$\mathrm{Re}(C_{ld})$	`CldR`	WC7R
$\mathrm{Im}(C_{ld})$	`CldI`	WC7I
$\mathrm{Re}(C_{qe})$	`CqeR`	WC7R
$\mathrm{Im}(C_{qe})$	`CqeI`	WC7I
$\mathrm{Re}(C_{qu1})$	`Cqu1R`	WC7R
$\mathrm{Im}(C_{qu1})$	`Cqu1I`	WC7I
$\mathrm{Re}(C_{qu8})$	`Cqu8R`	WC7R
$\mathrm{Im}(C_{qu8})$	`Cqu8I`	WC7I
$\mathrm{Re}(C_{qd1})$	`Cqd1R`	WC7R
$\mathrm{Im}(C_{qd1})$	`Cqd1I`	WC7I
$\mathrm{Re}(C_{qd8})$	`Cqd8R`	WC7R
$\mathrm{Im}(C_{qd8})$	`Cqd8I`	WC7I

Table 4.5 Symmetry categories for operators in the SMEFT. nF indicates the number of flavour indices for each category.

Parameter	Name
0	0F scalar object
WC1	2F generic real matrix
WC2R	2F Hermitian matrix (real part)
WC2I	2F Hermitian matrix (imaginary part)
WC5	4F generic real object
WC6R	4F two identical $\bar{\psi}\psi$ currents (real part)
WC6I	4F two identical $\bar{\psi}\psi$ currents (imaginary part)
WC7R	4F two independent $\bar{\psi}\psi$ currents (real part)

Parameter	Name
WC7I	4F two independent $\bar{\psi}\psi$ currents (imaginary part)
WC8R	$\mathcal{C}_{ee}$ (real part)
WC8I	$\mathcal{C}_{ee}$ (imaginary part)

Table 4.6 SM parameters used by default to generate SM initial conditions at an arbitrary scale. The scale at which these parameters are given is $\mu = 173.65$ GeV. We follow http://www.utfit.org/UTfit/Formalism for what concerns the conventions for the CKM matrix.

Parameter	Value
g_1	0.3573
g_2	0.6511
g_3	1.161
λ	0.1297
m_h^2 [GeV^2]	15650
$\sin(\theta_{12})$	0.225
$\sin(\theta_{13})$	0.042
$\sin(\theta_{23})$	0.003675
δ [rad]	1.1676

Parameter	Value [GeV]
m_u	0.0012
m_c	0.640
m_t	162.0
m_d	0.0027
m_s	0.052
m_b	2.75
m_e	0.000511
m_μ	0.1057
m_τ	1.776

4.1.2 Constructor & Destructor Documentation

4.1.2.1 RGESolver()

```
RGESolver::RGESolver ( )
```

The default constructor.

It initializes to 0 all the SMEFT coefficients.

4.1.3 Member Function Documentation

4.1.3.1 Evolve()

```
void RGESolver::Evolve (
            std::string method,
            double muI,
            double muF )
```

Performs the RGE evolution.

RGEs are solved with the chosen method from `muI` to `muF`. Currently, the available methods are "Numeric" and "Approximate".
The method takes as initial values the current values of the parameters, set with the SetCoefficient functions. After completing the evolution the values of the parameters are updated and are accessible with the GetCoefficient functions.

Parameters

method	solution method
muI	initial energy scale (in GeV)
muF	final energy scale (in GeV)

4.1.3.2 EvolveSMOnly()

```
void RGESolver::EvolveSMOnly (
            std::string method,
            double muI,
            double muF )
```

Same as Evolve, but only for the SM parameters. The user should use this method instead of Evolve when interested in pure SM running. Using this function is the same of using Evolve with all the SMEFT coefficients set to 0, but it is faster since it does compute only the evolution for the SM parameters.

Parameters

method	solution method
muI	initial energy scale (in GeV)
muF	final energy scale (in GeV)

4.1.3.3 EvolveToBasis()

```
void RGESolver::EvolveToBasis (
            std::string method,
            double muI,
            double muF,
            std::string basis )
```

Performs the RGE evolution and the back rotation on the coefficients with flavour indices.

After the evolution, the CKM matrix is computed. A flavour rotation is performed on the coefficients to go in the chosen basis.

Parameters

method	solution method
muI	initial energy scale (in GeV)
muF	final energy scale (in GeV)
basis	flavour basis after the evolution ("UP" or "DOWN").

4.1.3.4 GenerateSMInitialConditions() [1/2]

```
void RGESolver::GenerateSMInitialConditions (
            double mu,
            std::string basis,
            std::string method )
```

Generates the initial conditions for Standard Model's parameters (gauge couplings, Yukawa coupling, quartic coupling and Higgs' boson mass) at the scale `mu` (in GeV), using one-loop pure SM beta functions. Default low-energy input is used.

The initial conditions are generated at the scale `mu` starting from the values at $\mu = 173.65$ GeV in table 4.6. At the scale `mu` the CKM matrix is computed.

Parameters

mu	Scale (in GeV) at which the initial conditions are generated
basis	Flavour basis (`"UP"` or `"DOWN"`)
method	Method used by RGESolver to run the SM parameters to the scale `mu` (`"Numeric"` or `"Approximate"`)

4.1.3.5 GenerateSMInitialConditions() [2/2]

```
void RGESolver::GenerateSMInitialConditions (
            double muIn,
            double muFin,
            std::string basis,
            std::string method,
            double g1in,
            double g2in,
            double g3in,
            double lambdain,
            double mh2in,
            double Muin[3],
```

```
            double Mdin[3],
            double Mein[3],
            double s12in,
            double s13in,
            double s23in,
            double deltain )
```

Generates the initial conditions for Standard Model's parameters (gauge couplings, Yukawa coupling, quartic coupling and Higgs' boson mass) at the scale `mu` (in GeV), using one-loop pure SM beta functions. User-defined low energy input is used.

The initial conditions are generated at the scale `muFin` starting from the inserted parameters at the scale `muIn`. This method should be used with usual fermion hierarchy (smallest mass for the 1st generation and greatest mass for the 3rd withoud mass degeneracy for all up and down quarks and for charged leptons). The generation of the initial conditions is performed only if all the masses are non-negative and if $\sin\theta_{ij} \in (0,1)$, $\delta \in (\pi, \pi]$. We follow http://www.utfit.org/UTfit for what concerns the conventions for the CKM matrix. At the scale `mu` the CKM matrix is computed.

Parameters

muIn	Low-energy input scale (in GeV)
muFin	Scale (in GeV) at which the initial conditions are generated
basis	Flavour basis (`"UP"` or `"DOWN"`)
method	Method used by RGESolver to run the SM parameters to the scale `mu` (`"Numeric"` or `"Approximate"`)
g1in	g_1
g2in	g_2
g3in	g_3
lambdain	λ
mh2in	m_h^2 (in GeV^2)
Muin	Array containing the masses of the up-type quarks in GeV in the order (m_u, m_c, m_t)
Mdin	Array containing the masses of the down-type quarks in GeV in the order (m_d, m_s, m_b)
Mein	Array containing the masses of the charged leptons in GeV in the order (m_e, m_μ, m_τ)
s12in	The sine of the CKM matrix angle $\sin\theta_{12}$
s13in	The sine of the CKM matrix angle $\sin\theta_{13}$
s23in	The sine of the CKM matrix angle $\sin\theta_{23}$
deltain	The CKM matrix phase δ

4.1.3.6 GetCKMAngle()

```
double RGESolver::GetCKMAngle (
            std::string name )
```

Getter function for the CKM matrix angles $\theta_{12}, \theta_{13}, \theta_{23}$.

This method should be called only after methods that choose a specific flavour basis (as GenerateSMInitialConditions or EvolveToBasis), otherwise the CKM matrix is not updated.

Parameters

name	of the angle (see table 4.1)

Returns

The selected CKM angle.

4.1.3.7 GetCKMImagPart()

```
double RGESolver::GetCKMImagPart (
            int i,
            int j )  [inline]
```

Getter function for the CKM matrix (imaginary part)

This method should be called only after methods that choose a specific flavour basis (as GenerateSMInitialConditions or EvolveToBasis), otherwise the CKM matrix is not updated.

Returns

The imaginary part of the selected CKM matrix element.

4.1.3.8 GetCKMPhase()

```
double RGESolver::GetCKMPhase ( )
```

Getter function for the CKM matrix phase δ.

This method should be called only after methods that choose a specific flavour basis (as GenerateSMInitialConditions or EvolveToBasis), otherwise the CKM matrix is not updated.

Returns

The CKM matrix phase δ.

4.1.3.9 GetCKMRealPart()

```
double RGESolver::GetCKMRealPart (
            int i,
            int j )  [inline]
```

Getter function for the CKM matrix (real part)

This method should be called only after methods that choose a specific flavour basis (as GenerateSMInitialConditions or EvolveToBasis), otherwise the CKM matrix is not updated.

Returns

The real part of the selected CKM matrix element.

4.1.3.10 GetCoefficient() [1/3]

```
double RGESolver::GetCoefficient (
            std::string name )
```

Getter function for scalar/0F parameters (no flavour indices).

Parameters

name	name of the parameter (see table 4.2)

Returns

the requested parameter

4.1.3.11 GetCoefficient() [2/3]

```
double RGESolver::GetCoefficient (
            std::string name,
            int i,
            int j )
```

Getter function for 2F parameters (2 flavour indices).

If at least one of the inserted indices is outside the [0:2] range, an error message is printed and the value 0 is returned.

Parameters

name	name of the parameter (see table 4.3)
i	first flavour index
j	second flavour index

Returns

the requested parameter

4.1.3.12 GetCoefficient() [3/3]

```
double RGESolver::GetCoefficient (
            std::string name,
            int i,
            int j,
            int k,
            int l )
```

Getter function for 4F parameters (4 flavour indices). one of the inserted indices is outside the [0:2] range, an error message is printed and the value 0 is returned.

Parameters

name	name of the parameter (see table 4.4)
i	first flavour index
j	second flavour index
k	third flavour index
l	fourth flavour index

Returns

the requested parameter

4.1.3.13 SaveOutputFile()

```
void RGESolver::SaveOutputFile (
            std::string filename,
            std::string format )
```

Saves the current values of parameters in a file.

Currently, only "SLHA" format is implemented

Parameters

filename	Name of the output file
format	Format of the output file

4.1.3.14 SetCoefficient() [1/3]

```
void RGESolver::SetCoefficient (
            std::string name,
            double val )
```

Setter function for scalar/0F parameters (no flavour indices).

Parameters

name	name of the parameter (see table 4.2)
val	its value

4.1.3.15 SetCoefficient() [2/3]

```
void RGESolver::SetCoefficient (
            std::string name,
            double val,
            int i,
            int j )
```

Setter function for 2F parameters (2 flavour indices).

If at least one of the inserted indices is outside the [0:2] range, an error message is printed and no assignation is performed.

Parameters

name	name of the parameter (see table 4.3)
val	its value
i	first flavour index
j	second flavour index

4.1.3.16 SetCoefficient() [3/3]

```
void RGESolver::SetCoefficient (
            std::string name,
            double val,
            int i,
            int j,
            int k,
            int l )
```

Setter function for 4F parameters (4 flavour indices).

If at least one of the inserted indices is outside the [0:2] range, an error message is printed and no assignation is performed.

Parameters

name	name of the parameter (see table 4.4)
val	its value
i	first flavour index
j	second flavour index
k	third flavour index
l	fourth flavour index

The documentation for this class was generated from the following files:

- Solver/src/RGESolver.h
- Solver/src/RGESolver.cpp
- Solver/src/StaticMembers.cpp

- Solver/src/BetaFunction.cpp
- Solver/src/SettersAndGetters.cpp

Chapter 5

File Documentation

5.1 RGESolver.h

```
00001 #ifndef RGESolver_h
00002 #define RGESolver_h
00003 //RGESolver.h
00004
00005 #include <iostream>
00006 #include <cmath>
00007 #include <fstream>
00008 #include <sstream>
00009
00010 #include <gsl/gsl_errno.h>
00011 #include <gsl/gsl_matrix.h>
00012 #include <gsl/gsl_odeiv2.h>
00013 //#include "IndependentIndices.h"
00014
00015 #include <unordered_map>
00016 //#include <functionalz>
00017 #include <boost/function.hpp>
00018 //#include <boost/bind/bind.hpp>
00019
00020 #include "gslpp.h"
00021
00068 //tables with all the names of the coefficients.
00069
00331 class RGESolver {
00332 public:
00337     RGESolver();
00338
00342     ~RGESolver();
00343
00344
00350     double epsrel() {
00351         return epsrel_;
00352     }
00353
00357     double epsabs() {
00358         return epsabs_;
00359     }
00360
00365     void Setepsrel(double epsrel) {
00366         epsrel_ = epsrel;
00367     }
00368
00373     void Setepsabs(double epsabs) {
00374         epsabs_ = epsabs;
00375     }
00376
00377
00391     void Evolve(std::string method, double muI, double muF);
00392
00404     void EvolveToBasis(std::string method, double muI,
00405             double muF, std::string basis);
00406
00407
00425     void GenerateSMInitialConditions(double mu, std::string basis, std::string method);
00426
00427
00465     void GenerateSMInitialConditions(double muIn, double muFin, std::string basis, std::string method,
00466             double g1in, double g2in, double g3in, double lambdain, double mh2in,
```

```
00467             double Muin[3], double Mdin[3], double Mein[3],
00468             double s12in, double s13in, double s23in, double deltain);
00469
00470
00481     void EvolveSMOnly(std::string method, double muI, double muF);
00482
00483
00501     double GetCKMAngle(std::string name);
00502
00510     double GetCKMRealPart(int i, int j) {
00511         return (CKM(i, j).real());
00512     };
00513
00521     double GetCKMImagPart(int i, int j) {
00522         return (CKM(i, j).imag());
00523     };
00524
00525
00533     double GetCKMPhase();
00534
00535
00536
00537     //Setters for 0F,2F,4F
00538
00544     void SetCoefficient(std::string name, double val);
00555     void SetCoefficient(std::string name, double val, int i, int j);
00569     void SetCoefficient(std::string name, double val, int i, int j,
00570             int k, int l);
00571
00572
00573
00574     //Getters for 0F,2F,4F
00580     double GetCoefficient(std::string name);
00591     double GetCoefficient(std::string name, int i, int j);
00592
00604     double GetCoefficient(std::string name, int i, int j,
00605             int k, int l);
00606
00607
00615     void Reset();
00616
00623     void SaveOutputFile(std::string filename,
00624             std::string format);
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636 private:
00637
00638
00639
00640
00647     void GoToBasis(std::string basis);
00648
00654     void GoToBasisSMOnly(std::string basis);
00655
00656
00661     void ExtractParametersFromCKM();
00667     void FromMassesToYukawas(std::string basis);
00672     void UpdateCKM();
00679     void InitSMOnly();
00684     void UpdateSMOnly();
00685
00686
00691     void SetSMDefaultInput();
00692
00697     void Init();
00703     void Update();
00704
00713     static int func(double logmu, const double y[],
00714             double f[], void* params);
00715
00724     static int funcSMOnly(double logmu, const double y[],
00725             double f[], void* params);
00726
00727
00730
00735     double epsrel_; // = ;
00736
00740     double epsabs_;
```

```
00741
00745     void Resetepsabs() {
00746         epsabs_ = 0.0000000000001;
00747     };
00748
00752     void Resetepsrel() {
00753         epsrel_ = 0.005;
00754     };
00755
00756
00757
00758     gsl_odeiv2_system sys = {func, NULL, 2558, NULL};
00759     /*gsl_odeiv2_driver * d =
00760             gsl_odeiv2_driver_alloc_y_new(&sys,
00761             gsl_odeiv2_step_rkf45,
00762             0.1, epsrel_, epsabs_);*/
00763     gsl_odeiv2_step * s
00764             = gsl_odeiv2_step_alloc(gsl_odeiv2_step_rkf45, 2558);
00765     gsl_odeiv2_evolve * e
00766             = gsl_odeiv2_evolve_alloc(2558);
00767
00768     gsl_odeiv2_system sysSMOnly = {funcSMOnly, NULL, 59, NULL};
00769
00770     gsl_odeiv2_step * sSMOnly
00771             = gsl_odeiv2_step_alloc(gsl_odeiv2_step_rkf45, 59);
00772     gsl_odeiv2_evolve * eSMOnly
00773             = gsl_odeiv2_evolve_alloc(59);
00774
00775
00776
00777
00778     /*
00779     gsl_odeiv2_system sys_ = {func, NULL, 3};
00780     gsl_odeiv2_step * s_ = gsl_odeiv2_step_alloc(
00781             gsl_odeiv2_step_rkf45, 2558);
00782     gsl_odeiv2_control * con_ = gsl_odeiv2_control_standard_new(
00783             epsabs_, epsrel_, 1, 1);
00784     gsl_odeiv2_evolve* evo_ = gsl_odeiv2_evolve_alloc(2558);
00785
00786
00787     gsl_odeiv2_system sysSMOnly_ = {funcSMOnly, NULL, 3};
00788     gsl_odeiv2_step * sSMOnly_ = gsl_odeiv2_step_alloc(
00789             gsl_odeiv2_step_rkf45, 59);
00790     gsl_odeiv2_control * conSMOnly_ = gsl_odeiv2_control_standard_new(
00791             epsabs_, epsrel_, 1, 1);
00792     gsl_odeiv2_evolve* evoSMOnly_ = gsl_odeiv2_evolve_alloc(59);*/
00793
00794
00796     double x[2558];
00797
00799
00800
00801
00802     //-------------------------------------------------------------------------------------------
00803
00812
00815     static const int DWC2R = 6;
00818     static const int DWC2I = 3;
00821     static const int DWC6R = 27;
00824     static const int DWC6I = 18;
00827     static const int DWC7R = 45;
00830     static const int DWC7I = 36;
00833     static const int DWC8R = 21;
00836     static const int DWC8I = 15;
00837
00839
00840
00847
00848     static const int WC2R_indices[DWC2R][2];
00850     static const int WC2I_indices[DWC2I][2];
00852     static const int WC6R_indices[DWC6R][4];
00854     static const int WC6I_indices[DWC6I][4];
00856     static const int WC7R_indices[DWC7R][4];
00858     static const int WC7I_indices[DWC7I][4];
00860     static const int WC8R_indices[DWC8R][4];
00862     static const int WC8I_indices[DWC8I][4];
00863
00864
00866
00867
00868
00869
00870
00871
00872
00883
00884
```

```
00888     static const int NG = 3;
00892     static const int DF = 9;
00896     static const int DFs = (NG*NG + NG) / 2;
00900     static const int DFa = (NG*NG - NG) / 2;
00901
00904     static const int Ngauge = 3;
00907     static const int Nh = 2;
00910     static const int Nyukawa = 3;
00913     static const int Eyuk = (Nyukawa * 2 * DF);
00914     static const int N1 = 4;
00915     static const int N23 = 3;
00916     static const int N4 = 8;
00917
00918     static const int N5 = 3;
00919     static const int E5 = (N5 * 2 * DF);
00920
00921     static const int N6 = 8;
00922     static const int E6 = (N6 * 2 * DF);
00923
00924     static const int N7 = 8;
00926     static const int N7H = 7;
00928     static const int N7NH = 1;
00929     static const int E7 = (N7H*(DWC2R + DWC2I) + N7NH * 2 * DF);
00930
00931     static const int N8_LLLL = 5;
00932     static const int E8_LLLL = 2 * (DWC7R + DWC7I) + 3 * (DWC6R + DWC6I);
00933
00934     static const int N8_RRRR = 7;
00935     static const int E8_RRRR = 4 * (DWC7R + DWC7I) + 2 * (DWC6R + DWC6I) + 1 * (DWC8R + DWC8I);
00936
00937     static const int N8_LLRR = 8;
00938     static const int E8_LLRR = 8 * (DWC7R + DWC7I);
00939
00940     static const int N8_LRRL = 1;
00941     static const int E8_LRRL = 2 * NG*NG*NG*NG*N8_LRRL;
00942     static const int N8_LRLR = 4;
00943     static const int E8_LRLR = 2 * NG*NG*NG*NG*N8_LRLR;
00944
00946
00947
00948
00954     static const double TWO_THIRDS;
00955     static const double FOUR_THIRDS;
00956     static const double EIGHT_THIRDS;
00957     static const double ONE_THIRD;
00958     static const double ONE_SIXTH;
00959     static const double TEN_THIRDS;
00961
00962
00967
00970     static const double delta[3][3];
00971
00972
00974     static const int dim = (Ngauge + Nh + Eyuk + N1 + N23 + N4 + E5 + E6 + E7 + E8_LLLL + E8_RRRR +
    E8_LLRR + E8_LRRL + E8_LRLR);
00975
00977     static const double NC;
00979     static const double NC2;
00980
00983     static const double b01;
00985     static const double b02;
00987     static const double b03;
00988
00989     //Casimirs
00991     static const double cA2;
00993     static const double cA3;
00995     static const double cF2;
00997     static const double cF3;
00999
01000     //Hypercharges (and product of hypercharges)
01001     //
01002
01009     static const double Yh;
01010     static const double Yh2;
01011
01012     static const double Yq;
01013     static const double Yq2;
01014     static const double Yl;
01015     static const double Yl2;
01016
01017     static const double Yu;
01018     static const double Yu2;
01019     static const double Yd;
01020     static const double Yd2;
01021     static const double Ye;
01022     static const double Ye2;
01023
```

```
01024     static const double YhYu;
01025     static const double YhYd;
01026     static const double YhYe;
01027     static const double YhYq;
01028     static const double YhYl;
01029
01030     static const double YuYd;
01031     static const double YuYe;
01032     static const double YuYq;
01033     static const double YuYl;
01034
01035     static const double YdYe;
01036     static const double YdYq;
01037     static const double YdYl;
01038
01039     static const double YeYq;
01040     static const double YeYl;
01041
01042     static const double YlYq;
01044
01045
01046
01047
01052
01053     double g1,  g2,  g3, mh2, lambda;
01058     gslpp::matrix<double> yuR, yuI, ydR, ydI, yeR, yeI;
01060
01063     gslpp::matrix<gslpp::complex> CKM = gslpp::matrix<gslpp::complex>(3, 3, 0.);
01064
01070     double InputScale_SM; //GeV
01071
01072     //CKM parameters
01076     double CKM_delta;
01077
01081     double c12;
01085     double s12;
01089     double c13;
01093     double s13;
01097     double c23;
01101     double s23;
01102
01103     //masses
01107     double mu;
01111     double mc;
01114     double mt;
01115
01119     double md;
01123     double ms;
01126     double mb;
01127
01131     double mel;
01135     double mmu;
01138     double mtau;
01139
01148  double cG = 0.;double cGT = 0.; double cW = 0.;double cWT = 0.;
01153 double cH = 0.;
01155  double cHBOX = 0.; double cHD = 0.;
01158  double cHG = 0.; double cHGT = 0.; double cHW = 0.; double cHWT = 0.; double cHB = 0.; double cHBT =
       0.; double cHWB = 0.;double cHWBT = 0.;
01167
01168
01169
01171     double ceHR[3 * 3] = {0.};
01173     double ceHI[3 * 3] = {0.};
01175     double cuHR[3 * 3] = {0.};
01177     double cuHI[3 * 3] = {0.};
01179     double cdHR[3 * 3] = {0.};
01181     double cdHI[3 * 3] = {0.};
01182
01183  double ceWR[3 * 3] = {0.}; double ceWI[3 * 3] = {0.}; double ceBR[3 * 3] = {0.}; double ceBI[3 * 3] =
       {0.};double cuGR[3 * 3] = {0.};double cuGI[3 * 3] = {0.};double cuWR[3 * 3] = {0.};double cuWI[3 * 3]
       = {0.};double cuBR[3 * 3] = {0.};double cuBI[3 * 3] = {0.};double cdGR[3 * 3] = {0.};double cdGI[3 *
       3] = {0.};double cdWR[3 * 3] = {0.};double cdWI[3 * 3] = {0.};double cdBR[3 * 3] = {0.};double cdBI[3
       * 3] = {0.};
01200
01201
01202
01203  double cHl1R[3 * 3] = {0.}; double cHl1I[3 * 3] = {0.};double cHl3R[3 * 3] = {0.};double cHl3I[3 * 3]
       = {0.};double cHeR[3 * 3] = {0.};double cHeI[3 * 3] = {0.};double cHq1R[3 * 3] = {0.};double cHq1I[3 *
       3] = {0.};double cHq3R[3 * 3] = {0.};double cHq3I[3 * 3] = {0.};double cHuR[3 * 3] = {0.};double
       cHuI[3 * 3] = {0.};double cHdR[3 * 3] = {0.};double cHdI[3 * 3] = {0.};
01218 double cHudR[3 * 3] = {0.};double cHudI[3 * 3] = {0.};
01221
01222
01223
01224 double cllR[3 * 3 * 3 * 3] = {0.};double cllI[3 * 3 * 3 * 3] = {0.};double cqq1R[3 * 3 * 3 * 3] =
       {0.};double cqq1I[3 * 3 * 3 * 3] = {0.};double cqq3R[3 * 3 * 3 * 3] = {0.};double cqq3I[3 * 3 * 3 * 3]
```

```
      = {0.};double clq1R[3 * 3 * 3 * 3] = {0.};double clq1I[3 * 3 * 3 * 3] = {0.};double clq3R[3 * 3 * 3 *
      3] = {0.};double clq3I[3 * 3 * 3 * 3] = {0.};
01235
01236 double cuuR[3 * 3 * 3 * 3] = {0.};double cuuI[3 * 3 * 3 * 3] = {0.};double cddR[3 * 3 * 3 * 3] =
      {0.};double cddI[3 * 3 * 3 * 3] = {0.};double ceeR[3 * 3 * 3 * 3] = {0.};double ceeI[3 * 3 * 3 * 3] =
      {0.};double ceuR[3 * 3 * 3 * 3] = {0.};double ceuI[3 * 3 * 3 * 3] = {0.};double cedR[3 * 3 * 3 * 3] =
      {0.}; double cedI[3 * 3 * 3 * 3] = {0.};double cud1R[3 * 3 * 3 * 3] = {0.};double cud1I[3 * 3 * 3 * 3]
      = {0.};double cud8R[3 * 3 * 3 * 3] = {0.};double cud8I[3 * 3 * 3 * 3] = {0.};
01251
01252
01253 double cleR[3 * 3 * 3 * 3] = {0.};double cleI[3 * 3 * 3 * 3] = {0.};double cluR[3 * 3 * 3 * 3] =
      {0.};double cluI[3 * 3 * 3 * 3] = {0.};double cldR[3 * 3 * 3 * 3] = {0.};double cldI[3 * 3 * 3 * 3] =
      {0.};double cqeR[3 * 3 * 3 * 3] = {0.};double cqeI[3 * 3 * 3 * 3] = {0.};double cqu1R[3 * 3 * 3 * 3] =
      {0.};double cqu1I[3 * 3 * 3 * 3] = {0.};double cqu8R[3 * 3 * 3 * 3] = {0.};double cqu8I[3 * 3 * 3 * 3]
      = {0.};double cqd1R[3 * 3 * 3 * 3] = {0.}; double cqd1I[3 * 3 * 3 * 3] = {0.};double cqd8R[3 * 3 * 3 *
      3] = {0.}; double cqd8I[3 * 3 * 3 * 3] = {0.};
01270
01271
01272
01273  double cledqR[3 * 3 * 3 * 3] = {0.};double cledqI[3 * 3 * 3 * 3] = {0.};
01276
01277
01278 double clequ1R[3 * 3 * 3 * 3] = {0.};double clequ1I[3 * 3 * 3 * 3] = {0.};double clequ3R[3 * 3 * 3 *
      3] = {0.};double clequ3I[3 * 3 * 3 * 3] = {0.};double cquqd1R[3 * 3 * 3 * 3] = {0.};double cquqd1I[3 *
      3 * 3 * 3] = {0.};double cquqd8R[3 * 3 * 3 * 3] = {0.};double cquqd8I[3 * 3 * 3 * 3] = {0.};
01288
01289     //---------------------------------------------------------------------------
01290
01291
01292
01293
01294
01295
01303     std::unordered_map<std::string, double*> CKMAngles;
01304     std::unordered_map<std::string, double*> Operators0F;
01305     std::unordered_map<std::string, boost::function<void(int, int, double) » Setter2F;
01306     std::unordered_map<std::string, boost::function<double(int, int) » Getter2F;
01307     std::unordered_map<std::string, boost::function<void(int, int, int, int, double) » Setter4F;
01308     std::unordered_map<std::string, boost::function<double(int, int, int, int) » Getter4F;
01310
01311
01316
01331     static inline void Print(double* c, std::string name,
01332             std::string sym, std::string format,
01333             std::ofstream& f);
01335
01336
01337
01345
01346     static inline void Yukawa_set(gslpp::matrix<double> *y, int i, int j, double val);
01347     static inline double Yukawa(gslpp::matrix<double> *y, int i, int j);
01348
01349     static inline void WC1_set(double * c, int i, int j, double val);
01350     static inline double WC1(double * c, int i, int j);
01351
01352     static inline double WC2R(double * c, int i, int j);
01353     static inline void WC2R_set(double * c, int i, int j, double val);
01354     static inline double WC2I(double * c, int i, int j);
01355     static inline void WC2I_set(double * c, int i, int j, double val);
01356
01357     static inline double WC3(double * c, int i, int j);
01358     static inline void WC3_set(double * c, int i, int j, double val);
01359
01360     static inline void WC5_set(double * c, int i, int j, int k, int l, double val);
01361     static inline double WC5(double * c, int i, int j, int k, int l);
01362
01363     static inline double WC6R(double * c, int i, int j, int k, int l);
01364     static inline void WC6R_set(double * c, int i, int j, int k, int l, double val);
01365     static inline double WC6I(double * c, int i, int j, int k, int l);
01366     static inline void WC6I_set(double * c, int i, int j, int k, int l, double val);
01367
01368     static inline void WC7R_set(double * c, int i, int j, int k, int l, double val);
01369     static inline double WC7R(double * c, int i, int j, int k, int l);
01370     static inline double WC7I(double * c, int i, int j, int k, int l);
01371     static inline void WC7I_set(double * c, int i, int j, int k, int l, double val);
01372
01373     static inline double WC8R(double * c, int i, int j, int k, int l);
01374     static inline void WC8R_set(double * c, int i, int j, int k, int l, double val);
01375     static inline double WC8I(double * c, int i, int j, int k, int l);
01376     static inline void WC8I_set(double * c, int i, int j, int k, int l, double val);
01378
01379 };
01380
01381
01382 #endif
```

Index