

RGESolver

Generated by Doxygen 1.8.20

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 RGESolver Class Reference	5
3.1.1 Detailed Description	15
3.1.2 Constructor & Destructor Documentation	17
3.1.2.1 RGESolver()	17
3.1.2.2 ~RGESolver()	17
3.1.3 Member Function Documentation	18
3.1.3.1 epsabs()	18
3.1.3.2 epsrel()	18
3.1.3.3 Evolve()	18
3.1.3.4 EvolveSMAonly()	18
3.1.3.5 ExtractParametersFromCKM()	19
3.1.3.6 FromMassesToYukawas()	19
3.1.3.7 func()	19
3.1.3.8 funcSMAonly()	20
3.1.3.9 GenerateSMInitialConditions()	20
3.1.3.10 GetCoefficient() [1/3]	20
3.1.3.11 GetCoefficient() [2/3]	21
3.1.3.12 GetCoefficient() [3/3]	21
3.1.3.13 GetSMInputScale()	22
3.1.3.14 GoToBasisSMAonly()	22
3.1.3.15 Init()	22
3.1.3.16 InitSMAonly()	22
3.1.3.17 Print()	22
3.1.3.18 Reset()	23
3.1.3.19 SaveOutputFile()	23
3.1.3.20 Set_epsabs()	23
3.1.3.21 Set_epsrel()	23
3.1.3.22 Set_step()	24
3.1.3.23 SetCoefficient() [1/3]	24
3.1.3.24 SetCoefficient() [2/3]	24
3.1.3.25 SetCoefficient() [3/3]	25
3.1.3.26 SetSMInputScale()	25
3.1.3.27 step()	25
3.1.3.28 Update()	26
3.1.3.29 UpdateCKM()	26
3.1.3.30 UpdateSMAonly()	26

3.1.3.31 WC1()	26
3.1.3.32 WC1_set()	26
3.1.3.33 WC2I()	26
3.1.3.34 WC2I_set()	27
3.1.3.35 WC2R()	27
3.1.3.36 WC2R_set()	27
3.1.3.37 WC3()	27
3.1.3.38 WC3_set()	27
3.1.3.39 WC5()	28
3.1.3.40 WC5_set()	28
3.1.3.41 WC6I()	28
3.1.3.42 WC6I_set()	28
3.1.3.43 WC6R()	28
3.1.3.44 WC6R_set()	29
3.1.3.45 WC7I()	29
3.1.3.46 WC7I_set()	29
3.1.3.47 WC7R()	29
3.1.3.48 WC7R_set()	29
3.1.3.49 WC8I()	30
3.1.3.50 WC8I_set()	30
3.1.3.51 WC8R()	30
3.1.3.52 WC8R_set()	30
3.1.3.53 Yukawa()	30
3.1.3.54 Yukawa_set()	31
3.1.4 Member Data Documentation	31
3.1.4.1 b01	31
3.1.4.2 b02	31
3.1.4.3 b03	31
3.1.4.4 c12	31
3.1.4.5 c13	32
3.1.4.6 c23	32
3.1.4.7 cA2	32
3.1.4.8 cA3	32
3.1.4.9 cdBI	32
3.1.4.10 cdBR	32
3.1.4.11 cddI	33
3.1.4.12 cddR	33
3.1.4.13 cdGI	33
3.1.4.14 cdGR	33
3.1.4.15 cdHI	33
3.1.4.16 cdHR	33
3.1.4.17 cdWI	34

3.1.4.18 cdWR	34
3.1.4.19 ceBI	34
3.1.4.20 ceBR	34
3.1.4.21 cedI	34
3.1.4.22 cedR	34
3.1.4.23 ceel	35
3.1.4.24 ceer	35
3.1.4.25 ceHI	35
3.1.4.26 ceHR	35
3.1.4.27 ceul	35
3.1.4.28 ceuR	35
3.1.4.29 ceWI	36
3.1.4.30 ceWR	36
3.1.4.31 cF2	36
3.1.4.32 cF3	36
3.1.4.33 cG	36
3.1.4.34 cGT	36
3.1.4.35 cH	37
3.1.4.36 cHB	37
3.1.4.37 CHBOX	37
3.1.4.38 cHBT	37
3.1.4.39 cHD	37
3.1.4.40 cHdI	37
3.1.4.41 cHdR	38
3.1.4.42 cHel	38
3.1.4.43 cHeR	38
3.1.4.44 cHG	38
3.1.4.45 cHGT	38
3.1.4.46 cHI1I	38
3.1.4.47 cHI1R	39
3.1.4.48 cHI3I	39
3.1.4.49 cHI3R	39
3.1.4.50 cHq1I	39
3.1.4.51 cHq1R	39
3.1.4.52 cHq3I	39
3.1.4.53 cHq3R	40
3.1.4.54 cHudI	40
3.1.4.55 cHudR	40
3.1.4.56 cHuI	40
3.1.4.57 cHuR	40
3.1.4.58 cHW	40
3.1.4.59 cHWB	41

3.1.4.60 cHWBT	41
3.1.4.61 cHWT	41
3.1.4.62 CKM	41
3.1.4.63 CKM_delta	41
3.1.4.64 CKM_theta12	41
3.1.4.65 CKM_theta13	42
3.1.4.66 CKM_theta23	42
3.1.4.67 cldI	42
3.1.4.68 cldR	42
3.1.4.69 cledql	42
3.1.4.70 cledqR	42
3.1.4.71 cleI	43
3.1.4.72 clequ1I	43
3.1.4.73 clequ1R	43
3.1.4.74 clequ3I	43
3.1.4.75 clequ3R	43
3.1.4.76 cleR	43
3.1.4.77 clII	44
3.1.4.78 clIR	44
3.1.4.79 clq1I	44
3.1.4.80 clq1R	44
3.1.4.81 clq3I	44
3.1.4.82 clq3R	44
3.1.4.83 cluI	45
3.1.4.84 cluR	45
3.1.4.85 con_	45
3.1.4.86 cqd1I	45
3.1.4.87 cqd1R	45
3.1.4.88 cqd8I	45
3.1.4.89 cqd8R	46
3.1.4.90 cqel	46
3.1.4.91 cqer	46
3.1.4.92 cqq1I	46
3.1.4.93 cqq1R	46
3.1.4.94 cqq3I	46
3.1.4.95 cqq3R	47
3.1.4.96 cqu1I	47
3.1.4.97 cqu1R	47
3.1.4.98 cqu8I	47
3.1.4.99 cqu8R	47
3.1.4.100 cquqd1I	47
3.1.4.101 cquqd1R	48

3.1.4.102 cquqd8I	48
3.1.4.103 cquqd8R	48
3.1.4.104 cuBI	48
3.1.4.105 cuBR	48
3.1.4.106 cud1I	48
3.1.4.107 cud1R	49
3.1.4.108 cud8I	49
3.1.4.109 cud8R	49
3.1.4.110 cuGI	49
3.1.4.111 cuGR	49
3.1.4.112 cuHI	49
3.1.4.113 cuHR	50
3.1.4.114 cuul	50
3.1.4.115 cuuR	50
3.1.4.116 cuWI	50
3.1.4.117 cuWR	50
3.1.4.118 cW	50
3.1.4.119 cWT	51
3.1.4.120 delta	51
3.1.4.121 DF	51
3.1.4.122 DFa	51
3.1.4.123 DFs	51
3.1.4.124 dim	52
3.1.4.125 DWC2I	52
3.1.4.126 DWC2R	52
3.1.4.127 DWC6I	52
3.1.4.128 DWC6R	52
3.1.4.129 DWC7I	52
3.1.4.130 DWC7R	53
3.1.4.131 DWC8I	53
3.1.4.132 DWC8R	53
3.1.4.133 E5	53
3.1.4.134 E6	53
3.1.4.135 E7	53
3.1.4.136 E8_LLLL	53
3.1.4.137 E8_LLRR	54
3.1.4.138 E8_LRLR	54
3.1.4.139 E8_LRRL	54
3.1.4.140 E8_RRRR	54
3.1.4.141 EIGHT_THIRDS	54
3.1.4.142 epsabs_	54
3.1.4.143 epsrel_	54

3.1.4.144 evo_	55
3.1.4.145 Eyuk	55
3.1.4.146 FOUR_THIRDS	55
3.1.4.147 g1	55
3.1.4.148 g2	55
3.1.4.149 g3	55
3.1.4.150 Getter2F	56
3.1.4.151 Getter4F	56
3.1.4.152 InputScale_SM	56
3.1.4.153 lambda	56
3.1.4.154 mb	56
3.1.4.155 mc	56
3.1.4.156 md	56
3.1.4.157 mel	57
3.1.4.158 mh2	57
3.1.4.159 mmu	57
3.1.4.160 ms	57
3.1.4.161 mt	57
3.1.4.162 mtau	57
3.1.4.163 mu	57
3.1.4.164 N1	58
3.1.4.165 N23	58
3.1.4.166 N4	58
3.1.4.167 N5	58
3.1.4.168 N6	58
3.1.4.169 N7	58
3.1.4.170 N7H	58
3.1.4.171 N7NH	59
3.1.4.172 N8_LLLL	59
3.1.4.173 N8_LLRR	59
3.1.4.174 N8_LRLR	59
3.1.4.175 N8_LRRL	59
3.1.4.176 N8_RRRR	59
3.1.4.177 NC	59
3.1.4.178 NC2	60
3.1.4.179 NG	60
3.1.4.180 Ngauge	60
3.1.4.181 Nh	60
3.1.4.182 Nyukawa	60
3.1.4.183 ONE_SIXTH	60
3.1.4.184 ONE_THIRD	61
3.1.4.185 Operators0F	61

3.1.4.186 s12	61
3.1.4.187 s13	61
3.1.4.188 s23	61
3.1.4.189 s_	61
3.1.4.190 Setter2F	62
3.1.4.191 Setter4F	62
3.1.4.192 step_	62
3.1.4.193 sys_	62
3.1.4.194 TEN_THIRDS	62
3.1.4.195 TWO_THIRDS	62
3.1.4.196 WC2I_indices	62
3.1.4.197 WC2R_indices	63
3.1.4.198 WC6I_indices	63
3.1.4.199 WC6R_indices	63
3.1.4.200 WC7I_indices	64
3.1.4.201 WC7R_indices	64
3.1.4.202 WC8I_indices	64
3.1.4.203 WC8R_indices	65
3.1.4.204 x	65
3.1.4.205 Yd	65
3.1.4.206 Yd2	65
3.1.4.207 ydl	65
3.1.4.208 ydR	66
3.1.4.209 YdYe	66
3.1.4.210 YdYI	66
3.1.4.211 YdYq	66
3.1.4.212 Ye	66
3.1.4.213 Ye2	66
3.1.4.214 yel	66
3.1.4.215 yeR	66
3.1.4.216 YeYI	67
3.1.4.217 YeYq	67
3.1.4.218 Yh	67
3.1.4.219 Yh2	67
3.1.4.220 YhYd	67
3.1.4.221 YhYe	67
3.1.4.222 YhYI	67
3.1.4.223 YhYq	67
3.1.4.224 YhYu	68
3.1.4.225 YI	68
3.1.4.226 YI2	68
3.1.4.227 YIYq	68

3.1.4.228 Yq	68
3.1.4.229 Yq2	68
3.1.4.230 Yu	68
3.1.4.231 Yu2	68
3.1.4.232 yul	69
3.1.4.233 yuR	69
3.1.4.234 YuYd	69
3.1.4.235 YuYe	69
3.1.4.236 YuYl	69
3.1.4.237 YuYq	69
4 File Documentation	71
4.1 BetaFunction.cc File Reference	71
4.2 RGESolver.cc File Reference	71
4.3 RGESolver.h File Reference	71
4.4 SettersAndGetters.cc File Reference	71
4.5 StaticMembers.cc File Reference	71
Index	73

Chapter 1

Class Index

1.1 Class List

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

RGESolver	A class that performs renormalization group evolution in the context of the SMEFT	5
---------------------------	---	-------------------

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

BetaFunction.cc	71
RGESolver.cc	71
RGESolver.h	71
SettersAndGetters.cc	71
StaticMembers.cc	71

Chapter 3

Class Documentation

3.1 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.
- double [epsrel](#) ()
Getter for the relative error used in the numerical integration.
- double [epsabs](#) ()
Getter for the absolute error used in the numerical integration.
- double [step](#) ()
Getter for the step used in the numerical integration.
- void [Set_epsrel](#) (double [epsrel](#))
Setter for the relative error used in the numerical integration.
- void [Set_epsabs](#) (double [epsabs](#))
Setter for the absolute error used in the numerical integration.
- void [Set_step](#) (double [step](#))
Setter for the step used in the numerical integration.
- 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).
- void [Evolve](#) (std::string method, double muL, double muF)
Command to perform the RGE evolution.
- void [GenerateSMInitialConditions](#) (double mu, std::string basis, std::string method, bool inputCKM=true)
Generates the initial conditions for Standard Model's parameters (gauge couplings, Yukawa coupling, quartic coupling and Higgs' boson mass).
- void [EvolveSMOnly](#) (std::string method, double muL, double muF)
Same as [Evolve\(std::string method, double muL, double muF\)](#), but only for the SM parameters. The user should use this method instead of [Evolve](#) when interested in pure SM running.
- void [SetSMInputScale](#) (double mu)
Setter method for the scale at which the method [GenerateSMInitialConditions](#) takes the input values for SM parameters.
- double [GetSMInputScale](#) ()
Getter method for the scale at which the method [GenerateSMInitialConditions](#) takes the input values for SM parameters.
- void [SaveOutputFile](#) (std::string filename, std::string format)
Saves the current values of parameters in a file.
- void [Reset](#) ()
Resets all the parameters to 0.

Private Member Functions

- void [GoToBasisSMOnly](#) (std::string basis)
- void [ExtractParametersFromCKM](#) ()
- void [UpdateCKM](#) ()
- void [InitSMOnly](#) ()
- void [UpdateSMOnly](#) ()
- void [FromMassesToYukawas](#) (std::string basis)
- void [Init](#) ()
Inserts the initial values of the parameters in the array x .
- void [Update](#) ()
Saves the evolved values of the coefficients from x .

Static Private Member Functions

- static int [funcSMOnly](#) (double logmu, const double y[], double f[], void *params)
- static int [func](#) (double logmu, const double y[], double f[], void *params)
Computes the beta functions for the SMEFT.

Output file

Stuff for output on file

- static void [Print](#) (double *c, std::string name, std::string sym, std::string format, std::ofstream &f)
Prints on file the coefficient c .

Setters and Getters

Setter and getter function for each symmetry class.

- static void [Yukawa_set](#) (double y[3][3], int i, int j, double val)

- static double [Yukawa](#) (double y[3][3], int i, int j)
- static void [WC1_set](#) (double *c, int i, int j, double val)
- static double [WC1](#) (double *c, int i, int j)
- static double [WC2R](#) (double *c, int i, int j)
- static void [WC2R_set](#) (double *c, int i, int j, double val)
- static double [WC2I](#) (double *c, int i, int j)
- static void [WC2I_set](#) (double *c, int i, int j, double val)
- static double [WC3](#) (double *c, int i, int j)
- static void [WC3_set](#) (double *c, int i, int j, double val)
- static void [WC5_set](#) (double *c, int i, int j, int k, int l, double val)
- static double [WC5](#) (double *c, int i, int j, int k, int l)
- static double [WC6R](#) (double *c, int i, int j, int k, int l)
- static void [WC6R_set](#) (double *c, int i, int j, int k, int l, double val)
- static double [WC6I](#) (double *c, int i, int j, int k, int l)
- static void [WC6I_set](#) (double *c, int i, int j, int k, int l, double val)
- static void [WC7R_set](#) (double *c, int i, int j, int k, int l, double val)
- static double [WC7R](#) (double *c, int i, int j, int k, int l)
- static double [WC7I](#) (double *c, int i, int j, int k, int l)
- static void [WC7I_set](#) (double *c, int i, int j, int k, int l, double val)
- static double [WC8R](#) (double *c, int i, int j, int k, int l)
- static void [WC8R_set](#) (double *c, int i, int j, int k, int l, double val)
- static double [WC8I](#) (double *c, int i, int j, int k, int l)
- static void [WC8I_set](#) (double *c, int i, int j, int k, int l, double val)

Private Attributes

- `gslpp::matrix< gslpp::complex > CKM = gslpp::matrix<gslpp::complex>(3, 3, 0.)`
The CKM matrix.
- `double InputScale_SM = 91.`
the scale at which the method `GenerateSMInitialConditions` takes the input values for SM parameters.
- `double CKM_theta12 = 0.2`
 θ_{12} of the CKM matrix (in radians). The default value is ???
- `double CKM_theta13 = 0.1`
- `double CKM_theta23 = 0.3`
- `double CKM_delta = 3.14 / 4.`
- `double c12`
 $\cos \theta_{12}$
- `double s12`
 $\sin \theta_{12}$
- `double c13`
 $\cos \theta_{13}$
- `double s13`
 $\sin \theta_{13}$
- `double c23`
 $\cos \theta_{23}$
- `double s23`
 $\sin \theta_{23}$
- `double mu = 0.002`
 m_u , the mass of up quark in GeV (default value ????)
- `double mc = 1.2`
- `double mt = 170.`
- `double md = 0.006`
- `double ms = 0.05`
- `double mb = 5.2`
- `double mel = 0.0005`

- double `mmu` = 0.100
- double `mtau` = 1.2

GSL Objects

- double `epsrel_` = 0.0000000000000001
Relative error used in the integrator with its default value.
- double `epsabs_` = 0.0001
Absolute error used in the integrator with its default value.
- double `step_`
Last step used in the integrator.
- gsl_odeiv2_system `sys_` = {`func`, NULL, 3}
- gsl_odeiv2_step * `s_`
- gsl_odeiv2_control * `con_`
- gsl_odeiv2_evolve * `evo_` = `gsl_odeiv2_evolve_alloc(2558)`
- double `x` [2558]
1D array for the integration

Standard Model parameters

- double `g1`
 g_1
- double `g2`
 g_2
- double `g3`
 g_3
- double `mh2` = 126. * 126.
 m_h^2 (Higgs boson mass squared)
- double `lambda` = 0.2
 λ (Higgs quartic coupling)
- double `yuR` [3][3]
- double `yul` [3][3]
- double `ydR` [3][3]
- double `ydl` [3][3]
- double `yeR` [3][3]
- double `yel` [3][3]

SMEFT dimension-six operators

By default, all SMEFT dimension-six operators' coefficients are set to 0. See <https://arxiv.org/abs/1308.2627> tab. 1 for the full list of operators. Each member has a class from 1 to 8 depending on its field contents, as well as a flavour symmetry classification (WC1, WC2R, WC2I...).

- double `cG` = 0.
 C_G (class 1, scalar)
- double `cGT` = 0.
 $C_{\tilde{G}}$ (class 1, scalar)
- double `cW` = 0.
 C_W (class 1, scalar)
- double `cWT` = 0.
 $C_{\tilde{W}}$ (class 1, scalar)
- double `cH` = 0.
 C_H (class 2, scalar)
- double `cHBOX` = 0.
 $C_{H\Box}$ (class 3, scalar)

- double **cHD** = 0.
 C_{HD} (class 3, scalar)
- double **chG** = 0.
 C_{HG} (class 4, scalar)
- double **chGT** = 0.
 $C_{H\tilde{G}}$ (class 4, scalar)
- double **chW** = 0.
 C_{HW} (class 4, scalar)
- double **chWT** = 0.
 $C_{H\tilde{W}}$ (class 4, scalar)
- double **chB** = 0.
 C_{HB} (class 4, scalar)
- double **chBT** = 0.
 $C_{H\tilde{B}}$ (class 4, scalar)
- double **chWB** = 0.
 C_{HWB} (class 4, scalar)
- double **chWBT** = 0.
 $C_{H\tilde{W}B}$ (class 4, scalar)
- double **ceHR** [3 *3] = {0.}
 $\Re[C_{eH}]$ (class 5, WC1)
- double **ceHI** [3 *3] = {0.}
 $\Im[C_{eH}]$ (class 5, WC1)
- double **cuHR** [3 *3] = {0.}
 $\Re[C_{uH}]$ (class 5, WC1)
- double **cuHI** [3 *3] = {0.}
 $\Im[C_{uH}]$ (class 5, WC1)
- double **cdHR** [3 *3] = {0.}
 $\Re[C_{dH}]$ (class 5, WC1)
- double **cdHI** [3 *3] = {0.}
 $\Im[C_{dH}]$ (class 5, WC1)
- double **ceWR** [3 *3] = {0.}
 $\Re[C_{eW}]$ (class 6, WC1)
- double **ceWI** [3 *3] = {0.}
 $\Im[C_{eW}]$ (class 6, WC1)
- double **ceBR** [3 *3] = {0.}
 $\Re[C_{eB}]$ (class 6, WC1)
- double **ceBI** [3 *3] = {0.}
 $\Im[C_{eB}]$ (class 6, WC1)
- double **cuGR** [3 *3] = {0.}
 $\Re[C_{uG}]$ (class 6, WC1)
- double **cuGI** [3 *3] = {0.}
 $\Im[C_{uG}]$ (class 6, WC1)
- double **cuWR** [3 *3] = {0.}
 $\Re[C_{uW}]$ (class 6, WC1)
- double **cuWI** [3 *3] = {0.}
 $\Im[C_{uW}]$ (class 6, WC1)
- double **cuBR** [3 *3] = {0.}
 $\Re[C_{uB}]$ (class 6, WC1)
- double **cuBI** [3 *3] = {0.}
 $\Im[C_{uB}]$ (class 6, WC1)
- double **cdGR** [3 *3] = {0.}
 $\Re[C_{dG}]$ (class 6, WC1)
- double **cdGI** [3 *3] = {0.}
 $\Im[C_{dG}]$ (class 6, WC1)
- double **cdWR** [3 *3] = {0.}
 $\Re[C_{dW}]$ (class 6, WC1)
- double **cdWI** [3 *3] = {0.}
 $\Im[C_{dW}]$ (class 6, WC1)
- double **cdBR** [3 *3] = {0.}

- $\Re[C_{dB}]$ (class 6, WC1)
- double **cdBI** [3 *3]
- $\Im[C_{dW}]$ (class 6, WC1)
- double **chI1R** [3 *3] = {0.}
- $\Re[C_{Hl1}]$ (class 7, WC2R)
- double **chI1I** [3 *3] = {0.}
- $\Im[C_{Hl1}]$ (class 7, WC2I)
- double **chI3R** [3 *3] = {0.}
- $\Re[C_{Hl3}]$ (class 7, WC2R)
- double **chI3I** [3 *3] = {0.}
- $\Im[C_{Hl3}]$ (class 7, WC2I)
- double **cHeR** [3 *3] = {0.}
- $\Re[C_{He}]$ (class 7, WC2R)
- double **cHeI** [3 *3] = {0.}
- $\Im[C_{He}]$ (class 7, WC2I)
- double **chq1R** [3 *3] = {0.}
- $\Re[C_{Hq1}]$ (class 7, WC2R)
- double **chq1I** [3 *3] = {0.}
- $\Im[C_{Hq1}]$ (class 7, WC2I)
- double **chq3R** [3 *3] = {0.}
- $\Re[C_{Hq3}]$ (class 7, WC2R)
- double **chq3I** [3 *3] = {0.}
- $\Im[C_{Hq3}]$ (class 7, WC2I)
- double **chUR** [3 *3] = {0.}
- $\Re[C_{Hu}]$ (class 7, WC2R)
- double **chUI** [3 *3] = {0.}
- $\Im[C_{Hu}]$ (class 7, WC2I)
- double **chdR** [3 *3] = {0.}
- $\Re[C_{Hd}]$ (class 7, WC2R)
- double **chdI** [3 *3] = {0.}
- $\Im[C_{Hd}]$ (class 7, WC2I)
- double **chudR** [3 *3] = {0.}
- $\Re[C_{Hud}]$ (class 7, WC1)
- double **chudI** [3 *3] = {0.}
- $\Im[C_{Hud}]$ (class 7, WC1)
- double **clIR** [3 *3 *3 *3] = {0.}
- $\Re[C_{ll}]$ (class 8-[LL][LL], WC6R)
- double **clII** [3 *3 *3 *3] = {0.}
- $\Im[C_{ll}]$ (class 8-[LL][LL], WC6I)
- double **cqq1R** [3 *3 *3 *3] = {0.}
- $\Re[C_{qq1}]$ (class 8-[LL][LL], WC6R)
- double **cqq1I** [3 *3 *3 *3] = {0.}
- $\Im[C_{qq1}]$ (class 8-[LL][LL], WC6I)
- double **cqq3R** [3 *3 *3 *3] = {0.}
- $\Re[C_{qq3}]$ (class 8-[LL][LL], WC6R)
- double **cqq3I** [3 *3 *3 *3] = {0.}
- $\Im[C_{qq3}]$ (class 8-[LL][LL], WC6I)
- double **clq1R** [3 *3 *3 *3] = {0.}
- $\Re[C_{lq1}]$ (class 8-[LL][LL], WC7R)
- double **clq1I** [3 *3 *3 *3] = {0.}
- $\Im[C_{lq1}]$ (class 8-[LL][LL], WC7I)
- double **clq3R** [3 *3 *3 *3] = {0.}
- $\Re[C_{lq3}]$ (class 8-[LL][LL], WC7R)
- double **clq3I** [3 *3 *3 *3] = {0.}
- $\Im[C_{lq3}]$ (class 8-[LL][LL], WC7I)
- double **cuuR** [3 *3 *3 *3] = {0.}
- $\Re[C_{uu}]$ (class 8-[RR][RR], WC6R)
- double **cuuI** [3 *3 *3 *3] = {0.}
- $\Im[C_{uu}]$ (class 8-[RR][RR], WC6I)

- double **cddR** [3 *3 *3 *3] = {0.}
 $\Re[C_{dd}]$ (class 8-[RR][RR], WC6R)
- double **cddl** [3 *3 *3 *3] = {0.}
 $\Im[C_{dd}]$ (class 8-[RR][RR], WC6I)
- double **ceeR** [3 *3 *3 *3] = {0.}
 $\Re[C_{ee}]$ (class 8-[RR][RR], WC8R)
- double **ceel** [3 *3 *3 *3] = {0.}
 $\Im[C_{ee}]$ (class 8-[RR][RR], WC8I)
- double **ceuR** [3 *3 *3 *3] = {0.}
 $\Re[C_{eu}]$ (class 8-[RR][RR], WC7R)
- double **ceul** [3 *3 *3 *3] = {0.}
 $\Im[C_{eu}]$ (class 8-[RR][RR], WC7I)
- double **cedR** [3 *3 *3 *3] = {0.}
 $\Re[C_{ed}]$ (class 8-[RR][RR], WC7R)
- double **cedl** [3 *3 *3 *3] = {0.}
 $\Im[C_{ed}]$ (class 8-[RR][RR], WC7I)
- double **cud1R** [3 *3 *3 *3] = {0.}
 $\Re[C_{ud1}]$ (class 8-[RR][RR], WC7R)
- double **cud1l** [3 *3 *3 *3] = {0.}
 $\Im[C_{ud1}]$ (class 8-[RR][RR], WC7I)
- double **cud8R** [3 *3 *3 *3] = {0.}
 $\Re[C_{ud8}]$ (class 8-[RR][RR], WC7R)
- double **cud8l** [3 *3 *3 *3] = {0.}
 $\Im[C_{ud1}]$ (class 8-[RR][RR], WC7I)
- double **cleR** [3 *3 *3 *3] = {0.}
 $\Re[C_{le}]$ (class 8-[LL][RR], WC7R)
- double **clle** [3 *3 *3 *3] = {0.}
 $\Im[C_{le}]$ (class 8-[LL][RR], WC7I)
- double **cluR** [3 *3 *3 *3] = {0.}
 $\Re[C_{lu}]$ (class 8-[LL][RR], WC7R)
- double **clul** [3 *3 *3 *3] = {0.}
 $\Im[C_{lu}]$ (class 8-[LL][RR], WC7I)
- double **clldR** [3 *3 *3 *3] = {0.}
 $\Re[C_{ld}]$ (class 8-[LL][RR], WC7R)
- double **clldl** [3 *3 *3 *3] = {0.}
 $\Im[C_{ld}]$ (class 8-[LL][RR], WC7I)
- double **cqeR** [3 *3 *3 *3] = {0.}
 $\Re[C_{qe}]$ (class 8-[LL][RR], WC7R)
- double **cqel** [3 *3 *3 *3] = {0.}
 $\Im[C_{qe}]$ (class 8-[LL][RR], WC7I)
- double **cqu1R** [3 *3 *3 *3] = {0.}
 $\Re[C_{qu1}]$ (class 8-[LL][RR], WC7R)
- double **cqu1l** [3 *3 *3 *3] = {0.}
 $\Im[C_{qu1}]$ (class 8-[LL][RR], WC7I)
- double **cqu8R** [3 *3 *3 *3] = {0.}
 $\Re[C_{qu8}]$ (class 8-[LL][RR], WC7R)
- double **cqu8l** [3 *3 *3 *3] = {0.}
 $\Im[C_{qu8}]$ (class 8-[LL][RR], WC7I)
- double **cqd1R** [3 *3 *3 *3] = {0.}
 $\Re[C_{qd1}]$ (class 8-[LL][RR], WC7R)
- double **cqd1l** [3 *3 *3 *3] = {0.}
 $\Im[C_{qd1}]$ (class 8-[LL][RR], WC7I)
- double **cqd8R** [3 *3 *3 *3] = {0.}
 $\Re[C_{qd8}]$ (class 8-[LL][RR], WC7R)
- double **cqd8l** [3 *3 *3 *3] = {0.}
 $\Im[C_{qd8}]$ (class 8-[LL][RR], WC7I)
- double **clcdqR** [3 *3 *3 *3] = {0.}
 $\Re[C_{ledq}]$ (class 8-[LR][RL], WC5)
- double **clcdql** [3 *3 *3 *3] = {0.}

- $\Im [C_{ledq}]$ (class 8-[LR][RL], WC5)
- double [clequ1R](#) [3 *3 *3 *3] = {0.}
- $\Re [C_{lequ1}]$ (class 8-[LR][LR], WC5)
- double [clequ1I](#) [3 *3 *3 *3] = {0.}
- $\Im [C_{lequ1}]$ (class 8-[LR][LR], WC5)
- double [clequ3R](#) [3 *3 *3 *3] = {0.}
- $\Re [C_{lequ3}]$ (class 8-[LR][LR], WC5)
- double [clequ3I](#) [3 *3 *3 *3] = {0.}
- $\Im [C_{lequ3}]$ (class 8-[LR][LR], WC5)
- double [cquqd1R](#) [3 *3 *3 *3] = {0.}
- $\Re [C_{quqd1}]$ (class 8-[LR][LR], WC5)
- double [cquqd1I](#) [3 *3 *3 *3] = {0.}
- $\Im [C_{quqd1}]$ (class 8-[LR][LR], WC5)
- double [cquqd8R](#) [3 *3 *3 *3] = {0.}
- $\Re [C_{quqd8}]$ (class 8-[LR][LR], WC5)
- double [cquqd8I](#) [3 *3 *3 *3] = {0.}
- $\Im [C_{quqd8}]$ (class 8-[LR][LR], WC5)

Maps for I/O

Maps that connects the coefficients with the appropriate getter/setter functions (based on their symmetry properties)

- `std::unordered_map< std::string, double * >` [Operators0F](#)
- `std::unordered_map< std::string, boost::function< void(int, int, double) > >` [Setter2F](#)
- `std::unordered_map< std::string, boost::function< double(int, int) > >` [Getter2F](#)
- `std::unordered_map< std::string, boost::function< void(int, int, int, int, double) > >` [Setter4F](#)
- `std::unordered_map< std::string, boost::function< double(int, int, int, int) > >` [Getter4F](#)

Static Private Attributes

Independent entries

We follow <https://arxiv.org/abs/2010.16341> tab. 15, 16 for the symmetry class of the operators.

Notice that operators in classes WC1 and WC5 have no restrictions for neither real nor imaginary part, each having N_G^2 (for WC1) or N_G^4 (for WC5).

- static const int [DWC2R](#) = 6
Number of independent entries of the real part of operators in symmetry class WC2.
- static const int [DWC2I](#) = 3
Number of independent entries of the imaginary part of operators in symmetry class WC2.
- static const int [DWC6R](#) = 27
Number of independent entries of the real part of operators in symmetry class WC6.
- static const int [DWC6I](#) = 18
Number of independent entries of the imaginary part of operators in symmetry class WC6.
- static const int [DWC7R](#) = 45
Number of independent entries of the real part of operators in symmetry class WC7.
- static const int [DWC7I](#) = 36
Number of independent entries of the imaginary part of operators in symmetry class WC7.
- static const int [DWC8R](#) = 21
Number of independent entries of the real part of operators in symmetry class WC8.
- static const int [DWC8I](#) = 15
Number of independent entries of the imaginary part of operators in symmetry class WC8.

Indices chosen as independent entries

the element `WCn_indices[a][b]` must be interpreted as : the b-th index (there are 4 in 4F operators and 2 in 2F) of the a-th independent entry for the WCn category, where $n = 1, 2R, 2I, \dots$

- static const int [WC2R_indices](#) [DWC2R][2]
Independent indices for WC2R.
- static const int [WC2I_indices](#) [DWC2I][2]
Independent indices for WC2I.
- static const int [WC6R_indices](#) [DWC6R][4]
Independent indices for WC6R.
- static const int [WC6I_indices](#) [DWC6I][4]
Independent indices for WC6I.
- static const int [WC7R_indices](#) [DWC7R][4]
Independent indices for WC7R.
- static const int [WC7I_indices](#) [DWC7I][4]
Independent indices for WC7I.
- static const int [WC8R_indices](#) [DWC8R][4]
Independent indices for WC8R.
- static const int [WC8I_indices](#) [DWC8I][4]
Independent indices for WC8I.

Number of operators for each class

See <https://arxiv.org/abs/1308.2627> tab. 1 for the full list of operators. There are 8 different classes, depending on the field content. This classification is different from the symmetry classification WC1, WC2R, ...

For each class N is the number of operators and E the number of independent entries. When E is not explicitly defined is understood $E=N$ (no flavour structure)

- static const int [NG](#) = 3
Number of fermion flavours.
- static const int [DF](#) = 9
Dimension of matrices in flavour space.
- static const int [DFs](#) = ([NG](#)*[NG](#) + [NG](#)) / 2
Independent entries of a $N_G \times N_G$ real symmetric matrix.
- static const int [DFa](#) = ([NG](#)*[NG](#) - [NG](#)) / 2
Independent entries of a $N_G \times N_G$ real anti-symmetric matrix.
- static const int [Ngauge](#) = 3
Number of gauge couplings.
- static const int [Nh](#) = 2
Number of Higgs' sector parameters.
- static const int [Nyukawa](#) = 3
Number of Yukawa matrices.
- static const int [Eyuk](#) = ([Nyukawa](#) * 2 * [DF](#))
Number of real parameters for each Yukawa matrix.
- static const int [N1](#) = 4
- static const int [N23](#) = 3
- static const int [N4](#) = 8
- static const int [N5](#) = 3
- static const int [E5](#) = ([N5](#) * 2 * [DF](#))
- static const int [N6](#) = 8
- static const int [E6](#) = ([N6](#) * 2 * [DF](#))
- static const int [N7](#) = 8
- static const int [N7H](#) = 7
Number of Hermitian operators in class 7.
- static const int [N7NH](#) = 1
Number of non-Hermitian operators in class 7.
- static const int [E7](#) = ([N7H](#)*([DWC2R](#) + [DWC2I](#)) + [N7NH](#) * 2 * [DF](#))
- static const int [N8_LLLL](#) = 5
- static const int [E8_LLLL](#) = 2 * ([DWC7R](#) + [DWC7I](#)) + 3 * ([DWC6R](#) + [DWC6I](#))
- static const int [N8_RRRR](#) = 7
- static const int [E8_RRRR](#) = 4 * ([DWC7R](#) + [DWC7I](#)) + 2 * ([DWC6R](#) + [DWC6I](#)) + 1 * ([DWC8R](#) + [DWC8I](#))
- static const int [N8_LLRR](#) = 8
- static const int [E8_LLRR](#) = 8 * ([DWC7R](#) + [DWC7I](#))

- static const int `N8_LRRL` = 1
- static const int `E8_LRRL` = 2 * `NG*NG*NG*NG*N8_LRRL`
- static const int `N8_LRLR` = 4
- static const int `E8_LRLR` = 2 * `NG*NG*NG*NG*N8_LRLR`

Fractions

Recurring fractions defined in order to increase efficiency

- static const double `TWO_THIRDS` = (2. / 3.)
- static const double `FOUR_THIRDS` = (4. / 3.)
- static const double `EIGHT_THIRDS` = (8. / 3.)
- static const double `ONE_THIRD` = (1. / 3.)
- static const double `ONE_SIXTH` = (1. / 6.)
- static const double `TEN_THIRDS` = (10. / 3.)

Miscellaneous parameters

- static const double `delta` [3][3]
Kroenecker delta in flavour space.
- static const int `dim` = (`Ngauge` + `Nh` + `Eyuk` + `N1` + `N23` + `N4` + `E5` + `E6` + `E7` + `E8_LLLL` + `E8_RRRR` + `E8_LLRR` + `E8_LRRL` + `E8_LRLR`)
Dimension of the system.
- static const double `NC` = 3.
Number of colors.
- static const double `NC2` = 9.
Number of colors squared.
- static const double `b01` = (- 1. / 6. - 3. * 20. / 9.)
Leading-order g_1 beta function (with g_1 normalized as usual and not as in GUT theories)
- static const double `b02` = (43. / 6. - 4. * 3. / 3.)
Leading-order g_2 beta function.
- static const double `b03` = (11. - 4. * 3. / 3.)
Leading-order g_3 beta function.
- static const double `cA2` = double(2.)
SU(2) adjoint Casimir
- static const double `cA3` = double(`NC`)
SU(3) adjoint Casimir
- static const double `cF2` = double(3. / 4.)
SU(2) fundamental Casimir
- static const double `cF3` = double(0.5 * (`NC*NC` - 1) / `NC`)
SU(3) fundamental Casimir

Hypercharges (and products of hypercharges)

We use $Q = T_3 + Y$, being T_3 the z-component of the weak isospin and Q the electric charge. The other possibility is $Q = T_3 + \frac{Y}{2}$ as in <https://arxiv.org/abs/1308.2627>

- static const double `Yh` = (0.5)
- static const double `Yh2` = `Yh*Yh`
- static const double `Yq` = (1. / 6.)
- static const double `Yq2` = `Yq*Yq`
- static const double `Yl` = (- 0.5)
- static const double `Yl2` = `Yl*Yl`
- static const double `Yu` = (2. / 3.)
- static const double `Yu2` = `Yu*Yu`
- static const double `Yd` = (- 1. / 3.)
- static const double `Yd2` = `Yd*Yd`
- static const double `Ye` = (- 1.)
- static const double `Ye2` = `Ye*Ye`
- static const double `YhYu` = `Yh*Yu`
- static const double `YhYd` = `Yh*Yd`

- static const double $Y_h Y_e = Y_h * Y_e$
- static const double $Y_h Y_q = Y_h * Y_q$
- static const double $Y_h Y_l = Y_h * Y_l$
- static const double $Y_u Y_d = Y_u * Y_d$
- static const double $Y_u Y_e = Y_u * Y_e$
- static const double $Y_u Y_q = Y_u * Y_q$
- static const double $Y_u Y_l = Y_u * Y_l$
- static const double $Y_d Y_e = Y_d * Y_e$
- static const double $Y_d Y_q = Y_d * Y_q$
- static const double $Y_d Y_l = Y_d * Y_l$
- static const double $Y_e Y_q = Y_e * Y_q$
- static const double $Y_e Y_l = Y_e * Y_l$
- static const double $Y_l Y_q = Y_l * Y_q$

3.1.1 Detailed Description

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

The class solves the Renormalization Group Equations (RGEs) both numerically and in the leading-log approximations. 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>.

The user must set separately real and imaginary part of each complex parameter.

In tables 3.1, 3.2, 3.3 and 3.4 are listed all the parameters, together with their name (that must be used to correctly invoke getter and setter functions).

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} , ϵ_{abs} and the integration step using the dedicated getter functions.

Author

HEPfit Collaboration

Copyright

GNU General Public License

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

Table 3.1 Standard Model parameters

Classes 1-3		Class 4	
Coefficient	Name	Coefficient	Name
C_G	CG	C_{HG}	CHG
$C_{\tilde{G}}$	CGtilde	$C_{H\tilde{G}}$	CHGtilde
C_W	CW	C_{HW}	CHW
$C_{\tilde{W}}$	CWtilde	$C_{H\tilde{W}}$	CHWtilde
C_H	CH	C_{HB}	CHB
$C_{H\Box}$	CHbox	$C_{H\tilde{B}}$	CHBtilde
C_{HD}	CHD	C_{HWB}	CHWB
		$C_{H\tilde{W}B}$	CHWtildeB

Table 3.2 Scalar (and real) SMEFT operators.

Class 5			Class 6			Class 7		
Coefficient	Name	Symmetry	Coefficient	Name	Symmetry	Coefficient	Name	Symmetry
$\Re(C_{eH})$	CeHR	WC1	$\Re(C_{eW})$	CeWR	WC1	$\Re(C_{H11})$	CH11R	WC2R
$\Im(C_{eH})$	CeHI	WC1	$\Im(C_{eW})$	CeWI	WC1	$\Im(C_{H11})$	CH11I	WC2I
$\Re(C_{uH})$	CuHR	WC1	$\Re(C_{eB})$	CeBR	WC1	$\Re(C_{H13})$	CH13R	WC2R
$\Im(C_{uH})$	CuHI	WC1	$\Im(C_{eB})$	CeBI	WC1	$\Im(C_{H13})$	CH13I	WC2I
$\Re(C_{dH})$	CdHR	WC1	$\Re(C_{uG})$	CuGR	WC1	$\Re(C_{He})$	CHeR	WC2R
$\Im(C_{dH})$	CdHI	WC1	$\Im(C_{uG})$	CuGI	WC1	$\Im(C_{He})$	CHeI	WC2I
			$\Re(C_{uW})$	CuWR	WC1	$\Re(C_{Hq1})$	CHq1R	WC2R
			$\Im(C_{uW})$	CuWI	WC1	$\Im(C_{Hq1})$	CHq1I	WC2I
			$\Re(C_{uB})$	CuBR	WC1	$\Re(C_{Hq3})$	CHq3R	WC2R
			$\Im(C_{uB})$	CuBI	WC1	$\Im(C_{Hq3})$	CHq3I	WC2I
			$\Re(C_{dG})$	CdGR	WC1	$\Re(C_{Hu})$	CHuR	WC2R
			$\Im(C_{dG})$	CdGI	WC1	$\Im(C_{Hu})$	CHuI	WC2I
			$\Re(C_{dW})$	CdWR	WC1	$\Re(C_{Hd})$	CHdR	WC2R
			$\Im(C_{dW})$	CdWI	WC1	$\Im(C_{Hd})$	CHdI	WC2I
			$\Re(C_{dB})$	CdBR	WC1	$\Re(C_{Hud})$	CHudR	WC1
			$\Im(C_{dB})$	CdBI	WC1	$\Im(C_{Hud})$	CHudI	WC1

Table 3.3 2F SMEFT operators

Class 8 ($\bar{L}L$)($\bar{L}L$)			Class 8 ($\bar{R}R$)($\bar{R}R$)			Class 8 ($\bar{L}L$)($\bar{R}R$)		
Coefficient	Name	Symmetry	Coefficient	Name	Symmetry	Coefficient	Name	Symmetry
$\Re(C_{ll})$	C11R	WC6R	$\Re(C_{ee})$	CeeR	WC8R	$\Re(C_{le})$	C1eR	WC7R
$\Im(C_{ll})$	C11I	WC6I	$\Im(C_{ee})$	CeeI	WC8I	$\Im(C_{le})$	C1eI	WC7I
$\Re(C_{qq1})$	Cqql1R	WC6R	$\Re(C_{uu})$	CuuR	WC6R	$\Re(C_{lu})$	C1uR	WC7R
$\Im(C_{qq1})$	Cqql1I	WC6I	$\Im(C_{uu})$	CuuI	WC6I	$\Im(C_{lu})$	C1uI	WC7I
$\Re(C_{qq3})$	Cqql3R	WC6R	$\Re(C_{dd})$	CddR	WC6R	$\Re(C_{ld})$	C1dR	WC7R
$\Im(C_{qq3})$	Cqql3I	WC6I	$\Im(C_{dd})$	CddI	WC6I	$\Im(C_{ld})$	C1dI	WC7I
$\Re(C_{lq1})$	C1q1R	WC7R	$\Re(C_{eu})$	CeuR	WC7R	$\Re(C_{qe})$	CqeR	WC7R
$\Im(C_{lq1})$	C1q1I	WC7I	$\Im(C_{eu})$	CeuI	WC7I	$\Im(C_{qe})$	CqeI	WC7I
$\Re(C_{lq3})$	C1q3R	WC7R	$\Re(C_{ed})$	CedR	WC7R	$\Re(C_{qu1})$	Cqu1R	WC7R
$\Im(C_{lq3})$	C1q3I	WC7I	$\Im(C_{ed})$	CedI	WC7I	$\Im(C_{qu1})$	Cqu1I	WC7I
Class 8 ($\bar{L}R$)($\bar{L}R$)			$\Re(C_{ud1})$	Cud1R	WC7R	$\Re(C_{qu8})$	Cqu8R	WC7R
Coefficient	Name	Symmetry	$\Im(C_{ud1})$	Cud1I	WC7I	$\Im(C_{qu8})$	Cqu8I	WC7I
$\Re(C_{quqd1})$	Cquqd1R	WC5	$\Re(C_{ud8})$	Cud8R	WC7R	$\Re(C_{qd1})$	Cqd1R	WC7R
$\Im(C_{quqd1})$	Cquqd1I	WC5	$\Im(C_{ud8})$	Cud8I	WC7I	$\Im(C_{qd1})$	Cqd1I	WC7I
$\Re(C_{quqd8})$	Cquqd8R	WC5	Class 8 ($\bar{L}R$)($\bar{R}L$)			$\Re(C_{qd8})$	Cqd8R	WC7R
$\Im(C_{quqd8})$	Cquqd8I	WC5	Coefficient	Name	Symmetry	$\Im(C_{qd8})$	Cqd8I	WC7I
$\Re(C_{lequ1})$	Clequ1R	WC5	$\Re(C_{ledq})$	CledqR	WC5			
$\Im(C_{lequ1})$	Clequ1I	WC5	$\Im(C_{ledq})$	CledqI	WC5			
$\Re(C_{lequ3})$	Clequ3R	WC5						
$\Im(C_{lequ3})$	Clequ3I	WC5						

Table 3.4 4F SMEFT Operators.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 RGESolver()

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

The default constructor.

3.1.2.2 ~RGESolver()

```
RGESolver::~~RGESolver ( ) [inline]
```

The default destructor.

3.1.3 Member Function Documentation

3.1.3.1 epsabs()

```
double RGESolver::epsabs ( ) [inline]
```

Getter for the absolute error used in the numerical integration.

3.1.3.2 epsrel()

```
double RGESolver::epsrel ( ) [inline]
```

Getter for the relative error used in the numerical integration.

3.1.3.3 Evolve()

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

Command to perform the RGE evolution.

RGEs are solved with the chosen method from `muI` to `muF`. Currently, the available methods are "Numeric" and "Leading-Log".

The evolver takes as initial values the current values of the parameters, set with the `SetCoefficient(...)` function. After completing the evolution the values of the parameters are updated and are accessible with the `GetCoefficient(...)` function.

Parameters

<i>method</i>	resolution method
<i>muI</i>	initial energy scale
<i>muF</i>	final energy scale

3.1.3.4 EvolveSMOnly()

```
void RGESolver::EvolveSMOnly (
    std::string method,
```

```
double muI,
double muF )
```

Same as [Evolve\(std::string method, double muI, double muF\)](#), but only for the SM parameters. The user should use this method instead of `Evolve` when interested in pure SM running.

Parameters

<i>method</i>	
<i>muI</i>	
<i>muF</i>	

3.1.3.5 ExtractParametersFromCKM()

```
void RGESolver::ExtractParametersFromCKM ( ) [private]
```

3.1.3.6 FromMassesToYukawas()

```
void RGESolver::FromMassesToYukawas (
    std::string basis ) [private]
```

3.1.3.7 func()

```
int RGESolver::func (
    double logmu,
    const double y[],
    double f[],
    void * params ) [static], [private]
```

Computes the beta functions for the SMEFT.

Parameters

<i>logmu</i>	value of the logarithm of the energy scale at which the beta functions are computed
<i>y</i>	1D array in which are stored the current values of the parameters
<i>f</i>	1D array in which the beta functions for each parameters are saved
<i>params</i>	eventual additional parameters (not used)

Returns

GSL_SUCCESS

3.1.3.8 funcSMOnly()

```
static int RGESolver::funcSMOnly (
    double logmu,
    const double y[],
    double f[],
    void * params ) [static], [private]
```

3.1.3.9 GenerateSMInitialConditions()

```
void RGESolver::GenerateSMInitialConditions (
    double mu,
    std::string basis,
    std::string method,
    bool inputCKM = true )
```

Generates the initial conditions for Standard Model's parameters (gauge couplings, Yukawa coupling, quartic coupling and Higgs' boson mass).

Parameters

<i>mu</i>	Scale (in GeV) at which the initial conditions are generated. If <i>mu</i> is different from the scale at which the input is given, RGESolver will use the pure SM RGEs to run the parameters to the scale <i>mu</i> .
<i>basis</i>	Flavour basis ("UP" or "DOWN")
<i>method</i>	Method used by RGESolver to run the SM parameters to the scale <i>mu</i> ("Numeric" or "Leading-Log")
<i>inputCKM</i>	If set to true (default), the input for the Yukawa matrices will be generated from the current value of the CKM matrix and the masses of the fermions (default).

3.1.3.10 GetCoefficient() [1/3]

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

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

If the parameter name does not match with any of the parameters, an error message is printed and the value 0 is returned.

Parameters

<i>name</i>	name of the parameter
-------------	-----------------------

Returns

the requested parameter (if it exists), otherwise returns 0.

3.1.3.11 GetCoefficient() [2/3]

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

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

If the parameter name does not match with any of the parameters or 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

<i>name</i>	name of the parameter
<i>i</i>	first flavour index
<i>j</i>	second flavour index

Returns

the requested parameter (if it exists), otherwise returns 0.

3.1.3.12 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).

If the parameter name does not match with any of the parameters or 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

<i>name</i>	name of the parameter
<i>i</i>	first flavour index
<i>j</i>	second flavour index
<i>k</i>	third flavour index
<i>l</i>	fourth flavour index

Returns

the requested parameter (if it exists), otherwise returns 0.

3.1.3.13 GetSMInputScale()

```
double RGESolver::GetSMInputScale ( ) [inline]
```

Getter method for the scale at which the method `GenerateSMInitialConditions` takes the input values for SM parameters.

3.1.3.14 GoToBasisSMOnly()

```
void RGESolver::GoToBasisSMOnly (
    std::string basis ) [private]
```

3.1.3.15 Init()

```
void RGESolver::Init ( ) [private]
```

Inserts the initial values of the parameters in the array `x`.

Only used in `Evolve`

3.1.3.16 InitSMOnly()

```
void RGESolver::InitSMOnly ( ) [private]
```

3.1.3.17 Print()

```
void RGESolver::Print (
    double * c,
    std::string name,
    std::string sym,
    std::string format,
    std::ofstream & f ) [inline], [static], [private]
```

Prints on file the coefficient `c`.

This function is used only in the function `SaveOutputFile`. Currently only "SLHA" printing for WC1,WC2R/I, WC5,WC6R/I,WC7R/I,WC8R/I is implemented.

Parameters

<i>c</i>	coefficient
<i>name</i>	printed name
<i>sym</i>	symmetry category of the operator
<i>format</i>	chosen format
<i>f</i>	pointer to file

3.1.3.18 Reset()

```
void RGESolver::Reset ( )
```

Resets all the parameters to 0.

3.1.3.19 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

<i>filename</i>	Name of the output file
<i>format</i>	Format of the output file

3.1.3.20 Set_epsabs()

```
void RGESolver::Set_epsabs (
    double epsabs ) [inline]
```

Setter for the absolute error used in the numerical integration.

3.1.3.21 Set_epsrel()

```
void RGESolver::Set_epsrel (
    double epsrel ) [inline]
```

Setter for the relative error used in the numerical integration.

3.1.3.22 Set_step()

```
void RGESolver::Set_step (
    double step ) [inline]
```

Setter for the step used in the numerical integration.

3.1.3.23 SetCoefficient() [1/3]

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

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

If the parameter name does not match with any of the parameters, an error message is printed and no assignation is performed.

Parameters

<i>name</i>	name of the parameter
<i>val</i>	its value

3.1.3.24 SetCoefficient() [2/3]

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

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

If the parameter name does not match with any of the parameters or if at least one of the inserted indices is outside the [0:2] range,

Parameters

<i>name</i>	name of the parameter
<i>val</i>	its value
<i>i</i>	first flavour index
<i>j</i>	second flavour index

3.1.3.25 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 the parameter name does not match with any of the parameters or 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

<i>name</i>	name of the parameter
<i>val</i>	its value
<i>i</i>	first flavour index
<i>j</i>	second flavour index
<i>k</i>	third flavour index
<i>l</i>	fourth flavour index

3.1.3.26 SetSMInputScale()

```
void RGESolver::SetSMInputScale (
    double mu ) [inline]
```

Setter method for the scale at which the method `GenerateSMInitialConditions` takes the input values for SM parameters.

Parameters

<i>mu</i>	
-----------	--

3.1.3.27 step()

```
double RGESolver::step ( ) [inline]
```

Getter for the step used in the numerical integration.

3.1.3.28 Update()

```
void RGESolver::Update ( ) [private]
```

Saves the evolved values of the coefficients from `x`.

Only used in `Evolve`

3.1.3.29 UpdateCKM()

```
void RGESolver::UpdateCKM ( ) [private]
```

3.1.3.30 UpdateSOnly()

```
void RGESolver::UpdateSOnly ( ) [private]
```

3.1.3.31 WC1()

```
double RGESolver::WC1 (
    double * c,
    int i,
    int j ) [inline], [static], [private]
```

3.1.3.32 WC1_set()

```
void RGESolver::WC1_set (
    double * c,
    int i,
    int j,
    double val ) [inline], [static], [private]
```

3.1.3.33 WC2I()

```
double RGESolver::WC2I (
    double * c,
    int i,
    int j ) [inline], [static], [private]
```

3.1.3.34 WC2I_set()

```
void RGESolver::WC2I_set (
    double * c,
    int i,
    int j,
    double val ) [inline], [static], [private]
```

3.1.3.35 WC2R()

```
double RGESolver::WC2R (
    double * c,
    int i,
    int j ) [inline], [static], [private]
```

3.1.3.36 WC2R_set()

```
void RGESolver::WC2R_set (
    double * c,
    int i,
    int j,
    double val ) [inline], [static], [private]
```

3.1.3.37 WC3()

```
double RGESolver::WC3 (
    double * c,
    int i,
    int j ) [inline], [static], [private]
```

3.1.3.38 WC3_set()

```
void RGESolver::WC3_set (
    double * c,
    int i,
    int j,
    double val ) [inline], [static], [private]
```

3.1.3.39 WC5()

```
double RGESolver::WC5 (
    double * c,
    int i,
    int j,
    int k,
    int l ) [inline], [static], [private]
```

3.1.3.40 WC5_set()

```
void RGESolver::WC5_set (
    double * c,
    int i,
    int j,
    int k,
    int l,
    double val ) [inline], [static], [private]
```

3.1.3.41 WC6I()

```
double RGESolver::WC6I (
    double * c,
    int i,
    int j,
    int k,
    int l ) [inline], [static], [private]
```

3.1.3.42 WC6I_set()

```
void RGESolver::WC6I_set (
    double * c,
    int i,
    int j,
    int k,
    int l,
    double val ) [inline], [static], [private]
```

3.1.3.43 WC6R()

```
double RGESolver::WC6R (
    double * c,
    int i,
    int j,
    int k,
    int l ) [inline], [static], [private]
```

3.1.3.44 WC6R_set()

```
void RGESolver::WC6R_set (
    double * c,
    int i,
    int j,
    int k,
    int l,
    double val ) [inline], [static], [private]
```

3.1.3.45 WC7I()

```
double RGESolver::WC7I (
    double * c,
    int i,
    int j,
    int k,
    int l ) [inline], [static], [private]
```

3.1.3.46 WC7I_set()

```
void RGESolver::WC7I_set (
    double * c,
    int i,
    int j,
    int k,
    int l,
    double val ) [inline], [static], [private]
```

3.1.3.47 WC7R()

```
double RGESolver::WC7R (
    double * c,
    int i,
    int j,
    int k,
    int l ) [inline], [static], [private]
```

3.1.3.48 WC7R_set()

```
void RGESolver::WC7R_set (
    double * c,
    int i,
    int j,
    int k,
    int l,
    double val ) [inline], [static], [private]
```

3.1.3.49 WC8I()

```
double RGESolver::WC8I (
    double * c,
    int i,
    int j,
    int k,
    int l ) [inline], [static], [private]
```

3.1.3.50 WC8I_set()

```
void RGESolver::WC8I_set (
    double * c,
    int i,
    int j,
    int k,
    int l,
    double val ) [inline], [static], [private]
```

3.1.3.51 WC8R()

```
double RGESolver::WC8R (
    double * c,
    int i,
    int j,
    int k,
    int l ) [inline], [static], [private]
```

3.1.3.52 WC8R_set()

```
void RGESolver::WC8R_set (
    double * c,
    int i,
    int j,
    int k,
    int l,
    double val ) [inline], [static], [private]
```

3.1.3.53 Yukawa()

```
double RGESolver::Yukawa (
    double y[3][3],
    int i,
    int j ) [inline], [static], [private]
```


3.1.3.54 Yukawa_set()

```
void RGESolver::Yukawa_set (
    double y[3][3],
    int i,
    int j,
    double val ) [inline], [static], [private]
```

3.1.4 Member Data Documentation**3.1.4.1 b01**

```
const double RGESolver::b01 = (- 1. / 6. - 3. * 20. / 9.) [static], [private]
```

Leading-order g_1 beta function (with g_1 normalized as usual and not as in GUT theories)

3.1.4.2 b02

```
const double RGESolver::b02 = (43. / 6. - 4. * 3. / 3.) [static], [private]
```

Leading-order g_2 beta function.

3.1.4.3 b03

```
const double RGESolver::b03 = (11. - 4. * 3. / 3.) [static], [private]
```

Leading-order g_3 beta function.

3.1.4.4 c12

```
double RGESolver::c12 [private]
```

$\cos \theta_{12}$

3.1.4.5 c13

```
double RGESolver::c13 [private]
```

$\cos \theta_{13}$

3.1.4.6 c23

```
double RGESolver::c23 [private]
```

$\cos \theta_{23}$

3.1.4.7 cA2

```
const double RGESolver::cA2 = double(2.) [static], [private]
```

$SU(2)$ adjoint Casimir

3.1.4.8 cA3

```
const double RGESolver::cA3 = double(NC) [static], [private]
```

$SU(3)$ adjoint Casimir

3.1.4.9 cdBI

```
double RGESolver::cdBI[3 *3] [private]
```

$\Im[C_{dW}]$ (class 6, WC1)

3.1.4.10 cdBR

```
double RGESolver::cdBR[3 *3] = {0.} [private]
```

$\Re[C_{dB}]$ (class 6, WC1)

3.1.4.11 cddI

```
double RGESolver::cddI[3 * 3 * 3 * 3] = {0.} [private]
```

$\Im[C_{dd}]$ (class 8-[RR][RR], WC6I)

3.1.4.12 cddR

```
double RGESolver::cddR[3 * 3 * 3 * 3] = {0.} [private]
```

$\Re[C_{dd}]$ (class 8-[RR][RR], WC6R)

3.1.4.13 cdGI

```
double RGESolver::cdGI[3 * 3] = {0.} [private]
```

$\Im[C_{dG}]$ (class 6, WC1)

3.1.4.14 cdGR

```
double RGESolver::cdGR[3 * 3] = {0.} [private]
```

$\Re[C_{dG}]$ (class 6, WC1)

3.1.4.15 cdHI

```
double RGESolver::cdHI[3 * 3] = {0.} [private]
```

$\Im[C_{dH}]$ (class 5, WC1)

3.1.4.16 cdHR

```
double RGESolver::cdHR[3 * 3] = {0.} [private]
```

$\Re[C_{dH}]$ (class 5, WC1)

3.1.4.17 cdWI

```
double RGESolver::cdWI[3 * 3] = {0.} [private]
```

$\Im[C_{dW}]$ (class 6, WC1)

3.1.4.18 cdWR

```
double RGESolver::cdWR[3 * 3] = {0.} [private]
```

$\Re[C_{dW}]$ (class 6, WC1)

3.1.4.19 ceBI

```
double RGESolver::ceBI[3 * 3] = {0.} [private]
```

$\Im[C_{eB}]$ (class 6, WC1)

3.1.4.20 ceBR

```
double RGESolver::ceBR[3 * 3] = {0.} [private]
```

$\Re[C_{eB}]$ (class 6, WC1)

3.1.4.21 cedI

```
double RGESolver::cedI[3 * 3 * 3 * 3] = {0.} [private]
```

$\Im[C_{ed}]$ (class 8-[RR][RR], WC7I)

3.1.4.22 cedR

```
double RGESolver::cedR[3 * 3 * 3 * 3] = {0.} [private]
```

$\Re[C_{ed}]$ (class 8-[RR][RR], WC7R)

3.1.4.23 ceel

```
double RGESolver::ceel[3 * 3 * 3 * 3] = {0.} [private]
```

$\Im[C_{ee}]$ (class 8-[RR][RR], WC8I)

3.1.4.24 ceer

```
double RGESolver::ceer[3 * 3 * 3 * 3] = {0.} [private]
```

$\Re[C_{ee}]$ (class 8-[RR][RR], WC8R)

3.1.4.25 ceHI

```
double RGESolver::ceHI[3 * 3] = {0.} [private]
```

$\Im[C_{eH}]$ (class 5, WC1)

3.1.4.26 ceHR

```
double RGESolver::ceHR[3 * 3] = {0.} [private]
```

$\Re[C_{eH}]$ (class 5, WC1)

3.1.4.27 ceul

```
double RGESolver::ceul[3 * 3 * 3 * 3] = {0.} [private]
```

$\Im[C_{eu}]$ (class 8-[RR][RR], WC7I)

3.1.4.28 ceuR

```
double RGESolver::ceuR[3 * 3 * 3 * 3] = {0.} [private]
```

$\Re[C_{eu}]$ (class 8-[RR][RR], WC7R)

3.1.4.29 ceWI

```
double RGESolver::ceWI[3 * 3] = {0.} [private]
```

$\Im[C_{eW}]$ (class 6, WC1)

3.1.4.30 ceWR

```
double RGESolver::ceWR[3 * 3] = {0.} [private]
```

$\Re[C_{eW}]$ (class 6, WC1)

3.1.4.31 cF2

```
const double RGESolver::cF2 = double(3. / 4.) [static], [private]
```

$SU(2)$ fundamental Casimir

3.1.4.32 cF3

```
const double RGESolver::cF3 = double(0.5 * (NC*NC - 1) / NC) [static], [private]
```

$SU(3)$ fundamental Casimir

3.1.4.33 cG

```
double RGESolver::cG = 0. [private]
```

C_G (class 1, scalar)

3.1.4.34 cGT

```
double RGESolver::cGT = 0. [private]
```

$C_{\tilde{G}}$ (class 1, scalar)

3.1.4.35 cH

```
double RGESolver::cH = 0. [private]
```

C_H (class 2, scalar)

3.1.4.36 cHB

```
double RGESolver::cHB = 0. [private]
```

C_{HB} (class 4, scalar)

3.1.4.37 cHBOX

```
double RGESolver::cHBOX = 0. [private]
```

$C_{H\Box}$ (class 3, scalar)

3.1.4.38 cHBT

```
double RGESolver::cHBT = 0. [private]
```

$C_{H\bar{B}}$ (class 4, scalar)

3.1.4.39 cHD

```
double RGESolver::cHD = 0. [private]
```

C_{HD} (class 3, scalar)

3.1.4.40 cHdI

```
double RGESolver::cHdI[3 * 3] = {0.} [private]
```

$\Im[C_{Hd}]$ (class 7, WC2I)

3.1.4.41 cHdR

```
double RGESolver::cHdR[3 * 3] = {0.} [private]
```

$\Re[C_{Hd}]$ (class 7, WC2R)

3.1.4.42 cHeI

```
double RGESolver::cHeI[3 * 3] = {0.} [private]
```

$\Im[C_{He}]$ (class 7, WC2I)

3.1.4.43 cHeR

```
double RGESolver::cHeR[3 * 3] = {0.} [private]
```

$\Re[C_{He}]$ (class 7, WC2R)

3.1.4.44 cHG

```
double RGESolver::cHG = 0. [private]
```

C_{HG} (class 4, scalar)

3.1.4.45 cHGT

```
double RGESolver::cHGT = 0. [private]
```

$C_{H\tilde{G}}$ (class 4, scalar)

3.1.4.46 cHlI

```
double RGESolver::cHlI[3 * 3] = {0.} [private]
```

$\Im[C_{Hl}]$ (class 7, WC2I)

3.1.4.47 cH11R

```
double RGESolver::cH11R[3 * 3] = {0.} [private]
```

$\Re[C_{H11}]$ (class 7, WC2R)

3.1.4.48 cH13I

```
double RGESolver::cH13I[3 * 3] = {0.} [private]
```

$\Im[C_{H13}]$ (class 7, WC2I)

3.1.4.49 cH13R

```
double RGESolver::cH13R[3 * 3] = {0.} [private]
```

$\Re[C_{H13}]$ (class 7, WC2R)

3.1.4.50 cHq1I

```
double RGESolver::cHq1I[3 * 3] = {0.} [private]
```

$\Im[C_{Hq1}]$ (class 7, WC2I)

3.1.4.51 cHq1R

```
double RGESolver::cHq1R[3 * 3] = {0.} [private]
```

$\Re[C_{Hq1}]$ (class 7, WC2R)

3.1.4.52 cHq3I

```
double RGESolver::cHq3I[3 * 3] = {0.} [private]
```

$\Im[C_{Hq3}]$ (class 7, WC2I)

3.1.4.53 cHq3R

```
double RGESolver::cHq3R[3 * 3] = {0.} [private]
```

$\Re[C_{Hq3}]$ (class 7, WC2R)

3.1.4.54 cHudI

```
double RGESolver::cHudI[3 * 3] = {0.} [private]
```

$\Im[C_{Hud}]$ (class 7, WC1)

3.1.4.55 cHudR

```
double RGESolver::cHudR[3 * 3] = {0.} [private]
```

$\Re[C_{Hud}]$ (class 7, WC1)

3.1.4.56 cHuI

```
double RGESolver::cHuI[3 * 3] = {0.} [private]
```

$\Im[C_{Hu}]$ (class 7, WC2I)

3.1.4.57 cHuR

```
double RGESolver::cHuR[3 * 3] = {0.} [private]
```

$\Re[C_{Hu}]$ (class 7, WC2R)

3.1.4.58 cHW

```
double RGESolver::cHW = 0. [private]
```

C_{HW} (class 4, scalar)

3.1.4.59 cHWB

```
double RGESolver::cHWB = 0. [private]
```

C_{HWB} (class 4, scalar)

3.1.4.60 cHWBT

```
double RGESolver::cHWBT = 0. [private]
```

$C_{H\bar{W}B}$ (class 4, scalar)

3.1.4.61 cHWT

```
double RGESolver::cHWT = 0. [private]
```

$C_{H\bar{W}}$ (class 4, scalar)

3.1.4.62 CKM

```
gslpp::matrix<gslpp::complex> RGESolver::CKM = gslpp::matrix<gslpp::complex>(3, 3, 0.) [private]
```

The CKM matrix.

3.1.4.63 CKM_delta

```
double RGESolver::CKM_delta = 3.14 / 4. [private]
```

3.1.4.64 CKM_theta12

```
double RGESolver::CKM_theta12 = 0.2 [private]
```

θ_{12} of the CKM matrix (in radians). The default value is ???

3.1.4.65 CKM_theta13

```
double RGESolver::CKM_theta13 = 0.1 [private]
```

3.1.4.66 CKM_theta23

```
double RGESolver::CKM_theta23 = 0.3 [private]
```

3.1.4.67 cldI

```
double RGESolver::cldI[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{ld}]$ (class 8-[LL][RR], WC7I)

3.1.4.68 cldR

```
double RGESolver::cldR[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{ld}]$ (class 8-[LL][RR], WC7R)

3.1.4.69 cledqI

```
double RGESolver::cledqI[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{ledq}]$ (class 8-[LR][RL], WC5)

3.1.4.70 cledqR

```
double RGESolver::cledqR[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{ledq}]$ (class 8-[LR][RL], WC5)

3.1.4.71 cleI

```
double RGESolver::cleI[3 * 3 * 3] = {0.} [private]
```

 $\Im [C_{le}]$ (class 8-[LL][RR], WC7I)
3.1.4.72 clequ1I

```
double RGESolver::clequ1I[3 * 3 * 3] = {0.} [private]
```

 $\Im [C_{lequ1}]$ (class 8-[LR][LR], WC5)
3.1.4.73 clequ1R

```
double RGESolver::clequ1R[3 * 3 * 3] = {0.} [private]
```

 $\Re [C_{lequ1}]$ (class 8-[LR][LR], WC5)
3.1.4.74 clequ3I

```
double RGESolver::clequ3I[3 * 3 * 3] = {0.} [private]
```

 $\Im [C_{lequ3}]$ (class 8-[LR][LR], WC5)
3.1.4.75 clequ3R

```
double RGESolver::clequ3R[3 * 3 * 3] = {0.} [private]
```

 $\Re [C_{lequ3}]$ (class 8-[LR][LR], WC5)
3.1.4.76 cleR

```
double RGESolver::cleR[3 * 3 * 3] = {0.} [private]
```

 $\Re [C_{le}]$ (class 8-[LL][RR], WC7R)

3.1.4.77 cII

```
double RGESolver::cII[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{II}]$ (class 8-[LL][LL], WC6I)

3.1.4.78 cIIR

```
double RGESolver::cIIR[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{II}]$ (class 8-[LL][LL], WC6R)

3.1.4.79 clq1I

```
double RGESolver::clq1I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{Iq1}]$ (class 8-[LL][LL], WC7I)

3.1.4.80 clq1R

```
double RGESolver::clq1R[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{Iq1}]$ (class 8-[LL][LL], WC7R)

3.1.4.81 clq3I

```
double RGESolver::clq3I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{Iq3}]$ (class 8-[LL][LL], WC7I)

3.1.4.82 clq3R

```
double RGESolver::clq3R[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{Iq3}]$ (class 8-[LL][LL], WC7R)

3.1.4.83 cluI

```
double RGESolver::cluI[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{lu}]$ (class 8-[LL][RR], WC7I)

3.1.4.84 cluR

```
double RGESolver::cluR[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{lu}]$ (class 8-[LL][RR], WC7R)

3.1.4.85 con_

```
gsl_odeiv2_control* RGESolver::con_ [private]
```

Initial value:

```
= gsl_odeiv2_control_standard_new(
    epsabs_, epsrel_, 1, 1)
```

3.1.4.86 cqd1I

```
double RGESolver::cqd1I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{qd1}]$ (class 8-[LL][RR], WC7I)

3.1.4.87 cqd1R

```
double RGESolver::cqd1R[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{qd1}]$ (class 8-[LL][RR], WC7R)

3.1.4.88 cqd8I

```
double RGESolver::cqd8I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{qd8}]$ (class 8-[LL][RR], WC7I)

3.1.4.89 cqd8R

```
double RGESolver::cqd8R[3 * 3 * 3 * 3] = {0.} [private]
```

 $\Re[C_{qd8}]$ (class 8-[LL][RR], WC7R)
3.1.4.90 cqel

```
double RGESolver::cqelI[3 * 3 * 3 * 3] = {0.} [private]
```

 $\Im[C_{qe}]$ (class 8-[LL][RR], WC7I)
3.1.4.91 cqeR

```
double RGESolver::cqeR[3 * 3 * 3 * 3] = {0.} [private]
```

 $\Re[C_{qe}]$ (class 8-[LL][RR], WC7R)
3.1.4.92 cqqlI

```
double RGESolver::cqqlI[3 * 3 * 3 * 3] = {0.} [private]
```

 $\Im[C_{qq1}]$ (class 8-[LL][LL], WC6I)
3.1.4.93 cqqlR

```
double RGESolver::cqqlR[3 * 3 * 3 * 3] = {0.} [private]
```

 $\Re[C_{qq1}]$ (class 8-[LL][LL], WC6R)
3.1.4.94 cqq3I

```
double RGESolver::cqq3I[3 * 3 * 3 * 3] = {0.} [private]
```

 $\Im[C_{qq3}]$ (class 8-[LL][LL], WC6I)

3.1.4.95 cq3R

```
double RGESolver::cq3R[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{qq3}]$ (class 8-[LL][LL], WC6R)

3.1.4.96 qu1I

```
double RGESolver::qu1I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{qu1}]$ (class 8-[LL][RR], WC7I)

3.1.4.97 qu1R

```
double RGESolver::qu1R[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{qu1}]$ (class 8-[LL][RR], WC7R)

3.1.4.98 qu8I

```
double RGESolver::qu8I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{qu8}]$ (class 8-[LL][RR], WC7I)

3.1.4.99 qu8R

```
double RGESolver::qu8R[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{qu8}]$ (class 8-[LL][RR], WC7R)

3.1.4.100 quqd1I

```
double RGESolver::quqd1I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{quqd1}]$ (class 8-[LR][LR], WC5)

3.1.4.101 cquqd1R

```
double RGESolver::cquqd1R[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{quqd1}]$ (class 8-[LR][LR], WC5)

3.1.4.102 cquqd8I

```
double RGESolver::cquqd8I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{quqd8}]$ (class 8-[LR][LR], WC5)

3.1.4.103 cquqd8R

```
double RGESolver::cquqd8R[3 * 3 * 3] = {0.} [private]
```

$\Re[C_{quqd8}]$ (class 8-[LR][LR], WC5)

3.1.4.104 cuBI

```
double RGESolver::cuBI[3 * 3] = {0.} [private]
```

$\Im[C_{uB}]$ (class 6, WC1)

3.1.4.105 cuBR

```
double RGESolver::cuBR[3 * 3] = {0.} [private]
```

$\Re[C_{uB}]$ (class 6, WC1)

3.1.4.106 cud1I

```
double RGESolver::cud1I[3 * 3 * 3] = {0.} [private]
```

$\Im[C_{ud1}]$ (class 8-[RR][RR], WC7I)

3.1.4.107 cud1R

```
double RGESolver::cud1R[3 * 3 * 3] = {0.} [private]
```

 $\Re[C_{ud1}]$ (class 8-[RR][RR], WC7R)
3.1.4.108 cud8I

```
double RGESolver::cud8I[3 * 3 * 3] = {0.} [private]
```

 $\Im[C_{ud1}]$ (class 8-[RR][RR], WC7I)
3.1.4.109 cud8R

```
double RGESolver::cud8R[3 * 3 * 3] = {0.} [private]
```

 $\Re[C_{ud8}]$ (class 8-[RR][RR], WC7R)
3.1.4.110 cuGI

```
double RGESolver::cuGI[3 * 3] = {0.} [private]
```

 $\Im[C_{uG}]$ (class 6, WC1)
3.1.4.111 cuGR

```
double RGESolver::cuGR[3 * 3] = {0.} [private]
```

 $\Re[C_{uG}]$ (class 6, WC1)
3.1.4.112 cuHI

```
double RGESolver::cuHI[3 * 3] = {0.} [private]
```

 $\Im[C_{uH}]$ (class 5, WC1)

3.1.4.113 cuHR

```
double RGESolver::cuHR[3 * 3] = {0.} [private]
```

$\Re[C_{uH}]$ (class 5, WC1)

3.1.4.114 cuuI

```
double RGESolver::cuuI[3 * 3 * 3 * 3] = {0.} [private]
```

$\Im[C_{uu}]$ (class 8-[RR][RR], WC6I)

3.1.4.115 cuuR

```
double RGESolver::cuuR[3 * 3 * 3 * 3] = {0.} [private]
```

$\Re[C_{uu}]$ (class 8-[RR][RR], WC6R)

3.1.4.116 cuWI

```
double RGESolver::cuWI[3 * 3] = {0.} [private]
```

$\Im[C_{uW}]$ (class 6, WC1)

3.1.4.117 cuWR

```
double RGESolver::cuWR[3 * 3] = {0.} [private]
```

$\Re[C_{uW}]$ (class 6, WC1)

3.1.4.118 cW

```
double RGESolver::cW = 0. [private]
```

C_W (class 1, scalar)

3.1.4.119 cWT

```
double RGESolver::cWT = 0. [private]
```

C_W (class 1, scalar)

3.1.4.120 delta

```
const double RGESolver::delta [static], [private]
```

Initial value:

```
= {
    {1., 0., 0.},
    { 0., 1., 0.},
    { 0., 0., 1.}
}
```

Kronecker delta in flavour space.

3.1.4.121 DF

```
const int RGESolver::DF = 9 [static], [private]
```

Dimension of matrices in flavour space.

3.1.4.122 DFa

```
const int RGESolver::DFa = (NG*NG - NG) / 2 [static], [private]
```

Independent entries of a $N_G \times N_G$ real anti-symmetric matrix.

3.1.4.123 DFs

```
const int RGESolver::DFs = (NG*NG + NG) / 2 [static], [private]
```

Independent entries of a $N_G \times N_G$ real symmetric matrix.

3.1.4.124 dim

```
const int RGESolver::dim = (Ngauge + Nh + Eyuk + N1 + N23 + N4 + E5 + E6 + E7 + E8_LLLL +  
E8_RRRR + E8_LLRR + E8_LRRL + E8_LRLR) [static], [private]
```

Dimension of the system.

3.1.4.125 DWC2I

```
const int RGESolver::DWC2I = 3 [static], [private]
```

Number of independent entries of the imaginary part of operators in symmetry class WC2.

3.1.4.126 DWC2R

```
const int RGESolver::DWC2R = 6 [static], [private]
```

Number of independent entries of the real part of operators in symmetry class WC2.

3.1.4.127 DWC6I

```
const int RGESolver::DWC6I = 18 [static], [private]
```

Number of independent entries of the imaginary part of operators in symmetry class WC6.

3.1.4.128 DWC6R

```
const int RGESolver::DWC6R = 27 [static], [private]
```

Number of independent entries of the real part of operators in symmetry class WC6.

3.1.4.129 DWC7I

```
const int RGESolver::DWC7I = 36 [static], [private]
```

Number of independent entries of the imaginary part of operators in symmetry class WC7.

3.1.4.130 DWC7R

```
const int RGESolver::DWC7R = 45 [static], [private]
```

Number of independent entries of the real part of operators in symmetry class WC7.

3.1.4.131 DWC8I

```
const int RGESolver::DWC8I = 15 [static], [private]
```

Number of independent entries of the imaginary part of operators in symmetry class WC8.

3.1.4.132 DWC8R

```
const int RGESolver::DWC8R = 21 [static], [private]
```

Number of independent entries of the real part of operators in symmetry class WC8.

3.1.4.133 E5

```
const int RGESolver::E5 = (N5 * 2 * DF) [static], [private]
```

3.1.4.134 E6

```
const int RGESolver::E6 = (N6 * 2 * DF) [static], [private]
```

3.1.4.135 E7

```
const int RGESolver::E7 = (N7H*(DWC2R + DWC2I) + N7NH * 2 * DF) [static], [private]
```

3.1.4.136 E8_LLLL

```
const int RGESolver::E8_LLLL = 2 * (DWC7R + DWC7I) + 3 * (DWC6R + DWC6I) [static], [private]
```

3.1.4.137 E8_LLRR

```
const int RGESolver::E8_LLRR = 8 * (DWC7R + DWC7I) [static], [private]
```

3.1.4.138 E8_LRLR

```
const int RGESolver::E8_LRLR = 2 * NG*NG*NG*NG*N8_LRLR [static], [private]
```

3.1.4.139 E8_LRRL

```
const int RGESolver::E8_LRRL = 2 * NG*NG*NG*NG*N8_LRRL [static], [private]
```

3.1.4.140 E8_RRRR

```
const int RGESolver::E8_RRRR = 4 * (DWC7R + DWC7I) + 2 * (DWC6R + DWC6I) + 1 * (DWC8R + DWC8I)
[static], [private]
```

3.1.4.141 EIGHT_THIRDS

```
const double RGESolver::EIGHT_THIRDS = (8. / 3.) [static], [private]
```

3.1.4.142 epsabs_

```
double RGESolver::epsabs_ = 0.0001 [private]
```

Absolute error used in the integrator with its default value.

3.1.4.143 epsrel_

```
double RGESolver::epsrel_ = 0.0000000000000001 [private]
```

Relative error used in the integrator with its default value.

().

3.1.4.144 evo_

```
gsl_odeiv2_evolve* RGESolver::evo_ = gsl_odeiv2_evolve_alloc(2558) [private]
```

3.1.4.145 Eyuk

```
const int RGESolver::Eyuk = (Nyukawa * 2 * DF) [static], [private]
```

Number of real parameters for each Yukawa matrix.

3.1.4.146 FOUR_THIRDS

```
const double RGESolver::FOUR_THIRDS = (4. / 3.) [static], [private]
```

3.1.4.147 g1

```
double RGESolver::g1 [private]
```

g_1

3.1.4.148 g2

```
double RGESolver::g2 [private]
```

g_2

3.1.4.149 g3

```
double RGESolver::g3 [private]
```

g_3

3.1.4.150 Getter2F

```
std::unordered_map<std::string, boost::function<double(int, int) > > RGESolver::Getter2F  
[private]
```

3.1.4.151 Getter4F

```
std::unordered_map<std::string, boost::function<double(int, int, int, int) > > RGESolver::↵  
Getter4F [private]
```

3.1.4.152 InputScale_SM

```
double RGESolver::InputScale_SM = 91. [private]
```

the scale at which the method `GenerateSMInitialConditions` takes the input values for SM parameters.

3.1.4.153 lambda

```
double RGESolver::lambda = 0.2 [private]
```

λ (Higgs quartic coupling)

See <https://arxiv.org/pdf/1308.2627.pdf> for the normalization

3.1.4.154 mb

```
double RGESolver::mb = 5.2 [private]
```

3.1.4.155 mc

```
double RGESolver::mc = 1.2 [private]
```

3.1.4.156 md

```
double RGESolver::md = 0.006 [private]
```

3.1.4.157 mel

```
double RGESolver::mel = 0.0005 [private]
```

3.1.4.158 mh2

```
double RGESolver::mh2 = 126. * 126. [private]
```

m_h^2 (Higgs boson mass squared)

See <https://arxiv.org/pdf/1308.2627.pdf> for the normalization

3.1.4.159 mmu

```
double RGESolver::mmu = 0.100 [private]
```

3.1.4.160 ms

```
double RGESolver::ms = 0.05 [private]
```

3.1.4.161 mt

```
double RGESolver::mt = 170. [private]
```

3.1.4.162 mtau

```
double RGESolver::mtau = 1.2 [private]
```

3.1.4.163 mu

```
double RGESolver::mu = 0.002 [private]
```

m_u , the mass of up quark in GeV (default value ?????).

3.1.4.164 N1

```
const int RGESolver::N1 = 4 [static], [private]
```

3.1.4.165 N23

```
const int RGESolver::N23 = 3 [static], [private]
```

3.1.4.166 N4

```
const int RGESolver::N4 = 8 [static], [private]
```

3.1.4.167 N5

```
const int RGESolver::N5 = 3 [static], [private]
```

3.1.4.168 N6

```
const int RGESolver::N6 = 8 [static], [private]
```

3.1.4.169 N7

```
const int RGESolver::N7 = 8 [static], [private]
```

3.1.4.170 N7H

```
const int RGESolver::N7H = 7 [static], [private]
```

Number of Hermitian operators in class 7.

3.1.4.171 N7NH

```
const int RGESolver::N7NH = 1 [static], [private]
```

Number of non-Hermitian operators in class 7.

3.1.4.172 N8_LLLL

```
const int RGESolver::N8_LLLL = 5 [static], [private]
```

3.1.4.173 N8_LLRR

```
const int RGESolver::N8_LLRR = 8 [static], [private]
```

3.1.4.174 N8_LRLR

```
const int RGESolver::N8_LRLR = 4 [static], [private]
```

3.1.4.175 N8_LRRL

```
const int RGESolver::N8_LRRL = 1 [static], [private]
```

3.1.4.176 N8_RRRR

```
const int RGESolver::N8_RRRR = 7 [static], [private]
```

3.1.4.177 NC

```
const double RGESolver::NC = 3. [static], [private]
```

Number of colors.

3.1.4.178 NC2

```
const double RGESolver::NC2 = 9. [static], [private]
```

Number of colors squared.

3.1.4.179 NG

```
const int RGESolver::NG = 3 [static], [private]
```

Number of fermion flavours.

3.1.4.180 Ngauge

```
const int RGESolver::Ngauge = 3 [static], [private]
```

Number of gauge couplings.

3.1.4.181 Nh

```
const int RGESolver::Nh = 2 [static], [private]
```

Number of Higgs' sector parameters.

3.1.4.182 Nyukawa

```
const int RGESolver::Nyukawa = 3 [static], [private]
```

Number of Yukawa matrices.

3.1.4.183 ONE_SIXTH

```
const double RGESolver::ONE_SIXTH = (1. / 6.) [static], [private]
```

3.1.4.184 ONE_THIRD

```
const double RGESolver::ONE_THIRD = (1. / 3.) [static], [private]
```

3.1.4.185 Operators0F

```
std::unordered_map<std::string, double*> RGESolver::Operators0F [private]
```

3.1.4.186 s12

```
double RGESolver::s12 [private]
```

$\sin \theta_{12}$

3.1.4.187 s13

```
double RGESolver::s13 [private]
```

$\sin \theta_{13}$

3.1.4.188 s23

```
double RGESolver::s23 [private]
```

$\sin \theta_{23}$

3.1.4.189 s_

```
gsl_odeiv2_step* RGESolver::s_ [private]
```

Initial value:

```
= gsl_odeiv2_step_alloc(  
    gsl_odeiv2_step_rkf45, 2558)
```

3.1.4.190 Setter2F

```
std::unordered_map<std::string, boost::function<void(int, int, double) > > RGESolver::↵
Setter2F [private]
```

3.1.4.191 Setter4F

```
std::unordered_map<std::string, boost::function<void(int, int, int, int, double) > > RGE↵
Solver::Setter4F [private]
```

3.1.4.192 step_

```
double RGESolver::step_ [private]
```

Last step used in the integrator.

3.1.4.193 sys_

```
gsl_odeiv2_system RGESolver::sys_ = {func, NULL, 3} [private]
```

3.1.4.194 TEN_THIRDS

```
const double RGESolver::TEN_THIRDS = (10. / 3.) [static], [private]
```

3.1.4.195 TWO_THIRDS

```
const double RGESolver::TWO_THIRDS = (2. / 3.) [static], [private]
```

3.1.4.196 WC2I_indices

```
const int RGESolver::WC2I_indices [static], [private]
```

Initial value:

```
= {
    {0, 1},
    {0, 2},
    {1, 2}
}
```

Independent indices for WC2I.

3.1.4.197 WC2R_indices

```
const int RGESolver::WC2R_indices [static], [private]
```

Initial value:

```
= {
    {0, 0},
    {0, 1},
    {0, 2},
    {1, 1},
    {1, 2},
    {2, 2}
}
```

Independent indices for WC2R.

3.1.4.198 WC6I_indices

```
const int RGESolver::WC6I_indices [static], [private]
```

Initial value:

```
= {
    {0, 0, 0, 1},
    {0, 0, 0, 2},
    {0, 0, 1, 2},
    {0, 1, 0, 1},
    {0, 1, 0, 2},
    {0, 1, 1, 1},
    {0, 1, 1, 2},
    {0, 1, 2, 0},
    {0, 1, 2, 1},
    {0, 1, 2, 2},
    {0, 2, 0, 2},
    {0, 2, 1, 1},
    {0, 2, 1, 2},
    {0, 2, 2, 1},
    {0, 2, 2, 2},
    {1, 1, 1, 2},
    {1, 2, 1, 2},
    {1, 2, 2, 2}
}
```

Independent indices for WC6I.

3.1.4.199 WC6R_indices

```
const int RGESolver::WC6R_indices [static], [private]
```

Initial value:

```
= {
    {0, 0, 0, 0},
    {0, 0, 0, 1},
    {0, 0, 0, 2},
    {0, 0, 1, 1},
    {0, 0, 1, 2},
    {0, 0, 2, 2},
    {0, 1, 0, 1},
    {0, 1, 0, 2},
    {0, 1, 1, 0},
    {0, 1, 1, 1},
    {0, 1, 1, 2},
    {0, 1, 2, 0},
    {0, 1, 2, 1},
    {0, 1, 2, 2},
    {0, 2, 0, 2},
    {0, 2, 1, 1},

```

```

    {0, 2, 1, 2},
    {0, 2, 2, 0},
    {0, 2, 2, 1},
    {0, 2, 2, 2},
    {1, 1, 1, 1},
    {1, 1, 1, 2},
    {1, 1, 2, 2},
    {1, 2, 1, 2},
    {1, 2, 2, 1},
    {1, 2, 2, 2},
    {2, 2, 2, 2}
}

```

Independent indices for WC6R.

3.1.4.200 WC7I_indices

```
const int RGESolver::WC7I_indices [static], [private]
```

Independent indices for WC7I.

3.1.4.201 WC7R_indices

```
const int RGESolver::WC7R_indices [static], [private]
```

Independent indices for WC7R.

3.1.4.202 WC8I_indices

```
const int RGESolver::WC8I_indices [static], [private]
```

Initial value:

```

= {
    {0, 0, 0, 1},
    {0, 0, 0, 2},
    {0, 0, 1, 2},
    {0, 1, 0, 1},
    {0, 1, 0, 2},
    {0, 1, 1, 1},
    {0, 1, 1, 2},
    {0, 1, 2, 1},
    {0, 1, 2, 2},
    {0, 2, 0, 2},
    {0, 2, 1, 2},
    {0, 2, 2, 2},
    {1, 1, 1, 2},
    {1, 2, 1, 2},
    {1, 2, 2, 2}
}

```

Independent indices for WC8I.

3.1.4.203 WC8R_indices

```
const int RGESolver::WC8R_indices [static], [private]
```

Initial value:

```
= {
    {0, 0, 0, 0},
    {0, 0, 0, 1},
    {0, 0, 0, 2},
    {0, 0, 1, 1},
    {0, 0, 1, 2},
    {0, 0, 2, 2},
    {0, 1, 0, 1},
    {0, 1, 0, 2},
    {0, 1, 1, 1},
    {0, 1, 1, 2},
    {0, 1, 2, 1},
    {0, 1, 2, 2},
    {0, 2, 0, 2},
    {0, 2, 1, 2},
    {0, 2, 2, 2},
    {1, 1, 1, 1},
    {1, 1, 1, 2},
    {1, 1, 2, 2},
    {1, 2, 1, 2},
    {1, 2, 2, 2},
    {2, 2, 2, 2}
}
```

Independent indices for WC8R.

3.1.4.204 x

```
double RGESolver::x[2558] [private]
```

1D array for the integration

3.1.4.205 Yd

```
const double RGESolver::Yd = (- 1. / 3.) [static], [private]
```

3.1.4.206 Yd2

```
const double RGESolver::Yd2 = Yd*Yd [static], [private]
```

3.1.4.207 ydI

```
double RGESolver::ydI[3][3] [private]
```

3.1.4.208 ydR

```
double RGESolver::ydR[3][3] [private]
```

3.1.4.209 YdYe

```
const double RGESolver::YdYe = Yd*Ye [static], [private]
```

3.1.4.210 YdYl

```
const double RGESolver::YdYl = Yd*Yl [static], [private]
```

3.1.4.211 YdYq

```
const double RGESolver::YdYq = Yd*Yq [static], [private]
```

3.1.4.212 Ye

```
const double RGESolver::Ye = (- 1.) [static], [private]
```

3.1.4.213 Ye2

```
const double RGESolver::Ye2 = Ye*Ye [static], [private]
```

3.1.4.214 yeI

```
double RGESolver::yeI[3][3] [private]
```

3.1.4.215 yeR

```
double RGESolver::yeR[3][3] [private]
```

3.1.4.216 YeYl

```
const double RGESolver::YeYl = Ye*Yl [static], [private]
```

3.1.4.217 YeYq

```
const double RGESolver::YeYq = Ye*Yq [static], [private]
```

3.1.4.218 Yh

```
const double RGESolver::Yh = (0.5) [static], [private]
```

3.1.4.219 Yh2

```
const double RGESolver::Yh2 = Yh*Yh [static], [private]
```

3.1.4.220 YhYd

```
const double RGESolver::YhYd = Yh*Yd [static], [private]
```

3.1.4.221 YhYe

```
const double RGESolver::YhYe = Yh*Ye [static], [private]
```

3.1.4.222 YhYl

```
const double RGESolver::YhYl = Yh*Yl [static], [private]
```

3.1.4.223 YhYq

```
const double RGESolver::YhYq = Yh*Yq [static], [private]
```

3.1.4.224 YhYu

```
const double RGESolver::YhYu = Yh*Yu [static], [private]
```

3.1.4.225 Yl

```
const double RGESolver::Yl = (- 0.5) [static], [private]
```

3.1.4.226 Yl2

```
const double RGESolver::Yl2 = Yl*Yl [static], [private]
```

3.1.4.227 YlYq

```
const double RGESolver::YlYq = Yl*Yq [static], [private]
```

3.1.4.228 Yq

```
const double RGESolver::Yq = (1. / 6.) [static], [private]
```

3.1.4.229 Yq2

```
const double RGESolver::Yq2 = Yq*Yq [static], [private]
```

3.1.4.230 Yu

```
const double RGESolver::Yu = (2. / 3.) [static], [private]
```

3.1.4.231 Yu2

```
const double RGESolver::Yu2 = Yu*Yu [static], [private]
```

3.1.4.232 yuI

```
double RGESolver::yuI[3][3] [private]
```

3.1.4.233 yuR

```
double RGESolver::yuR[3][3] [private]
```

3.1.4.234 YuYd

```
const double RGESolver::YuYd = Yu*Yd [static], [private]
```

3.1.4.235 YuYe

```
const double RGESolver::YuYe = Yu*Ye [static], [private]
```

3.1.4.236 YuYl

```
const double RGESolver::YuYl = Yu*Yl [static], [private]
```

3.1.4.237 YuYq

```
const double RGESolver::YuYq = Yu*Yq [static], [private]
```

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

- [RGESolver.h](#)
- [RGESolver.cc](#)
- [StaticMembers.cc](#)
- [BetaFunction.cc](#)
- [SettersAndGetters.cc](#)

Chapter 4

File Documentation

4.1 BetaFunction.cc File Reference

4.2 RGESolver.cc File Reference

```
#include "RGESolver.h"  
#include <boost/bind/bind.hpp>  
#include "StaticMembers.cc"  
#include "SettersAndGetters.cc"  
#include "SMInput.cc"  
#include "BetaFunction.cc"
```

4.3 RGESolver.h File Reference

```
#include <iostream>  
#include <cmath>  
#include <fstream>  
#include <sstream>  
#include <gsl/gsl_errno.h>  
#include <gsl/gsl_matrix.h>  
#include <gsl/gsl_odeiv2.h>  
#include <unordered_map>  
#include <boost/function.hpp>  
#include "src/gslpp.h"
```

Classes

- class [RGESolver](#)

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

4.4 SettersAndGetters.cc File Reference

4.5 StaticMembers.cc File Reference

Index

~RGESolver
 RGESolver, [17](#)

b01
 RGESolver, [31](#)

b02
 RGESolver, [31](#)

b03
 RGESolver, [31](#)

BetaFunction.cc, [71](#)

c12
 RGESolver, [31](#)

c13
 RGESolver, [31](#)

c23
 RGESolver, [32](#)

cA2
 RGESolver, [32](#)

cA3
 RGESolver, [32](#)

cdBl
 RGESolver, [32](#)

cdBR
 RGESolver, [32](#)

cddl
 RGESolver, [32](#)

cddR
 RGESolver, [33](#)

cdGl
 RGESolver, [33](#)

cdGR
 RGESolver, [33](#)

cdHl
 RGESolver, [33](#)

cdHR
 RGESolver, [33](#)

cdWl
 RGESolver, [33](#)

cdWR
 RGESolver, [34](#)

ceBl
 RGESolver, [34](#)

ceBR
 RGESolver, [34](#)

cedl
 RGESolver, [34](#)

cedR
 RGESolver, [34](#)

ceel
 RGESolver, [34](#)

ceeR
 RGESolver, [35](#)

ceHl
 RGESolver, [35](#)

ceHR
 RGESolver, [35](#)

ceul
 RGESolver, [35](#)

ceuR
 RGESolver, [35](#)

ceWl
 RGESolver, [35](#)

ceWR
 RGESolver, [36](#)

cF2
 RGESolver, [36](#)

cF3
 RGESolver, [36](#)

cG
 RGESolver, [36](#)

cGT
 RGESolver, [36](#)

cH
 RGESolver, [36](#)

cHB
 RGESolver, [37](#)

cHBOX
 RGESolver, [37](#)

cHBT
 RGESolver, [37](#)

cHD
 RGESolver, [37](#)

cHdl
 RGESolver, [37](#)

cHdR
 RGESolver, [37](#)

cHel
 RGESolver, [38](#)

cHeR
 RGESolver, [38](#)

cHG
 RGESolver, [38](#)

cHGT
 RGESolver, [38](#)

cHI1l
 RGESolver, [38](#)

cHI1R
 RGESolver, [38](#)

- cHI3I
 - RGESolver, [39](#)
- cHI3R
 - RGESolver, [39](#)
- cHq1I
 - RGESolver, [39](#)
- cHq1R
 - RGESolver, [39](#)
- cHq3I
 - RGESolver, [39](#)
- cHq3R
 - RGESolver, [39](#)
- cHudI
 - RGESolver, [40](#)
- cHudR
 - RGESolver, [40](#)
- cHul
 - RGESolver, [40](#)
- cHuR
 - RGESolver, [40](#)
- cHW
 - RGESolver, [40](#)
- cHWB
 - RGESolver, [40](#)
- cHWBT
 - RGESolver, [41](#)
- cHWT
 - RGESolver, [41](#)
- CKM
 - RGESolver, [41](#)
- CKM_delta
 - RGESolver, [41](#)
- CKM_theta12
 - RGESolver, [41](#)
- CKM_theta13
 - RGESolver, [41](#)
- CKM_theta23
 - RGESolver, [42](#)
- cIdI
 - RGESolver, [42](#)
- cIdR
 - RGESolver, [42](#)
- cIedqI
 - RGESolver, [42](#)
- cIedqR
 - RGESolver, [42](#)
- cIel
 - RGESolver, [42](#)
- cIequ1I
 - RGESolver, [43](#)
- cIequ1R
 - RGESolver, [43](#)
- cIequ3I
 - RGESolver, [43](#)
- cIequ3R
 - RGESolver, [43](#)
- cIeR
 - RGESolver, [43](#)
- cII
 - RGESolver, [43](#)
- cIIR
 - RGESolver, [44](#)
- clq1I
 - RGESolver, [44](#)
- clq1R
 - RGESolver, [44](#)
- clq3I
 - RGESolver, [44](#)
- clq3R
 - RGESolver, [44](#)
- clul
 - RGESolver, [44](#)
- cluR
 - RGESolver, [45](#)
- con_
 - RGESolver, [45](#)
- cqd1I
 - RGESolver, [45](#)
- cqd1R
 - RGESolver, [45](#)
- cqd8I
 - RGESolver, [45](#)
- cqd8R
 - RGESolver, [45](#)
- cqeI
 - RGESolver, [46](#)
- cqeR
 - RGESolver, [46](#)
- cqq1I
 - RGESolver, [46](#)
- cqq1R
 - RGESolver, [46](#)
- cqq3I
 - RGESolver, [46](#)
- cqq3R
 - RGESolver, [46](#)
- cqu1I
 - RGESolver, [47](#)
- cqu1R
 - RGESolver, [47](#)
- cqu8I
 - RGESolver, [47](#)
- cqu8R
 - RGESolver, [47](#)
- cquqd1I
 - RGESolver, [47](#)
- cquqd1R
 - RGESolver, [47](#)
- cquqd8I
 - RGESolver, [48](#)
- cquqd8R
 - RGESolver, [48](#)
- cuBI
 - RGESolver, [48](#)
- cuBR
 - RGESolver, [48](#)

- cud1I
 - RGESolver, [48](#)
- cud1R
 - RGESolver, [48](#)
- cud8I
 - RGESolver, [49](#)
- cud8R
 - RGESolver, [49](#)
- cuGI
 - RGESolver, [49](#)
- cuGR
 - RGESolver, [49](#)
- cuHI
 - RGESolver, [49](#)
- cuHR
 - RGESolver, [49](#)
- cuul
 - RGESolver, [50](#)
- cuuR
 - RGESolver, [50](#)
- cuWI
 - RGESolver, [50](#)
- cuWR
 - RGESolver, [50](#)
- cW
 - RGESolver, [50](#)
- cWT
 - RGESolver, [50](#)
- delta
 - RGESolver, [51](#)
- DF
 - RGESolver, [51](#)
- DFa
 - RGESolver, [51](#)
- DFs
 - RGESolver, [51](#)
- dim
 - RGESolver, [51](#)
- DWC2I
 - RGESolver, [52](#)
- DWC2R
 - RGESolver, [52](#)
- DWC6I
 - RGESolver, [52](#)
- DWC6R
 - RGESolver, [52](#)
- DWC7I
 - RGESolver, [52](#)
- DWC7R
 - RGESolver, [52](#)
- DWC8I
 - RGESolver, [53](#)
- DWC8R
 - RGESolver, [53](#)
- E5
 - RGESolver, [53](#)
- E6
 - RGESolver, [53](#)
- E7
 - RGESolver, [53](#)
- E8_LLLL
 - RGESolver, [53](#)
- E8_LLRR
 - RGESolver, [53](#)
- E8_LRLR
 - RGESolver, [54](#)
- E8_LRRL
 - RGESolver, [54](#)
- E8_RRRR
 - RGESolver, [54](#)
- EIGHT_THIRDS
 - RGESolver, [54](#)
- epsabs
 - RGESolver, [18](#)
- epsabs_
 - RGESolver, [54](#)
- epsrel
 - RGESolver, [18](#)
- epsrel_
 - RGESolver, [54](#)
- evo_
 - RGESolver, [54](#)
- Evolve
 - RGESolver, [18](#)
- EvolveSMOnly
 - RGESolver, [18](#)
- ExtractParametersFromCKM
 - RGESolver, [19](#)
- Eyuk
 - RGESolver, [55](#)
- FOUR_THIRDS
 - RGESolver, [55](#)
- FromMassesToYukawas
 - RGESolver, [19](#)
- func
 - RGESolver, [19](#)
- funcSMOnly
 - RGESolver, [19](#)
- g1
 - RGESolver, [55](#)
- g2
 - RGESolver, [55](#)
- g3
 - RGESolver, [55](#)
- GenerateSMInitialConditions
 - RGESolver, [20](#)
- GetCoefficient
 - RGESolver, [20](#), [21](#)
- GetSMInputScale
 - RGESolver, [22](#)
- Getter2F
 - RGESolver, [55](#)
- Getter4F
 - RGESolver, [56](#)

- GoToBasisSOnly
 - RGESolver, [22](#)
- Init
 - RGESolver, [22](#)
- InitSOnly
 - RGESolver, [22](#)
- InputScale_SM
 - RGESolver, [56](#)
- lambda
 - RGESolver, [56](#)
- mb
 - RGESolver, [56](#)
- mc
 - RGESolver, [56](#)
- md
 - RGESolver, [56](#)
- mel
 - RGESolver, [56](#)
- mh2
 - RGESolver, [57](#)
- mmu
 - RGESolver, [57](#)
- ms
 - RGESolver, [57](#)
- mt
 - RGESolver, [57](#)
- mtau
 - RGESolver, [57](#)
- mu
 - RGESolver, [57](#)
- N1
 - RGESolver, [57](#)
- N23
 - RGESolver, [58](#)
- N4
 - RGESolver, [58](#)
- N5
 - RGESolver, [58](#)
- N6
 - RGESolver, [58](#)
- N7
 - RGESolver, [58](#)
- N7H
 - RGESolver, [58](#)
- N7NH
 - RGESolver, [58](#)
- N8_LLLL
 - RGESolver, [59](#)
- N8_LLRR
 - RGESolver, [59](#)
- N8_LRLR
 - RGESolver, [59](#)
- N8_LRRL
 - RGESolver, [59](#)
- N8_RRRR
 - RGESolver, [59](#)
- NC
 - RGESolver, [59](#)
- NC2
 - RGESolver, [59](#)
- NG
 - RGESolver, [60](#)
- Ngauge
 - RGESolver, [60](#)
- Nh
 - RGESolver, [60](#)
- Nyukawa
 - RGESolver, [60](#)
- ONE_SIXTH
 - RGESolver, [60](#)
- ONE_THIRD
 - RGESolver, [60](#)
- OperatorsOF
 - RGESolver, [61](#)
- Print
 - RGESolver, [22](#)
- Reset
 - RGESolver, [23](#)
- RGESolver, [5](#)
 - ~RGESolver, [17](#)
 - b01, [31](#)
 - b02, [31](#)
 - b03, [31](#)
 - c12, [31](#)
 - c13, [31](#)
 - c23, [32](#)
 - cA2, [32](#)
 - cA3, [32](#)
 - cdBI, [32](#)
 - cdBR, [32](#)
 - cddl, [32](#)
 - cddR, [33](#)
 - cdGI, [33](#)
 - cdGR, [33](#)
 - cdHI, [33](#)
 - cdHR, [33](#)
 - cdWI, [33](#)
 - cdWR, [34](#)
 - ceBI, [34](#)
 - ceBR, [34](#)
 - cedI, [34](#)
 - cedR, [34](#)
 - ceel, [34](#)
 - ceeR, [35](#)
 - ceHI, [35](#)
 - ceHR, [35](#)
 - ceul, [35](#)
 - ceuR, [35](#)
 - ceWI, [35](#)
 - ceWR, [36](#)
 - cF2, [36](#)

cF3, 36
cG, 36
cGT, 36
cH, 36
cHB, 37
cHBOX, 37
cHBT, 37
cHD, 37
cHdI, 37
cHdR, 37
cHeI, 38
cHeR, 38
cHG, 38
cHGT, 38
cHI1I, 38
cHI1R, 38
cHI3I, 39
cHI3R, 39
cHq1I, 39
cHq1R, 39
cHq3I, 39
cHq3R, 39
cHudI, 40
cHudR, 40
cHul, 40
cHuR, 40
cHW, 40
cHWB, 40
cHWBT, 41
cHWT, 41
CKM, 41
CKM_delta, 41
CKM_theta12, 41
CKM_theta13, 41
CKM_theta23, 42
cIdI, 42
cIdR, 42
cledqI, 42
cledqR, 42
cleI, 42
clequ1I, 43
clequ1R, 43
clequ3I, 43
clequ3R, 43
cleR, 43
cIII, 43
cIIR, 44
clq1I, 44
clq1R, 44
clq3I, 44
clq3R, 44
clul, 44
cluR, 45
con_, 45
cqdl, 45
cqdlR, 45
cqdl, 45
cqdlR, 45
cqel, 46
cqelR, 46
cqql, 46
cqqlR, 46
cqql, 46
cqqlR, 46
cqu1I, 47
cqu1R, 47
cqu8I, 47
cqu8R, 47
cquqd1I, 47
cquqd1R, 47
cquqd8I, 48
cquqd8R, 48
cuBI, 48
cuBR, 48
cud1I, 48
cud1R, 48
cud8I, 49
cud8R, 49
cuGI, 49
cuGR, 49
cuHI, 49
cuHR, 49
cuul, 50
cuuR, 50
cuWI, 50
cuWR, 50
cW, 50
cWT, 50
delta, 51
DF, 51
DFa, 51
DFs, 51
dim, 51
DWC2I, 52
DWC2R, 52
DWC6I, 52
DWC6R, 52
DWC7I, 52
DWC7R, 52
DWC8I, 53
DWC8R, 53
E5, 53
E6, 53
E7, 53
E8_LLLL, 53
E8_LLRR, 53
E8_LRLR, 54
E8_LRRL, 54
E8_RRRR, 54
EIGHT_THIRDS, 54
epsabs, 18
epsabs_, 54
epsrel, 18
epsrel_, 54
evo_, 54
Evolve, 18

EvolveSMAOnly, 18
 ExtractParametersFromCKM, 19
 Eyuk, 55
 FOUR_THIRDS, 55
 FromMassesToYukawas, 19
 func, 19
 funcSMAOnly, 19
 g1, 55
 g2, 55
 g3, 55
 GenerateSMInitialConditions, 20
 GetCoefficient, 20, 21
 GetSMInputScale, 22
 Getter2F, 55
 Getter4F, 56
 GoToBasisSMAOnly, 22
 Init, 22
 InitSMAOnly, 22
 InputScale_SM, 56
 lambda, 56
 mb, 56
 mc, 56
 md, 56
 mel, 56
 mh2, 57
 mmu, 57
 ms, 57
 mt, 57
 mtau, 57
 mu, 57
 N1, 57
 N23, 58
 N4, 58
 N5, 58
 N6, 58
 N7, 58
 N7H, 58
 N7NH, 58
 N8_LLLL, 59
 N8_LLRR, 59
 N8_LRLR, 59
 N8_LRRL, 59
 N8_RRRR, 59
 NC, 59
 NC2, 59
 NG, 60
 Ngauge, 60
 Nh, 60
 Nyukawa, 60
 ONE_SIXTH, 60
 ONE_THIRD, 60
 Operators0F, 61
 Print, 22
 Reset, 23
 RGESolver, 17
 s12, 61
 s13, 61
 s23, 61
 s_, 61
 SaveOutputFile, 23
 Set_epsabs, 23
 Set_epsrel, 23
 Set_step, 23
 SetCoefficient, 24
 SetSMInputScale, 25
 Setter2F, 61
 Setter4F, 62
 step, 25
 step_, 62
 sys_, 62
 TEN_THIRDS, 62
 TWO_THIRDS, 62
 Update, 25
 UpdateCKM, 26
 UpdateSMAOnly, 26
 WC1, 26
 WC1_set, 26
 WC2I, 26
 WC2I_indices, 62
 WC2I_set, 26
 WC2R, 27
 WC2R_indices, 62
 WC2R_set, 27
 WC3, 27
 WC3_set, 27
 WC5, 27
 WC5_set, 28
 WC6I, 28
 WC6I_indices, 63
 WC6I_set, 28
 WC6R, 28
 WC6R_indices, 63
 WC6R_set, 28
 WC7I, 29
 WC7I_indices, 64
 WC7I_set, 29
 WC7R, 29
 WC7R_indices, 64
 WC7R_set, 29
 WC8I, 29
 WC8I_indices, 64
 WC8I_set, 30
 WC8R, 30
 WC8R_indices, 64
 WC8R_set, 30
 x, 65
 Yd, 65
 Yd2, 65
 ydl, 65
 ydR, 65
 YdYe, 66
 YdYl, 66
 YdYq, 66
 Ye, 66
 Ye2, 66
 yel, 66

- yeR, [66](#)
- YeYl, [66](#)
- YeYq, [67](#)
- Yh, [67](#)
- Yh2, [67](#)
- YhYd, [67](#)
- YhYe, [67](#)
- YhYl, [67](#)
- YhYq, [67](#)
- YhYu, [67](#)
- Yl, [68](#)
- Yl2, [68](#)
- YlYq, [68](#)
- Yq, [68](#)
- Yq2, [68](#)
- Yu, [68](#)
- Yu2, [68](#)
- yul, [68](#)
- Yukawa, [30](#)
- Yukawa_set, [30](#)
- yuR, [69](#)
- YuYd, [69](#)
- YuYe, [69](#)
- YuYl, [69](#)
- YuYq, [69](#)
- RGESolver.cc, [71](#)
- RGESolver.h, [71](#)
- s12
 - RGESolver, [61](#)
- s13
 - RGESolver, [61](#)
- s23
 - RGESolver, [61](#)
- s_
 - RGESolver, [61](#)
- SaveOutputFile
 - RGESolver, [23](#)
- Set_epsabs
 - RGESolver, [23](#)
- Set_epsrel
 - RGESolver, [23](#)
- Set_step
 - RGESolver, [23](#)
- SetCoefficient
 - RGESolver, [24](#)
- SetSMInputScale
 - RGESolver, [25](#)
- Setter2F
 - RGESolver, [61](#)
- Setter4F
 - RGESolver, [62](#)
- SettersAndGetters.cc, [71](#)
- StaticMembers.cc, [71](#)
- step
 - RGESolver, [25](#)
- step_
 - RGESolver, [62](#)
- sys_
 - RGESolver, [62](#)
- TEN_THIRDS
 - RGESolver, [62](#)
- TWO_THIRDS
 - RGESolver, [62](#)
- Update
 - RGESolver, [25](#)
- UpdateCKM
 - RGESolver, [26](#)
- UpdateSMOnly
 - RGESolver, [26](#)
- WC1
 - RGESolver, [26](#)
- WC1_set
 - RGESolver, [26](#)
- WC2l
 - RGESolver, [26](#)
- WC2l_indices
 - RGESolver, [62](#)
- WC2l_set
 - RGESolver, [26](#)
- WC2R
 - RGESolver, [27](#)
- WC2R_indices
 - RGESolver, [62](#)
- WC2R_set
 - RGESolver, [27](#)
- WC3
 - RGESolver, [27](#)
- WC3_set
 - RGESolver, [27](#)
- WC5
 - RGESolver, [27](#)
- WC5_set
 - RGESolver, [28](#)
- WC6l
 - RGESolver, [28](#)
- WC6l_indices
 - RGESolver, [63](#)
- WC6l_set
 - RGESolver, [28](#)
- WC6R
 - RGESolver, [28](#)
- WC6R_indices
 - RGESolver, [63](#)
- WC6R_set
 - RGESolver, [28](#)
- WC7l
 - RGESolver, [29](#)
- WC7l_indices
 - RGESolver, [64](#)
- WC7l_set
 - RGESolver, [29](#)
- WC7R
 - RGESolver, [29](#)
- WC7R_indices

RGESolver, [64](#)
 WC7R_set
 RGESolver, [29](#)
 WC8I
 RGESolver, [29](#)
 WC8I_indices
 RGESolver, [64](#)
 WC8I_set
 RGESolver, [30](#)
 WC8R
 RGESolver, [30](#)
 WC8R_indices
 RGESolver, [64](#)
 WC8R_set
 RGESolver, [30](#)

 x
 RGESolver, [65](#)

 Yd
 RGESolver, [65](#)
 Yd2
 RGESolver, [65](#)
 ydl
 RGESolver, [65](#)
 ydR
 RGESolver, [65](#)
 YdYe
 RGESolver, [66](#)
 YdYl
 RGESolver, [66](#)
 YdYq
 RGESolver, [66](#)
 Ye
 RGESolver, [66](#)
 Ye2
 RGESolver, [66](#)
 yel
 RGESolver, [66](#)
 yeR
 RGESolver, [66](#)
 YeYl
 RGESolver, [66](#)
 YeYq
 RGESolver, [67](#)
 Yh
 RGESolver, [67](#)
 Yh2
 RGESolver, [67](#)
 YhYd
 RGESolver, [67](#)
 YhYe
 RGESolver, [67](#)
 YhYl
 RGESolver, [67](#)
 YhYq
 RGESolver, [67](#)
 YhYu
 RGESolver, [67](#)

 Yl
 RGESolver, [68](#)
 Yl2
 RGESolver, [68](#)
 YlYq
 RGESolver, [68](#)
 Yq
 RGESolver, [68](#)
 Yq2
 RGESolver, [68](#)
 Yu
 RGESolver, [68](#)
 Yu2
 RGESolver, [68](#)
 yul
 RGESolver, [68](#)
 Yukawa
 RGESolver, [30](#)
 Yukawa_set
 RGESolver, [30](#)
 yuR
 RGESolver, [69](#)
 YuYd
 RGESolver, [69](#)
 YuYe
 RGESolver, [69](#)
 YuYl
 RGESolver, [69](#)
 YuYq
 RGESolver, [69](#)