Date: March 13, 2025

The micrOMEGAs user's manual, version 6.2

G. Alguero[4], D. Barducci[1], G. Bélanger[2], A. Belyaev[10], F. Boudjema[2], S.Chakraborti[9], J. Da Silva[2], A. Goudelis[3], S. Kraml[4], U. Laa[5], A. Mjallal[2], A. Pukhov[6], A. Semenov[7], B. Zaldivar[8].

1) Università degli Studi di Roma la Sapienza and INFN Section of Roma 1, Piazzale Aldo Moro 5, 00185, Roma, Italy
2) LAPTh, CNRS, USMB, 9 Chemin de Bellevue, F-74940 Annecy, France
3) LPC, CNRS/IN2P3, Univ. Clermont- Auvergne, 4 Av. Blaise Pascal, F-63178 Aubière Cedex, France
4) Laboratoire de Physique Subatomique et de Cosmologie, Université Grenoble-Alpes, CNRS/IN2P3, 53 Avenue des Martyrs, F-38026 Grenoble, France
5) Institute of Statistics, BOKU, Peter-Jordan-Straße 82, 1190 Wien, Austria
6) Skobeltsyn Inst. of Nuclear Physics, Moscow State Univ., Moscow 119992, Russia
7) Joint Institute for Nuclear Research (JINR) 141980, Dubna, Russia
8) Departamento de Física Teórica, UAM, 28049 Madrid, Spain
9) IPPP, Department of Physics, Durham University, Durham, DH1 3LE, United Kingdom
10) School of Physics and Astronomy, University of Southampton, Highfield, Southampton SO17 1BJ, UK

Abstract

We give an up-to-date description of the `micrOMEGAs` functions. Only the routines which are available for the users are described. Examples on how to use these functions can be found in the sample main programs distributed with the code.

Contents

1 Introduction

`micrOMEGAs` is a code to calculate the properties of cold dark matter (CDM) in a generic model of particle physics. First developed to compute the relic density of dark matter, the code also computes the rates for dark matter direct and indirect detection. `micrOMEGAs` computes CDM properties in the framework of a model of particle interactions presented in `CalcHEP` format [**?**]. It is assumed that the model is invariant under a discrete symmetry like R-parity (even for all standard particles and odd for some new particles including the dark matter candidate) which ensures the stability of the lightest odd particle. Similarly in multi-component dark matter models, a discrete symmetry that guarantees the stability of the lightest particle in each of the dark matter sectors is assumed. The `CalcHEP` package is included in `micrOMEGAs` and used for matrix elements calculations. All annihilation and coannihilation channels are included in the computation of the relic density. This manual gives an up-to-date description of all `micrOMEGAs` functions. The methods used to compute the different dark matter properties are described in references [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]. These references also contain a more complete description of the code. In the following the cold dark matter candidate also called weakly-interactive massive particle (WIMP) will be denoted by χ. Starting with version 5.0, `micrOMEGAs` also allows to compute the abundance of feebly interacting dark matter candidates (FIMP) through the freeze-in mechanism [**?**]

micrOMEGAs is written in C and uses some Fortran routines mostly in external packages. The complete format for all functions can be found in include/.h (for C). Examples on how to use these functions are provided in the MSSM/main.c file.

2 Discrete symmetry in micrOMEGAs

micrOMEGAs exploits the fact that models of dark matter exhibit a discrete symmetry and that the fields of the model transform as $\phi \to e^{i2\pi X_\phi}\phi$ where the charge $|X_\phi| < 1$. The particles of the Standard Model are assumed to transform trivially under the discrete symmetry, $X_\phi = 0$. In the following all particles with charge $X_\phi \neq 0$ will be called *odd* and the lightest odd particle will be stable. If neutral, it can be considered as a DM candidate. Typical examples of discrete symmetries used for constructing single DM models are Z_2 and Z_3. Multi-component DM can arise in models with larger discrete groups. A simple example is a model with $Z_2 \times Z_2'$ symmetry, the particles charged under $Z_2(Z_2')$ will belong to the first (second) dark sector. The lightest particle of each sector will be stable and therefore a potential DM candidate. Another example is a model with a Z_4 symmetry. The two dark sectors contain particles with $X_\phi = \pm 1/4$ and $X_\phi = 1/2$ respectively. The lightest particle with charge 1/4 is always stable while the lightest particle of charge 1/2 is stable only if its decay into two particles of charge 1/4 is kinematically forbidden. micrOMEGAs assumes that all odd particles have names starting with '~', for example, ~o1 for the lightest neutralino. To distinguish the particles with different transformation properties with respect to the discrete group, that is particles belonging to different 'dark' sectors, we use the number of '~' at the beginning of the name of the particles. The maximal number of sectors is limited by the maximal length of the particle names (11 in the current version). For the relic density calculation we also assume that all particles in a given sector are in thermal equilibrium with each other. This last assumption is not necessarily valid, therefore the user has the possibility to split default sectors in subsectors where thermal equilibrium is maintained. See Section 6.5 for details.

Note that micrOMEGAs does not check the symmetry of the Lagrangian, it assumes that the name convention correctly identifies all particles with the same discrete symmetry quantum numbers. For models with FIMPs, new particles are considered to be in thermal equilibrium with the SM bath ($\mathcal{B}$) at high temperatures unless explicitly defined as being feeble, ie belonging to $\mathcal{F}$. Both $\mathcal{B}$ and $\mathcal{F}$ can contain odd or even particles, see section 6.7.

3 Downloading and compilation of micrOMEGAs

To download micrOMEGAs, go to

```
http://lapth.cnrs.fr/micromegas
```

and unpack the file received, micromegas_6.2.tgz, with the command

```
tar -xvzf micromegas_6.2.tgz
```

This should create the directory micromegas_6.2/ which occupies about 116Mb of disk space. You will need more disk space after compilation of specific models and generation of matrix elements. In case of problems and questions

email: micromegas@lapth.cnrs.fr

3.1 File structure of micrOMEGAs

calc	calculator
calchep.ini	specify the fonts for graphics in CalCHEP
Makefile	to compile the kernel of the package
CalcHEP_src/	generator of matrix elements for micrOMEGAs
Packages/	external codes
clean	to remove compiled files
fileMap.txt	contains the list of all files in the code
history	contains a list of changes in recent versions
man/	contains the manual: description of micrOMEGAs routines
newProject	to create a new model directory structure
sources/	micrOMEGAs code
include/	include files for micrOMEGAs routines or external codes
lib/	contains library micromegas.a when micrOMEGAs is compiled
MSSM model directory	
MSSM/	
Makefile	to compile the code and executable for this model
main.c[pp]	files with sample *main* programs
README	Brief description on how to use the code
mssmX.par	Sample input files
lib/	directory for routines specific to this model
Makefile	to compile the auxiliary code library *lib/aLib.a*
*.c *.f *.h	source codes of auxiliary functions
work/	CalcHEP working directory for the generation of matrix elements
Makefile	to compile the library *work/work_aux.a*
calchep	Executable for CalcHEP
lanhep/	directory containing lanhep source model files
models/	directory for files which specifies the model files *1.mdl are used in micrOMEGAs sessions. Other *.mdl files are intended for CalcHEP interactive sessions
vars1.mdl	free variables
func1.mdl	constrained variables
prtcls1.mdl	particles
lgrng1.mdl	Feynman rules
results/	
so_generated/	storage of matrix elements generated by CalcHEP
Directories of other models which have the same structure as MSSM/	
NMSSM/	Next-to-Minimal Supersymmetric Model [?, ?]
CPVMSSM/	MSSM with complex parameters [?, ?]
IDM/	Inert Doublet Model [?]
LLL_scalar/	Simplified model with singlet charged lepton and real scalar DM
LHM/	Little Higgs Model [?]

`RDM/`	Scalar Leptoquark and two singlet fermions [?]
`SingletDM/`	Singlet scalar DM model with Z_2 symmetry [?]
`STFM/`	Singlet-triplet fermionic model [?]
`Z3IDM/`	Inert doublet model with Z_3 discrete symmetry [?, ?]
`Z4IDSM/`	Inert doublet and singlet model with Z_4 symmetry [?, ?]
`Z5M/`	Two scalar singlets model with Z_5 symmetry [?]
`ZpPortal/`	Simplified model with a Z' portal and fermion DM
`mdlIndep/`	For model independent computation of DM signals

Other models can be downloaded on the web, `http://lapth.cnrs.fr/micromegas`, for example : `RHNM`, a right-handed Neutrino Model [?], `SM4`, a toy model with a 4th generation of leptons and neutrino DM, `UMSSM`, an U(1) extension of the MSSM [?, ?].

3.2 Compilation of `CalcHEP` and `micrOMEGAs` routines

The graphical interface of CalcHEP and micrOMEGAs uses X11 routines. Therefore X11 header files

```
/use/include/X11/*.h
```

are needed to compile these codes. If you do not have these files on your computer, you have to install the *X11-devel* package. Its name depends on the operating system, namely,

```
libX11-devel    for Fedora/Scientific, old Darwin(Mac)
Xquartz ( https://www.xquartz.org)     new Mac
libX11-dev      for Ubuntu/Debian      [old Ubuntu]
libx11-dev      for Ubuntu/Debian      [new Ubuntu]
xorg-x11-devel  for SUSE
```

`CalcHEP` and `micrOMEGAs` are compiled by *gmake*. Go to the `micrOMEGAs` directory and launch

```
gmake
```

If `gmake` is not available, then `make` should work like `gmake`. In principle `micrOMEGAs` defines automatically the names of *C* and *Fortran* compilers and the flags for compilation. If you meet a problem, open the file which contains the compiler specifications, `CalcHEP_src/FlagsForSh`, improve it, and launch `[g]make` again. The file is written in **sh** script format and looks like

```
# C compiler
CC="gcc"
# Flags for C compiler
CFLAGS="-g -fsigned-char"
# Disposition of header files for X11
HX11=
# Disposition of lX11
LX11="-lX11"
# Fortran compiler
FC="gfortran"
```

```
FFLAGS="-fno-automatic"
........
```

After a successful definition of compilers and their flags, `micrOMEGAs` rewrites the file *FlagsForSh* into *FlagsForMake* and substitutes its contents in all *Makefile*s of the package.

`[g]make clean` deletes all generated files, but asks permission to delete *FlagsForSh*.

`[g]make flags` only generates FlagsForSh. It allows to check and change flags before compiling the codes.

3.3 External packages

`micrOMEGAs` is interfaced with a number of external packages, some of which are directly included in the `micrOMEGAs` distributions, those are stored in in the directory `/Packages`, while others are downloaded automatically upon request.

The codes included in the `micrOMEGAs` distribution are:

`Suspect2.41` [?]	spectrum calculator for MSSM
NMSSMTools5.6.0 [?, ?]	spectrum calculator for NMSSM
`CPsuperH2.3` [?, ?]	spectrum calculator for the MSSM and CPVMSSM
`LoopTools` [?]	for computing some loop-induced processes
`lf2c`	auxilliary library for Fortran-C converter
`LanHEP` [?]	for generating model files
`maxGap` [?]	for recasting experimental limits using the Optimal Intervals method
`Lilith-2.1` [?, ?, ?]	for checking Higgs signal strengths
`FermiDwarfs` [?, ?, ?]	To determine the limit from the FermiLAT experiment on photons from Dwarf Spheroidal galaxies

The packages that are downloaded automatically during runtime are:

`SOFTSUSY` [?]	SUSY spectrum generator
`SPheno` [?]	SUSY spectrum generator
`SModelS` [?, ?, ?, ?, ?]	for simplified-model constraints from LHC searches
`superIso` [?]	for flavor constraints
`HiggsBounds` [?, ?]	for checking Higgs-sector constraints
`HiggsSignals` [?, ?]	

The versions of the codes to be downloaded can be defined by the user via the parameter VERSION in the corresponding files:

```
MSSM/lib/ssusy_call.c    include/SMODELS.inc  include/hBandS.inc
MSSM/lib/spheno_call.c   sources/superIso.c
```

The URL for the source code and compilation instructions are presented in *makefile*s

```
SSUSY.makef      SMODELS.makef    HBOUNDS.makef
SPHENO.makef     SUPERISO.makef   HSIGNALS.makef
```

Note however, that care must be taken that new versions remain compatible with the existing interface structure. We also point out that `Lilith` `SModelS` and `FermiDwarfs` are python packages; for the latest versions of these codes, a python3 installation is required in addition to the compilers mentioned above.

3.4 Module structure of main programs

Each model included in `micrOMEGAs` is accompanied with sample files for C programs which call `micrOMEGAs` routines, the *main.c* files. These files consist of several modules enclosed between the instructions

```
#ifdef XXXXX
  ..................
#endif
```

Each of these blocks contains some code for a specific problem

```
#define MASSES_INFO        //Displays information about mass spectrum
#define CONSTRAINTS        //Displays B_>sgamma, Bs->mumu, etc
#define HIGGSBOUNDS        //Calls HiggsBounds to constrain the Higgs sector
#define HIGGSSIGNALS       //Calls HiggsSignal to constrain the Higgs boson
#define SUPERISO           //calls SuperISO to compute flavour observables
#define LILITH             //Calls LiLith to constrain the Higgs sector
#define SMODELS            //Calls SModelS to constrain the new physics sector
#define MONOJET            //Constrain the new physics sector using the LHC monojet limit
#define OMEGA              //Calculates the relic density
#define FREEZEIN           //Calculates the relic density in the freeze-in mechanism
#define INDIRECT_DETECTION //Signals of DM annihilation in galactic halo
#define LoopGAMMA          //Gamma-Ray lines - available only in some models
#define RESET_FORMFACTORS  //Redefinition of Form Factors and other
                           //parameters
#define CDM_NUCLEON        //Calculates amplitudes and cross-sections
                           //for DM-nucleon collisions
#define CDM_NUCLEUS        //Calculates number of events for 1kg*day,
                           //recoil energy distribution for various nuclei
                           //and compares with experimental data.
#define NEUTRINO           //Calculates flux of solar neutrinos and
                           //the corresponding muon flux
#define DECAYS             //Calculates decay widths and branching ratios
#define CROSS_SECTIONS     //Calculates cross sections
#define CLEAN              //Removes intermediate files.
```

The flag

```
#define SHOWPLOTS          //switches on graphic facilities of micrOMEGAs.
```

All these modules are completely independent. The user can comment or uncomment any set of *define* instructions to suit his/her need.

3.5 Compilation of codes for specific models

After the compilation of `micrOMEGAs` one has to compile the executable to compute DM related observables in a specific model. To do this, go to the model directory, say MSSM, and launch

`[g]make`

This should generate the executable `main` using the `main.c` source file. In general

`gmake main=`*filename.ext*

generates the executable `filename` based on the source file *filename.ext*. For *ext* we support 2 options: *'c'* , *'cpp'* which correspond to `C` and `C++` sources. `[g]make` called in the model directory automatically launches `[g]make` in the subdirectories `lib` and `work` to compile

`lib/aLib.a` – the library of auxiliary model functions, and
`work/work_aux.a` – the library of model particles, free and dependent parameters.

3.6 Command line parameters of main programs

The default versions of *main.c* programs need some arguments which have to be specified in command lines. If launched without arguments *main* explains which parameter are needed. As a rule *main* needs the name of a file containing the numerical values of the free parameters of the model. The structure of a file record should be

```
Name       Value # comment ( optional)
```

For instance, an Inert Doublet model (IDM) input file contains

```
Mh    125   # mass of SM Higgs
MHC   200   # mass of charged Higgs ~H+
MH3   200   # mass of odd Higgs ~H3
MHX   63.2  # mass of ~X particle
la2  0.01   # \lambda_2  coupling
laL  0.01   # 0.5*(\lambda_3+\lambda_4+\lambda_5)
```

In other cases, different inputs can be required. For example, in the MSSM with input parameters defined at the GUT scale, the parameters have to be provided in a command line. Launching `./main` will return

```
This program needs 4 parameters:
  m0      common scalar mass at GUT scale
  mhf common gaugino mass at GUT scale
  a0     trilinear soft breaking parameter at GUT scale
  tb    tan(beta)
Auxiliary parameters are:
  sgn +/-1, sign of Higgsino mass term (default 1)
  Mtp top quark pole mass
  MbMb Mb(Mb) scale independent b-quark mass
  alfSMZ strong coupling at MZ
Example: ./main 120 500 -350 10 1 173.1
```

4 Global Parameters and constants

The list of the global parameters and their default values are given in Tables 1 and 2. The numerical value for any of these parameters can be simply reset anywhere in the code. The numerical values of the scalar quark form factors can also be reset by the `calcScalarQuarkFF` routine presented below. Some physical values evaluated by `micrOMEGAs` also are presented as global variables, see Table 3.

Table 1: Global input parameters of `micrOMEGAs`

Name	default value	units	comments
deltaY	0		Difference between DM/anti-DM abundances
K_dif	0.0112	kpc^2/Myr	The normalized diffusion coefficient
L_dif	4	kpc	Vertical size of the Galaxy diffusive halo
Delta_dif	0.7		Slope of the diffusion coefficient
Tau_dif	10^{16}	s	Electron energy loss time
Vc_dif	0	km/s	Convective Galactic wind
Fermi_a	0.52	fm	nuclei surface thickness
Fermi_b	-0.6	fm	parameters to set the nuclei radius with
Fermi_c	1.23	fm	$R_A = cA^{1/3} + b$
Rsun	8.5	kpc	Distance from the Sun to the center of the Galaxy
Rdisk	20	kpc	Radius of the galactic diffusion disk
rhoDM	0.3	GeV/cm^3	Dark Matter density at Rsun
vEarth	232	km/s	Galaxy velocity of the Earth
vRot	220	km/s	Galaxy rotation velocity at Rsun
vEsc	544	km/s	Escape velocity at Rsun
etaSHMpp	0.2		η parameter of `SHM++`
betaSHMpp	0.9		β parameter of `SHM++`

Table 2: Global parameters of `micrOMEGAs`: nucleon quark form factors

Proton		Neutron		
Name	value	Name	value	comments
ScalarFFPd	0.0191	ScalarFFNd	0.0273	
ScalarFFPu	0.0153	ScalarFFNu	0.011	Scalar form factor
ScalarFFPs	0.0447	ScalarFFNs	0.0447	
pVectorFFPd	-0.427	pVectorFFNd	0.842	
pVectorFFPu	0.842	pVectorFFNu	-0.427	Axial-vector form factor
pVectorFFPs	-0.085	pVectorFFNs	-0.085	
SigmaFFPd	-0.23	SigmaFFNd	0.84	
SigmaFFPu	0.84	SigmaFFNu	-0.23	Tensor form factor
SigmaFFPs	-0.046	SigmaFFNs	-0.046	

All physical constants used in relic density calculations are defined in the file `include/micromegas_aux.h`, they are listed in Table 4.

Table 3: Evaluated global variables

Name	units	comments	Evaluated by
Ncdm	integer	number of thermal sectors for DM particles.	sortOddParticles
CDM[k]	*character*	name of the lightest particles in each sector k=1...Ncdm	sortOddParticles
McdmN[k]	GeV	Mass of CDM[k]	sortOddParticles
Mcdm	GeV	minimal mass of odd particles	sortOddParticles
fracCDM[k]		fraction of CDM[k] in relic density.	darkOmega*
dmAsymm		Asymmetry between relic density of DM- $\overline{DM}$	darkOmega[FO]
Tstart, Tend	GeV	Temperature interval for solving the differential equation	darkOmega[2]

Table 4: Some useful constants included in `micrOMEGAs`

Name	Value	Units	Description
`MPlank`	1.22091×10^{19}	GeV	Planck mass
`EntropyNow`	2.8912×10^{9}	m^{-3}	Present day entropy, s_0
`RhoCrit100`	10.537	GeVm^{-3}	ρ_c/h^2 or ρ for $H = 100\mathrm{km/s/Mpc}$

5 Setting of model parameters, spectrum calculation, parameter display

The independent parameters of a given model are specified in `work/models/vars1.mdl`. Three functions can be used to set the values of these parameters:

- `assignVal(name,val)`
- `assignValW(name,val)`

assign value *val* to parameter *name*. The function `assignVal` returns a non-zero value if it cannot recognize a parameter name while `assignValW` writes an error message.

- `readVar(fileName)`

reads parameters from a file. The file should contain two columns with the following format (see also Section 3.6)

```
name    value
```

`readVar` returns zero when the file has been read successfully, a negative value when the file cannot be opened for reading and a positive value corresponding to the line where a wrong file record was found.

The constrained parameters of the model are stored in `work/models/func1.mdl`. Some of these parameters are treated as *public* parameters. The *public* parameters include by default all particle masses and all parameters whose calculation requires external functions (except simple mathematical functions like $\sin, \cos$, ...). The parameters needed for the calculation of any *public* parameters in `work/models/func1.mdl` are also treated as *public*. It is possible to enlarge the list of *public* parameters. There are two ways to do

this. One can type * before a parameter name to make it *public* or one can add a special record in `work/models/func1.mdl`

```
%Local! |
```

Then all parameters listed above this record become *public*.

The calculation of the particle spectrum and of all *public* model constraints is done with:

• `sortOddParticles(txt)`
This routine has to be called after a reassignment of any input model parameter, after changing the sets of particles in thermal equilibrium (section 6.5), and after defining the set of feeble particles (section 6.7). The routine calculates the constrained parameters of the model. This routine returns a non zero error code for a wrong set of parameters, for example parameters for which some constraint cannot be calculated. The name of the corresponding constraint is written in `txt`. This routine also defines the number of dark sectors containing particles in chemical equilibrium, `Ncdm`, and finds the name of the lightest particle in each sector `CDM[k]` (k=1...Ncdm) as well as the minimal mass in each sector, `McdmN[k]`. It also defines the mass of the lightest dark particle `Mcdm`, this particle can either be a WIMP or a FIMP. [1]

• `qNumbers(pName, &spin2,&charge3,&cdim)`
returns the quantum numbers for the particle `pName`. Here `spin2` is twice the spin of the particle; `charge3` is three times the electric charge; `cdim` is the dimension of the representation of $SU(3)_c$, it can be $1, 3, -3, 6, -6$ or 8. The parameters `spin2, charge3, cdim` are variables of type `int`. The value returned is the `PDG` code. If `pName` does not correspond to any particle of the model then `qNumbers` returns zero.

• `pdg2name(nPDG)`
returns the name of the particle which PDG code is *nPDG*. If this particle does not exist in the model the return value is NULL.

• `antiParticle(pName)`
returns the name of the anti-particle for the particle `pName`.

• `pMass(pName)`
returns the numerical value of the particle mass.

• `nextOdd(n, &pMass)`
returns the name and mass of the n^{th} odd particle assuming that particles are sorted according to increasing masses. For $n = 0$ the output specifies the name and the mass of the CDM candidate.

• `findVal(name,&val)`
finds the value of variable *name* and assigns it to parameter *val*. It returns a non-zero value if it cannot recognize a parameter name.

• `findValW(name)`
returns the value of variable *name* and writes an error message if it cannot recognize a parameter name.

[1] Note that `Mcdm1` and `Mcdm2` that were defined for two DM models are not defined anymore and have been replaced by `McdmN[1]` and `McdmN[2]`.

The variables accessible by these last two commands are all free parameters and the constrained parameters of the model (in file `model/func1.mdl`) treated as *public*.

The following routines are used to display the value of the independent and the constrained *public* parameters:

• `printVar(FD)`
prints the numerical values of all independent and *public* constrained parameters into `FD`

• `printMasses(FD, sort)`
prints the masses of 'odd' particles (those whose names started with `~`). If $sort \neq 0$ the masses are sorted so the mass of the CDM is given first.

• `printHiggs(FD, sort)`
prints the masses and widths of 'even' colorless scalars.

6 Relic density calculation

6.1 Switches and auxilary routines

• `VWdecay,VZdecay`
Switches to turn on/off processes with off-shell gauge bosons in the final state for DM annihilation and particle decays. If `VW/VZdecay=1`, the 3-body final states will be computed for annihilation processes only while if `VW/VZdecay=2` they will be included in coannihilation processes as well. By default the switches are set to (`VW/VZdecay=1`).[2] Note that `micrOMEGAs` calculates the width of each particle only once and stores the result in *Decay Table*. A second call to the function `pWidth` (whether an explicit call or within the computation of a cross section) will return the same result even if the user has changed the `VW/VZdecay` switch. We recommend to call

• `cleanDecayTable()`
after changing the switches to force `micrOMEGAs` to recalculate the widths taking into account the new value of `VW/VZdecay`. The `sortOddParticles` command which must be used to recompute the particle spectrum after changing the model parameters also clears the decay table.

• `useSLHAwidth`
Switch to determine how the particle widths are computed. If `=1` the particle widths stored in a SLHA file (SUSY Les Houches Accord [**?**]) are downloaded by `micrOMEGAs`. These widths then do not depend on the `VW/VZdecay` switches. If `=0` micrOMEGAs will calculate the widths, it will also do so if the switch is set to 1 and the widths are not provided in the SLHA file. By default this swith is set to 0.

The thermodynamics of the Universe is determined by the effective numbers of degrees of freedom $h_{eff}(T)$ and $g_{eff}(T)$,

$$\rho_R(T) = \frac{\pi^2}{30} g_{eff}(T) T^4. \quad \text{and} \quad s(T) = \frac{2\pi^2}{45} h_{eff}(T) T^3. \tag{1}$$

[2]Including the 3-body final states can significantly increase the execution time for the relic density computation.

Note that h_{eff} and g_{eff} are related by the equation [3]

$$\frac{d\rho}{dT} = T\frac{ds}{dT} \tag{2}$$

These functions can be called with

• `gEff(T)`

which returns the effective number of degrees of freedom for the energy density of radiation at a bath temperature `T`, only SM particles are included.

• `hEff(T)`

which returns the effective number of degrees of freedom for the entropy density of radiation at a bath temperature `T`.

By default the tables for h_{eff}, g_{eff} correspond to the ones in Ref. [**?**] and can be found in the file `sources/hgEff/DHS.thg`. These default tables can be changed using

• `loadHeffGeff(char*fname)`

that reads the file *fname* located in the directory `sources/hgEff`. This file should contain 3 columns for T, $h_{eff}(T)$ and $g_{eff}(T)$. A positive return value corresponds to the number of lines in the table. A negative return value indicates the line which creates a problem (e.g. wrong format), the routine returns zero when the file `fname` cannot be opened.

The directory `sources/hgEff` also contains solutions described in Ref. [**?**], `LM.thg`, and in Ref. [**?**], `HP_B.thg, HP_C.thg`, as well as the tables used in DarkSUSY, `GG.thg`. The latter was used as default in previous versions of micrOMEGAs and does not include the contribution of the Higgs boson.

• `hEffLnDiff(T)`

returns the derivative of h_{eff} with respect to the *log* of the bath temperature, $\frac{d\log(h_{eff}(T))}{d\log(T)}$.

• `Hubble(T)`

returns the Hubble expansion rate in `GeV` units at a bath temperature `T`[GeV]. `Hubble` is defined via the density of the Universe and includes the contribution of radiation, dark matter, baryonic matter and dark energy,

$$H = \sqrt{\frac{8\pi\rho(T)}{3M_P^2}} \tag{3}$$

$$\rho(T) = \frac{\pi^2}{30}g_{eff}(T)T^4 + \mu_M\frac{2\pi^2}{45}h_{eff}(T)T^3 + \mu_{DE}^4 \tag{4}$$

where M_P is the Planck mass, $\mu_M = 0.519$ eV, $\mu_{DE} = 2.24\ 10^{-3}$ eV.

Entropy conservation

$$\frac{ds(T)}{dt} = -3Hs(T) \tag{5}$$

allows to write a relation between time and temperature:

$$\frac{dt}{dT} = -\left(1 + \frac{1}{3}\frac{d\log(h_{eff}(T))}{d\log(T)}\right)\frac{1}{H(T)T} \tag{6}$$

In particular micrOMEGAs has a function

• `HubbleTime(T1,T2)`

[3] This equation is not valid for $T \leq 1$ MeV, where photons and neutrinos have different temperatures.

which calculates the time interval in seconds during which the temperature of the Universe decreases from `T1[GeV]` to `T2[GeV]`.

$$t = \int_{T_2}^{T_1} \left(-\frac{dt}{dT}\right) dT \tag{7}$$

The constant `T2_73K` gives the current temperature T= 2.725K. The age of the Universe calculated as `Hubble(10,T2_73K)` is 13.806 Gyr in agreement with the value quoted in the Particle Data Group [?] of 13.797(23) Gyr.

• `freeStreaming(p/m, T1,T2)`
calculates the length of the trajectory in `Mpc` units for a freely propagating particle between `T1[GeV]` and `T2`. The parameter `p/m` characterizes the initial velocity of the particle $v = \frac{p/m}{\sqrt{1+(p/m)^2}}$. Usually the free streaming length, λ_{FS}, is obtained by the integral over time

$$\lambda_{FS} = \int_{t_2}^{t_1} \frac{v(t)}{a(t)} dt \tag{8}$$

where $a(t)$ is the scale factor, in particular $a = (s(T2\_73K)/s(T))^{\frac{1}{3}}$. In the code we compute the free-streaming length as integral over T.

$$\lambda_{FS} = \int_{T_2}^{T_1} \left(1 + \left(\frac{a(T)m}{a(T_1)p}\right)^2\right)^{-\frac{1}{2}} \frac{1}{a(t)} \left(-\frac{dt}{dT}\right) dT \tag{9}$$

• `improveCrossSection( n1,n2,n3,n4, Pcm, &cs)`
allows to substitute a new cross-section for a given process instead of the one calculated by micrOMEGAs at tree level. Here `n1,n2` are the PDG codes for particles in the initial state and `n3,n4` for those in the final state. `Pcm` is the center of mass momentum and `cs` is the cross-section in $[GeV^{-2}]$. This function is called just after the calculation of the annihilation cross section in routines that calculates the relic density and indirect detection. `micrOMEGAs` calls this routine substituting for the last parameter the address of the memory where the calculated tree level cross section `cs` is stored. This function is useful if, for example, the user wants to include their loop improved cross-section calculation and/or the Sommerfeld effect. `micrOMEGAs` contains a dummy version of this routine disposed in `sources/improveCS.c` which does not modify the default cross section. This file also contains some commented out example of the code for the IDM model. To activate this facility the user has to write their own version of the `improveCrossSection` routine and place it in the directory `MODEL/lib`, then the *dummy* version will be ignored.

• `SYukawa(a,b)` and `SHulthen(a,b)`
These functions can be used to calculate Sommerfeld enhancement in the `improveCrossSection` routine for Yukawa and Hulthen potentials for s-channel scattering. The arguments are

$$a = \alpha/v \tag{10}$$

$$b = m_{med}/(m_r v) \tag{11}$$

where v is the relative velocity of colliding particles, m_{med} is the mass of the mediator, $m_r = \frac{m_1 m_2}{m_1+m_2}$ - the reduced mass of colliding particles and $\alpha = \frac{e^2}{4\pi}$ where the coupling e is

defined from the Lagrangian that describes interactions of a Dirac particle (f_D) or of a charged scalar field (ϕ) with a scalar (h) or a vector (V_μ) mediator,

$$\begin{aligned} \mathcal{L} &= eh\bar{f}_D f_D, \quad \mathcal{L} = ev_\mu \bar{f}_D \gamma^\mu f_D \\ \mathcal{L} &= 2eM_\phi h\phi\phi^*, \quad \mathcal{L} = ieV_\mu(\partial^\mu\phi h\phi^* - \phi\partial^\mu\phi^*) \end{aligned} \tag{12}$$

For Majorana particles $\mathcal{L}$ contains an extra factor $\frac{1}{2}$. The Sommerfeld factor for pseudo-scalar and axial-vector interactions is not available in micrOMEGAs, since it has been shown to be negligible [**?**]. The calculation of `SYukawa` is based on Eq.5.1-5.4 of [**?**]. Here we solve numerically the differential equation for DM elastic scattering in spherical coordinates and compute the wave function at zero to get the enhancement factor. For the Hulthen potential we use an analytical solution described in Eq. 5.6 of [**?**] . A demonstration of the Sommerfeld enhancement for both `SYukawa` and `SHulthen` can be found in the file `mdlIndep/Sommerfeld.c`.

6.2 Temperature interval and Error codes for routines calculating relic density

micrOMEGAs provides three routines, `darkOmega, darkOmega2, darkOmegaN` for calculating the relic density of one-component, two-component and N-component DM respectively by solving differential equations for the abundances

$$Y_i = n_i/\mathfrak{s} \tag{13}$$

where n_i is the number density of the i^{th}-component DM and $\mathfrak{s}$ is the entropy density. The equations contain the thermal equilibrium abundance $\overline{Y_i} = \frac{\overline{n_i}}{\mathfrak{s}}$ as well as $\chi\chi \to SM, SM$ annihilation cross sections $v\sigma(T)$. Decays of dark sector particles and processes such as $\chi, SM \to \chi', SM$ are taken into account only in `darkOmegaN`. Abundances are calculated at a temperature `Tend` that can be defined by the user. The default value is `Tend`=10^{-3}`GeV` since in general the evolution of the relic abundance stops at higher temperatures. The initial temperature for integration, `Tstart`, is set by the condition

$$Y(Tstart) - \bar{Y}(Tstart) \approx 0.1\bar{Y}(Tstart).$$

These functions are described in the following subsections.

micrOMEGAs provides as well the routines `darkOmegaTR, darkOmega2TR, darkOmegaNTR` which also calculate the DM relic density. However the input parameters for these routines specify the initial temperature and abundances. The initial temperature is assigned to the variable `Tstart`.

All the `darkOmega*` routines have an *&err* parameter which returns the following error code :

```
32  - no WIMP
64  - Tstart is not found. It means that one of the  DM component was never
        in thermal equilibrium  with SM particles.
128 - problem in solution of differential equation. It can appear if the
      equation is stiff because Tstart is very large.
```

The `darkOmega*` routines return *NAN* if one of these error appears.

For calculating $v\sigma(T)$ we use the program *simpson* to evaluate the integrals over scattering angle and energy of collisions. This program can return the following error codes

```
1 - NAN in integrand;
2 - too deep recursion;
4 - loss of precision.
```

which are passed to *&err*. In general, these codes can be treated as warnings, although it can be useful to check the calculation of the problematic integrals using e.g. the `gdb` debugging tools. More information on this tool can be found in section 17.1. The error code `err` is a binary code which can signal several problems simultaneously.

6.3 Calculation of relic density for one-component Dark Matter models

All routines to calculate the relic density in version 3 are available in further versions. For these routines, the difference between dark sectors is ignored and the dark matter if the lightest particle among all those whose names starts with a `~`. These routines are intended for models with either a Z_2 or Z_3 discrete symmetry.

• `vSigmaA(T,fast,Beps)`, `vSigmaS(T,fast,Beps)`
calculates the thermally averaged cross section for DM annihilation times velocity at a temperature T [GeV], $\sigma_v = \langle v\sigma \rangle$ for DM annihilation (A) and semi-annihilation. (S),

$$< v\sigma^A >_T = \frac{T}{8\pi^4 \overline{n}(T)^2} \int ds \sqrt{s} K_1\left(\frac{\sqrt{s}}{T}\right) \sum_{\tilde{\alpha},\tilde{\beta}} p^2_{\tilde{\alpha}\tilde{\beta}}(s) g_{\tilde{\alpha}} g_{\tilde{\beta}} \sum_{x \geq y} \sigma_{\tilde{\alpha}\tilde{\beta} \to xy}(s) \qquad (14)$$

$$< v\sigma^S >_T = \frac{T}{8\pi^4 \overline{n}(T)^2} \int ds \sqrt{s} K_1\left(\frac{\sqrt{s}}{T}\right) \sum_{\tilde{\alpha},\tilde{\beta}} p^2_{\tilde{\alpha}\tilde{\beta}}(s) g_{\tilde{\alpha}} g_{\tilde{\beta}} \sum_{x\tilde{\gamma}} \sigma_{\tilde{\alpha}\tilde{\beta} \to x\tilde{\gamma}}(s) \qquad (15)$$

where

$$\overline{n}(T) = \frac{T}{2\pi^2} \sum_{\tilde{\alpha}} g_{\tilde{\alpha}} m^2_{\tilde{\alpha}} K_2(\frac{m_{\tilde{\alpha}}}{T}), \qquad (16)$$

is the equilibrium number density of DM particles. Here $\tilde{\alpha}$, $\tilde{\beta}$, $\tilde{\gamma}$ is used for `Odd` particles and x,y for `Even` particles. Here only $2 \to 2$ processes are included and $\sigma_{\tilde{\alpha}\tilde{\beta} \to x[\tilde{\gamma}/y]}$ is the cross section for the corresponding process averaged over the spins of incoming particles and summed over the spins of outgoing particles. K_1, K_2 are modified Bessel functions of the second kind, and $m_{\tilde{\alpha}}$ and $g_{\tilde{\alpha}}$ stand for the mass and the number of degrees of freedom of particle $\tilde{\alpha}$. Note, that if $\tilde{\alpha} \neq \tilde{\beta}$ then each $\sigma_{\tilde{\alpha}\tilde{\beta}}$ term will be presented twice. The value for $v\sigma$ is expressed in $[pb \cdot c]$. The parameter *Beps* defines the criteria for including coannihilation channels as for `darkOmega` described below. The $fast = 1/0$ option switches between the *fast*/*accurate* calculations.

Note, that if $\tilde{\alpha} \neq \tilde{\beta}$, both $\sigma_{\tilde{\alpha}\tilde{\beta}}$ and $\sigma_{\tilde{\beta}\tilde{\alpha}} = \sigma_{\tilde{\alpha}\tilde{\beta}}$ will contribute to the sum. Moreover, the sum over α in Eq.14,15,16 runs over each particle and anti-particle separately.

In the low temperature limit

$$< v\sigma^A >_T = \frac{1}{\overline{n}(T)^2} \sum_{\tilde{\alpha},\tilde{\beta}} \sum_{x \geq y} \bar{n}_{\tilde{\alpha}}(T) \bar{n}_{\tilde{\beta}}(T) \lim_{v \to 0} v\sigma_{\tilde{\alpha}\tilde{\beta} \to xy}(v) \tag{17}$$

where v is the relative velocity. When only one particle and its anti-particle contribute to DM annihilation, Eq.17 reads

$$< v\sigma^A >_T = \frac{1}{2} \lim_{v \to 0} (v\sigma_{\tilde{\alpha},\tilde{\alpha} \to SM,SM}(v) + v\sigma_{\tilde{\alpha},\tilde{\alpha} \to SM,SM}(v)) \tag{18}$$

In many models, for example when DM is a Dirac fermion, there are no process $\tilde{\alpha}, \tilde{\alpha} \to SM, SM$ and our definition of $< v\sigma >_T$ looks unnatural. However this definition has the advantage that it leads to a universal formula for the number of annihilation events in unit of space-time volume, regardless of the nature of DM,

$$\frac{dN}{dt d^3x} = \frac{1}{2} < v\sigma >_T n^2 \tag{19}$$

where n is the number density of DM particles[4].

After a call to `vSigmaA` or `vSigmaS`, the global array `vSigmaTCh` contains the contribution of different channels to `vSigma`. `vSigmaTCh[i].weight` specifies the relative weight of the i^{th} channel,
`vSigmaTCh[i].prtcl[j]` *(j=0, 4)* defines the particles names for the i^{th} channel.
The last record in `vSigmaTCh` array has zero weight and NULL particle names.

In terms of $v\sigma^A$ and $v\sigma^S$ the abundance equation reads

$$\frac{dY}{dt} = - < v\sigma^A >_T (Y^2 - \bar{Y}^2) - \frac{< v\sigma^S >_T}{2} (Y^2 - Y\bar{Y}) \tag{20}$$

• `vSigmaCC(T,cc,mode)`
calculates the thermally averaged *cross section* × *velocity* for $2 \to 2$, $2 \to 3$, and $2 \to 4$ processes. `T` is the temperature in [GeV], *cc* is the address of the code for each process. This address can be obtained by the function `newProcess` presented in Section 12. The returned value is given in [c·pb].

If *mode* $\neq 0$, `vSigmaCC` calculates the contribution of a given process to the total annihilation cross section, see Eq.14,15. The incoming particles should belong to the odd sector. For $2 \to 2$ processes the result after summation over all subprocesses should be identical to the one obtained via `vSigma` above. For this mode, `vSigmaCC` includes combinatorial factors: 2 if $\tilde{\alpha} \neq \tilde{\beta}$, an additional factor 2 if the incoming state is not self-conjugated, and a factor $\frac{1}{2}$ for semi-annihilation.

If *mode* $= 0$, `vSigmaCC` is defined by the integral

$$< v\sigma^{\tilde{\alpha}\tilde{\beta} \to X} >_T = \frac{1}{2T m_{\tilde{\alpha}}^2 m_{\tilde{\beta}}^2 K_2(\frac{m_{\tilde{\alpha}}}{T}) K_2(\frac{m_{\tilde{\beta}}}{T})} \int ds \sqrt{s} K_1(\frac{\sqrt{s}}{T}) p_{cm}^2(s) \sigma^{\tilde{\alpha}\tilde{\beta} \to X}(p_{cm}(s))$$

[4]{Eq.19 is valid if we assume a Maxwell-Boltzmann distribution, as is done for all routines that compute the relic density of WIMPs, the treatment of quantum statistics in the case of FIMPs is discussed in section 6.7 .

where p_{cm} is the center of mass momentum of incoming particles. Note that

$$\lim_{T\to 0} \text{vSigmaCC(T, cc, 0)} = \lim_{p_{cm}\to 0} \sigma(p_{cm}) v_{rel}(p_{cm})$$

where $v_{rel}(p_{cm})$ is the relative velocity of incoming particles. The result of `vSigmaCC` can be different from that of `vSigma` described above when there is an important contribution from NLSP's to the total number density of DM particles.

• `darkOmega(&Xf,fast,Beps,&err)`
calculates the dark matter relic density Ωh^2. This routine solves the differential evolution equation using the Runge-Kutta method. $X_f = Mcdm/T_f$ characterizes the freeze-out temperature which is defined by the condition $Y(T_f) = 2.5\overline{Y}(T_f)$. For asymmetric DM this condition reads $2\sqrt{Y^+(T_f)Y^-(T_f)} = 2.5\overline{Y}(T_f)$. The value of X_f is given for information and is also used as an input for the routine that gives the relative contribution of each channel to Ωh^2, see `printChannels` below. The $fast = 1$ flag forces the fast calculation (for more details see Ref. [**?**]). This is the recommended option and gives an accuracy around 1%. The parameter `Beps` defines the criteria for including a given coannihilation channel in the computation of the thermally averaged cross-section, [**?**]. The recommended value is $Beps = 10^{-4} - 10^{-6}$ whereas if $Beps = 1$ only annihilation of the lightest odd particle is computed. Non-zero error code means that the temperature where thermal equilibrium between the DM and SM sectors is too large $Mcdm/T < 2$ or $T > 10^5$GeV.

`darkOmega` solves the differential equation for the abundance $Y(T)$ in two temperature intervals `[Tstart,Tf]` and `[Tf,Tend]` [**?**]. The temperatures are defined by the conditions $Y(T_{start}) \approx 1.1\overline{Y}(T_{start})$, $Y(T_f) \approx 10\overline{Y}(T_f)$ while `Tend` is defined by the user. In the second interval, `[Tf,Tend]`, the contribution of $\overline{Y}$ is neglected and the differential equation is integrated explicitly. The solution in the interval `[Tstart,Tf]` is tabulated and can be displayed via the function `YF(T)`. The equilibrium abundance can be accessed with the function `Yeq(T)`.

• `darkOmegaFO(&Xf, fast, Beps)`
calculates the dark matter relic density Ωh^2 using the freeze-out approximation.

• `printChannels(Xf,cut,Beps,prcnt,FD)`
writes into `FD` the contributions of different channels to $(\Omega h^2)^{-1}$. Here `Xf` is an input parameter which should be evaluated first in `darkOmega[FO]`. Only the channels whose relative contribution is larger than `cut` will be displayed. `Beps` plays the same role as in the `darkOmega[FO]` routine. If $prcnt \neq 0$ the contributions are given in percent. Note that for this specific purpose we use the freeze-out approximation.

• `oneChannel(Xf,Beps,p1,p2,p3,p4)`
calculates the relative contribution of the channel `p1,p2` → `p3,p4` to $(\Omega h^2)^{-1}$. p1,...,p4 are particle names. To sum over several channels one can write `"*"` instead of a particle name, *e.g* `"*"` in place of p1.

• `omegaCh` is an array that contains the relative contribution and particle names for each annihilation channel. These array and function are similar to `vSigmaTCh` described above.

The array `omegaCh` if filled after calling either `darkOmegaFO` or `printChannels`.

• `darkOmegaTR(Tstart,Ystart, fast, Besp)`
This function is similar to `darkOmega` except that the initial temperature (`Tstart`) and abundance (`Ystart`) that are needed for solving the evolution equation have to be provided by the user as arguments to the function.

There is an option to calculate the relic density in models with DM-$\overline{\text{DM}}$ asymmetry. In this case, we assume that the number difference of DM-$\overline{\text{DM}}$ is conserved in all reactions. Thus a small difference in initial abundances can lead to a large DM asymmetry after freeze-out as is the case for the baryon asymmetry.

• `deltaY`
describes the difference between the DM and anti-DM abundances for the models where the number of DM particles minus the number of anti-DM is conserved in decays and collisions. In such models `deltaY` is a constant during the thermal evolution of the Universe, see Ref. [**?**].
• `dmAsymm`
is defined by the equation

$$\Omega_{\pm} = \Omega \frac{1 \pm \text{dmAsymm}}{2}$$

and evaluated by `micrOMEGAs` while calculating the relic density with an initial asymmetry `deltaY`, see [**?**]. This parameter can also be reset after the relic density computation and will then be taken into account for direct and indirect detection rates.
• `darkOmegaExt(&Xf, vs_a, vs_sa)`
calculates the dark matter relic density Ωh^2 using annihilation cross sections provided by external functions. Here `vs_a` is the annihilation cross section in [c·pb] as a function of the temperature in [GeV] units, Eq.14, while `vs_sa` is the semi-annihilation cross section, Eq.15. `vs_a` is required for all models, while `vs_sa` is relevant only for models where semi-annihilation occurs. The user can substitute `NULL` for `vs_sa` when semi-annihilation is not possible.

`darkOmegaExt` can also be used if processes other than $2 \to 2$ processes contribute to DM annihilation. In this case the appropriate annihilation or semi-annihilation cross sections can be calculated by `vSigmaCC` and the tabulated results stored in `vs_a` and `vs_sa`.

`darkOmegaExt` solves the Runge-Kutta equation in the interval `[Tstart, Tend]` where Tstart is defined automatically while `Tend` has a fixed value 10^{-3} GeV. `darkOmegaExt` is sensitive to effect of DM asymmetry.

One important application of `darkOmegaExt` is that it can be used to take into account off shell resonances in the calculation of the relic density. For this one has to first compute the corresponding $2 \to 3$ and/or $2 \to 4$ processes, this can be done with the functions:

• `vSigmaPlus24(proc22, T, &err)`
which calculates the contribution to $v\sigma$ for a $2 \to 4$ process associated with the $2 \to 2$ process for which outgoing particles are off-shell. Here `proc22` is the name of the $2 \to 2$ process and `T` is the temperature. For each virtual particle, the decay channel with the largest branching fraction is chosen, the result is then divided by the corresponding branching fractions.

• `vSigmaPlus23(proc22, T, &err)`
calculates the contribution to $v\sigma$ for a $2 \to 3$ process associated with the $2 \to 2$ process (`proc22`) for which one of the outgoing particles can be off-shell. The contribution of the on-shell $2 \to 2$ process (for example $\chi\chi \to BB'$) is subtracted. First, the 3-body cross-section for the process corresponding to one real and one virtual final state (corresponding to the main decay channel of the virtual particle) is computed. To ensure the proper behaviour near threshold one would need to compute the 4-body final state, rather we use a trick to approximate the 4-body result from the 3-body result after applying a K-factor. To obtain the K factor we assume that the 4-body matrix element is proportional to

$$|\mathcal{M}_4^2| \propto \left[m_{B'}^2 m_B^2 + \frac{1}{8}(s - m_{B'}^2 - m_B^2)^2\right] \times \tag{21}$$

$$\frac{\Gamma_B}{((m_B^2 - m_B'^2)^2 - \Gamma_B^2 m_B'^2)} \frac{\Gamma'_B}{((m_{B'}^2 - m_{B'}'^2)^2 - \Gamma_{B'}^2 m_{B'}'^2)} \tag{22}$$

where $m_B, m_{B'}$ and $m'_B, m'_{B'}$ are respectively the virtual and pole masses of B and B', $\Gamma_B, \Gamma_{B'}$ the widths of the virtual particles, and s is taken in the range $2m_\chi < \sqrt{s} < m_B + m_{B'} + 10(\Gamma_B + \Gamma_{B'})$. We integrate this matrix element with $\Gamma_B = 0$ in order to simulate the 3-body result, then we integrate the same matrix element with the correct value for Γ_B. The ratio of these integrals gives the K factor. The same trick is applied automatically for calculating matrix elements with virtual W and Z if the flags `VWdecay` and `VZdecay` are activated. see Section 6.1 [**?**].

When first called, both `vSigmaPlus23` and `vSigmaPlus24` tabulate the cross sections of $2 \to 3$ and $2 \to 4$ processes respectively and keep results in memory for subsequent calls to calculate integrals over s. A call to `sortOddParticles()` cleans the tabulated cross sections.

To take into account the off-shell contribution in the computation of the relic density one has to create a new function which sums the contributions of `vSigmaA` and `vSigmaPlus23` and `vSigmaPlus24` and pass this function to `darkOmegaExt`. We have checked that both `vSigmaPlus23` and `vSigmaPlus24` give similar results.

The functions presented in this section are used to calculate the relic density of a single WIMP particle. If they are used for a model with multiple dark sectors, it will assume that all particles of the dark sectors are in chemical equilibrium with each other and will solve only for the relic density of the lightest particle of the dark sectors. The fractional contribution of heavier DM particles to the total relic density will be set to zero. A proper calculation of multi-component DM must rather rely on the routines described in sections 6.4 and 6.5. If the DM is feeble (FIMP), special routines have to be used to compute the relic density, see Section 6.7. The particles that are marked as *feeble* are just ignored by the routines described in this section.

6.4 Calculation of relic density for two-component Dark Matter models

• `darkOmega2(fast, Beps)`
Calculates Ωh^2 for either one- or two-components DM models. In the former case it should give the same result as `darkOmega`. The parameters `fast` and `Beps` have the same

meaning as for the darkOmega routine. The returned value corresponds to the sum of the contribution of the two DM components to Ωh^2.

The two-component DM are referred to as CDM[1] and CDM[2] and correspond to the lightest particle in each sector. darkOmega2 also fills the elements of the array fracCDM[1], fracCDM[2] which contains the mass fraction of CDM[1] and CDM[2] in the total relic density, namely,

$$\texttt{fracCDM[i]} = \frac{\Omega_i}{\Omega_1 + \Omega_2} \tag{23}$$

These fractions are then used in routines which calculate the total signal from both DM components in direct and indirect detection experiments, nucleusRecoil, calcSpectrum, and neutrinoFlux. The user can change the global fracCDM[i] parameter before the calculation of these observables to take into account the fact that the value of the DM fraction in the Milky Way could be different than in the early Universe.

The routines that were described in section 6.3 are not available for two-component DM models. In particular the individual channel contribution to the relic density cannot be computed and DM asymmetry is ignored. After calling darkOmega2 the user can check the cross sections for each class of reactions (but not for individual processes) which were tabulated during the calculation of the relic density. The functions

• vs*abcd* F(T)

computes the sum of the cross sections for each class of reactions $(a, b, c, d = 0, 1, 2)$ tabulated during the calculation of the relic density. Here T is the temperature in [GeV] and the return value is $v\sigma$ in [c·pb]. These functions are defined in the interval [Tstart , Tend], where Tstart is a global parameter defined by darkOmega2, Tend=10^{-3}GeV. Specifically the functions available are

vs1100F	vs1110F	vs1120F	vs1112F	vs1122F	vs1210F	vs1211F
vs1220F	vs1222F	vs2200F	vs2210F	vs2220F	vs2211F	vs2221F

The temperature dependence of the abundances can also be called by the user, the functions are named Y1F(T) and Y2F(T) and are defined only in the interval T ∈ [Tend, Tstart]. The equilibrium abundances are accessible via the Yeq1(T), Yeq2(T) functions and the deviation from equilibrium by the functions dY1F(T)= Y1F(T)-Y1eq(T) and dY2F(T)=Y2F(T)-Y2eq(T).

• darkOmega2TR(Tstart, Y1start, Y2start, fast, Beps)

This function is similar to darkOmega2 described above, however the initial temperature (Tstart) and abundances (Y1start, Y2start) that are needed for solving the evolution equations should be passed by the user as arguments to the function.

As was the case for routines for one component DM, darkOmega2 ignores particles declared as feeble. However one can use darkOmega2TR for a model which contains both a FIMP and a WIMP by specifying e.g. Y1start= Yeq for the WIMP in sector 1 and Y2start= 0 for the FIMP in sector 2. Note that the decay processes between particles of different sectors are not taken into account, for models, where these can be important we recommend rather to use darkOmegaN described in the next section.

6.5 Calculation of relic density for N-component DM taking into account coscattering processes.

In this section we present routines for the calculation of the relic density of N-component DM. The dark particles need to be divided into *sectors*, within each of which chemical equilibrium is observed. By default, this separation is defined by the number of ~ symbols in the beginning of the particle names. Thus, `~x1` and `~~x2` denote dark particles of two different sectors. Usually, the sector assignment corresponds to the charge of the discrete symmetry responsible for DM stability, cf. section 2. However, in the absence of chemical equilibrium, the splitting into sectors needs to be done differently.

There are two types of processes which are responsible for chemical equilibrium: decays and coscattering (or DM conversion) processes. The latter corresponds to processes of the type

$$\chi_i, SM \to \chi_i', SM' \tag{24}$$

In general the decay processes are responsible for chemical equilibrium at low temperatures, while coscattering processes maintain chemical equilibrium at high temperatures. The darkOmegaN routine calculates DM abundances taking into account both coscattering and decay processes which relate different sectors.

In the absence of chemical equilibrium, the splitting into sectors is done using the function

• `defThermalSet(n, particles_list)` which moves all particles mentioned in *particles_list* to sector n. All particles that were assigned to sector n before this command are returned to their default sectors specified by the number of ~ in the beginning of their names. Particles in the *particles_list* have to be separated by commas, and particle and anti-particle automatically belong to the same sector. By definition, sector 0 is the SM bath while sector -1 is used to define *feeble* particles which do not take part in freeze out. Such particles will be ignored when solving for the relic density. Sectors $n > 0$, are used for all other cases.

In general, `defThermalSet` can define a set which includes particles with different charges of the discrete symmetry group (different number of ~ symbols) — in particular the set could include Z_2 odd particles as well as SM particles. In this case the user must keep in mind that the abundance equations are solved for sectors n ¿ 0 only. This entails that a Z_2 odd particle assigned to sector 0 will not be considered as potential DM candidate. The function returns an error code if *particles_list* contains a particle name which is not defined in the model.

• `printThermalSets()` prints the contents of all particle sets specified by `defChEset` on the screen.

To verify whether chemical equilibrium is reached in one sector, one can use

• `checkTE( n, T, mode, Beps)` which checks the condition for chemical equilibrium in the n^{th} sector at temperature `T`. If mode=0, then both decay and co-scattering are taken into account. If mode=1 (2), then only decay (co-scattering) processes are taken into account. `checkTE` returns the minimal value of $\Gamma/H(T)$ obtained after testing all possible subsets of particles in sector n. The particle assignment corresponding to the minimal value of $\Gamma/H(T)$ is printed on the screen. This value has to be $\gg X_f$ to have chemical equilibrium, when this condition is satisfied the correction to the abundance calculated assuming chemical equilibrium is approximately $\Delta Y/Y \approx X_f H/\Gamma$.

• `YdmNEq(T,`α`)` calculates the thermal equilibrium abundance for particles of sector α,

where α has to be presented by a text label. For instance, `YdmNeq(T,"1")`.
• `vSigmaN(T, channel)` calculates the thermally averaged cross section $\langle v\sigma\rangle$ in `[pb·c]` units. Here *channel* is a text code specifying the reaction, e.g. `vSigmaN(T,"1100")` for $1,1 \leftrightarrow 0,0$ processes. If *channel* starts with an exclamation mark, then *vSigmaN* returns the results of the previous call of `darkOmegaN`. Otherwise `vSigmaN` recalculates all needed cross sections using the parameters *fast, Beps* defined in previous call of `darkOmegaN` or the parameters defined by a call to
• `setFastBeps(fast,Beps)`

To find the contribution of different processes to `vSigmaN`, one can call
• `vSigmaNCh(T, channel, fast, Beps, &vsPb)` which returns an array of annihilation processes together with their relative contributions to the total annihilation cross section. The cross section is given by the return parameter `vsPb` in [pb c] units. The elements of the array are sorted according to weights and the last element has weight=0. The structure of this array is identical to vSigmaTCh which was defined for one-DM models, see 6.3. The input parameter `channel` is written in text format. The memory allocated by outCh can be cleaned after usage with the command free(outCh). The following lines of code give an example on how to use this function

```
aChannel*outCh=vSigmaNCh(T, "1100",  Beps, &vsPb);
for(int n=0;;n++)
{  if(outCh[n].weight==0) break;
   printf(" %.2E  %s %s -> %s %s\n", outCh[n].weight,
   outCh[n].prtcl[0], outCh[n].prtcl[1], outCh[n].prtcl[2], outCh[n].prtcl[3]);
}
free(outCh);
```

• `darkOmegaNTR(TR, Y, fast, Beps, &err)` solves the equation of the thermal evolution of abundances starting from the initial temperature `TR` and returns the total Ωh^2 as described in [?]. The array `Y` has to contain the initial abundances at the temperature `TR`. After completion, `Y[k]` contains the abundances of sector $k-1$ at the temperature `Tend` defined by the user [5]. The parameter `TR` is assigned to the global variable `Tstart`.

• `darkOmegaN(fast, Beps, &err)` calls `darkOmegaNTR` to solve the equations of the thermal evolution of abundances in the temperature interval `[Tend,Tstart]`. In each sector, the function looks for the temperature T_i where $Y_i(T_i) \approx Y_{eq}(T_i)$. The minimum value of T_i is assigned to `Tstart`. i If `Tstart` is not found, then the error code 64 is generated and `darkOmegaN` returns NaN.
• `YdmN(T,`α`)` presents the evolution of abundances for particles of sector α calculated by `darkOmegaN` or `darkOmegaNTR` for $T \in$ `[Tend,Tstart]`.

For the above functions, `micrOMEGAs` provides the possibility to selectively exclude part of the terms in the evolution equation. This is realized via the string `ExcludedForNDM`,

[5] By default `Tend`=10^{-3}GeV, however when the decay contribution is important it is preferable to choose a smaller value such as `Tend`=10^{-8}GeV .

which can be assigned specific keywords. The keyword `"DMdecay"` excludes decay processes which contribute to the DM evolution, while the keyword `"1100"` excludes $1,1 \leftrightarrow 0,0$ processes. to exclude coscattering ($2,0 \leftrightarrow 1,0$ or $1,0 \leftrightarrow 2,0$) processes, set

```
ExcludedForNDM="2010";
```

To reset and include all channels one must use `ExcludedForNDM=NULL;`.

Note that, for the computation of coscattering, the user defines which particles belong to sector 1 and sector 2. If all particles are in thermal equilibrium but one of them is nevertheless assigned to sector 2 which contains no other particle, the two abundance equations will be solved and should give the same result as the single abundance equation, that is $Y(\text{heavier particles}) = 0$ and $Y(\text{lightest particle}) = Y$ of the single equation.

Finally, in the coscattering phase kinetic equilibrium may be lost; it is the responsibility of the user to verify that this is not the case for the parameters considered, so that the momentum-integrated Boltzmann equations remain a good approximation.

6.6 Scenario with slow Inflaton decay

During cosmic reheating, the inflaton ϕ is assumed to decay into SM radiation with a total decay width Γ. The dynamics of the background is driven by the set of Boltzmann equations for the inflaton energy density ρ_ϕ and the SM entropy density, [**?**]

$$\frac{d\rho_\phi}{dt} = -\Gamma\rho_\phi - 3H\rho_\phi \tag{25}$$

$$\frac{ds}{dt} = \Gamma\frac{\rho_\phi}{T} - 3Hs \tag{26}$$

where $s = \frac{2\pi^2}{45}h_{eff}(T)T^3$. The Hubble expansion rate is given by

$$H = \sqrt{\frac{8\pi}{3}\frac{\rho_\phi + \rho_R}{M_P^2}} \tag{27}$$

and the SM energy density, ρ_R, by

$$\rho_R = \frac{\pi^2}{30}g_{eff}(T)T^4. \tag{28}$$

The function

•`getInflDecay(HI, Gamma, &Trh, &Tmax, &aEnd)`

solves Eqs. (25) and (26). The solution is stored in the tabulated functions Ta(a) and Ha(a), which respectively return the temperature and the Hubble parameter as a function of the scale parameter $a \in [1, \text{aEnd}]$. These functions are available to the user. Here, `HI` is the Hubble rate at the end of inflation when HI is generated by the inflaton only. `Gamma` is Γ_ϕ, the decay width of the inflaton. `Trh, Tmax` and `aEnd` are return parameters. They represent respectively the temperature when the energy density of inflation becomes equal to the energy density of SM particles, the maximal temperature reached, and the expansion factor at the temperature `Tend`. This temperature is defined by the user as a

global parameter. The function `getInflDecay` returns 0 when it can find a solution to the differential equations or 128 otherwise.

The function

•`darkOmegaInfl(Beps, &isWimp, &err)`

solves the DM evolution equation in Eq. (??). It uses Ta and Ha constructed by `getInflDecay`. The main return value of this routine is the DM relic density $\Omega_\chi h^2$. The auxiliary output parameters are *integer* `isWIMP`, and *integer* `err`. The parameter `isWIMP` indicates whether the DM candidate behaves as a WIMP, that is, if the final relic density decreases when the cross section increases, then `isWIMP=1`, otherwise `isWIMP=0`. The error code `err` indicates whether there is a problem with the solution of the differential equations (26) and (25), in which case the 8^{th} bit of the error code `err` is 1, or with the solution of Eq. (??), then the 9^{th} bit is 1. The lowest bits contain information about problems with numerical integration and should be treated as warnings. The parameter `Beps` has the same meaning as in other `darkOmega` routines; it determines the condition to include Boltzmann suppressed channels [?].

The evolution of the DM abundance is stored in an array and is available via the functions `Za(a)`, `ZaEq(a)`, `Ya(a)`, while the equilibrium abundance is stored in `YaEq(a)`.

There is a similar function for N-components DM:

•`darkOmegaNInfl`(H_I,Γ, Beps, &a_{end}, &T_{rh}, &err)

Here the argument `isWimp` is omitted because some components can present Wimp, whereas other prFimps. Solution can be controlled by functions `ZaN, YaN, ZaNEq, YaNEq, vSigmaN` similar to ones presented in section 6.5.

6.7 Calculation of relic density for freeze-in

Several routines are provided in `micrOMEGAs` to compute the DM abundance in freeze-in scenarios. These can be found in the file `sources/freezein.c`. The first line of this file contains the statement

`//#define NOSTATISTICS`

This statement can be uncommented for `micrOMEGAs` to compute the relic density assuming a Maxwell-Boltzman distribution. This option is faster.

The auxiliary functions that are needed for the computation of the factors from statistical quantum mechanics are

• `Stat2`$(P/T, x_Y, x_1, x_2, \eta_1, \eta_2)$,

returns the S function defined in Eq. (29), that takes into account particle statistical distributions for the decay of a mediator $Y \to a, b$ of fixed momentum P.

$$S\left(P/T, x_Y, x_a, x_b, \eta_a, \eta_b\right) = \frac{1}{2}\int_{-1}^{1} dc_\theta \frac{e^{E_Y/T}}{(e^{E_a(c_\theta)/T} - \eta_a)(e^{E_b(c_\theta)/T} - \eta_b)} \ . \tag{29}$$

where $x_i \equiv m_i/T$, E_a, E_b are the energies of the outgoing particles and $\eta_i \equiv \pm e^{\mu_i/T}$ and μ_i is the chemical potential.

• `K1to2`$(x_1, x_2, x_3, \eta_1, \eta_2, \eta_3)$,

returns the $\tilde{K}_1$ function defined in Eq. (30), that takes into account particle statistical

distributions.

$$\tilde{K}_1\left(x_1,x_2,x_3,\eta_1,\eta_2,\eta_3\right) \equiv \frac{1}{(4\pi)^2 p_{\rm CM} T}\int \prod_{i=1}^{3}\left(\frac{d^3p_i}{E_i}\frac{1}{e^{E_i/T}-\eta_i}\right) e^{E_1/T}\delta^4\left(P_1-P_2-P_3\right) \tag{30}$$

The code does *not* check whether or not a particle is in thermal equilibrium with the SM thermal bath and that it is the responsability of the user to specify which particles belong to the bath, $\mathcal{B}$, or are out of equilibrium, $\mathcal{F}$. This can be done through the function

• `toFeebleList(particle_name)`

which assigns the particle `particle_name` to the list of feebly interacting ones (*i.e.* those which belong to $\mathcal{F}$). Feebly interacting particles can be odd or even. This function can be called several times to include more than one particle. All odd or even particles that are not in this list are assumed to be in thermal equilibrium with the SM and belong to $\mathcal{B}$. The treatment of the particles that belong to $\mathcal{F}$ for the computation of Ωh^2 within the freeze-in routines is described below. Calling `toFeebleList(NULL)` will reassign all particles to $\mathcal{B}$.

The actual computation of the freeze-in DM abundance can be performed with the help of three functions:

• `darkOmegaFiDecay(TR, bathParticle, feebleParticle)`

calculates the abundance of `feebleParticle` resulting from the decay of `bathParticle` The equations used for the three different cases are described in [**?**].

For the freeze-in routines that compute $2 \to 2$ processes described below (`darkOmegaFi` and `darkOmegaFi22`, the effect of the thermal masses of particles in the t/u - channel can be simulated by introducing a cut on the Mandelstam variables t or u. The thermal masses, M_T, are defined by the user in the array `Tkappa`,

$$M_T = T \cdot Tkappa[k] \tag{31}$$

where `k` in the internal particle number. This number can be obtained by

```
k=abs(pTabPos(pName))-1
```

The cut is applied on $t > t_{max} - M_T^2$ for a t-channel propagator and on $u > u_{max} - M_T^2$ for a u-channel propagator. By default `Tkappa[k]=0` for all particles.

• `darkOmegaFi22(TR, Process, feebleParticle, &err)`

calculates the DM abundance of `feebleParticle` taking into account only $2 \to 2$ `Process`. For example `"b,B ->~x1,~x1"` for the production of DM (here `~x1`) via $b\bar{b}$ scattering. This routine allows the user to extract the contribution of individual processes. `TR` is the reheating temperature. `err` is the returned error code, it has the following meaning

1: the requested processes does not exist
2: $2 \to 2$ process is expected
3: can not calculate local parameters // some constrain parameters can not be calculated.
4: $T_{end} >= T_R$, or $T_{end} = 0$
5: one of the incoming particles belong to $\mathcal{F}$.
6: None of the outgoing particles are odd and feeble.

7: Lost of precision in temperature integrand
8: Pole in temperature integrand
9: NAN in temperature integrand
10: Lost of precision in sqrt(s) integrand
11: Pole in sqrt(s) integrand
12: NAN in sqrt(s) integrand
13: Lost of precision in angle integrand
14: Pole in angle integrand
15: NAN in angle integrand
16: lost of precision caused by diagram cancellation

• `darkOmegaFi(TR,feebleParticle  &err)`
calculates the DM abundance after summing over all $2 \to 2$ processes involving particles in the bath $\mathcal{B}$ in the initial state and at least one particle in $\mathcal{F}$ in the final state. The routine checks the decay modes of all bath particles and if one of them has no decay modes into two other bath particles, the $2 \to 2$ processes involving this particle are removed from the summation and instead the contribution to the DM abundance computed from the routine `darkOmegaFiDecay` is included in the sum. This is done to avoid appearance of poles in the corresponding $2 \to 2$ cross-section. We recommend for such models to compute individual $2 \to 2$ processes with `darkOmegaFi22` described above. As before, we assume that all odd FIMPs will decay into the lightest one which is the DM. T_R has the same meaning as above. `err` is the returned error code, `err=1` if feeble particles have not been defined.

• `printChannelsFi(cut,prcnt,filename)`
writes into the file `filename` the contribution of different channels to Ωh^2. The `cut` parameter specifies the lowest relative contribution to be printed. If `prcnt` $\neq 0$, the contributions are given in percent. The routine `darkOmegaFi` fills the array `omegaFiCh` which contains the contribution of different channels ($2 \to 2$ or $1 \to 2$) to Ωh^2. omegaFiCh[i].weight specifies the relative weight of the ith channel, omegaFiCh[i].prtcl[j] (j=0, 4) defines the particles names for the ith channel. The last record in the array `omegaFiCh` has zero weight and NULL particle names.

The temperature evolution of the abundances generated by these three routines can be obtained by calling the function `YFi(T)` in the interval $T \in$ `[Tend, Tstart]` where `Tstart` is set internally by the three routines and is assigned to the variable `TR`.

Note that if no particle has been declared as being feebly interacting, the freeze-out routines `darkOmega`, `darkOmegaFO`, and `darkOmega2` [?] will work exactly like in previous versions of `micrOMEGAs`. A non-empty `feeblelist`, however, will affect these routines since `micrOMEGAs` will exclude all the particles in this list from the computation of the relic density via freeze-out. To compute the relic density of particles in `feeblelist` one has to use the `darkOmegaFi` function (or the other routines describe in this section) for each of these particles . Alternatively the relic density of feeble particles can be computed using the `darkOmega*TR` routines described in section 6.4,6.5 by setting the initial abundance of these particles to zero. Note that the result can differ from the one obtained by the

darkOmegaFi routines since these take into account the Maxwell-Boltzmann statistics. [6]

7 Direct detection

7.1 Amplitudes for elastic scattering

• nucleonAmplitudes(CDM,pAsi,pAsd,nAsi,nAsd)
calculates the amplitudes for CDM-nucleon elastic scattering at zero momentum. pAsi(nAsi) are spin independent amplitudes for protons(neutrons) whereas pAsd(nAsd) are the corresponding spin dependent amplitudes. Each of these four parameters is an array of dimension 2. The zeroth (first) element of these arrays gives the χ-nucleon amplitudes whereas the second element gives $\overline{\chi}$-nucleon amplitudes. Amplitudes (in GeV^{-2}) are normalized such that the total cross section for either χ or $\overline{\chi}$ cross sections is

$$\sigma_{tot} = \frac{4M_\chi^2 M_N^2}{\pi(M_\chi + M_N)^2}(|A^{SI}|^2 + 3|A^{SD}|^2) \tag{32}$$

nucleonAmplitudes depends implicitly on form factors which describe the quark contents in the nucleon. These form factors are global parameters (see Table 1 for default values)

$$Type\text{FFP}q \quad Type\text{FFN}q$$

where $Type$ is either "Scalar", "pVector", or "Sigma", FFP and FFN denote proton and neutron and q specifies the quark, d, u or s. Heavy quark coefficients are calculated automatically.

micrOMEGAs automatically takes into account loop contributions from box diagrams as calculated in [?] (DM spin 1/2 case) and [?] (DM spin 0 and 1 cases).

nucleonAmplitudes does not take into account an operator involving DM-DM-gluon-gluon interactions. It is by default assumed that such interactions are generated at one-loop from DM interactions with heavy colored particles and this loop-induced contribution of heavy coloured particles to DM-DM-gluon-gluon interactions [7] is taken into account automatically. Thus a direct contribution might lead to double counting. Note that to avoid double counting an Hgg vertex in the model file will also be ignored. To include the contribution of an additional operator that involves DM interactions with gluons, it is rather recommended to introduce a new auxiliary heavy color (dim=3) fermion in the model file, the direct detection amplitude that results from the effective operator $\alpha_s G_{\mu\nu} G^{\mu\nu} \bar{\chi}\chi$ can be represented by a term $-12\pi M_F \bar{F} F \bar{\chi}\chi$, where M_F is the fermion mass which should be chosen large enough to ensure that F does not contribute to other processes.

• calcScalarQuarkFF(m_u/m_d,m_s/m_d,$\sigma_{\pi N}$,σ_s)
computes the scalar coefficients for the quark content in the nucleon from the quark mass ratios $m_u/m_d, m_s/m_d$ as well as from $\sigma_{\pi N}$ and σ_s. The default values given in Table 2 are obtained for $\sigma_s = 42\text{MeV}, \sigma_{\pi N} = 34\text{MeV}$, $m_u/m_d = 0.56, m_s/m_d = 20.2$ [?]. The function calcScalarQuarkFF(0.553,18.9,55.,243.5) will reproduce the default values of the scalar quark form factors used in micrOMEGAs2.4 and earlier versions.

[6] Note that in previous versions of micrOMEGAs the relic density of darkOmegaFi was rescaled when it was not the lightest particle in the dark sector to take into account the fact that the FIMP will eventually decay into the DM. In this version this rescaling has to be done by the user.

[7] Heavy quarks include c,b,t quarks

7.2 Scattering on nuclei

• `nucleusRecoil(f,A,Z,J,Sxx,dNdE)`
This is the main routine of the direct detection module. The input parameters are:

◇ `f` - the DM velocity distribution normalised such that

$$\int_0^\infty f(v)dv = 1 \tag{33}$$

The units are km/s for v and s/km for f(v).

◇ `A` - atomic number of nucleus;

◇ `Z` - number of protons in the nucleus;

◇ `J` - nucleus spin;

◇ `void Sxx(double p, double*S00,double*S01,double*S11)` - is a routine which calculates nucleus form factors for spin-dependent interactions (`S00,S01,S11`), it depends on the momentum transfer p in fm^{-1}. For nucleus with zero spin one has to substitute NULL for this parameter;

◇ The distribution over recoil energy is stored in the array `dNdE`

$$dNdE[n] = \frac{dN}{dE_n}$$

in units of (1/keV/kg/day) where

```
En=RE_START*RE_STEP**n;   0<=n<RE_DIM
#define RE_DIM     150    // dimension of recoil energy array
#define RE_START   1.E-2  // energy [keV] corresponding to zero'th element
#define RE_STEP    1.08   // factor for  recoil energy grid
                          // E_{n+1}=E_n*RE_STEP
```

The `include/micromegas.h` file contains predefined values for charges and spins of atomic nucleus. They are called with $Z\_\{Name\}$ and $J\_\{Name\}\{atomic\_number\}$ where $Name$ is the name of an atom. For example, for ^{73}Ge, a call to this routine will be:

```
nucleusRecoil(Maxwell,73,Z_Ge,J_Ge73,SxxGe73,dNdE);
```

The returned value of the function `nucleusRecoil` gives the number of events per day and per kilogram of detector material. The result depends implicitly on the global parameter `rhoDM`, the density of DM near the Earth. This routine is based on the spin-dependent and spin-independent cross sections, but also takes into account the modification of the recoil energy distribution caused by a light t- channel propagator. For a light mediator, the nucleus recoil energy distribution reads

$$\frac{dN_A}{dE} = \frac{M_{\rm med}^4}{(M_{\rm med}^2 + 2M_A E)^2} \frac{dN_A^{std}(M_{DM}, \sigma_0)}{dE} \tag{34}$$

where N_A^{std} is the standard expression for the number of recoil events for a point-like interaction with a DM- nucleon elastic scattering cross-section σ_0, $M_{\rm med}$ is the mass of the t-channel mediator. The factor to correct the recoil energy distribution as in Eq.34 is introduced automatically in the code after an internal check for the existence of a light mediator. For a complex WIMP and if DM has only one component, `nucleusRecoil` averages over χ and $\overline{\chi}$ taking into account the asymmetry between χ and $\overline{\chi}$. For models with 2 DM particles the result takes into account the relative contribution of each DM particle through the parameter `fracCDM2`.

• `dNdERecoil(E,dNdE)` interpolates the `dNdE` table.

7.2.1 Nucleus form factors

The form factors for the spin independent (SI) cross section are defined by the Fermi distribution

$$F_A(q) = \int e^{-qx}\rho_A(|x|)d^3x \quad (35)$$

$$\rho_A(r) = \frac{c_{\rm norm}}{1+exp((r-R_A)/a} \quad (36)$$

$$R_A = cA^{\frac{1}{3}}+b \quad (37)$$

where a, b, c are defined with the global parameters `Fermi_a`, `Fermi_b`, `Fermi_c`.

The spin dependent form factors collected in [?] are implemented in micrOMEGAs [?]:

```
SxxF19   SxxNa23   SxxNa23A  SxxAl27   SxxSi29   SxxSi29A
SxxK39   SxxGe73   SxxGe73A  SxxNb92   SxxTe125  SxxTe125A
SxxI127  SxxI127A  SxxXe129  SxxXe129A SxxXe129M
SxxXe131 SxxXe131A SxxXe131B SxxXe131Me
```

Here the characters after the atomic number are used to distinguish different implementations of the form factor for the same isotope. Recently we have added some form factors used in XENON1T and PICO-60 experiments. See Table 5.

Name	reference	ID
SxxF19EF SxxXe131EFT SxxXe129EFT	[?]	EFT
SxxF19SHELL SxxXe129SHELL SxxXe131SHELL	[?]	SHELL
SxxF19SHELLm SxxXe129SHELLm SxxXe131SHELLm	[?]	SHELLm

Table 5: Implemented form factors

The minimal and maximal values for the SD form factors, $S_{00}(q), S_{01}(q), S_{11}(q)$, are computed in Ref. [?] within the shell model. The ID `SHELL` corresponds to the average

$$S_{ab} = (S_{ab}^{min} + S_{ab}^{max})/2 \tag{38}$$

which are obtained from the minimum and maximum fitted values in Table VI in [?].

The ID `SHELLm` corresponds to the form factors which lead to the most robust exclusion. Since S_{ab}^{min} often leads to a negative value for the subdominant component of the form factor, and this has no physical meaning, we use rather the minimum value of the proton-only, S_p^{min}, and neutron-only, S_n^{min}, form factors also given in [?]. These correspond to the minimal form factor for the case when only one type of interaction (with proton or neutron) is included. With this we construct the nucleus form factors

$$\begin{aligned} S_{00} &= \frac{1}{4}\left(S_p^{min} + S_n^{min} \pm 2\sqrt{S_p^{min} S_n^{min}}\right) \\ S_{11} &= \frac{1}{4}\left(S_p^{min} + S_n^{min} \mp 2\sqrt{S_p^{min} S_n^{min}}\right) \\ S_{01} &= \frac{1}{2}(S_p^{min} - S_n^{min}) \end{aligned} \tag{39}$$

The sign in Eq.39 is chosen to reproduce the ratio $S_{00}(0)/S_{11}(0)$ for the central value of the form factors in Ref. [?].

For nuclei whose form factors are not known one can use the routine

• `nucleusRecoil0(f,A,Z,J,Sp,Sn,dNdE)`

which is similar to the function `nucleusRecoil` except that the spin dependent nuclei form factors are described by Gauss functions [?] whose values at zero momentum transfer are defined by the coefficients `Sp,Sn`. Predefined values for the coefficients `Sp,Sn` in the format

$$Sp\_\{Nucleus\ Name\}\{Atomic\ Number\}$$
$$Sn\_\{Nucleus\ Name\}\{Atomic\ Number\}$$

are presented in the file `include/micromegas.h`. For example,

```
#define Sn_He3      0.552
#define Sn_O17      0.5
```

7.2.2 Velocity distribution

Ignoring the direction of motion of DM particles and the small effect of DM acceleration by the gravitational field of the Sun, the DM velocity distribution in the vicinity of the direct detection experiment is given by

$$f(\mathrm{v}) = \int\limits_{|\vec{v}|<\mathrm{vEsc}} d^3\vec{v} F_G(\vec{v} - \vec{v}_{Earth})\delta(\mathrm{v} - |\vec{v}|) \tag{40}$$

where F_G is the DM velocity distribution in the frame of the galaxy, $\vec{v}_{Earth}$ is the velocity of the Earth in the Galaxy and `vEsc` is the maximal DM velocity in the Sun's orbit due

to the finite gravitational potential of our Galaxy. `vEsc` and `vEarth`=$|\vec{v}_{Earth}|$ are global parameters of micrOMEGAs.

The velocity distributions that are available in `micrOMEGAs` are the following

• `Maxwell(v)`

returns

$$F_G^M(\mathrm{v}) = c_{\mathrm{norm}} \frac{1}{(2\pi \mathrm{vRot}^2)^{3/2}} \exp\left(-\frac{(\vec{v})^2}{\mathrm{vRot}^2}\right) \theta(\mathrm{vEsc} - |\vec{v}|) \tag{41}$$

which corresponds to the isothermal model. Here `vRot` is the orbital velocity of stars in the Milky Way, it is also a global parameter of micrOMEGAs. $c_{\mathtt{norm}}$ is the normalization factor,

$$c_{\mathtt{norm}}^{-1} = \mathrm{erf}\left(\frac{\mathrm{vEsc}}{\mathrm{vRot}}\right) - \frac{2}{\sqrt{\pi}} \frac{\mathrm{vEsc}}{\mathrm{vRot}} \exp\left(-\frac{\mathrm{vEsc}^2}{\mathrm{vRot}^2}\right) \tag{42}$$

• `SHMpp(v)`

returns the velocity distribution `SHM++` proposed in [**?**].

$$F_G(\vec{v}) = (1-\eta) F_G^M(\mathrm{v}) + \eta F_G^S(\mathrm{v}) \tag{43}$$

This distribution consists of two components. The first, $F_G^M(\vec{v})$, is the standard Maxwell velocity distribution described above. The second component is the velocity distribution from the *Gaia* sausage [**?**, **?**], it is not spherically symmetric and is defined by the anisotropy parameter β with

$$F_G^S(\vec{v}) = \frac{c_{\mathrm{norm}}}{(2\pi)^{3/2} \Delta v_r \Delta v_\theta \Delta v_\phi} \exp\left(-\left(\frac{v_r}{\Delta v_r}\right)^2 - \left(\frac{v_\theta}{\Delta v_\theta}\right)^2 - \left(\frac{v_\phi}{\Delta v_\phi}\right)^2\right) \theta(\mathrm{vEsc} - |\vec{v}|) \tag{44}$$

where

$$\Delta v_r = \frac{\mathrm{vRot}}{\sqrt{1 - \frac{2}{3}\beta}}\,, \quad \Delta v_\phi = \Delta v_\theta = \frac{\mathrm{vRot}\sqrt{1-\beta}}{\sqrt{1 - \frac{2}{3}\beta}} \tag{45}$$

and

$$c_{\mathtt{norm}}^{-1} = \mathrm{erf}\left(\frac{\mathrm{vEsc}}{\mathrm{vRot}}\right) - \left(\frac{1-\beta}{\beta}\right)^{1/2} \exp\left(-\frac{\mathrm{vEsc}^2}{\mathrm{vRot}^2}\right) \mathrm{erfi}\left(\frac{\mathrm{vEsc}}{\mathrm{vRot}} \frac{\beta^{1/2}}{(1-\beta)^{1/2}}\right) \tag{46}$$

where erfi is the imaginary error function.

The central values and uncertainties of the `SHM++` parameters are

$$\begin{aligned} \mathsf{rhoDM} &= 0.55 \pm 0.17 \text{ GeV/cm}^3 \\ \mathsf{vRot} &= 233 \pm 3 \text{ km/s} \\ \mathsf{vEsc} &= 580 \pm 63 \text{ km/s} \\ \beta = \mathsf{betaSHMpp} &= 0.9 \pm 0.05 \\ \eta = \mathsf{etaSHMpp} &= 0.2 \pm 0.1 \end{aligned} \tag{47}$$

Note that these central values for the global parameters, `vRot, vEsc` and `rhoDM` are different from the ones in Table 1, thus the user has to change these parameters before using SHMpp.

7.3 Comparison with Direct Detection experiments

Examples on how to use the routines described below to impose constraints from direct detection experiments can be found in `mdlIndep/dd_exp.c` of micrOMEGAs.

7.3.1 Experimental data

The SI 90% DD limits tabulated from the results presented by LZ5T [?], LZ$_2$024 []XENON1T [?], DarkSide-50 [?], PICO-60 [?] and CRESST-III [?] are accessible through the following functions

- `LZ_2024(Mdm)` for $9 < M_{DM} < 10000$ GeV, [?]
- `LZ5T(Mdm)` for $9 < M_{DM} < 10000$ GeV, [?]
- `PandaX4T(Mdm)` for $5 < M_{DM} < 10000$ GeV, [?]
- `XENON1T_90(Mdm)` for $6 < M_{DM} < 1000$ GeV, [?]
- `DS50_90(Mdm)` for $0.7 < M_{DM} < 15$ GeV, [?]
- `PICO60_90(Mdm)` for $3 < M_{DM} < 10000$ GeV, [?]
- `CRESST_III_90(Mdm)` for $0.35 < M_{DM} < 12$ GeV. [?]

The corresponding SD 90% exclusion limits are contained in the functions

- `PICO60_SDp_90(Mdm)` for $3 < M_{DM} < 10000$ GeV, [?]
- `XENON1T_SDp_90(Mdm)` for $6 < M_{DM} < 1000$ GeV, [?]
- `XENON1T_SDn_90(Mdm)` for $6 < M_{DM} < 1000$ GeV, [?]
- `CRESST_III_SDn_90(Mdm)` for $0.35 < M_{DM} < 12$ GeV. [?]

These functions give the excluded cross sections in cm^2 . For a DM mass outside the range specified the function returns NAN.

7.3.2 Recasting the experimental limits with micrOMEGAs

- `DD_pvalCS`(`expCode`, f_v, $\sigma_{SI_P}, \sigma_{SI_N}, \sigma_{SD_P}, \sigma_{SD_N}$,&expName)
calculates the value $\alpha = 1 - C.L.$ for a model with DM-nucleon cross sections $\sigma_{SI_P}, \sigma_{SI_N}, \sigma_{SD_P}, \sigma_{SD_N}$. Cross sections are specified in [pb] units. The return value 0.1 corresponds to a 90% exclusion. The `expCode` parameter can be any of the codes `LZ5Tmedian`,`XENON1T_2018`,`DarkSide_2018`, `CRESST_2019`,`PICO_2019` or their combination concatenated with the symbol |. There is also a predefined parameter that currently combines these experiments

```
AllDDexp=LZ5Tmedian|XENON1T_2018|DarkSide_2018|PICO_2019|CRESST_2019;
```

The parameter `char* expName` is used to indicate the experiment that provides the best exclusion among those specified in `expCode`. The function `DD_pvalCS` calculates the exclusion for each experiment independently, returns the smallest α, and assigns the name of the corresponding experiment to `expName` if it is not `NULL`.

The f_v parameter specifies the DM velocity distribution in the detector frame. For example, one can use `Maxwell` or `SHMpp` which are included in `micrOMEGAs`, otherwise the user can define another distribution. The DM velocity distribution has to be normalized as in Eq.33. The units are km/s for v and s/km for $f_v(v)$. `DD_pvalCS` implicitly depends

on the global parameters `Mcdm` and `rhoDM` which specify the DM mass and DM local density respectively.

For Xenon1T one can chose between three recasting, p^q_{eff} with $q = 0, 1, 2$, see Ref. [**?**]. The flag `Xe1TnEvents=q` allows to choose the corresponding recasting, otherwise and by default the code uses p^1_{eff}. The three approaches agree within 5%. For PICO-60, the user can choose between the recasting based on Feldman-Cousins statistics, `PICO60Flag=0` which is the default value, or the one based on Neyman one side belt exclusion, `PICO60Flag=1`.

• `DD_factorCS(expCode,` α, f_v, $\sigma_{SI_P}, \sigma_{SI_N}, \sigma_{SD_P}, \sigma_{SD_N}$,&expName)
returns the overall factor which should be applied to the cross sections, $\sigma_{SI_P}, \sigma_{SI_N}, \sigma_{SD_P}, \sigma_{SD_N}$ to reach the exclusion level α. All parameters are the same as in `DD_pvalCS` above.

• `*dNdEFact(Enr_kev, A)`
is the address of the function which modifies the nucleus recoil distribution for `DD_pvalCS` and `DD_factorCS` to take into account a t-channel propagator with small or zero mass. By default `dNdEfact=NULL` and this function does not contribute to the calculation of the direct detection cross sections. Otherwise it is taken as an additional factor in the nucleus recoil distribution, see Eq.34. The parameter `Enr_kev` is the recoil energy in [keV] units, A is the atomic number of the nucleus. This function should be defined by the user, an example is given in `mdlIndep/dd_exp.c`.

• `DD_pval(expCode,` f_v,&expName)
• `DD_factor(expCode,` α, f_v,&expName)
These functions are similar to `DD_pvalCS` and `DD_factorCS` described above but use the cross section calculated from the DM model under consideration in micrOMEGAs. The necessary corrections for a light mediator are implemented automatically, these functions do not use `dNdEFact`.

7.3.3 Setting spin-dependent form factors

The spin dependent form factors for XENON1T and PICO60 are defined via the parameters

`XENON1T_2018_sdXe129` `XENON1T_2018_sdXe131` `PICO_2019_sdF19`

The default values for these parameters correspond to the ones used by the experiments, namely

`SxxXe129SHELL` `SxxXe131SHELL` `SxxF19EFT`

They can be replaced by any form factor listed in Section 7.2.1 through direct assignment. Form factors can also be changed using the routine
• `setSpinDepFF(ExperimentID, setID)`
where the choice for `ExperimentID` is given in Section7.3.2 and `setID` can be
`EFT` - corresponding to the form factors in [**?**]
`SHELL` - corresponding to the average form factors in [**?**], Eq. 38
`SHELLm` - corresponding to the minimal form factor of [**?**], Eq. 39. See section 7.2.1.

8 Indirect detection

8.1 Interpolation and display of spectra

Various spectra and fluxes of particles relevant for indirect detection are stored in arrays with **NZ** elements. The first (zeroth) element of the array contains the maximum energy E_{max}. As a rule E_{max} is the mass of the DM particle. The i^{th} element ($1 \leq i \leq NZ-1$) of the spectrum array contains the value of $E_i \frac{dN}{dE_i}$ where $E_i = E_{max} e^{Zi(i)}$, `Zi(i)=`$-7\log(10)\left(\frac{i-1}{250}\right)^{1.5}$. Here E is the kinetic energy $E = \sqrt{p^2+m^2} - m$. By default **NZ=250**, thus the array covers the energy interval $E_{max} \geq E > 1.101 \times 10^{-7} E_{max}$. The user can change the value of **NZ** defined in the file `include/micromegas.h` in order to treat smaller energies as described in section 8.2. To decode and interpolate the spectrum array one can use the following functions:

• `SpectdNdE(E,spectTab)`
interpolates the tabulated spectra and returns the particle distribution dN/dE where `E` is the energy in GeV. For a particle number distribution the returned value is given in `GeV`$^{-1}$ while a particle flux is expressed in `(sec cm`^{2} `sr GeV )`$^{-1}$.

• `eSpectdNdE(E,spectTab)`
returns $E \times$`SpectdNdE(E,spectTab)`.

• `addSpectrum(SpectTab,toAdd)`
sums the spectra `toAdd` and `SpectTab` and writes the result in `SpectTab`. For example, this routine can be useful for summing spectra with different maximal energy.
• `spectrMult(SpectTab, func)`
allows to multiply the spectrum `SpecTab` by any energy dependent function `func`
• `spectrInt(Emin,Emax,SpectTab)`
integrates a spectrum/flux, `SpecTab` from `Emin` to `Emax`.
• `spectrInfo(Emin,Spec,&Etot)`
provides information on the spectra. The returned value and `Etot` corresponds respectively to

$$N_{tot} = \int_{E_{min}}^{E_{max}} SpectdNdE(E, Spec)dE = spectrInt(E_{min}, E_{max}, Spec) \quad (48)$$

$$E_{tot} = \int_{E_{min}}^{E_{max}} E\ SpectdNdE(E, Spec)dE \quad (49)$$

where the first element of the table `Spec` contains the value of E_{max}. Alternatively the user can directly integrate the spectra using standard simpson routines, for example

$$\begin{aligned} N_{tot} &= \text{simpson\_arg}(\text{SpectdNdE}, \text{Spec}, \text{E}_{\text{min}}, \text{Spec}[0], 0.01, \text{NULL}) \quad (50) \\ E_{tot} &= \text{simpson\_arg}(\text{eSpectdNdE}, \text{Spec}, \text{E}_{\text{min}}, \text{Spec}[0], 0.01, \text{NULL}) \end{aligned}$$

The spectrum can be displayed on the screen with
`displayPlot("Spectrum","E",E_min, Spec[0],0,1,"dNdE",0, SpectdNdE,Spec)`

To boost a spectrum one can use the command

• `boost(`γ`,Emax,m,SpectTab)`

where γ is the boost parameter, `SpectTab` is an array of NZ lines which contains the initial spectrum for the particle of mass `m`. After calling the boost command the array `SpectTab` will be overwritten with the new spectrum. The first element of the resulting array is `SpectTab[0]=Emax` unless the maximal energy of the particle after boost, E, exceeds `Emax`, then `SpectTab[0]=E`.

•`fillSpect(dNdE, Emax, SpectArray)`

This routine converts the spectrum `dNdE` which is a function of energy E into an array `SpectArray` which contains NZ elements. The $Emax$ parameter is assigned to `SpectArray[0]`.

• `FSRdNdE(E, p, m, q, spin2, med)`

computes the Final State Radiation spectrum, $\frac{dN}{dE_\gamma}$, for DM annihilation in charged particles with momentum `p` and mass `m`. The $spin2$ parameter is 0 for bosons and 1 for fermions. The parameter $med = 0$ if the particle and antiparticle are produced via a scalar mediator and 1 for a vector mediator. `E` is the energy of the photon. Here we use formulas 4.8 - 4.11 of [**?**]. Note that FSRdNdE contains the photon spectrum generated by both particle and anti-particle.

8.2 Annihilation spectra

DM pair annihilation into SM particles will lead to stable particles

$$\gamma, e^+, p^-, \nu_e, \nu_\mu, \nu_\tau, D^- \tag{51}$$

The resulting spectra which take into account final state radiation, hadronisation and decays were calculated by Pythia and tabulated.

• `basicSpectra(Mass,pdg,outN,Spectr)`

computes the spectra of outgoing particles and writes the result in an array of dimension NZ, `Spectr`, `pdg` is the PDG code of the particles produced in the annihilation of a pair of WIMPs. `outN` specifies the outgoing particle,

$$\text{outN} = \{0, 1, 2, 3, 4, 5, 6\} \text{ for } \{\gamma, e^+, p^-, \nu_e, \nu_\mu, \nu_\tau, D^-\}$$

The `Mass` parameter defines the mass of the DM particle.

`basicSpectra` returns an error code if the parameter `outN` (err=1) or `pdg` (err=2) is out or range. The spectrum depends on `SpectraFlag` which specifies the external package used to calculated the spectra, see below. In particular, the antideuterons spectra is provided only within PPPC. The `Mass` parameter has to be larger than 2 GeV if `SpectraFlag=0` and larger than 5 GeV otherwise. When `basicSpectra` is called for lower masses, `micrOMEGAs` extrapolates the spectra assuming that the xdN/dx distribution, where $x = E/Mass$, depends only weakly on the DM mass.

• `decaySpectrum(pName,outN,spectTab)`

calculates the spectrum of the outgoing particle `outN` from the decay of particle `pName` . The result is written in an array of dimension NZ, `spectTab`, `outN` specifies the outgoing

particle. `decaySpectrum` returns the error code 1 if `pName` is stable and 2 if $1 \to 2$ and $1 \to 3$ decay channels are not kinematically closed.

• `SpectraFlag`

is a switch to choose the tables that contain the spectra for $\gamma, e^+, \bar{p}, \nu$ from DM pair annihilation into two particle final states. This Flag has to be set before calling the functions `calcSpectrum` and `basicSpectra`.

`SpectraFlag=0` corresponds to the tables obtained with Pythia-6 [**?**] for DM mass in the range $2\text{GeV} - 5\text{TeV}$ for $e^+, \bar{p}, \nu$. For the photon channel the tables have been extended to cover the range $110\text{MeV} - 5\text{TeV}$. These tables include also the spectra for polarized W's and Z's. To get the spectra generated by transverse and longitudinal W's substitute `pdgN`$= 24 +' T'$ and $24 +' L'$ correspondingly. In the same manner `pdgN`$= 23 +' T'$ and $23 +' L'$ provides the spectra produced by a polarized Z boson.

`SpectraFlag=1` corresponds to the spectra generated by *Pythia-8* [**?**, **?**] for DM mass in the range $5\text{GeV} - 5\text{TeV}$, here polarization is not taken into account.

`SpectraFlag=2` corresponds to the spectra generated with *PPPC* [**?**, **?**] for DM mass larger than 5GeV. The spectra take into account polarized W and Z, as well as polarized leptons, *pgd= 11 +'R"* or *pgd= 11 +'L"*, the latter are however not supported in the current version of micrOMEGAs.

Finally `SpectraFlag=3` corresponds to the spectra provided in *CosmiXs* for DM in the range $5\text{GeV} - 100\text{TeV}$ [**?**]. These spectra are generated with PYTHIA and include electroweak corrections. These spectra also included longitudinal/transverse W and Z as well as left/right polarized charged leptons.

The *Pythia-8, PPPC,* and *CosmiXs* tables contain spectra that start at $x \gtrapprox 10^{-9}$, while the default spectra `SpectraFlag=0` start at $x \gtrapprox 10^{-7}$. To download the spectra including smaller energies one has to recompile `micrOMEGAs` after setting the parameter **NZ=295** in the file `include/micromegas.h`.

The maximal values of the x parameter in *Pythia-8* and *CosmiXs* tables are 0.94 and 0.9 respectively. For this reason, the spectral line for $DM, DM \to \gamma, \gamma$ which corresponds to a contribution $2\delta(1-x)$ in the photon spectra is not included, only the sub-dominant part of the photon spectra from photon radiation is included. In `micrOMEGAs` the δ-like contribution to photon spectra in the γ, γ channel are added to the *Pythia-8* and *CosmiXs* tables. In the same manner we add a $\delta(1-x)$ contribution to the neutrino spectra forthe $\nu, \bar{\nu}$ channel. In *PPPC* the corresponding δ-like contributions are simulated by a linear function in the interval $x \in [0.89, 1]$.

The QCD uncertainties that can impact the annihilation spectra were evaluated in [**?**]. For each DM mass, annihilation channel and choice of final state, Ref. [**?**] have estimated three types of QCD uncertainties and provide tabulated spectra including these uncertainties. These tables are now incorporated in the code. We have defined three parameters that determines which uncertainties are taken into account,

- DHad: uncertainties on the flux from the variations of the hadronisation model parameters.
- DScale: shower uncertainties on the flux from the variation of the shower evolution variable by a factor of 2.
- DcNS: uncertainties from the variations of the non-singular terms of the DGLAP splitting kernels.

By default these parameters are set to 1 and all types of uncertainties are taken into account, however any one can be switched off by setting the corresponding parameter to zero. The total uncertainties $\Delta^2(E)_{min,max}$ are obtained by taking the sum of the square of the individual uncertainties. The global parameter, `spectUncert` determines whether or not uncertainties are included in the spectra. When `spectUncert=0` , the default spectra without uncertainties, $(dN/dE)_{wo}$, are used, `spectUncert=1` means that `basicSpectra` returns the maximal spectra corresponding to $(dN/dE)_{wo}+\Delta(E)_{max}$ while `spectUncert=-1` the minimal spectra corresponding to $(dN/dE)_{wo}-\Delta(E)_{min}$ are used. Note that although these uncertainties were calculated for the *Pythia-8* spectra, micrOMEGAs uses the same uncertainties for other spectra as well. One can access the spectra corresponding to the specified value of `spectUncert` by calling

• `spectraUncertainty(Mass,pdgN,outN,Spectr)`

which has the same parameters as `basicSpectra`.

The test code `mdlIndep/basicSpectra.c` contains an example of a call to `basicSpectra` and compares the four different spectra that are included in micrOMEGAs, checks energy conservation and shows uncertainties.

• `calcSpectrum(key,Sg,Se,Sp,Sne,Snm,Snl,&err)`

calculates the spectra of DM annihilation at rest and returns σv in cm^3/s . For multicomponent DM the number of annihilation events in one cm^3 during one second at a distance R from the galactic center is given by

$$N(R) = \frac{1}{2}\sigma v \left(\frac{\rho(R)}{M_{cdm}^{eff}}\right)^2 \tag{52}$$

where

$$M_{cdm}^{eff} = \left(\sum_{i=1}^{Ncdm} \frac{fracCDM[i]}{McdmN[i]}\right)^{-1} \tag{53}$$

and $\rho(R)[GeV/cm^3]$ is the DM density. In the special case of one-component DM, $M_{cdm}^{eff} =$ `Mcdm` is the mass of DM.

The calculated spectra for γ, e^+, $\bar{p}$, ν_e, ν_μ, ν_τ are stored in arrays of dimension `NZ` as described above: `Sg`, `Se`, `Sp`, `Sne`, `Snm`, `Snl`. To remove the calculation of a given spectra, substitute `NULL` for the corresponding argument. `key` is a switch to include the polarisation of the W, Z bosons (`key=1`) or photon radiation (`key=2`). Note that final state photon radiation (FSR) is always included. When `key=2` the 3-body process $\chi\chi' \to XX + \gamma$ is computed for those subprocesses which either contain a light particle in the t-channel (of mass less than 1.2 Mcdm) or an outgoing W when `Mcdm>500GeV`. The FSR is then subtracted to avoid double counting. Only the electron/positron spectrum is modified with this switch. When `key=4` the contributions for each channel to the total annihilation rate are written on the screen. More than one option can be switched on simultaneously by adding the corresponding values for `key`. For example both the W polarization and photon radiation effects are included if `key=3`. A problem in the spectrum calculation will produce a non zero error code, $err \neq 0$. `calcSpectrum` uses `basicSpectra` to interpolate and sum spectra obtained by Pythia, thus it depends on the switch `SpectraFlag` .

• `vSigmaCh`
is an array that contains the relative contribution and particle names for each annihilation channel. It is similar to `vSigmaTCh` described in Section 6.2. Note that the list of particles contains five elements to allow to include gamma radiation. For `2->2` processes vSigmaCh[n].prtcl[4]=NULL. The array `vSigmaCh` is filled by `calcSpectrum`.

• `calcSpectrumPlus(proc22, outP, Spect, &err)`
calculates DM annihilation into 3-body and 4-body final states via the exchange of virtual particles. These processes are not taken into account by `calcSpectrum`. Here `proc22` specifies the $2 \to 2$ process such as "$\chi, \chi' \to X, Y$" which is kinematically forbidden at small relative velocity. `calcSpectrumPlus` includes all reactions with virtual `X` and `Y` particles. The parameter `outP` specifies the spectrum: 0 - photons, 1- positrons, 2- anti-protons, 3,4,5 - neutrinos. `Spect` is an array of size NZ which stores the calculated spectrum. The routine returns the value of $v\sigma$ in $[cm^3/s]$ units. If DM particles are not selfconjugated, the corresponding conjugated channel is added automatically.

8.3 Distribution of Dark Matter in Galaxy

The indirect DM detection signals depend on the DM density in our Galaxy. The DM density is given as the product of the local density at the Sun with the halo profile function

$$\rho(r) = \rho_\odot F_{halo}(r) \tag{54}$$

where we assume that

$$F_{halo}(R_\odot) = 1 \tag{55}$$

In `micrOMEGAs` $\rho_\odot$ is a global parameter `rhoDM` and $R_\odot$ is a global parameter `Rsun`. The default profile is the Zhao profile [**?**].

$$F_{halo}(r) = \left(\frac{R_\odot}{r}\right)^\gamma \left(\frac{r_c^\alpha + R_\odot^\alpha}{r_c^\alpha + r^\alpha}\right)^{\frac{\beta-\gamma}{\alpha}} \tag{56}$$

By default we use $\alpha = 1, \beta = 3, \gamma = 1$ which corresponds to the NFW profile and we set $r_c = 8.1[kpc]$ [**?**]. Note that we have modified the default value, it was $r_c = 20[kpc]$ in previous versions. The parameters of the Zhao profile can be reset by
• `setProfileZhao(`α,β,γ, r_c`)`
The function to set another density profile is
• `setHaloProfile(`F_{halo}`)`
where $F_{halo}(r)$ is any function which depends on the distance from the galactic center, r, defined in [kpc] units. For instance, `setHaloProfile(hProfileEinasto)` sets the Einasto profile

$$F_{halo}(r) = exp\left[-\frac{2}{\alpha}\left(\left(\frac{r}{r_2}\right)^\alpha - \left(\frac{R_\odot}{r_2}\right)^\alpha\right)\right] \tag{57}$$

where by default $\alpha = 2.33$, r_2 =10.7 [**?**], note that the default values in previous versions were $\alpha = 0.17$, r_2 =`Rsun`. The values can be changed by
• `setProfileEinasto(`α, r_2`)`.
The command `setHaloProfile(hProfileZhao)` sets back the Zhao profile.

Dark matter annihilation in the Galaxy depends on the average of the square of the DM density, $<\rho^2>$. This quantity can be significantly larger than $<\rho>^2$ when clumps of DM are present [**?**]. In `micrOMEGAs`, we use a simple model where f_{cl} is a constant that characterizes the fraction of the total density due to clumps and where all clumps occupy the same volume V_{cl} and have a constant density ρ_{cl}. Assuming clumps do not overlap, we get

$$<\rho^2>=\rho^2+f_{cl}\rho_{cl}\rho. \tag{58}$$

This simple description allows to demonstrate the main effect of clumps: far from the Galactic center the rate of DM annihilation falls as $\rho(r)$ rather than as $\rho(r)^2$. The parameters ρ_{cl} and f_{cl} have zero default values. The routine to change these values is

• `setClumpConst(`f_{cl},ρ_{cl}`)`

To be more general, one could assume that ρ_{cl} and f_{cl} depend on the distance from the galactic center. The effect of clumping is then described by the equation

$$<\rho^2>(r)=\rho(r)(\rho(r)+\rho^{eff}_{clump}(r)), \tag{59}$$

and the function

• `setRhoClumps(`ρ^{eff}_{clump}`)`

allows to implement a more sophisticated clump structure. To return to the default treatment of clumps call `setRhoClumps(rhoClumpsConst)` or `setClumpConst`.

8.4 Photon signal

The photon flux does not depend on the diffusion model parameters but on the angle ϕ between the line of sight and the center of the galaxy as well as on the annihilation spectrum into photons

• `gammaFluxTab(fi,dfi,sigmav,Sg,Sobs)`

multiplies the annihilation photon spectrum with the integral over the line of sight and over the opening angle to give the photon flux. `fi` is the angle between the line of sight and the center of the galaxy, `dfi` is half the cone angle which characterizes the detector resolution (the solid angle is $2\pi(1-cos(dfi))$, `sigmav` is the annihilation cross section, `Sg` is the DM annihilation spectra. `Sobs` is the spectra observed in `1/(GeV cm^2 s )` units.

The function `gammaFluxTab` can be used for the neutrino spectra as well.

• `gammaFluxTabGC(l,b,dl,db,sigmav,Sg,Sobs)`

is similar to `gammaFluxTab` but uses standard galactic coordinates. Here l is the galactic longitude (measured along the galactic equator from the galactic center, and b is the latitude (the angle above the galactic plane). Both l and b are given in radians. The relation between the angle `fi` used above and the galactic coordinates is `fi`$=\cos^{-1}(\cos(l)\cos(b))$. `gammaFluxTabGC` integrates the flux over a rectangle $[(l,b)-(l+dl,b+db)]$.

• `loopGamma(&vcs_gz,&vcs_gg)`

calculates σv for loop induced processes of DM pair annihilation into γZ and into $\gamma\gamma$. The result is given in $\frac{cm^3}{s}$. The function returns a non-zero value to signal a problem. This function is available only for MSSM [**?**], NMSSM [**?**], CPVMSSM and IDM. Note that this function does not include non-perturbative effects that are in particular important when the mass of DM is much above the weak scale [**?**, **?**].

• `gammaFlux(fi,dfi,dSigmavdE)`

computes the photon flux for a given energy E and a differential cross section for photon

production, dSigmavdE. For example, one can substitute dSigmavdE=σvSpectdNdE(E,SpA) where σv and SpA are obtained by calcSpectrum. This function can also be used to compute the flux from a monochromatic gamma-ray line by substituting the cross section at fixed energy (in cm^3/s) instead of dSigmavdE, for example the cross sections obtained with the loopGamma function in the MSSM, NMSSM, CPVMSSM models (vcsAA and vcsAZ). In this case the flux of photons can be calculated with
gammaFlux(fi,dfi,2*vcsAA+vcsAZ).

• gammaFluxGC(l,b,dl,db,vcs)
is the analog of gammaFlux when using standard galactic coordinates.

8.5 Propagation of charged particles

The observed spectrum of charged particles strongly depends on their propagation in the Galactic Halo. The propagation depends on the global parameters

```
K_dif, Delta_dif, L_dif, Rsun, Rdisk
```

as well as

```
Tau_dif (positrons), Vc_dif (antiprotons)
```

• posiFluxTab(Emin,sigmav, Se, Sobs)
computes the positron flux at the Earth. Here sigmav and Se are values obtained by calcSpectrum. Sobs is the positron spectrum after propagation. Emin is the energy cut to be defined by the user. Note that a low value for Emin increases the computation time. The format is the same as for the initial spectrum. The function SpectrdNdE(E,Sobs) described above can also be used for the interpolation, in this case the flux is returned in (GeV s cm^{2}sr)$^{-1}$.

• pbarFlux(E,dSigmavdE)
computes the antiproton flux for a given energy E and a differential cross section for antiproton production, dSigmavdE. For example, one can substitute
dSigmavdE=σvSpectdNdE(E,SpP)
where σv and SpP are obtained by calcSpectrum. This function does not depend on the details of the particle physics model and allows to analyse the dependence on the parameters of the propagation model.

• pbarFluxTab(Emin,sigmav, Sp, Sobs)
computes the antiproton flux, this function works like posiFluxTab,

• solarModulation(Phi, mass, stellarTab, earthTab)
takes into account modification of the interstellar positron/antiproton flux caused by the electro-magnetic fields in the solar system. Here Phi is the effective Fisk potential in MeV, mass is the particle mass, stellarTab describes the interstellar flux, earthTab is the calculated particle flux in the Earth orbit.

Note that for solarModulation and for all *FluxTab routines one can use the same array for the spectrum before and after propagation.

9 Planck CMB constraint

The function

• PlanckCMB(vSigma, SpA, SpE)

determines whether the DM model under consideration is consistent with Planck measurements of the CMB anisotropies caused by the injection of electrons and photons coming from DM annihilation [**?**]. The spectra of DM annihilation into photons and positrons are provided in the arrays `SpA` and `SpE`. Here it is assumed that the DM relic density $\Omega h^2 = 0.12$, for multi-component DM the fraction of each DM component described by the `fracCDM` array is taken into account. We also assume that the input parameter `vSigma` represents the s-wave cross section in `[cm^3/s]` units, this limit does not apply if the DM annihilation cross-section depends on velocity at small velocities. A return value more than 1 indicates that the model is excluded at 95% C.L. Note that if the DM model gives $\Omega h^2 < 0.12$ and one assumes that DM forms only a fraction ξ of the total DM in the Universe, then the value returned by `PlanckCDM` should be scaled by ξ^2.

By default we assume that the annihilation is s-wave and that `vSigma` is the cross-section computed from calcSpectrum, the number of events will be computed using M_{cdm}^{eff}, as given in Eq.53. If a model of elementary particles is not loaded, and the user just defined the spectra assuming some specific annihilation channels, then the number of events is computed using the `Mcdm` parameter instead. This allows for a model independent test of the function `PlanckCMB`.

In the file `mdlIndep/CMB.c` we reproduce the top panels of Fig.3 and Fig.4 of Ref. [**?**], here we use the PPPC spectra.

10 Photons from Dwarf Spheroidal galaxies

• `DwarfSignal(vSigma, SpA)`
determines whether the photon spectra `SpA` is compatible with the 95% limits obtained from observations of Dwarf Spheroidal galaxies [**?**, **?**, **?**]. Here `vSigma` is the DM annihilation cross section at low velocities($< v\sigma >$ in $[cm^3/s]$ units) as obtained by `calcSpectrum`. The return value is the ratio of $< v\sigma >$ in the given model to the limit extracted from observations, a value more than 1 indicates that the model is excluded. Conversely dividing $< v\sigma >$ in the given model by the value returned by the routine gives the excluded cross-section.

To compute the signal `vSigma` is divided by the squared mass M_{cdm}^{eff}, Eq.53. For a model independent test of this function the `Mcdm` parameter is used instead. For an example, see the file `mdlIndep/Dwarf.c` which reproduces the *combined* curve in [**?**], Fig.5 (Right).

11 Neutrino signal from the Sun and the Earth

This module only works for single component dark matter

After being captured, DM particles concentrate in the center of the Sun/Earth and then annihilate into Standard Model particles. These SM particles further decay producing neutrinos that can be observed at the Earth. The neutrino spectra originating from different annihilation channels into SM particles and taking into account oscillations and Sun medium effects were computed both in `WimpSim` [**?**] and in PPPC4DMν [**?**]. We use the set of tables provided by these two groups as well as those from `DM`ν [**?**] which were included in previous versions of `micrOMEGAs`. The new global parameter

WIMPSIM allows to choose the neutrino spectra. The default value WIMPSIM=0 [8] corresponds to the PPPC4DMν spectra while WIMPSIM=1 corresponds to the WimpSim spectra and WIMPSIM=-1 to the DMν spectra.

• neutrinoFlux(f,forSun,nu, nu_bar)
calculates the muon neutrino/anti-neutrino fluxes near the surface of the Earth. Here f is the DM velocity distribution normalized such that $\int_0^\infty vf(v)dv = 1$. The units are km/s for v and s^2/km^2 for f(v). For example, one can use the same Maxwell function introduced for direct detection. This routine implicitly depends on the WIMPSIM switch.

If forSun==0 then the flux of neutrinos from the Earth is calculated, otherwise this function computes the flux of neutrinos from the Sun. The calculated fluxes are stored in nu and nu_bar arrays of dimension NZ=250. The neutrino fluxes are expressed in [1/Year/km^2].

• muonUpward(nu,Nu,muon)
calculates the muon flux which results from interactions of neutrinos with rocks below the detector. Here nu and Nu are input arrays containing the neutrino/anti-neutrino fluxes calculated by neutrinoFlux. muon is an array which stores the resulting sum of μ^+, μ^- fluxes. SpectdNdE(E,muon) gives the differential muon flux in [1/Year/km^2/GeV] units. The muon flux weakly depends on the propagation medium, e.g. rock or ice. The energy lost during propagation is described by the equation [?]

$$\frac{dE}{dx} = -(\alpha + \beta E)\rho \tag{60}$$

For propagation in ice (the switch forRocks=0), micrOMEGAs substitutes $\rho = 1.0\,\mathrm{g/cm^3}$, $\alpha = 0.00262\,\mathrm{GeVcm^2/g}$, $\beta = 3.5 \times 10^{-6}\,\mathrm{cm^2/g}$ [?], while for propagation in rocks, $\rho = 2.6\,\mathrm{g/cm^3}$, $\alpha = 0.002\,\mathrm{GeVcm^2/g}$, $\beta = 3.0 \times 10^{-6}\,\mathrm{cm^2/g}$ [?]. The result depends on the ratio α/β .

• muonContained(nu,Nu,rho, muon) calculates the flux of muons produced in a given detector volume.This function has the same parameters as muonUpward except that the outgoing array gives the differential muon flux resulting from neutrinos converted to muons in a km^3 volume given in [1/Year/km^3/GeV] units. rho is the density of the detector in g/cm^3.

• atmNuFlux(nu,cs,E)
returns the atmospheric muon neutrinos ($nu > 0$) and anti-neutrinos spectrum ($nu < 0$) in [1/Year/km^2/GeV] units for a given cosine of the zenith angle, $cs > 0$ and $1 \le E \le 10^4$GeV. This function is based on [?].

• atmNuFluxes(nuPdg,cos,E)
returns the atmospheric neutrinos fluxes in $(m^2 \cdot sec \cdot sr \cdot GeV)^{-1}$ units as presented in [?]. Only $\nu_e, \bar{\nu}_e, \nu_\mu, \bar{\nu}_\mu$ are included corresponding to PDG codes, nuPdg = 12,-12,14,-14. Input parameters should be specified in the range, $-1 \le cos \le 1$, $0.1 \le E \le 10^4$GeV.
Ref. [?] provides several tables of calculated neutrino fluxes depending on destination, solar activity, with and without mountains over the detector. By default we use the file *"grn-ally-20-01-mtn-solmax.d"* which gives the neutrino flux for a detector under the mountains in Gran Sasso at the time of large solar activity. To use other tables provided

[8] Since PPPC4DMν does not provide neutrino specrtra produced at the center of the Earth, in this case and for WIMPSIM=0 micrOMEGAs uses the DMν spectra.

in
http://www-rccn.icrr.u-tokyo.ac.jp/mhonda/public/nflx2014/index.html
the user has to copy the corresponding file in the micrOMEGAs directory Data/nu_data or in the current directory and write down the name of the file in the public variable *atmNuFile*. For example

```
atmNuFile="grn-ally-20-01-mtn-solmin.d";
```

Two functions allow to estimate the background from atmospheric neutrinos creating muons after interaction with rocks below the detector or with water inside the detector.
• `ATMmuonUpward(cosFi,E)` calculates the sum of muon and anti-muon fluxes resulting from the interaction of atmospheric neutrinos with rocks in units of `[1/Year/km`^{2}`/GeV/Sr]`. `cosFi` is the energy between the direction of observation and the direction to the center of Earth. `E` is the muon energy in `GeV`. The result depends on the `forRock` switch.
• `ATMmuonContained(cosFi, E, rho)` calculates the muon flux caused by atmospheric neutrinos produced in a given (detector) volume. The returned value for the flux is given in `1/Year/km`^{3}`/GeV/Sr`. `rho` is the density of the detector in `g/cm`^{3} units. `cosFi` and `E` are the same as above.

11.1 Comparison with IceCube results

These functions are described in [?] and allow to compare the predictions for the neutrino flux from DM captured in the Sun with results of IceCube22.
• `IC22nuAr(E)`
effective area in $[km^2]$ as a function of the neutrino energy, $A_{\nu_\mu}(E)$
• `IC22nuBarAr(E)`
effective area in $[km^2]$ as a function of the anti-neutrino energy, $A_{\bar{\nu}_\mu}(E)$).
• `IC22BGdCos(cs)`
angular distribution of the number of background events as a function of $\cos\phi$, $\frac{dN_{bg}}{d\cos\varphi}$.
• `IC22sigma(E)`
neutrino angular resolution in radians as a function of energy.
• `exLevIC22( nu_flux, nuB_flux,&B)`
calculates the exclusion confidence level for number of signal events generated by given ν_μ and $\bar{\nu}_\mu$ fluxes, [?]. The fluxes are assumed to be in [GeV km^2 Year]$^{-1}$. This function uses the `IC22BGdCos(cs)` and `IC22sigma(E)` angular distribution for background and signal as well as the event files distributed by IceCube22 with $\phi < \phi_{cut} = 8^\circ$. The returned parameter `B` is a Bayesian factor representing the ratio of likelihood functions for the model with given fluxes and the model with null signal. See details in [?].
• `fluxFactorIC22(exLev, nu,nuBar)`
For given neutrino, `nu`, and anti-neutrino fluxes, `nuBar`, this function returns the factor that should be applied to the fluxes (neutralino-proton cross sections) to obtain a given exclusion level `exLev` in exLevIC22. This is used to obtain limits on the SD cross section for a given annihilation channel.

12 Cross sections and decays

The calculation of particle widths, decay channels and branching fractions can be done by the following functions

• `pWidth(pName,&address)`
returns directly the width for particle `pName` and the returned parameter `address` gives an address where information about the branchings is stored. See below description of routines which use this parameters. We first describe default work of `pWidth` which can be modified specially for each particle by the `pWidthPref` command. For models which read a SLHA parameter file, the values of the widths and branchings are taken from the SLHA file when `useSLHAwidth=1` .

If `useSLHAwidth=0` or decay of given particle is not presented in SLHA file, then `micrOMEGAs` applies its own calculation of width. If the `1->2` decay channels are kinematically accessible then by default[9] only these channels are included in the width. If not, `pWidth` compiles all open `1->3` channels and use these for computing the width. If all `1->3` channels are kinematically forbidden, `micrOMEGAs` compiles `1->4` channels. If `VWdecay(VZdecay)`$\neq 0$, then `micrOMEGAs` also computes the `1->3` processes with virtual $W(Z)$ and adds these to the list of `1->2` decay channels. Note that `1->3` decay channels with a virtual W will be computed even if the mass of the decaying particle exceeds the threshold for `1->2` decays by several GeV's. This is done to ensure a proper matching of `1->2` and `1->3` processes.

The routine
•`pWidthPref(pName, pref)` changes the mode of the `pWidth` calculation individually for the particle `pName`. Here `pref=4` restores the default `pWidth` mode. Otherwize the `useSLHAwidth` flag is ignored and for `pref=0,1,2,3` we have

- 0 - widths are calculated using processes with minimal number of outgoing particles.
- 1 – widths are calculated using processes with minimal and next to minimal number of outgoing particles excluding processes with s-channel resonances to avoid double counting.
- 2- widths are read from the SLHA file - if the SLHA file does not contain widths, they are calculated as in `pref=0` case.
- 3– widths are read from the SLHA file - if the SLHA file does not contain widths, they are calculated as in `pref=1` case.

• `printTxtList(address,FD)`
lists the decays and their branching fractions and writes them in a file. `address` is the address returned by `pWidth`.

• `findBr(address,pattern)`
finds the branching fraction for a specific decay channel specified in `pattern`, a string containing the particle names in the `CalcHEP` notation. The names are separated by commas or spaces and can be specified in any order. If the `pattern` contains "*", then `findBr` sums all branching fractions which contain particles included in the `pattern`.

• `printPartialWidth(width,address,FD)`

[9] See `pWidthPref` below.

prints the branching fractions and partial widths for each decay channel of a particle in the file `FD`. Here `width` and `address` are the result of the function `pWidth` described above.

• `slhaDecayPrint(pname,delVirt,FD)`
uses `pWidth` described above to calculate the width and branching ratios of particle `pname` and writes down the result in SLHA format. The return value is the PDG particles code. In case of problem, for instance wrong particle names, this function returns zero. This function first computes $1 \to 2$ decays. If such decays are kinematically forbidden then $1 \to 3$ decay channels are computed. Decays via virtual W/Z bosons will be listed via their decay products when `delVirt`$\neq 0$.

• `newProcess(procName)`
compiles the codes for any $2 \to 2$ or $1 \to 2$ reaction. The result of the compilation is stored in the shared library in the directory `work/so-generated/`. The name of the library is generated automatically.

The `newProcess` routine returns the *address* of the compiled code for further usage. If the process can not be compiled, then a NULL address is returned.

Note that it is also possible to compute processes with polarized massless beams, for example for a polarized electron beam use `e%` to designate the initial electron.

• `procInfo1(address,&ntot,&nin,&nout)`
provides information about the total number of subprocesses (ntot) stored in the library specified by `address` as well as the number of incoming (`nin`) and outgoing (`nout`) particles for these subprocesses. Typically, for collisions (decays), `nin`=2(1) and `nout`=2,3. `NULL` can be substitute if this information is not needed.

• `procInfo2(address,nsub,N,M)`
fills an array of particle names `N` and an array of particle masses `M` for the subprocess `nsub` ($1 \leq nsub \leq ntot$) . These arrays have size $nin + nout$ and the elements are listed in the same order as in `CalcHEP` starting with the initial state, see the example in `MSSM/main.c`.

• `cs22(address,nsub,P,c1,c2,&err)`
calculates the cross-section for a given $2 \to 2$ process, `nsub`, with center of mass momentum P(GeV). All model parameters except the strong coupling `GG` can be specified with the functions `findVal[W]/assignVal[W]` described in Section 5. The strong coupling `GG` is defined via the scale parameter `GGscale`. The differential cross-section is integrated within the range $c1 < \cos\theta < c2$. θ is the angle between $\vec{p}_1$ and $\vec{p}_3$ in the center-of-mass frame. Here $\vec{p}_1$ ($\vec{p}_3$) denote respectively the momentum of the first initial(final) particle. *err* contains a non zero error code if *nsub* exceeds the maximum value for the number of subprocesses (given by the argument `ntot` in the routine `procInfo1`). To set the polarization of the initial massless beam, define `Helicity[i]` where $i = 0, 1$ for the 1^{st} and 2^{nd} particles respectively. The helicity is defined as the projection of the particle spin on the direction of motion. It ranges from [-1,1] for spin 1 particles and from [-0.5,0.5] for spin 1/2 particles. By definition a left handed particle has a positive helicity.

• `hCollider(Pcm,pp,nf, Qren,Qfac, pName1,pName2,Tmin,wrt)` calculates the cross section for particle production at hadron colliders. Here `Pcm` is the beam energy in the center-of-mass frame. `pp` is $1(-1)$ for $pp(p\bar{p})$ collisions, $nf \leq 5$ defines the number of quark flavors taken into account. The parameters `Qren` and `Qfac` define the renormalisation and factorization scales respectively. `pName1` and `pName2` are the names of outgoing

particles. If $T_{\min} \leq 0$ then `hCollider` calculates the total cross section for the 2-body final state process. Otherwise it calculates the cross section for

```
proton, [a]proton -> pName1, pName2, jet
```

where $T_{\min} > 0$ defines the cut on the jet transversee momentum. The jet content is defined by the parameter `nf`. If $Q_{\rm fac} \leq 0$, then running $\hat{s}$ is used for the factorization scale. If $Q_{\rm ren} \leq 0$, $\hat{s}$ is used for the renormalization scale for a $2 \to 2$ process and p_T of the jet is used for the renormalization scale for a process with a jet in the final state. The last argument in the `hCollider` routine allows to switch on/off (`wrt=1/0`) the printing of the contribution of individual channels to the total cross section. The value returned is the total cross section in [pb].

One of the arguments `pName1,pName2` can be NULL. Then the cross section for $2 \to 1$ or $2 \to 1 +$ jet process will be calculated.

By default hCollider uses the `cteq66` [?] structure function built-in the `micrOMEGAs` code. One can set any other parton distribution stored in `micrOMEGAs/CalcHEP_src/pdTables` (PDT-distributions) or by linking the LHAPDF library [?]. The list of PDT structure functions can be obtained with the command

- `PDTList()`

and one of these can be activated by

- `setPDT(name)`

To work with other PDF's available in `LHAPDF` one should recompile `micrOMEGAs` with a parameter which defines the path to the LHAPDF library:

`make LHAPDF=Path_to_LHAPDF_library.` [10]

`micrOMEGAs assumes the existence of a file` *$LHAPDF/lib/libLHAPDF.so*
`The list of available LHAPDF distributions can be obtained with the command`

- `LHAPDFList()`

`and one of these can be activated by`

- `setLHAPDF(name,nset)`

`where nset specifies the subset number. Note, that if a wrong input is provided, setLHAPDF terminates the execution.`

`Alternatively, one can convert a PDF distribution to the PDT format with the command lhapdf2pdt disposed in` *CalcHEP_src/bin*.

`lhapdf2pdt` *directory_with_PDF_disctribution*

`the generated .pdt file has to be placed in the folder /CalcHEP_src/pdTables.`

The parton densities are defined by the function

- `parton_distr(pdg,x,q)`

where `pdg` is the PDG code of a particle. Note that `parton_distr` defines the parton number density and contains a factor $1/x$ with respect to the definition used in LHAPDF.

[10] The path will be stored in `FlagsForSh` and `FlagsForMake` files and subsequent call of `make` without parameters in the `micrOMEGAs` directory will not clean the corresponding records.

13 Tools for model independent analysis

A model independent calculation of the DM observables is also available. After specifying the DM mass, the cross sections for DM spin dependent and spin independent scattering on proton and neutron, the DM annihilation cross section times velocity at rest and the relative contribution of each annihilation channel to the total DM annihilation cross section, one can compute the direct detection rate on various nuclei, the fluxes for photons, neutrinos and antimatter resulting from DM annihilation in the galaxy and the neutrino/muon fluxes in neutrino telescopes. All the routines presented here depend implicitly on the global parameter `Mcdm` except for `basicSpectra` and `basicNuSpectra`. They also explicitly depend on spin-independent and spin-dependent cross sections of DM scattering on proton and neutron: csIp,csIn,csDp, csDn. These cross sections have to be specified in [pb]units. A negative value for one of these cross sections is interpreted as a destructive interference between the proton and neutron amplitudes.

These routines do not take into account the multi-component structure of DM and, in particular, possible differences between DM and anti-DM. For multi-component `DM` the user has to perform a summation over the different DM components.

The routines available include:

• `nucleusRecoilCS(`f_v`,A,Z,J,Sxx,csIp,csIn,csDp,csDn,dNdE)`
is similar to `nucleusRecoil`, see Section 7.2.

• `nucleusRecoil0CS(`f_v`,A,Z,J,Sp,Sn,csIp,csIn,csDp,csDn,dNdE)` is the corresponding modification of `nucleusRecoil0`.

• `DD_pvalCS(Exp, $f_v$, Sxx,csIp,csIn,csDp,csDn,\&expName)`
and

• `DD_factorCS(Exp, pval, $f_v$,Sxx,csIp,csIn,csDp,csDn,\&expName)`
are similar to `DD_pval`, `DD_factor` described in Section 7.3.

`DD_pvalCS`, `DD_factorCS` have an option to handle a low mass mediator which modifies the recoil energy distribution, see Eq.34.

• `(*dNdEfact)(Enr,MA)`
is the address of the function which modifies the nucleus recoil distribution to take into account a t-channel propagator with small or zero mass as described in Section 7.3.

For indirect detection, we also provide a tool for model independent studies

• `basicSpectra(Mass,pdgN,outN,Spectr)`
computes the spectra of outgoing particles and writes the result in an array of dimension NZ, `Spectr`. `pdgN` is the PDG code of the particles produced in the annihilation of a pair of WIMPs. To get the spectra generated by transverse and longitudinal W's substitute `pdgN`$= 24 +' T'$ and $24 +' L'$ correspondingly. In the same manner `pdgN`$= 23 +' T'$ and $23 +' L'$ provides the spectra produced by a polarized Z boson. `outN` specifies the outgoing particle,

$$\text{outN} = \{0, 1, 2, 3, 4, 5\} \text{ for } \{\gamma, e^+, p^-, \nu_e, \nu_\mu, \nu_\tau\}$$

The `Mass` parameter defines the mass of the DM particle. Note that the propagation routines for e^+, p^-, γ can be used after this routine as usual. Note that the result of `basicSpectra` are not valid for Mcdm $<$ 2GeV as explained in the description of `calcSpectrum`.

To get indirect detection signals one can substitute the obtained spectra in the `[photon/posi/pbar]FluxTab` routines.

• `captureCS(f,forSun,Mass,csIp,csIn,csDp,csDn)`
calculates the number of DM particles captured per second assuming the cross sections for spin-independent and spin-dependent interactions with protons and neutrons `csIp, csIn, csDp, csDn` are given as input parameters (in `[pb]`). A negative value for one of the cross sections is interpreted as a destructive interference between the proton and neutron amplitudes. The first two parameters have the same meaning as in the `neutrinoFlux` routine Section 11. The result depends implicitly on the global parameters `rhoDM` and `Mcdm` in Table 1.
• `basicNuSpectra(forSun,Mass,pdg, pol, nu_tab, nuB_tab)`
calculates the ν_μ and $\bar{\nu}_\mu$ spectra corresponding to the pair annihilation of DM in the center of the Sun/Earth into a particle-antiparticle pair with PDG code `pdg`. `Mass` is the DM mass. Note that this routine depends implicitly on the global parameter `WIMPSIM` (1,0,-1) which selects the neutrino spectra computed by `WimpSim` [?], PPPC4DMν [?] and `DM`ν [?]. The parameter `pol` selects the spectra for polarized particles available in PPPC4DMν. `pol=-1(1)` corresponds to longitudinal (transverse) polarisation of vector bosons or to left-handed (right-handed) polarisation of fermions, `pol=0` is used for unpolarized spectra. When polarized spectra are not available, the unpolarized ones are generated irrespective of the value of `pol`. The parameter `outN` is 1 for muon neutrino and -1 for anti-neutrino. The resulting spectrum is stored in the arrays `nu_tab` and `nuB_tab` with NZ=250 elements.

The files `main.c/F` in the directory `mdlIndep` contain an example of the calculation of the direct detection, indirect detection and neutrino telescope signals using the routines described in this section. The numerical input data in this sample file corresponds to 'MSSM/mssmh.dat'.

14 Constraints from colliders

14.1 The Higgs sector

To obtain the limits on the Higgs sector for models with one or several Higgs bosons, the predictions for the signal strengths of the 125 GeV Higgs can be compared to the latest results of the LHC, for this an interface to the public code HiggsSignals [?] or Lilith [?, ?, ?] is provided. Moreover the exclusion limits obtained from Higgs searches in different channels at LEP, Tevatron and the LHC can be applied to other Higgses in the model using HiggsBounds [?].

14.1.1 Lilith

`Lilith` [?, ?, ?] is a Python library that is used to construct a global likelihood function $\mathcal{L}$ from the latest ATLAS and CMS results on the 125 GeV Higgs.[11] The `Lilith` inputs are the set of reduced couplings of the 125 GeV state, *i.e.*, couplings normalized by the SM ones, as well as the branching ratios of Higgs decays to invisible, $\mathrm{BR}_{\mathrm{inv}}$, or to undetected non-SM final states, $\mathrm{BR}_{\mathrm{undetected}} = 1 - \mathrm{BR}_{\mathrm{inv}} - \sum \mathrm{BR}(H \to \mathrm{SM\,SM})$. By default, the automatic generation of the input file assumes that only DM contributes to

[11] `Lilith` can test Higgs bosons with masses within the $[123, 128]$ GeV interval, a warning will be issued if no such state can be found. In the case where two or more states have masses within this interval, their signal strengths will be summed incoherently and an effective Higgs state will be tested against the LHC measurements.

the invisible width. Note that the reduced couplings of the 125 GeV Higgs are defined for all the models provided with `micrOMEGAs` with the exception of the $Z\gamma$ coupling. The latter is computed within `Lilith` assuming that only SM particles run in the loop. The file `include/Lilith.inc` (or `Lilith.inc_f`) contains the instructions to launch `Lilith` using a system call. The input file `Lilith_in.xml` for `Lilith` can be created by two commands

```
LilithMDL("Lilith_in.xml")
LilithMO("Lilith_in.xml")
```

both also return the number of neutral Higgs particles. `LilithMDL` requires that the reduced couplings be defined in `lib/lilith.c` of the model. These files are defined for all models provided with `micrOMEGAs` and should be written by the user for new models. On the other hand `LilithMO` generates automatically the input file required by `Lilith`. The functionality of `LilithMO` is described in Section 14.1.3. As input parameters, `Lilith` also requires the number of free parameters, `n_par`, and a reference likelihood point, `m2logL_reference`. Those are defined in the `main.c` file of each model and set to 0 by default.

The `SLHA` output file, `Lilith_out.slha`, consists of six entries which are respectively the log-likelihood evaluated at a parameter space point $\mathcal{P}$, $-2\log\mathcal{L}(\mathcal{P})$, the number of experimental degrees of freedom, n^{exp}, the reference likelihood point, the number of degrees of freedom, ndf, the p-value, p and the database version. For a detailed description of the various outputs and their interpretation see [**?**, **?**]. For example, one could flag or exclude points with too low p-value. In this context, a point with a p-value smaller than 0.3173, 0.0455, 0.0027 could be excluded at more than the 1σ, 2σ, 3σ levels, respectively.

14.1.2 HiggsBounds and HiggsSignals

Constraints on the properties of the 125 GeV Higgs boson can also be obtained with `HiggsSignals`. Moreover exclusion limits provided by the experimental LHC and Tevatron collaborations on additional Higgs bosons are obtained through an interface to `HiggsBounds` [**?**]. Theses codes are no longer distributed with `micrOMEGAs` but are downloaded and compiled when required[12]. The file `include/hBandS.inc` contains the instructions to call both `HiggsBounds` and `HiggsSignals`, in particular the options for running `HiggsSignals` are set in this file by fixing the `Dataset`, `Method` and `PDF` parameters. Moreover the interface uses the effective coupling option for specifying the input see [**?**] for more details. Two functions can be used to generate the input SLHA file, `HB.in`

```
hbBlocksMDL("HB.in",&NchHiggs)
hbBlocksMO("HB.in",&NchHiggs)
```

Both return the number of neutral Higgses and `NchHiggs` gives the number of charged Higgs particles. The function `hbBlocksMDL` is located in `lib/hbBlocks.c` for each MSSM-like model and contains the appropriate definition of the reduced couplings. The function `hbBlocksMO` generates automatically the required input file for any model and is thus very convenient to use for new models. The content of this function is described in

[12]For compilation one needs Fortran f90 compiler, for instance `gfortran`, and `cmake`

Section 14.1.3. Note that the PDG numbers which will be considered by HiggsBounds for Higgs-like particles are only 25, 35, 36, 37, 45, and 46. Moreover HiggsBounds will check the contribution to the Higgs invisible width only for particles with PDG codes 1000022, 1000012, 1000014, 1000016, 1000017, 1000018 or 1000019. Also note that the theoretical uncertainty on the mass of the Higgs boson should be specified in the SLHA BLOCK DMASS, see the example given in main.[c/F] to set the uncertainty at 2 GeV.

The complete outputs of `HiggsBounds` and `HiggsSignals` are stored in the files `HB.out` and `HS.out` respectively and can be accessed and read by the user using the `slhaval` function [**?**]. The screen output of `micrOMEGAs` contains the following information

```
HiggsBounds(version number)
id   result  obsratio  channel
```

where three channels are listed, `result` $= 0, 1, -1$ denotes respectively whether a parameter point is excluded at 95% CL, not excluded, or invalid; `obsratio` gives the ratio of the theoretical expectation relative to the observed value for the channel specified in `channel`. The first channel in this list is the one with the highest expected sensitivity and is the one that determines the overall exclusion, see [**?**], other channels are to be interpreted by the user.

The `HiggsSignals` output displayed on the screen is simply

```
HiggsSignals(version number)
Nobservables chi^2  pval
```

where `Nobservables` gives the number of observables used in the fit, `chi^2` the associated χ^2 and `pval` the p-value. The interpretation of these values is left to the user, see [**?**] for a detailed description.

14.1.3 Automatic generation of interface files

The functions `LiLithMO` and `hbBlocksMO` contain two routines that allow to extract the couplings contained in the model file, `lgrng1.mdl`. The first routine returns a description of a given vertex. The format used is

```
lVert* vv=getLagrVertex(name1,name2,name3,name4);
```

where *name1,..,name4* are the names of the particles included in the vertex; for vertices with three particles, *name4* should be replaced by `NULL`. The return parameter `vv` is the memory address of a structure which contains information about the vertex:

- `vv->GGpower` - power of strong coupling included in vertex
- `vv->nTerms` - number of different Lorentz structures in vertex
- `vv->SymbVert[i]` - text form of Lorentz structures $i \in [0, nTerms]$

The second routine allows to obtain the numerical coefficients corresponding to each Lorentz structure. The command is

```
getNumCoeff(vv,coeff)
```

with `coeff[i]` the numerical coefficient for `SymbVert[i]`. Note that the strong coupling is factored out of the coefficients.

All the QCD-neutral scalars belonging to the even sector (not designated with ~) are considered as Higgs particles. For each of these, `micrOMEGAs` calculates the couplings to SM fermions and massive bosons and writes down into the interface file the ratio of these couplings to the corresponding SM Higgs coupling. Note that the couplings of the Higgs to SM fermions can significantly depend on the QCD scale; `micrOMEGAs` assumes that the quark masses entering the vertices are obtained at the same scale in both the SM and the new model, thus the scale dependence in the reduced couplings to fermions disappears. The loop-induced couplings of the Higgs to gluons and photons are calculated by the `LiLithMO/hbBlocksMO` routines whether or not the Hgg and $H\gamma\gamma$ vertices are already implemented in the Lagrangian. This includes NLO-QCD corrections and is performed as described in [**?**, **?**].

The various branching ratios and widths of the Higgs required by `HiggsBounds`,`HiggsSignals` or `Lilith` are also written automatically in the interface file. When these values are provided in an SLHA file, they will be used. Otherwise `LiLithMO/hbBlocksMO` check the existence of $H \to gg$ and $H \to \gamma\gamma$ in the table of decays generated from the model Lagrangian. If found, the branching ratios and total widths are written in the interface file without comparing with the internal calculations. If not found, then `LiLithMO/hbBlocksMO` add these channels and recompute the total widths and all branching ratios.

14.2 Searches for New particles

14.2.1 SModelS

LHC limits on new (odd) particles can be obtained using `SModelS` [**?**, **?**, **?**, **?**, **?**], a code which tests Beyond the Standard Model (BSM) predictions against Simplified Model Spectra (SMS) results from ATLAS and CMS searches for new physics. The basic working principle of `SModelS` is to decompose a full input model (consisting of particle content, masses, production cross sections and decay widths) of a given BSM scenario into SMS components with their corresponding signal weights. These are then used to evaluate the constraints from a large database of experimental results. The outcome is presented as the ratio of predicted over excluded cross section, so-called r-values, for each experimental analysis in the database that provides a constraint. For efficiency-map type experimental results, the output also reports likelihoods. The (user-defined) combination of likelihoods from uncorrelated analyses is also possible [**?**, **?**].

The `SModelS` approach provides a fast way to cast BSM predictions for the LHC in a model-independent framework, which can be directly confronted with the relevant experimental constraints. The underlying assumption is that differences in the event kinematics (e.g. from different production mechanisms or from the spin of the BSM particle) do not significantly affect the signal selection efficiencies. The majority of experimental results in the `SModelS` database are from searches for promptly decaying SUSY particles, but there are also a number of 'exotics' results and a growing number of results from searches for long-lived particles.

`micrOMEGAs` is now interfaced to version 3.0.0 of `SModelS`, which, among other novel-

ties, features an improved treatment of width-dependent constraints. Most importantly, this new version can handle arbitrary simplified model topologies, without the need of an imposed $\mathcal{Z}_2$-like symmetry. It is thus capable of treating also resonant production of new particles, RPV signatures, etc., in addition to the pair-production of new particles followed by linear cascade decays, which is typical for models with a $\mathcal{Z}_2$-like symmetry.

For running `SModelS`, `micrOMEGAs` automatically generates an SLHA-type input file `smodels.slha` which contains

- QNUMBER blocks for the BSM particles as specification of the model;
- MASS block for masses of BSM particles;
- DECAY blocks for decay widths and branchings for BSM and Higgs particles;
- XSECTION blocks which contains cross sections of LHC processes with production of BSM particles.[13]

This input file, per default called `smodels.slha`, is located in the same directory as `main.c` and is written by `micrOMEGAs` by calling the function

```
smodels(Pcm, nf, csMinFb, fileName, version, wrt)
```

where `Pcm` is the proton beam energy in GeV and `nf` is the number of parton flavors used to compute the production cross sections of the odd-sector particles. (Note that u, d, $\bar{u}$, $\bar{d}$ and gluons are always included while s, c, and b quarks are included for `nf` $= 3, 4, 5$ respectively.) By default, `micrOMEGAs` uses the `cteq66` [**?**] structure functions (set in the `hCollider()` function, see section 12). `csMinFb` defines the minimum production cross section in pb for odd particles; processes with lower cross sections are not added to the SLHA file passed to `SModelS`, here denoted by `fileName`. Finally, `wrt` is a steering flag for the screen output; if `wrt` $\neq 0$ the computed cross sections will be also written on the screen. The call of `smodels()` is per default done in the file
`micromegas_6.1.X/include/SMODELS.inc`

For this functionality, the user first needs to specify which LHC energies should be considered. This is done by setting in `main.c` the switch

`LHCrun` = `LHC8` and/or `LHC13`

for Run 1 (8 TeV), Run 2 (13 TeV) or both. This determines whether 8 TeV or 13 TeV cross sections or both will be appended to the SLHA file, which in turn determines which simplified model results will be considered from the `SModelS` database.

The run parameters for `SModelS` can be set in the file

`micromegas_6.1.X/include/smodels_parameters.ini`.

This concerns for example the parameter `sigmacut`, which is the cutoff cross section for topologies to be considered in the decomposition. Also the database version to be used is set in this parameters file; the default is `path = official`, which means that the official

[13] At present, cross sections are computed automatically for $pp \to$ BSM BSM production. Resonant (s-channel) or associated production of one new particle, i.e. $pp \to$ BSM or $pp \to$ BSM SM, which is relevant for models without a $\mathcal{Z}_2$-like symmetry, will be added in a future release of the `micrOMEGAs-SModelS` interface.

database pickle file corresponding to the SModelS code version will be downloaded and used. An interesting alternative may be to set `path = latest`, in which case always the latest available database will be used.

The SModelS parameters `combineSRs` and `combineAnas` can now be defined in `main.c`.

- `combineSRs != 0` turns on the combination of signal regions on when a covariance matrix or full JSON likelihood is available, see [?, ?].[14]
- `combineAnas` = [list of analysis IDs] is a text parameter, which lists the analyses to be combined in a global likelihood. This switch works only for analyses with efficiency-map results. All the analyses are assumed to be fully uncorrelated, so use with caution!

By default, `char*combineAnas=NULL;`. For example, the following can be set by the user `char*combineAnas= "ATLAS-SUSY-2018-41,ATLAS-SUSY-2019-08,CMS-SUS-21-002";` For detailed information on the different options and parameters available, see the online SModelS manual.

The output is written in SLHA-type format to `smodels.slha.smodelsslha` (or an alternative name selected by the user in `SMODELS.inc`). This output consists of the following blocks:

- `SModelS_Settings`, which lists the SModelS code and database versions as well as input parameters for the decomposition;
- `SModelS_Exclusion`, which contains as the first line the status information if a point is excluded (1), not excluded (0), or not tested (-1), the latter being reported when no matching SMS results are found. This is followed by the list of experimental results. If `expandedOutput = True` (default), all results are printed. Otherwise, if the model is excluded, all results with r-value greater than one are shown and if the point is not excluded, only the result with the highest r-value is displayed;
- `SModelS_CombinedAnas` containing information about the combination of results, if `combineAnas` is defined (see above);
- `SModelS_Coverage`, which, if `testCoverage` is set to True (default: False) in the parameters file, lists the cross sections of missing topologies with prompt and/or displaced decays, and of topologies outside the grids of the experimental results

If a point is excluded (status 1), this is followed by a list of all results with $r > 1$, starting from the highest r-value. We remind the user that the SModelS r-values are defined as the ratio of the predicted theory cross section and the corresponding experimental upper limit at 95% confidence level. For each of these results, the SMS topology identifier as entry 0 (so-called Tx-name, see the online SMS dictionary for an explanation of the terminology), the r-value as entry 1, a measure of condition violation as entry 2, and the analysis identifier as entry 3 are listed. If the point is not excluded (status 0), the result with the highest r-value is given instead to show whether a point is close to the exclusion limit or not.

[14] Signal region combination can significantly increase the sensitivity of an analysis. However, since it can require much more CPU time, considerably increasing the run time, it is turned off by default.

Last but not least, in order to exploit decay channels involving a SM-like Higgs for which the experimental collaborations assumed SM branching ratios for the h with the mass fixed to $m_h = 125$ GeV, `micrOMEGAs` checks whether neutral scalar particles with a mass in the range 123–128 GeV have branching ratios to WW^*, ZZ^*, $\tau\tau$, $b\bar{b}$ within 15% of those of a SM Higgs of the same mass. The corresponding particle will be identified as a SM Higgs through PDG code 25 in the `smodels.slha` file. Note that, for `SModelS`, PDG code 25 is reserved for a SM-like Higgs and should not be assigned generically.

14.2.2 Other limits

Limits from searches for a new massive Abelian gauge boson at the LHC, from LEP on an invisible Z as well as limits on light neutralinos from LEP are provided through the functions:

• `Zinvisible()`
returns 1 and prints a WARNING if the invisible width of the Z boson of the Standard Model is larger than 0.5 MeV ([**?**]) and returns 0 if this constraint is fulfilled. This routine can be used in any model with one DM where the Z boson is uniquely defined by its PDG=23 and whether the neutral LSP is its own antiparticle or not.

• `Zprimelimits()`
returns 0 if the scenario considered is not excluded by Z' constraints, 1 if the point is excluded and 2 if both subroutines dealing with Z' constraints cannot test the given scenario. The routine can be used for any Z' uniquely defined by the PDG code 32. Currently two types of searches defined in different subroutines of `Zprimelimits()` are implemented. The 3.2/fb Z' search in the dilepton final state at $\sqrt{s} = 13$ TeV from ATLAS [**?**] is considered in the first subroutine `Zprime_dilepton`. If the scenario considered is allowed or not tested by `Zprime_dilepton`, a second subroutine called `Zprime_dijet` analyses the point using constraints from LHC dijet searches at $\sqrt{s} = 8$ TeV [**?**, **?**, **?**] and at $\sqrt{s} = 13$ TeV [**?**, **?**]. This subroutine uses the recasting performed in [**?**] for a combination of ATLAS and CMS searches. `Zprimelimits()` returns 1 if $M_{Z'} < 0.5$ TeV and 2 for points for which the narrow-width approximation is not valid, *i.e.* $\Gamma/M_{Z'} > 0.3$.

• `LspNlsp_LEP(&ncs)`
checks the compatibility with the upper limit obtained at LEP [**?**] on the cross section for the production of neutralinos $\sigma(e^+e^- \to \tilde{\chi}_1^0\tilde{\chi}_i^0)$, $i \neq 1$, when the heavier neutralino decays into quark pairs and the LSP, $\tilde{\chi}_i^0 \to \tilde{\chi}_1^0 q\bar{q}$. The function calculates ncs= $\sigma \times BR = \sum_i \sigma(e^+e^- \to \tilde{\chi}_1^0\tilde{\chi}_i^0) \times \mathrm{BR}(\tilde{\chi}_i^0 \to \tilde{\chi}_1^0 q\bar{q})/cs_{Lim}$ where $cs_{Lim} = 0.1$pb if $m_{\mathrm{NLSP}} > 100$GeV and 0.5pb otherwise. The function returns 1 if $ncs >= 1$ and 0 otherwise. This function can also be applied for non-SUSY models which feature the same signature, in this case the function will compute the cross section for production of the LSP and any other neutral particle from the odd sector which can decay into the LSP and a Z boson.

• `monoJet(pName1,pName2)`
computes the cross section for $p, p \to$ `pName1,pName2`$+jet$ at $\sqrt{s} = 8$ TeV where `pName1`, `pName2` are the names of neutral outgoing particles and jet includes light quarks (u,d,s) and gluons. The function returns the resulting confidence level obtained with the CLs technique [**?**, **?**] for each signal region of the 8 TeV CMS mono-jet analysis [**?**] and chooses the most constraining one.

15 Additional routines for specific models

The models included in `micrOMEGAs` contain some specific routines which we describe here for the sake of completeness. The current distribution includes the following models: `MSSM, NMSSM, CPVMSSM, IDM` (inert doublet model), `LHM`(little Higgs model), `Z3IDM` (Inert doublet model with Z_3 symmetry), `Z4IDSM` (Inert doublet and singlet model with Z_4 symmetry, `Z5M` (Two scalar singlets model with Z_4 symmetry), `RDM` (scalar leptoquark and two singlet fermions), `STFM` (singlet-triplet fermionic model) as well as three models which serve as examples for freeze-in and for which there are currently no additional routines (`SingletDM, ZpPortal, LLL_scalar`).

Some of these models contain a special routine for reading the input parameters:

- `readVarMSSM`, `readVarNMSSM`, `readVarCPVMSSM`, `readVarlHiggs`, `readVarRHNM`.

These routines are similar to the general `readVar` routine described in Section 5 but they write a warning when a parameter is not found in the input file and display the default values for these parameters.

The supersymmetric models contain several additional routines to calculate the spectrum and compute various constraints on the parameter space of the models. Some functions are common to the `MSSM,NMSSM,CPVMSSM,UMSSM` models:

- `o1Contents(FD)`

prints the neutralino LSP components as the $\tilde{B}, \tilde{W}, \tilde{h_1}, \tilde{h_2}$ fractions. For the `NMSSM` the fifth component is the singlino fraction $\tilde{S}$ and for the `UMSSM` the sixth component is the bino' fraction $\tilde{B}'$. The sum of the squares of the LSP components should add up to 1.

15.1 The MSSM

The `MSSM` has a long list of low scale independent model parameters, those are specified in the SLHA file [**?**, **?**]. They are directly implemented as parameters of the model. micrOMEGAs can either read an SLHA file computed externally or generate this file in the framework of popular SUSY scenarios, these are specified by one of the instructions

```
#define SUGRA
#define SUGRANUH
#define AMSB
#define EWSB
```

If none of these instructions are given then micrOMEGAs reads the external SLHA file which should be passed as an argument when executing the *main* routine. To generate an SLHA file, micrOMEGAs uses either SuSpect, SPHENO, or SoftSusy. These codes, if need be, solve the RGE equations and calculate the masses of Higgs and SUSY particles including one-loop corrections. Note that SuSpect is included within the micrOMEGAs distribution while `SPHENO` or `SOFTSUSY` will be downloaded automatically when needed (see section 3.4). To specify the spectrum calculator the user has to define the parameter RGE in *main.c/main.cpp*. For instance

```
#define RGE suspect
```

Other possibilities for `RGE` are `softSusy`, `spheno` and `tree`. The option `tree` will generate the needed SLHA file for the spectrum using EWSB input parameters and tree-level formulas. One-loop corrections to the Higgs masses and to the effective Higgs potential

are based on [**?**]. In this case, the realization of the MSSM is completely gauge invariant. This option can be useful to check the effect of loop corrections.

For **EWSB** scenarios the input parameters are the soft parameters, the names of these parameters are given in the `MSSM/mssm[1/2].par` files. The user can assign new values to these parameters by means of `assignVal` or `readVarMSSM`.

• *spect*`EwsbMSSM()`

calculates the masses of Higgs and supersymmetric particles in the MSSM including one-loop corrections starting from weak scale input parameters.

In these functions *spect* is defined by `RGE`.

For other MSSM scenarios, the parameters at the electroweak symmetry breaking scale are derived from an input at high scale. The same codes `suspect`, `spheno`, or `softSusy` are used for this.The corresponding routines are:

• *spect*`SUGRA(tb,MG1,MG2,MG3,Al,At,Ab,signMu,MHu,MHd,Ml1,Ml3,Mr1,Mr3,Mq1,Mq3,`
`Mu1,Mu3,Md1,Md3)`

assumes that all input parameters except `tb` and `signMu` are defined at the GUT scale. The `SUGRA/CMSSM` scenario is a special case of this general routine.

• *spect*`SUGRAnuh(tb,MG1,MG2,MG3,Al,At,Ab,Ml1,Ml3,Mr1,Mr3,Mq1,Mq3,`
`Mu1,Mu3,Md1,Md3,mu,MA)`

realizes a SUGRA scenario with non universal Higgs parameters. Here the `Mhu, MHd` parameters in the Higgs potential are replaced with the `mu` parameter defined at the EWSB scale and `MA`, the pole mass of the CP-odd Higgs. The `signMu` parameter is omitted because `mu` is defined explicitly.

•*spect*`AMSB(am0,m32,tb,sng)`.

does the same as above within the `AMSB` model.

We have an option to directly read a `SLHA` input file, this uses the function

• `lesHinput(file_name)`

which returns a non-zero number in case of problem.

The routines for computing constraints are (see details in [**?**]).

• `deltarho()`

calculates the $\Delta\rho$ parameter in the MSSM. It contains for example the stop/sbottom contributions, as well as the two-loop QCD corrections due to gluon exchange and the correction due to gluino exchange in the heavy gluino limit.

• `bsgnlo(&SMbsg)`

returns the value of the branching ratio for $b \to s\gamma$, see Appendix A. We have included some new contributions beyond the leading order that are especially important for high $\tan\beta$. `SMbsg` gives the SM contribution.

• `bsmumu()`

returns the value of the branching ratio $B_s \to \mu^+\mu^-$ in the MSSM. It includes the loop contributions due to chargino, sneutrino, stop and Higgs exchange. The Δm_b effect relevant for high $\tan\beta$ is taken into account. Code is bases on [**?**].

• `btaunu()`

computes the ratio between the MSSM and SM branching fractions for $\bar{B}^+ \to \tau^+\nu_\tau$.

• `gmuon()`

returns the value of the supersymmetric contribution to the anomalous magnetic moment of the muon.

• `Rl23()`

computes the ratio of the MSSM to SM value for R_{l23} in $K^+ \to \mu\nu$ due to a charged higgs contribution, see Eq.70 in [?].

• `dtaunu(&dmunu)`

computes the branching ratio for $D_s^+ \to \tau^+\nu_\tau$. `dmunu` gives the branching ratio for $D_s^+ \to \mu^+\nu_\mu$

• `masslimits()`

returns a positive value and prints a WARNING when the choice of parameters conflicts with a direct accelerator limits on sparticle masses from LEP. The constraint on the light Higgs mass from the LHC is included.

• `treeMSSM()`

This function transforms a loop improved model defined in an SLHA file into a tree level one. The masses and mixing angles are stored in the file `tree.slha`. This option can be useful to check the effect of loop induced masses and mixings. This option guarantees gauge invariance for all processes.

15.2 The NMSSM

As in the `MSSM` there are specific routines to compute the parameters of the model as specified in SLHA. The spectrum calculator is `NMSPEC` [?] in the `NMSSMTools_5.0.1` package [?].

• `nmssmEWSB()`

calculates the masses of Higgs and supersymmetric particles in the NMSSM starting from the weak scale input parameters [?]. These can be downloaded by the `readVarNMSSM` routine.

• `nmssmSUGRA(m0,mhf,a0,tb,sgn,Lambda,aLambda,aKappa)`

calculates the parameters of the NMSSM starting from the input parameters of the `CNMSSM`.

The routines for computing constraints are taken from NMSSMTools (see details in [?]).

• `bsgnlo(&M,&P), bsmumu(&M,&P), btaunu(&M,&P), gmuon(&M,&P)`

are the same as in the MSSM case. Here the output parameters `M` and `P` give information on the lower/upper experimental limits [?]

• `deltaMd(&M,&P),deltaMs(&M,&P)`

compute the supersymmetric contribution to the $B^0_{d,s} - \overline{B^0}_{d,s}$ mass differences, ΔM_d and ΔM_s.

• `NMHwarn(FD)`

is similar to `masslimits_` except that it also checks the constraints on the Higgs masses, returns the number of warnings and writes down warnings in the file `FD`.

15.3 The CPVMSSM

The independent parameters of the model include, in addition to some standard model parameters, only the weak scale soft SUSY parameters. The independent parameters are listed in `CPVMSSM/work/models/vars1.mdl`. Masses, mixing matrices and parameters of the effective Higgs potential are read directly from CPsuperH [?, ?], together with the masses and the mixing matrices of the neutralinos, charginos and third generation

sfermions. Masses of the first two generations of sfermions are evaluated (at tree-level) within `micrOMEGAs` in terms of the independent parameters of the model.

The routines for computing constraints are taken from CPsuperH, [**?**]

• `bsgnlo(), bsmumu(), btaunu(), gmuon()`
are the same as in the MSSM case.

• `deltaMd(),deltaMs()`
are the same as in the NMSSM case.

• `Bdll()`
computes the supersymmetric contribution to the branching fractions for $B_d \to \tau^+\tau^-$ in the CPVMSSM.

• `ABsg()`
computes the supersymmetric contribution to the asymmetry for $B \to X_s\gamma$.

• `EDMel(),EDMmu(),EDMTl()`
return the value of the electric dipole moment of the electron, d_e, the muon,d_μ, and of Thallium, d_{Tl} in units of ecm.

15.4 The UMSSM

The independent parameters of the UMSSM are the standard model parameters and weak scale soft SUSY parameters listed in `UMSSM/work/models/vars1.mdl`. All masses, mixing matrices and parameters of the different sectors of the model are computed by `micrOMEGAs` [**?**, **?**].

Some routines for computing constraints were taken from the `MSSM` and were adapted to the UMSSM. For example

• `masslimits()` which is essentially the same as in the `MSSM` except that the constraint on the light Higgs mass from the LHC was removed as other routines in the UMSSM include this constrain, or

• `deltarho()`
calculates the $\Delta\rho$ parameter in the UMSSM where in addition to MSSM contributions a pure UMSSM tree-level contribution from the extended Abelian gauge boson sector is included. Two other routines in C are included, `Zinvisible()` and `Zprimelimits()` that were defined above.

The remaining routines for computing B-physics, K-physics and Higgs observables as well as the anomalous magnetic moment of the muon were taken from `NMSSMTools_5.0.1` and adapted to the UMSSM [**?**,**?**]. To call these routines use `umssmtools(PDG_LSP)` where `PDG_LSP` is the PDG code of the LSP. The result is contained in four files (`UMSSM_inp.dat`, `UMSSM_spectr.dat`, `UMSSM_decay.dat` and `SM_decay.dat`) and the WARNING messages from these routines can be displayed with `slhaWarnings(stdout)`.

As in the NMSSM the following routines are available :

• `bsg(&M,&P), bsmumu(&M,&P), btaunu(&M,&P), gmuon(&M,&P),deltamd(&M,&P), deltams(&M,&P).`

as well as the routines

• `bdg(&M,&P)`, `bdmumu(&M,&P)`, `bxislllow(&M,&P)`, `bxisllhigh(&M,&P)`, `bxisnunu(&M,&P)`, `bpkpnunu(&M,&P)`, `bksnunu(&M,&P)`, `rdtaul(&M,&P)`, `rdstaul(&M,&P)`
to compute respectively $b \to d\gamma$, $B_d \to \mu^+\mu^-$, the $b \to sl^+l^-$ transition in the low ($[1,6]$ GeV^2) and high (≥ 14.4 GeV^2) $m^2_{l^+l^-}$ ranges, $B \to X_s\nu\bar{\nu}$, $B^+ \to K^+\nu\bar{\nu}$, $B \to K^*\nu\bar{\nu}$, $R_D \equiv \frac{\mathrm{BR}(B^+\to D\tau^+\nu_\tau)}{\mathrm{BR}(B^+\to Dl^+\nu_l)}$ and $R_{D^*} \equiv \frac{\mathrm{BR}(B^+\to D^*\tau^+\nu_\tau)}{\mathrm{BR}(B^+\to D^*l^+\nu_l)}$.

The K-physics observables $K^+ \to \pi^+\nu\bar{\nu}$, $K_L \to \pi^0\nu\bar{\nu}$, ΔM_K and ϵ_K can be computed respectively with
• `kppipnunu(&M,&P)`, `klpi0nunu(&M,&P)`, `deltamk(&M,&P)`, `epsk(&M,&P)`.

See the `README` of the UMSSM for further details.

16 Tools for new model implementation

It is possible to implement a new particle physics model in `micrOMEGAs`. For this the model must be specified in the `CalcHEP` format. `micrOMEGAs` then relies on `CalcHEP` to generate the libraries for all matrix elements entering DM calculations. Below we describe the main steps and tools for implementing a new model.

16.1 Main steps

- The command `./newProject MODEL`
  `MODEL` launched from the root `micrOMEGAs` directory creates the directory `MODEL`. This directory and the subdirectories contain all files needed to run `micrOMEGAs` with the exception of the new model files.

- The new model files in the `CalcHEP` format should then be included in the subdirectory `MODEL/work/models`. The files needed are `vars1.mdl`, `func1.mdl`, `prtcls1.mdl`, `lgrng1.mdl`, `extlib1.mdl`. For more details on the format and content of model files see [**?**].

- For odd particles and for the Higgs sector it is recommended to use the widths that are (automatically) calculated internally by `CalcHEP`/`micrOMEGAs`. For this one has to add the '!' symbol before the definition of the particle's width in the file `prtcls1.mdl`, for example

```
Full  name  | P | aP|PDG   |2*spin| mass |width |color|
Higgs 1     |h1 |h1 |25    |0     |Mh1   |!wh1  |1    |
```

- Some models contain external functions, if this is the case they have to be compiled and stored in the `MODEL/lib/aLib.a` library. These functions should be written in C and both functions and their arguments have to be of type `double`. The library `aLib.a` can also contain some functions which are called directly from the *main* program. The `MODEL/Makefile` automatically launches `make` in the `lib` directory and compiles the external functions provided the prototypes of these external functions are specified in `MODEL/lib/pmodel.h`. The user can of course rewrite his own —`lib/Makefile` if need be.

If the new `aLib.a` library needs some other libraries, their names should be added to the *SSS* variable defined in `MODEL/Makefile`.

The `MODEL` directory contains samples of *main* routines. In these sample main programs it is assumed that input parameters are provided in a separate file. In this case the program can be launched with the command:

```
./main data1.par
```

Note that for the direct detection module all quarks must be massive. However the cross sections do not depend significantly on the exact numerical values for the masses of light quarks.

16.2 Automatic width calculation

Automatic width calculation can be implemented by inserting the '!' symbol before the name of the particle width in the `CalcHEP` particle table (file prtcls1.mdl). In this case the width parameter should not be defined as a free or constrained parameter. Actually the `pWidth` function described in section 12 is used for width calculation in this case. We recommend to use the automatic width calculation for all particles from the 'odd' sector and for Higgs particles. For models which use SLHA parameter transfer (Section 16.6), the automatic width option will use the widths computed internally unless the user chooses to use those contained in the SLHA file by setting `useSLHAwidth=1`.

16.3 One-loop scalar vertices

The loop-induced couplings of scalars and pseudoscalars to gluons and photons can be calculated internally by micrOMEGAs with the use of the functions described below. First, effective vertices mush be defined in the model file. These vertices correspond to the Lagrangian

$$\mathcal{L} = \lambda h F_{\mu\nu} F^{\mu\nu} + \lambda_5 h F_{\mu\nu} \tilde{F}^{\mu\nu} \tag{61}$$

where the first term is the effective operator for the CP-even part while the second term is for the CP-odd part. Similar operators can be defined for the couplings to gluons with $F^{\mu\nu}$ replaced by $G^{\mu\nu}$. In the CalcHEP notation, the vertices will appear in the file `lgrng1.mdl` as

```
A    |A    |h    |     |-4*LAAh                |p1.p2*m1.m2-m2.p1*m1.p2
A    |A    |h    |     |-4*LAA5h               |eps(p1,m1,p2,m2)
```

where `LAAh, LAA5h` correspond respectively to λ, λ_5 in Eq. 61. These coefficients are computed internally by micrOMEGAs with the functions

•`lAAhiggs(Mass,Name)`

which calculates the coefficient of the loop-induced Scalar - photon - photon vertex based on the generic contributions of scalars/fermions/vector bosons given in Ref. [**?**]. Here `Name` specifies the scalar and `Mass` its mass. To do this, micrOMEGAs searches the model file to find the couplings of the scalar to all possible coloured scalars/fermions or vector bosons that can enter the one-loop process.

•`lAA5higgs(Mass,Name)`

which calculates the coefficient of the loop-induced Pseudoscalar - photon - photon vertex based on the results in Ref. [**?**] where `Name` specifies the pseudoscalar and `Mass` its mass.

As explained in [?, ?], an overall QCD correction factor is included for coloured particles in the loop. HDECAY [?] is used to generate this factor and is therefore tuned for the SM Higgs.

Similar effective operators can be defined for hgg vertices. Tthe vertex describing the couplings of the Higgs to gluon pairs is defined

```
G    |G    |h   |      |-4*LGGh*RQCDh                    |p1.p2*m1.m2-m2.p1*m1.p2
```

where `RQCDh` is the NLO QCD correction corresponding to the radiation of gluons for the SM Higgs. Other NLO QCD vertex corrections for heavy quarks, light quarks and coloured scalars are included in the coefficient of the vertex as described in [?, ?]. As for the photon, the coefficients including NLO QCD vertex corrections are computed internally by micrOMEGAs with the functions

•`lGGhiggs(Mass,Name)`
which calculates the coefficient of the loop-induced Scalar - gluon - gluon vertex based on the formulae in Ref. [?] where `Name` specifies the scalar and `Mass` its mass.

•`lGG5higgs(Mass,Name)`
which calculates the coefficient of the loop-induced Pseudoscalar - gluon - gluon vertex based on the formulae in Ref. [?] where `Name` specifies which scalar and `Mass` its mass.

The coefficient of the vertices should be defined in func1.mdl, for example for the scalar vertices

```
LAAh    |-cabs(lAAhiggs(Mh, "h"))
LGGh    |-cabs(lGGhiggs(Mh, "h"))
```

and the overall QCD corrections are defined with

```
aQCDh  |alphaQCD(Mh)/acos(-1)
RQCDh  |sqrt(1+149/12*aQCDh+68.6482*aQCDh^2-212.447*aQCDh^3)|
```

where `alphaQCD(Mh)` is the strong coupling, $\alpha_s(m_h)$ evaluated at the Higgs mass. Note that this construction is tuned for Higgs decays. In particular the `RQCDh` factor is caused by hadronization of gluons. For $g, g \to H$ production, other NLO factors need to be introduced by the user. The difference between NLO production and decays is ignored in micrOMEGAs. The scale entering the definition of α_s can be adapted to the process under consideration.

The four functions for loop-induced vertices were not available in micrOMEGAs' versions before v5.1, hence most of the models implemented use the functions introduced in [?] where the user had to give by hand the names of all particles that could contribute to the triangle diagrams. The IDM model is an example where this newer and simpler implementation of the loop-induced vertices can be found.

Note that in the interface to Lilith and HiggsBounds, the loop induced couplings to the Higgs are calculated by the `LiLithMO/hbBlocksMO` routines whether or not the Hgg and $H\gamma\gamma$ vertices are already implemented in the Lagrangian, as mentioned in section 14.1.

16.4 Using LanHEP for model file generation

For models with a large number of parameters and various types of fields/particles such as the MSSM, it is more convenient to use an automatic tool to implement the model.

LanHEP is a tool for Feynman rules generation. A few minor modifications to the default format of LanHEP have to be taken into account to get the model files into the `micrOMEGAs` format.

- The **lhep** command has to be launched with the -ca flag

  ```
  lhep  -ca source_file
  ```

- The default format for the file `prtcls1.mdl` which specifies the particle content has to be modified to include a column containing the PDG code of particles. For this, first add the following command in the LanHEP source code, before specifying the particles

  ```
  prtcformat fullname:
  'Full   Name ', name:' P ', aname:' aP', pdg:'  number  ',
  spin2,mass,width, color, aux, texname: '> LaTeX(A) <',
  atexname:'>  LateX(A+)  <' .
  ```

  Then for each particle define the PDG code. For instance:
  `vector  'W+'/'W-': ('W boson', pdg 24, mass MW, width wW).`

- LanHEP does not generate the file `extlib1.mdl`. `micrOMEGAs` works without this file but it is required for a `CalcHEP` interactive session. The role of this file is to provide the linker with the paths to all user's libraries needed at compilation. For example for the `lib/aLib.a` library define

  ```
  $CALCHEP/../MODEL/lib/aLib.a
  ```

  For examples see the `extlib1.mdl` files in the directory of the models provided.

16.5 QCD functions

Here we describe some QCD functions which can be useful for the implementation of a new model.

• `initQCD(alfsMZ,McMc,MbMb,Mtp)`

This function initializes the parameters needed for the functions listed below. It has to be called before any of these functions. The input parameters are the QCD coupling at the Z scale, $\alpha_s(M_Z)$, the quark masses, $m_c(m_c), m_b(m_b)$ and $m_t(pole)$.

• `alphaQCD(Q)`

calculates the running α_s at the scale `Q` in the $\overline{MS}$ scheme. The calculation is done using the `NNLO` formula in [**?**]. Thresholds for the b-quark and t-quark are included in n_f at the scales $m_b(m_b)$ and $m_t(m_t)$ respectively.

• `MtRun(Q), MbRun(Q), McRun(Q)`

calculates top, bottom and charm quarks running masses evaluated at NNLO.

• `MtEff(Q), MbEff(Q), McEff(Q),`

calculates effective top, bottom and charm quark masses using [**?**]

$$\begin{aligned} M^2_{eff}(Q) &= M(Q)^2\left[1+5.67a+(35.94-1.36n_f)a^2\right. \\ &+ \left.(164.14-n_f(25.77-0.259n_f))a^3\right] \end{aligned} \tag{62}$$

where $a = \alpha_s(Q)/\pi$, $M(Q)$ and $\alpha_s(Q)$ are the quark masses and running strong coupling in the $\overline{MS}$-scheme. In `micrOMEGAs`, we use the effective quark masses calculated at the scale $Q = 2$`Mcdm`. In some special cases one needs a precise treatment of the light quarks masses. The function

- `MqRun(M2GeV, Q)`

returns the running quark mass defined at a scale of 2 GeV. The corresponding effective mass needed for the Higgs decay width is given by

- `Mqeff(M2GeV, Q)`

16.6 SLHA reader

Very often the calculation of the particle spectra for specific models is done by some external program which writes down the particle masses, mixing angles and other model parameters in a file with the so-called **SLHA** format [**?**, **?**]. The `micrOMEGAs` program contains routines for reading files in the SLHA format. Such routines can be very useful for the implementation of new models.

In general a SLHA file contains several pieces of information which are called blocks. A block is characterized by its name and, sometimes, by its energy scale. Each block contains the values of several physical parameters characterized by a *key*. The key consists in a sequence of integer numbers. For example:

```
BLOCK MASS   # Mass spectrum
#  PDG Code     mass             particle
        25     1.15137179E+02   # lightest neutral scalar
        37     1.48428409E+03   # charged Higgs

BLOCK NMIX  # Neutralino Mixing Matrix
  1  1     9.98499129E-01   # Zn11
  1  2    -1.54392008E-02   # Zn12

BLOCK Au Q=  4.42653237E+02  # The trilinear couplings
  1  1    -8.22783075E+02   # A_u(Q) DRbar
  2  2    -8.22783075E+02   # A_c(Q) DRbar
```

- `slhaRead(filename,mode)`

downloads all or part of the data from the file `filename`. `mode` is an integer which determines which part of the data should be read form the file, `mode= 1*m1+2*m2+4*m4` where

```
  m1 = 0/1 -   overwrites all/keeps old data
  m2 = 0/1 -   reads DECAY /does not read   DECAY
  m4 = 0/1 -   reads BLOCK/does not  read   BLOCK
```

For example `mode=2` (`m1=0,m2=1`) is an instruction to overwrite all previous data and read only the information stored in the BLOCK sections of `filename`. In the same manner `mode=3` is an instruction to add information from DECAY to the data obtained previously. `slhaRead` returns the values:

```
  0  - successful reading
 -1  - can not open the file
```

```
 -2  - error in spectrum calculator
 -3  - no data
 n>0 - wrong file format at line n
```

• `slhaValExists(BlockName, keylength, key1, key2,...)`
checks the existence of specific data in a given block. `BlockName` can be substituted with any case spelling. The `keylength` parameter defines the length of the key set `{key1,key2,...}`. For example `slhaValExists("Nmix",2,1,2)` will return 1 if the neutralino mass mixing element `Zn12` is given in the file and 0 otherwise.
• `slhaVal(BlockName,Q, keylength, key1, key2,.....)`
is the main routine which allows to extract the numerical values of parameters. `BlockName` and `keylength` are defined above. The parameter `Q` defines the scale dependence. This parameter is relevant only for the blocks that contain scale dependent parameters, it will be ignored for other blocks, for example those that give the particle pole masses. In general a SLHA file can contain several blocks with the same name but different scales (the scale is specified after the name of the block). `slhaVal` uses the following algorithm to read the scale dependent parameters. If `Q` is less(greater) than all the scales used in the different blocks for a given parameter `slhaVal` returns the value corresponding to the minimum(maximum) scale contained in the file. Otherwise `slhaVal` reads the values corresponding to the two scales Q_1 and Q_2 just below and above `Q` and performs a linear interpolation with respect to log(`Q`) to evaluate the returned values.

Recently it was proposed to use an extension of the SLHA interface to transfer Flavour Physics data [**?**]. Unfortunately the structure of the new blocks is such that they cannot be read with the `slhaVal` routine. We have added two new routines for reading such data
• `slhaValFormat(BlockName, Q, format)`
where the *format* string allows to specify data which one would like to extract from the given block `BlockName`. For instance, to get the $b \to s\gamma$ branching ratio from the block

```
Block FOBS # Flavour observables
# ParentPDG type value       q   NDA ID1 ID2 ID3 ... comment
    5    1   2.95061156e-04  0     2   3  22      # BR(b->s gamma)
  521    4   8.35442304e-02  0     2 313  22      # Delta0(B->K* gamma)
  531    1   3.24270419e-09  0     2  13 -13      # BR(B_s->mu+ mu-)
  ...
```

one has to use the command `slhaValFormat("FOBS", 0., "5 1 %E 0 2 3 22")`. In this command the *format* string is specified in C-style. The same routine can be used to read HiggsBound SLHA output.

A block can also contain a textual information. For example, in `HIGGSBOUNDS` a block contains the following records,

```
Block HiggsBoundsResults
#CHANNELTYPE 1: channel with the highest statistical sensitivity
    1          1         328                 # channel id number
    1          2           1                 # HBresult
    1          3  0.72692779334500290         # obsratio
    1          4           1                 # ncombined
    1          5 ||(p p)->h+..., h=1 where h is SM-like (CMS-PAS-HIG-12-008)|| # text description of channel
```

In particular, the last record contains the name of the channel which gives the strongest constraint on the Higgs. To extract the name of this channel one can use the new function

• `slhaSTRFormat("HiggsBoundsResults","1 5 || %[^|]||",channel);`
which will write the channel name in the text parameter *channel*.
• `slhaWarnings(fileName)`
writes into the file the warnings or error message stored in the SPINFO block and returns the number of warnings. If `FD=NULL` the warnings are not written in a file.
• `slhaWrite(fileName)`
writes down the information stored by `readSLHA` into the file. This function can be used for testing purposes.

SLHA also describes the format of the information about particle decay widths. Even though `micrOMEGAs` also performs the width calculation, one might choose to read the information from the SLHA file.
• `slhaDecayExists(pNum)`
checks whether information about the decay of particle `pNum` exists in the SLHA file. `pNum` is the particle PDG code. This function returns the number of decay channels. In particular zero means that the SLHA file contains information only about the total width, not on branching ratios while -1 means that even the total width is not given.
• `slhaWidth(pNum)`
returns the value of particle width.
• `slhaBranch(pNum,N, nCh)`
returns the branching ratio of particle `pNum` into the N-th decay channel. Here `0<N<=slhaDecayExists(pNum)`. The array `nCh` is an output which specifies the PDG numbers of the decay products, the list is terminated by zero.

The functions `slhaValExists`, `slhaVal`, `slhaDecayExists`, `slhaWidth` can be used directly in `CalcHEP` model files, see an example in
`MSSM/work/models/func3.mdl`. Note that in this example the call to `slhaRead` is done within the function `suspectSUGRAc`.

16.6.1 Writing an SLHA input file

We have included in the `micrOMEGAs` package some routines which allow to write an SLHA input file and launch the spectrum generator via the `CalcHEP` *constraints* menu. This way a new model can be implemented without the use of external libraries. The routines are called from `func1.mdl`, see example below.
• `openAppend(fileName)`
deletes the input file `fileName` and stores its name. This file will then be filled with the function `aPrintF`.
• `aPrintF(format,...)`
opens the file `fileName` and writes at the end of the file the input parameters needed in the SLHA format or in any other format understood by the spectrum calculator. The arguments of `aPrintF` are similar to the arguments of the standard `printf` function.
• `System(command, ...)` generates a command line which is launched by the standard `system` C-function. The parameter *command* works here like a format string and can contain %s, %d elements. These are replaced by the next parameters of the `System` call.

For example to write directly the SLHA model file needed by `SuSpect` to compute the spectrum in a CMSSM(SUGRA) model, one needs to add the following sequence in the `func1.mdl` model file.

```
open  |openAppend("suspect2_lha.in")
```

```
input1|aPrintF("Block MODSEL  # Select model\n  1  1   # SUGRA\n")
input2|aPrintF("Block SMINPUTS\n 5 %E#mb(mb)\n 6 %E#mt(pole)\n",MbMb,Mtp)
input3|aPrintF("BLOCK MINPAR\n 1 %E #m0\n 2 %E #m1/2\n ",Mzero,Mhalf)
input4|aPrintF("3 %E #tb\n 4 %E #sign(mu)\n 5 %E #A0\n",tb,sgn,A0)
sys   |System("./suspect2.exe")
rd    |slhaRead("suspect2_lha.out",0)
```

It is possible to cancel the execution of a program launched with `System` if it runs for too long. For this we have introduced two global parameters `sysTimeLim` and `sysTimeQuant`. `sysTimeLim` sets a time limit in milliseconds for `System` execution, if `sysTimeLim==0` (the default value) the execution time is not checked. The time interval between checks of the status of the program launched with `System` is specified by the parameter `sysTimeQuant`, the default value is set to 10. Note that it is preferable not too use too large a value for `sysTimeQuant` as it defines the lower time limit for a system call.

The function prototypes are available in
`CalcHEP_src/c_source/SLHAplus/include/SLHAplus.h`

16.7 Routines for diagonalisation

Very often in a new model one has to diagonalize mass matrices. Here we present some numerical routines for diagonalizing matrices. Our code is based on the `jacobi` routine provided in [**?**]. To use the same routine for a matrix of arbitrary size, we use a C option that allows to write routines with an arbitrary number of arguments.

• `initDiagonal()` should be called once before any other `rDiagonal(A)` routine described below. `initDiagonal()` assigns zero value to the internal counter of eigenvalues and rotation matrices. Returns zero.

• `rDiagonal(d,M11,M12,..M1d,M22,M23...Mdd)`
diagonalizes a symmetric matrix of dimension `d`. The $d(d+1)/2$ matrix elements, `Mij` ($i \leq j$), are given as arguments. The function returns an integer number `id` which serves as an identifier of eigenvalues vector and rotation matrix.

• `MassArray(id, i)`
returns the eigenvalues m_i ordered according to their absolute values.

• `MixMatrix(id,i,j)`
returns the rotation matrix R_{ij} where

$$M_{ij} = \sum_k R_{ki} m_k R_{kj}$$

A non-symmetric matrix, for example the chargino mass matrix in the MSSM, is diagonalized by two rotation matrices,

$$M_{ij} = \sum_k U_{ki} m_k V_{kj}.$$

• `rDiagonalA(d,M11,M12..M1d,M21,M22...Mdd)`
diagonalizes a non-symmetric matrix, the d^2 matrix elements, `Mij`, are given as arguments. The eigenvalues and the V rotation matrix are calculated as above with `MassArray` and `MixMatrix`.

• `MixMatrixU(id,i,j)`
returns the rotation matrix U_{ij}.

One can find example of work of diagonalozation routines in `mdlIndep/diagonal.c`. The function prototypes can be found in
`CalcHEP_src/c_source/SLHAplus/include/SLHAplus.h`

17 Mathematical tools

Some mathematical tools used by `micrOMEGAs` are available only in C format. Prototypes of these functions can be found in

`include/micromegas_aux.h`

17.1 Integration

• `simpson(F, x1, x2, eps,&err)`
performs numerical integration of the function $F(x)$ in the interval [`x1,x2`]. `simpson` tries to reach relative precision `eps` or absolute precision $\frac{eps}{10}\int_{x1}^{x2}|F(x)|dx$. For one interval `simpson` compares the results of 3, 5 and 9 points formulas. If they do not agree within the required precision, `simpson` splits the interval in two and applies the same procedure recursively. The smallest interval is $|x_2-x_1|/2^{25-log(eps)}$.

A non-zero error code *err* means
1: NAN in integrand: `simpson` replaces NAN by zero and continues integration
2: Too deep recursion: one needs the smallest minimal interval to reach the required precision. For instance, these two error codes will show up in the integration of $\int dx/|x|^{0.8}$ with a precision $eps = 10^{-3}$.
4: Lost of precision: If in a small interval $\approx |x1-x2|/2^{10}$ the derivative of the function changes sign more than twice, it is treated as a lost of precision in the integrand and `simpson` decreases precision to `eps=0.01` and treats uncertainties of each interval as a random numbers.

In case of several error messages, *err* contains their sum. If the last argument of `simpson` is $NULL$, the error messages are displayed on the screen. If the integrand has a discontinuity or a pole, the requirement of reaching a fixed relative precision can not be satisfied. Thus we also stop the recursion if the integration uncertainty on the interval is small as compared to the current estimation of $\frac{eps}{10}\int_{x1}^{x2}|F(x)|dx$. Since the current estimation of this integral can be significantly smaller than the final one, we keep the sum of errors obtained at the end of the recursion chains and compare it with the final estimation of the total integral. It the sum of errors is small, then the error 2 is not generated.

We have implemented a debugging tool for the `simpson` code based on `gdb`. To initiate it the user has to launch the `./main` code under `gdb` and set a breakpoint

```
gdb ./main
break verifySimpson
r inputParameters              // to launch code
```

The execution of the program will stop when `simpson` generates an error code. With the `bt` command one can see the sequence of calls leading to the problematic call of `simpson`. After

```
n       // next
```

command one reaches the cycle which is executed until the variable `show`$\neq 0$. The user should see on the screen

```
293   while(show)  {drawP(f, par, x1, x2,ans,nErr);
```

By default `show=0` and the cycle is not executed, but it can be changed by the `set` command

```
set show=1
n
```

and the user will see a plot of the function $F(x)$ on the screen. When the user closes the window with the plot, he/she will reach the next line of code

```
294      continue; }
```

Here the user can leave the cycle using the commands

```
set show=0
c                // continue
```

or verify a narrower range of the function $F(x)$ by redefining the variables `x1` and `x2`.

The file `mdlIndep/simpson.c` contains several examples which allow to test how this routine works.

• `gauss(F,x1,x2,N)`
performs Gauss N-point integration for $N < 8$.

• `peterson21(F, x1, x2,&aErr)`
performs numerical integration of the function $F(x)$ in the interval $[x1, x2]$ using Gauss-Kronrod-Peterson formula with 21 points. `aErr` is a parameter which returns the accuracy of the calculation using 10 points subset.

17.2 Solution of differential equations.

• `odeint(Y, Dim, x1, x2, eps,h1, deriv)`
solves a system of `Dim` differential equations in the interval $[x1, x2]$. The `Dim` component array Y contains the starting variables at `x1` as an input and is replaced by the resulting values at `x2` as an output. *eps* determines the precision of the calculation and `h1` gives an estimation of step of integration. The function `deriv` calculates $Y_i' = dY_i/dx$ with the call $deriv(x, Y, Y')$. The Runge-Kutta method is used, see details in [?].

• `stiff ( first, x1, x2, Dim, Y, Yscal, eps, &htry,derivs)`

• `stifbs( first, x1, x2, Dim, Y, Yscal, eps, &htry,derivs)`
these two functions solve stiff differential equations. Both routines are slightly adapted codes from [?]. Here the parameters `x1, x2, Dim, Y` have the same meaning as in the routine `odeint` above. The parameter `first` should be set to one for the first call to the routines with a given number of equations `Dim` and to zero for subsequent calls. The

flag `first` is used for memory allocation. If `Yscal=NULL` the parameter `eps` defines the absolute precision of calculation ($\delta Y_i < eps$). Otherwise, the precision is defined by the condition δ`Y`$_{\texttt{i}} <$ `epsYscale`$_i$. The parameter `htry` defines the initial step of integration and contains the last step of integration used during calculations. The function `derivs` evaluates the differential equation $F = dY/dx$ and its partial derivatives:

`derivs(x, Y, F, h, dFdx,dFdY)` where `dFdY[i*Dim+j]`$=\frac{dF_i}{dY_j}$

This routine can be called with parameters `dFdx=NULL` and `dFdY=NULL`. The parameter `h` presents current step of integration and can be used for numerical evaluation of `dFdx`.

17.3 Interpolation

• `polint3(x,Dim,X,Y)`
performs cubic interpolation for *Dim*-dimension arrays X,Y. Similar functions, `polint1` performs linear interpolations.

• `spline(x, y, dim, y2)`

• `splint(x, y, y2, dim, double x0, &y0)`

`spline` constructs cubic spline and `splint` calculates spline interpolation `y0` for a given point `x0`. Here `x` and `y` are a grid of function arguments and function values $y_i = Y(x_i)$. The function `spline` fills an array of second derivatives `y2` which is used by `splint`.

• `buildInterpolation(F,x1,x2,eps,delt, &Dim,&X,&Y)`
constructs a cubic interpolation of the function `F` in the interval $[x1, x2]$. *eps* controls the precision of interpolation. If $eps < 0$ the absolute precision is fixed, otherwise a relative precision is required. The *delt* parameter limits the distance between interpolation points: $|x_i - x_{i+1}| < delt|x2 - x1|$. The function checks that after removing any grid point, the function at that point can be reproduced with a precision *eps* using only the other points. It means that the expected precision of interpolation is about $eps/16$. `Dim` gives the number of points in the constructed grid. X and Y are variables of the `double*` type. The function allocates memory for the `Dim` array for each of these parameters. $X[i]$ contains the x-grid while $Y[i] = F(X[i])$.

17.4 Functions with additional parameters

We use routines that work with functions of type `double F(double x,void*arg)`, where the structure of `arg` is open, here `arg` presents the memory address where these parameters are stored. This is useful for example for integrating a function f(x,y) over x, y is then passed as 'arg'. The functions available are `simpson_arg`, `peterson21_arg`, `buildInterpolation_arg`. The additional argument is the address `arg` which is specified just after the function name.

The recent versions of C-compiler may refuse the substitution of a function if the type of the argument does not correspond. For example, if there is a problem for the function

```
double myF(double x, double *par)
```

to be used in `simpson_arg`, we recommend to use a type conversion:

```
simpson_arg( (funcArg) myF, ....)
```

17.5 Plots

• `displayPlot(title,xName,xMin,xMax,lScale, N, ...)`
displays several curves/histograms on one plot. Here `title` contains some text, `xName` is the name of a variable, `xMin,xMax` are the lower and upper limits. If `lScale` $\neq$ `0`, a logarithmic scale is used for the x axis.

`N` is the number of curves/histograms to display. After the parameter `N`, `displayPlot` expects $N \times 4$ parameters, where each tetrad can contain

text label	dimension of array	array of data	array of error
text label	dimension of array	array of data	NULL
text label	0	double *f(double x)	NULL
text label	0	double *f(double x, void *arg)	arg

where the first line is used for an histogram with error bars, the second line for a tabulated function, the third for a function f(x) and the fourth for a function f(x,arg) which also depends on some arguments contained in the structure `arg`. For a linear scale `lScale=0`, the arrays of data and errors should correspond to a grid

$$x_i = \text{xMin} + (i + 0.5)(\text{xMax} - \text{xMin})/\text{Dim} \ ,$$

where $i = 0, ..., \text{Dim} - 1$. For logarithmic scale

$$x_i = \text{xMin} \cdot \left(\frac{\text{xMax}}{\text{xMin}}\right)^{\frac{i+0.5}{\text{Dim}}}$$

`displayPlot` uses the X11 font presented in the file `CalcHEP_src/calchep.ini`. One can change the font by editing the file or via key signals: $Ctrl\pm$ increases/decreases font size, while $Ctrl\lessgtr$ - changes the font saving its size.

17.6 Bessel functions

• `bessI0(x), bessI1(x), bessK0(x), bessK1(x), bessK2(x)`
Bessel functions I_0, I_1, K_1, K_2.

• `K1pol(x)`=$K_1(\frac{1}{x})e^{\frac{1}{x}}\sqrt{\frac{2}{\pi x}}$
• `K2pol(x)`=$K_2(\frac{1}{x})e^{\frac{1}{x}}\sqrt{\frac{2}{\pi x}}$
`micrOMEGAs` uses these functions for calculating the relic density. For small values of $x = T/M_{cdm}$, where these functions can be presented as polynomials in x. `K1pol(0)=K2pol(0)=1`.

17.7 Statistics

• `FeldmanCousins(n0, b,cl)`
is the Feldman-Cousins [?] function for a Poisson distribution. Here `n0` is the observed number of events, `b` - the expected background, `cl`<1 - the requested confidence level. This function sets the upper limit on the number of signal events compatible with `n0` and `cl`.
• `ch2pval(ndf,chi2)`
returns the `p-value` assuming a χ^2 distribution with `ndf` degrees of freedom (expected

χ^2) and `chi2` is the observed χ^2.

$$\texttt{ch2pval}(k,q) = \int_q^\infty \frac{1}{2^k\Gamma(\frac{k}{2})} Q^{\frac{k}{2}-1} e^{-\frac{Q}{2}} dQ$$

A An updated routine for $b \to s\gamma$ in the MSSM

The calculation of $b \to s\gamma$ was described in micromegas1.3 [**?**]. The branching fraction reads

$$B(\bar{B} \to X_s\gamma) = B(\bar{B} \to X_c e\bar{\nu}) \left|\frac{V_{ts}^* V_{tb}}{V_{cb}}\right|^2 \frac{6\alpha_{em}}{\pi f(z_0)} K_{NLO}(\delta) \tag{63}$$

where $\alpha_{em} = 1./137.036$, the factor K_{NLO} involves the photon energy cut-off parameter δ and $f(z_0) = 0.542 - 2.23(\sqrt{z_0} - 0.29)$ depends on $z_0 = (m_c/m_b)^2$ defined in terms of pole masses. In the code the standard model and Higgs contribution at NLO were included as well as the leading order SUSY contributions. However in the last few years the NNLO standard model contribution has been computed [**?**] and shown to lead to large corrections, shifting the standard model value by over 10%. It was also argued that the NNLO SM result could be reproduced from the NLO calculation by appropriately choosing the scale for the c-quark mass [**?**, **?**].

In this improved version of the `bsgnlo` routine, we have changed the default value for the parameter $z_1 = (m_c/m_b)^2$ where m_c is the $\overline{MS}$ running charm mass $m_c(m_b)$. Taking $z_1 = 0.29$ allows to reproduce the NNLO result. It is therefore no longer necessary to apply a shift to the `micrOMEGAs` output of $b \to s\gamma$ to reproduce the SM value.

We have also updated the default values for the experimentally determined quantities in Eq. 63, see Table 6, and we have replaced the factor $f(z_0)$ by C_{sl} where

$$C_{sl} = \left|\frac{V_{ub}}{V_{cb}}\right|^2 \frac{\Gamma(\bar{B} \to X_c e\bar{\nu})}{\Gamma(\bar{B} \to X_u e\bar{\nu})} \tag{64}$$

accounts for the m_c dependence in $\bar{B} \to X_c e\bar{\nu}$.

$B(\bar{B} \to X_c e\bar{\nu})$	0.1064 [**?**]
C_{sl}	0.546 [**?**]
$\lvert V_{ts}^* V_{tb}/V_{cb}\rvert^2$	0.9613 [**?**]
A	0.808
λ	0.2253
$\bar{\rho}$	0.132
$\bar{\eta}$	0.341
m_b/m_s	50
$\lambda_2 \approx \frac{1}{4}(m_{B^*}^2 - m_B^2)$	0.12GeV^2 [**?**]
$\alpha_s(M_Z)$	0.1189

Table 6: Default values in `micrOMEGAs`

The CKM matrix elements in the Wolfenstein parametrisation given in Table 6 are used to compute the central value of $ckmf$ at order λ^4,

$$ckmf = \left|\frac{V_{ts}^* V_{tb}}{V_{cb}}\right|^2 = 1 + \lambda^2(2\bar{\rho} - 1) + \lambda^4(\bar{\rho}^2 + \bar{\eta}^2 - A^2) \tag{65}$$

With these default values the NLO- improved SM contribution is $B(\bar{B} \to X_s\gamma)|_{\rm SM} = 3.27 \times 10^{-4}$ which corresponds to the result of Gambino and Giordano [**?**] after correcting for the slightly different CKM parameter used ($ckmf = 0.963$).

We have performed a comparison with superIso which includes the NNLO SM calculation for 10^5 randomly generated MSSM scenarios. The results are presented in Fig. 1 after applying a correction factor in superISO to account for the different value for the overall factor $F = B(\bar{B} \to X_c e\bar{\nu}) \left|\frac{V_{ts}^* V_{tb}}{V_{cb}}\right|^2 / C_{sl}$. The ratio of $F_{micro}/F_{ISO} = 0.942$. The two codes agree within 5% most of the time.

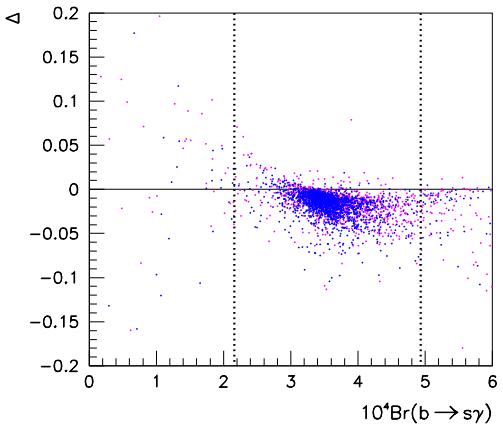

Figure 1: Relative difference for $B(\bar{B} \to s\gamma)$ between micromegas2.4 and superIso3.1. the vertical lines show the 3σ experimentally measured value.