Date: June 10, 2016

The micrOMEGAs user's manual, version 4.1

G. Bélanger[1], F. Boudjema[1], A. Pukhov[2], A. Semenov[3].

1) LAPTH, Univ. de Savoie, CNRS, B.P.110, F-74941 Annecy-le-Vieux, France
2) Skobeltsyn Inst. of Nuclear Physics, Moscow State Univ., Moscow 119992, Russia
3) Joint Institute for Nuclear Research (JINR) 141980, Dubna, Russia

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` calculates 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 (LOP). 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 LOP or weakly-interactive massive particle (WIMP) will be denoted by χ.

`micrOMEGAs` contains both C and Fortran routines. Below we describe only the C-version of the routines, in general we use the same names and the same types of argument for both C and Fortran functions. We always use `double(real*8)` variables for float point numbers and `int(INTEGER)` for integers. In this manual we use *FD* for file descriptor variables, the file descriptors are `FILE*` in C and `channel number` in Fortran. The symbol `&` before the names of variables in C-functions stands for the address of the variable. It is used for *output parameters*. In Fortran calls there is no need for `&` since all parameters are passed via addresses. In C programs one can substitute `NULL` for any output parameter which the user chooses to ignore. In Fortran one can substitute `cNull, iNull, r8Null` for unneeded parameters of *character*, *integer* and *real*8* type respectively.

A few C-functions use pointer variables that specify an *address* in the computer memory. Because pointers do not exist in Fortran, we use an `INTEGER*8` variable whose length is sufficient to store a computer address.

The complete format for all functions can be found in `sources/micromegas.h` (for C) or `sources/micromegas_f.h` (for Fortran). Examples on how to use these functions are provided in the MSSM/main.c[F] 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 symmetries. 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. In versions 4.X, 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 convention that the names of particles in the second 'dark' sector starts with '~~'. 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.

3 Downloading and compilation of micrOMEGAs.

To download micrOMEGAs, go to

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

and unpack the file received, `micromegas_4.2.tgz`, with the command

```
tar -xvzf micromegas_4.2.tgz
```

This should create the directory `micromegas_4.2/` which occupies about 67Mb 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.

`Makefile`	to compile the kernel of the package
`CalcHEP_src/`	generator of matrix elements for micrOMEGAs
`Packages/`	external codes
`clean`	to remove compiled files
`man/` — contains the manual: description of micrOMEGAs routines	
`newProject`	to create a new model directory structure
`sources/`	micrOMEGAs code
MSSM model directory	
`MSSM/`	
`Makefile`	to compile the code and executable for this model
`main.c[pp] main.F`	files with sample *main* programs
`lib/`	directory for routines specific to this model
`Makefile`	to compile the auxiliary code library *lib/aLib.a*
`*.c *.f`	source codes of auxiliary functions
`work/`	CalcHEP working directory for the generation of matrix elements
`Makefile`	to compile the library *work/work_aux.a*
`models/`	directory for files which specifies the model
`vars1.mdl`	free variables
`func1.mdl`	constrained variables
`prtcls1.mdl`	particles
`lgrng1.mdl`	Feynman rules

`tmp/`	auxiliary directories for CalcHEP sessions
`results/`	
`so_generated/`	storage of matrix elements generated by CalcHEP
`calchep/`	directory for interactive CalcHEP sessions

Directories of other models which have the same structure as `MSSM/`

`NMSSM/`	Next-to-Minimal Supersymmetric Model [?, ?]
`CPVMSSM/`	MSSM with complex parameters [?, ?]
`UMSSM/`	U(1) extensions of the MSSM [?, ?]
`IDM/`	Inert Doublet Model [?]
`LHM/`	Little Higgs Model [?]
`RHNM/`	Right-handed Neutrino Model [?]
`SM4/`	Toy model with a 4th generation of lepton and neutrino DM
`Z3M/`	A model with scalar DM and Z_3 discrete symmetry— [?, ?]
`Z4ID/`	A model with Z_4 symmetry— [?, ?]
`mdlIndep/`	For model independent computation of DM signals

3.2 Compilation of CalcHEP and micrOMEGAs routines.

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 is **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 compilation of codes.

3.3 Module structure of main programs.

Each model included in micrOMEGAs is accompanied with sample files for C and Fortran programs which call micrOMEGAs routines, the *main.c*, *main.F* 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 OMEGA              //Calculates the relic density
#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
                         //and recoil energy distribution for various nuclei
#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.
#define SHOWPLOTS  //Displays  graphical plots on the screen
```

Other modules which require a link to external programs can also be defined, in this case the path to the required code must be specified, for example

```
#define LILITH "../Packages/Lilith-1.1.2"
```

Note that HiggsBounds and HiggsSinals are no longer included in the `micrOMEGAs`'s distribution and must be installed separately by the user.

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

3.4 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
```

It 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 3 options: *'c'* , *'F'*, *'cpp'* which correspond to `C`, `FORTRAN` and `C++` sources. `[g]make` called in the model directory automatically launches `[g]make` in subdirectories *lib* and *work* to compile

`lib/aLib.a` - library of auxiliary model functions, e.g. constraints,
`work/work_aux.a` - library of model particles, free and dependent parameters.

3.5 Command line parameters of main programs.

The default versions of *main.c/F* 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

The list of the global parameters and their default values are given in Tables 1, 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	$\mathrm{kpc}^2/\mathrm{Myr}$	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	225.2	km/s	Galaxy velocity of the Earth
Vrot	220	km/s	Galaxy rotation velocity at Rsun
Vesc	600	km/s	Escape velocity at Rsun

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	

Table 3: Evaluated global variables

Name	units	comments	Evaluated by
CDM1	*character*	name of first DM particle	sortOddParticles
CDM2	*character*	name of second DM particle	sortOddParticles
Mcdm1	GeV	Mass of the first Dark Matter particle	sortOddParticles
Mcdm2	GeV	Mass of the second DM particles	sortOddParticles
Mcdm	GeV	$\min(Mcdm1, Mcdm2)$ if both exist	sortOddParticles
dmAsymm		Asymmetry between relic density of DM- $\overline{DM}$	darkOmega[FO]
fracCDM2		fraction of CDM2 in relic density.	darkOmega2
Tstart, Tend	GeV	Temperature interval	
		for solving the differential equation	darkOmega[2]

5 Setting of model parameters, spectrum calculation, parameter display.

The independent parameters that characterize a given model are listed in the file `work/models/vars1.mdl`. Three functions can be used to set the value 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 3.5)

```
    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.

Note that in Fortran, numerical constants should be specified as Real*8, for example

```
        call assignValW('SW', 0.473D0)
```

A common mistake is to use Real*4.

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 listed above 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*. See example in

```
MSSM/work/models/func1.mdl
```

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

• `sortOddParticles(txt)`
which also sorts the odd particles with increasing masses, writes the name of the lightest odd particle in `txt` and assigns the value of the mass of the lightest odd particle to the global parameter `Mcdm`. 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 has to be called after a reassignment of any input parameter. These routine was updated for the case of two DM particles. `sortOddParticles` fills text parameters *CDM1* and *CDM2* which present the name of lightest particle which names are started with one and two tildes respectively. `Mcdm1` and `Mcdm2` are masses of these particles. If we have only one kind of DM then for the absent component $Mcdm_i = 0$ and CDM_i=NULL (in Fortran the string if filled by space symbols).

• `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 $1, 3, -3$ 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. In the FORTRAN version this function is `Subroutine pdg2name(nPDG, pName)` and the character variable `pName` consists of white spaces if the particle does not exist in the model.

• `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. In the FORTRAN version this function is `Subroutine nextOdd(n,pName,pMass)`

• `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)` just returns the value of variable *name* and writes an error message if it cannot recognize a parameter name. The variables accessible by these 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.
• `printHiggsMasses(FD, sort)`
prints the masses and widths of 'even' 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`). [1] 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`. In Fortran, the subroutine
• `setVVdecay(VWdecay,VZdecay)` changes the switches and calls `cleanDecayTable()`. The `sortOddParticles` command which must be used to recompute the particle spectrum after changing the model parameters also clears the decay table.

If the particle widths were stored in a SLHA file (Susy Les Houches Accord [?]) downloaded by `micrOMEGAs`, then the SLHA value will be used, the widths then do not depend on the `VW/VZdecay` switches. To avoid downloading particle widths, one can use `slhaRead(`*fileName*`,mode=4)` to read the content of the SLHA file, see the description in Section 14.5.

The temperature dependence of the effective number of degrees of freedom can be set with
• `loadHeffGeff(char*fname)`
allows to modify the temperature dependence of the effective number of degrees of freedom by loading the file `fname` which contains a table of $h_{eff}(T), g_{eff}(T)$. 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 default file is `std_thg.tab` and is downloaded automatically if loadHeffGeff is not called is user's main program. Five other files are provided in the sources/data directory: `HP_A_thg.tab`, `HP_B_thg.tab`, `HP_B2_thg.tab`, `HP_B3_thg.tab`, and `HP_C_thg.tab`. They correspond to sets A, B, B2, B3, C in [?]. The user can substitute his/her own table as well, if so, the file must contain three columns containing the numerical values for T, h_{eff}, g_{eff}, the data file can also contain comments

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

for lines starting with #.
•`improveCrossSection( p1,p2,p3,p4, Pcm, &address)`
allows to substitute a new cross-section for a given process. Here `p1,p2` are the names of particles in the initial state and `p3,p4` those in the final state. `Pcm` is the center of mass momentum and `address` ... This function is useful if for example the user wants to include her/his one-loop improved cross-section calculation in the relic density computation.

6.2 Calculation of relic density for one-component Dark Matter models.

All routines to calculate the relic density in version 3 are still available in this version. For these routines, the difference between the two dark sectors is ignored. These routines are intended for models with either a Z_2 or Z_3 discrete symmetry.
• `vSigma(T,Beps,fast)`
calculates the thermally averaged cross section for DM annihilation times velocity at a temperature T [GeV],

$$\begin{aligned}\sigma_v(T) &= \frac{T}{8\pi^4\overline{n}(T)^2}\int ds\sqrt{s}K_1\left(\frac{\sqrt{s}}{T}\right)\sum_{\tilde{\alpha},\tilde{\beta}} p^2_{\tilde{\alpha}\tilde{\beta}}(s)g_{\tilde{\alpha}}g_{\tilde{\beta}} \\ &\quad \big(\sum_{x\geq y}\sigma_{\tilde{\alpha}\tilde{\beta}\to xy}(s)+\frac{1}{2}\sum_{x\tilde{\gamma}}\sigma_{\tilde{\alpha}\tilde{\beta}\to x\tilde{\gamma}}(s)\big) \qquad (1)\\ \overline{n}(T) &= \frac{T}{2\pi^2}\sum_{\tilde{\alpha}} g_{\tilde{\alpha}}m^2_{\tilde{\alpha}}K_2(\frac{m_{\tilde{\alpha}}}{T}), \qquad (2)\end{aligned}$$

Here $\tilde{\alpha}$, $\tilde{\beta}$, $\tilde{\gamma}$ is used for `Odd` particles and x,y for `Even` particles. $\sigma_{\tilde{\alpha}\tilde{\beta}\to x[\tilde{\gamma}/y]}$ is the cross section for the corresponding process averaged over the spins of incoming particles and summed over the spins of outgoing particles. σ_v should represent the rate of disappearance of `Odd` particles, therefore when a final state particle has a non-zero decay branching ratio to odd particles, the annihilation cross section for this process is multiplied by the corresponding branching ratio into all SM particles. For the same reason cross sections for semi - annihilation processes contribute to `vSigma` with a factor $\frac{1}{2}$. 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 $\tilde{\alpha} \neq \tilde{\beta}$ that each $\sigma_{\tilde{\alpha}\tilde{\beta}}$ term will be presented twice. The value for σ_v 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* calculation. 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 the Fortran version, the function `vSigmaTCh(i,weight,pdg,process)`serves the same purpose. This function returns 0 if i exceeds the number of annihilation channels and 1 otherwise, $i \geq 1$. The variable *real*8 weight* gives the relative contribution of each annihilation channel and *integer pdg(5)* contains the codes of incoming and outgoing particles in the annihilation process. *character*40 process* contains a textual description of annihilation

processes.

• `vSigmaCC(T,cc,mode)`
calculates the thermally averaged *cross section* $\times$ *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 10. The returned value is given in `[pb]`.

If *mode* $\neq 0$, `vSigmaCC` calculates the contribution of a given process to the total annihilation cross section, see Eq.1. The incoming particles should belong to the odd sector. For $2 \to 2$ processes the result should be identical to the one obtained via `vSigma` above. For this mode, verb—vSigmaCC— includes combinatoric 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 instead defined by the integral

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

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

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

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)`
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 $Y(T_f) = 2.5 Y_{eq}(T_f)$. For asymmetric DM this condition reads $2\sqrt{Y^+(T_f)Y^-(T_f)} = 2.5 Y_{eq}(T_f)$. 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.

`darkOmega` solves the differential equation for the abundance $Y(T)$ in the temperature interval `[Tend,Tstart]` defined by the conditions $Y(T_{start}) \approx 1.1 Y_{eq}(T_{start})$, $Y(T_{end}) \approx 10 Y_{eq}(T_{end})$. For temperatures below `Tend`, the contribution for Y_{eq} is neglected and the differential equation is integrated explicitely. The solution in the interval `[Tend,Tstart]` interval 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 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 $p1, p2 \to p3, p4$ 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. In the Fortran version one uses instead the function
`omegaCh(i,weight,pdg,process)`. These array and function are similar to `vSigmaTCh` described above. The array `omegaCh` if filled after calling either `darkOmegaFO` or `printChannels`.

There is an option to calculate the relic density in models with DM -$\overline{DM}$ asymmetry. In this case we assume that the number difference DM -$\overline{DM}$ 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 for annihilation cross sections provided by an external function. Here `vs_a` is the annihilation cross section in [pb] as a function of the temperature in [GeV] units while `vs_sa` is the semi-annihilation cross section. `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 $2 \to 2$ processes do not 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`. Note that if the user substitute some function which is not in tabular form, `darkOmegaExt` can be slow as it has not been optimized.

`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.

6.3 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. `darkOmega2` also calculates the global parameter `fracCDM2` which represents the mass fraction of CDM2 in the total relic density

$$\Omega = \Omega_1 + \Omega_2 \tag{3}$$
$$\texttt{fracCDM2} = \frac{\Omega_2}{\Omega} \tag{4}$$

This parameter is then used in routines which calculate the total signal from both DM candidates in direct and indirect detection experiments, `nucleusRecoil`, `calcSpectrum`, and `neutrinoFlux`. The user can change the global `fracCDM2` 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 6.2 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 [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` $\in$ [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)`.

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 are normalized such that the total cross section for either χ or $\overline{\chi}$ cross sections is[2]

$$\sigma_{tot} = \frac{4M_\chi^2 M_N^2}{\pi(M_\chi + M_N)^2}(|A^{SI}|^2 + 3|A^{SD}|^2) \tag{5}$$

`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.

• `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.

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 normalized such that

$$\int_0^\infty v f(v) dv = 1$$

The units are km/s for v and s^2/km^2 for f(v).

◇ `A` - atomic number of nucleus;

◇ `Z` - number of protons in the nucleus, predefined values for a wide set of isotopes are called with $Z\_\{Name\}$;

◇ `J` - nucleus spin, predefined values for a wide set of isotopes are called with $J\_\{Name\}\{atomic\_number\}$.

◇ `Sxx` - is a routine which calculates nucleus form factors for spin-dependent interactions (`S00,S01,S11`), it depends on the momentum transfer in fm^{-1}. The available form factors are

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

[2]All parameters are in GeV.

The last character is used to distinguish different implementations of the form factor for the same isotope, see details in [?].

The form factors for the spin independent (SI) cross section are defined by a Fermi distribution and depend on the global parameters `Fermi_a`, `Fermi_b`, `Fermi_c`.

The returned value 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. The distribution over recoil energy is stored in the array $dNdE$ which by default has $Nstep = 200$ elements. The value in the i^{th} element corresponds to

$$dNdE[i] = \frac{dN}{dE}|_{E=i*keV*step}$$

in units of (1/keV/kg/day). By default *step* is set to 1.

For a complex WIMP, `nucleusRecoil` averages over χ and $\overline{\chi}$. For example for ^{73}Ge, a call to this routine will be:

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

• `setRecoilEnergyGrid(step,Nstep)`
changes the values of `step` and `Nstep` for the computation of `dNdE`.
• `Maxwell(v)`
returns

$$f(\mathrm{v}) = \frac{c_{\mathrm{norm}}}{\mathrm{v}} \int\limits_{|\vec{v}|<v_{max}} d^3\vec{v} \exp\left(-\frac{(\vec{v}-V_{Earth})^2}{\Delta v^2}\right) \delta(\mathrm{v}-|\vec{v}|)$$

which corresponds to the isothermal model. Default values for the global parameters $\Delta v =$ Vrot, $v_{max} =$ Vesc, `Vearth` are listed in Table 1. c_{norm} is the normalization factor. This function is an argument of the `nucleusRecoil` function described above.
• `nucleusRecoil0(f,A,Z,J,Sp,Sn,dNdE)`
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` are included for the nuclei listed in `nucleusrecoil` as well as ^3He, ^{133}Cs. Their names are

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

One can use this routine for nuclei whose form factors are not known.

7.3 Auxiliary routines

Two auxiliary routines are provided to work with the energy spectrum computed with `nucleusRecoil` and `nucleusRecoil0`.
• `cutRecoilResult(dNdE,E1,E2)`
calculates the number of events in an energy interval defined by the values `E1,E2` in keV.
• `displayRecoilPlot(dNdE,title,E1,E2)`
plots the generated energy distribution dNdE. Here `title` is a character string specifying the title of the plot and `E1,E2` are minimal and maximal values for the displayed energy in keV.

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=250** elements. 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 $\texttt{GeV}^{-1}$ units while a particle flux is expressed in $(\texttt{sec cm}^2\ \texttt{sr GeV})^{-1}$.

To display the spectra as a function of energy one can use

• `displaySpectrum(message,Emin,Emax,spectTab)`
where `message` is a text string which gives a title to the plot and `Emin` and `Emax` define energy cuts.

• `displaySpectra(title, Emin, Emax, N, nu1,lab1,...)`
displays several spectra. Here `title` contains some text, `Emin,Emax` are the lower and upper limits, and `N` is the number of spectra to display. Each spectrum is defined with two arguments, `nu1` designates the spectrum array and `lab1` contains some text to label the spectrum.

Even though the user does not need to know the structure of the spectrum array, we describe it below. 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 ($1 \leq i < NZ-1$) of the spectrum array contains the value of $E_i \frac{dN}{dE_i}$ where $E_i = E_{max} e^{Zi(i)}$, `Zi(i)`$=-7\ln 10 \left(\frac{i-1}{NZ}\right)^{1.5}$.

That is the array covers the energy interval $E_{max} \geq E > 10^{-7} E_{max}$.

• `addSpectrum(Spect,toAdd)`
sums the spectra `toAdd` and `Spect` and writes the result in `Spect`. For example, this routine can be useful for summing spectra with different maximal energy.

• `spectrMult(Spec, func)`
allows to multiply the spectrum `Spec` by any energy dependent function `func`

• `spectrInt(Emin,Emax,Spec)`
integrates a spectrum/flux, `Spec` from `Emin` to `Emax`.

• `spectrInfo(Emin,Spec,&Etot)`
provides information on the spectra. The returned value and `Etot` corresponds respectively to

$$\begin{aligned} N_{tot} &= \int_{E_{min}}^{E_{max}} SpectdNdE(E, Spec) dE = spectrInt(E_{min}, E_{max}, Spec) \\ E_{tot} &= \int_{E_{min}}^{E_{max}} E\ SpectdNdE(E, Spec) dE \end{aligned}$$

where the first element of the table `Spec` contains the value of E_{max}.

8.2 Annihilation spectra

• `calcSpectrum(key,Sg,Se,Sp,Sne,Snm,Snl,&err)`
calculates the spectra of DM annihilation at rest and returns σv in cm^3/s . 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 contibutions 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` interpolates and sums spectra obtained by Pythia. The spectra tables are provided only for Mcdm> 2GeV. The results for a dark matter mass below 2 GeV will therefore be wrong, for example an antiproton spectrum with kinematically forbidden energies will be produced. A warning is issued for Mcdm< 2GeV.
• `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`. In the Fortran version one uses instead the function
`vSigmaCh(i,weight,pdg,process)`
which is similar to the Fortran `vSigmaTCh` described in Section 6.2.

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

$$\rho(r) = \rho_\odot F_{halo}(r) \tag{6}$$

In `micrOMEGAs` $\rho_\odot$ is a global parameter `rhoDM` and the Zhao profile [?]

$$F_{halo}(r) = \left(\frac{R_\odot}{r}\right)^\gamma \left(\frac{r_c^\alpha + R_\odot^\alpha}{r_c^\alpha + r^\alpha}\right)^{\frac{\beta-\gamma}{\alpha}} \tag{7}$$

with $\alpha = 1, \beta = 3, \gamma = 1, rc = 20[kpc]$ is used by default. $R_\odot$, the distance from the Sun to the galactic center, is also a global parameter, `Rsun`. The parameters of the Zhao profile can be reset by
• `setProfileZhao(`α,β,γ,rc`)`
The function to set another density profile is
• `setHaloProfile(`$F_{halo}(r)$`)`
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 Einasto profile

$$F_{halo}(r) = exp\left[-\frac{2}{\alpha}\left(\left(\frac{r}{R_\odot}\right)^\alpha - 1\right)\right]$$

where by default $\alpha = 0.17$, but can be changed by
• `setProfileEinasto(`α`)`
The command `setHaloProfile(hProfileZhao)` sets back the Zhao profile. Note that both `setProfileZhao` and `setProfileEinasto` call `setHaloProfile` to define the corresponding 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

$$<\rho^2>= \rho^2 + f_{cl}\rho_{cl}\rho. \tag{8}$$

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

$$<\rho^2>(r) = \rho(r)(\rho(r) + \rho^{eff}_{clump}(r)), \tag{9}$$

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 $[(l,b)-(l+dl,b+db)]$.
• `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=`σv`SpectdNdE(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=`σv`SpectdNdE(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 Neutrino signal from the Sun and the Earth

This module does not work yet in case of 2DM

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 recently 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` [3] corresponds to the PPPC4DMν spectra while `WIMPSIM=1` corresponds to the `WimpSim` spectra and `WIMPSIM=-1` to the `DM`ν 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 [?]

$$\frac{dE}{dx} = -(\alpha + \beta E)\rho \tag{10}$$

For propagation in ice (the switch `forRocks=0`), micrOMEGAs substitutes $\rho = 1.0\mathrm{g/cm}^3$, $\alpha = 0.00262 GeV \mathrm{cm}^2/\mathrm{g}$, $\beta = 3.5 \times 10^{-6}\mathrm{cm}^2/\mathrm{g}$ [?], while for propagation in rocks, $\rho = 2.6\mathrm{g/cm}^3$, $\alpha = 0.002 GeV \mathrm{cm}^2/\mathrm{g}$, $\beta = 3.0 \times 10^{-6}\mathrm{cm}^2/\mathrm{g}$ [?]. 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 ($nu < 0$)

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

in `[1/Year/km^2]` units for a given cosine of the zenith angle, `cs`. This function is based on [?].

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.

9.1 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 $\phi < \phi_{cut} = 8°$. The returned parameter `B` is a Basyan factor which presents the ratio of likelihood functions for model with given fluxes and 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 given annihilation channel.

10 Cross sections and decays.

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

• `pWidth(particleName,&address)`
returns directly the particle width. If the `1->2` decay channels are kinematically accessible then only these channels are included in the width when `VWdecay,VZdecay`= 0. 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 processes with virtual $W(Z)$ which are closed kinematically and adds these to the `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. For particles other than gauge bosons, an improved routine with 3-body processes and a matching between the `1->2` and `1->3` calculations is kept for the future. The returned parameter `address` gives an address where information about the decay channels is stored. In C, the address should be of type `txtList`. For models which read a SLHA parameter file, the values of the widths and branchings are taken from the SLHA file unless the user chooses not to read this data, see (Section 14.5) for details.

• `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.

• `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 tries to calculate $1 \to 2$ decays. If such decays are kinematically forbidden then $1 \to 3$ decay channels are computed. If decay list contains virtual W/Z bosons and $delVirt \neq 0$, then vector bosons will be presenter via their decay probucts.

• `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. [4]

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 particle.

• `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`.

[4]The Fortran version of newProcess returns integer*8.

• `widh1CC(address, &err)`
calculate width of external process which *address* should be obtained by `newProcess` command. `widh1CC` is able to calculate widths for $1 \to 2$, $1 \to 3$ and $1 \to 3$ processes. Internal resonances in tested decay processe are not expected. In case of on sell subdecays retun value can be incorrect. If *address* code contains several subproceses, then only the first one is evaluated. $err \neq 0$ is a signal of error.
• `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^{th} 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 $T_{\rm min} \leq 0$ 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 $T_{\rm min} > 0$ defines the cut on the jet transversee momentum. The jet contents is defined by the parameter nf. If $Q_{\rm fact} \leq 0$, 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. In Fortran, one should pass a blank string instead of NULL.

By default hCollider uses the `cteq6l` structure function built-in the micrOMEGAs code. One can set any other parton distribution included in either micrOMEGAs or LHAPDF. The list of structure functions in micrOMEGAs can be obtained with the command
• `PDFList`
and one of these can be activated by
• `setPDF(name)`
To work with other PDF's available in `LHAPDF` one should first define the environment variable `LHAPDFPATH` which specifies the path to the LHAPDF library. Then micrOMEGAs

Makefile links it automatically. The list of available LHAPDF distributions can be obtained with the command
• `LHAPDFList`
and one of these can be activated by
• `setLHAPDF(nset,name)`
where `nset` specifies the subset number. Note, that if a wrong input is provided, `setLHAPDF` terminates the execution.

11 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 global parameter `Mcdm` except for `basicSpectra` and `basicNuSpectra` . These routines do not take into account the multi-component structure of DM and, in particular, possible differences between DM and anti-DM. To use these for multi-component `DM` the user has to perform a summation over the different DM components.
• `nucleusRecoilAux(f,A,Z,J,Sxx,csIp,csIn,csDp,csDn,dNdE)`
This function is similar to `nucleusRecoil`. The additional input parameters include `csIp(csIn)` the SI cross sections for WIMP scattering on protons(neutrons) and `csDp(csDn)` the SD cross sections on protons(neutrons). A negative value for one of these cross sections is interpreted as a destructive interference between the proton and neutron amplitudes. Note that the rate of recoil depends implicitly on the WIMP mass, the global parameter `Mcdm`. The numerical value for the global parameter has to be set before calling this function.
• `nucleusRecoil0Aux(f,A,Z,J,Sp,Sn,csIp,csIn,csDp,csDn,dNdE)` is the corresponding modification of `nucleusRecoil0`.

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 250, `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. As long as one keeps the default setting CDM1=CDM2=NULL these routines will use the Mcdm parameter to calculate the number density of DM particles.

• captureAux(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 9. 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 (1,0,01) which selects the neutrino spectra computed by WimpSim [?], PPPC4DMν [?] and DMν [?]. 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'.

12 Constraints from colliders

12.1 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, furthermore 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. For the former an interface to the public code HiggsSignals [?] or Lilith [?] is provided while exclusion limits obtained by the experimental collaborations can be applied using HiggsBounds [?].

The interface to HiggsBounds and HiggsSignals is realized via SLHA files. More specifically we generate the files HB.in and HS.in which contain the normalized couplings squared of the Higgs to all SM particles, including the normalized couplings squared to $\gamma\gamma, \gamma Z, gg$ as well as all Higgs partial widths and the top decay width. The loop-induced couplings are introduced into the different models using effective operators [?, ?]. These files are then fed to HiggsBounds and/or HiggsSignals and the output files HB.out and

HS.out are created. The information stored in these files can be extracted using the SLHAplus commands presented in Section 14.5 in particular the slhaVal function, see for example the main.c file in the MSSM directory.

The output of HiggsBounds consists of two numbers, HBresult and obsratio which indicate whether a point has been excluded at the 95%$C.L.$ by one of the experimental results considered.

HBresult	obsratio	
0	> 1	parameter point is excluded
1	< 1	parameter point is not excluded
-1	< 0	invalid parameter point

Information on the most constraining channel is also given as output.

The output of HiggsSignals contains the number of channels analyzed and the corresponding χ^2 value. We print these values on the screen as well as the corresponding p-value - the probability that χ^2 exceeds the obtained number. Detailed information about each channel is stored in the HS.out file.

The options for running HiggsSignals are set by fixing the Dataset, Method and PDF parameters. In the main.[c/F] files they are specified by *define* instructions. A theoretical uncertainty on the mass of the Higgs bosons can be specified in the SLHA BLOCK DMASS. The code which adds this block to HS.in for the 125 GeV Higgs is presented in main.[c/F], by default we set the uncertainty at 2 GeV.

The input file to Lilith is created automatically and contains the reduced couplings of the Higgs to all SM particles. The output is written in the Lilith_out.slha file and contains the value of $-2log(L)$ as well as the number of experimental data used for the fit.

By default we delete the HB/HS.in/out, Lilith_out.slha files in the end of the session. To keep them the user has to modify the CLEAN section in the end of main.[c/F].

12.2 Searches for New particles

12.2.1 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 searches for R-parity conserving SUSY by ATLAS and CMS. SModelS v1.0.4 decomposes any BSM model featuring a Z_2 symmetry into its SMS components using a generic procedure where each SMS is defined by the vertex structure and the SM final state particles; BSM particles are described only by their masses, production cross sections and branching ratios. 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. Within this assumption, SModelS can be used for any BSM model with a Z_2 symmetry as long as all heavier odd particles decay promptly to the dark matter candidate. Note that due to the Z_2 symmetry only pair production is considered, and missing transverse energy (MET) is always implied in the final state description.

SModelS needs three input files:

- an SLHA-type input file, containing the mass spectrum, decay tables[5] and produc-

[5]Note that all decay products in the decay table need to be on-shell.

tion cross sections for the parameter point under investigation;

- `particles.py` defining the particle content of the model, specifically which particles are even ("R-even") and which ones are odd ("R-odd") under the Z_2 symmetry;
- a file for setting the run parameters, `parameters.ini`.

The first two are located in the same directory as `main.c` and are automatically written by `micrOMEGAs`. A file containing the instructions to call `SModelS` can be found in `micromegas_4.3/include/SMODELS.inc` (or `SMODELS.inc_f`).

An SLHA-type output is written to `smodels.res`, (or an alternative name selected by the user). This output consists of three blocks,

- `SModelS_Settings` lists the `SModelS` code and database versions as well as input parameters for the decomposition.
- `SModelS_Exclusion` contains as the first line the status information if a point is excluded (1), not excluded (0), or not tested (-1). The latter can occur in scenarios with long-lived charged particles or in scenarios where no matching SMS results are found.

  If a point is excluded (status 1), this is followed by a list of all results with $R > 1$, sorted by their R values, R is defined as the ratio of the predicted theory cross section and the corresponding experimental upper limit. For each of these results, the SMS topology identifier (entry 0) (so-called Tx-name, see [**?**] for an explanation of the terminology), the R value (entry 1), a measure of condition violation (entry 2), and the analysis identifier (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.
- `SModelS_Missing_Topos` lists up to ten missing topologies sorted by their weights ($= \sigma \times \mathrm{BR}$). Each entry consists of the line number, the $\sqrt{s}$ in TeV, the weight and a description of the topology in the `SModelS` bracket notation. Note that this information is useful mainly for points that are not excluded.

In order to exploit decay channels involving a SM-like Higgs for which the experimental collaborations assume 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 by an entry of type

```
25 : "higgs",
-25 : "higgs"
```

in the `rEven` dictionary in the file `particles.py`. Note that the name `higgs` is reserved for a SM-like Higgs and should not be assigned generically. If no particle of that name is identified in `particles.py`, and the corresponding SMS results requiring a Higgs in the final state are not used by `SModelS` to constrain the parameter point.

12.2.2 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 1 if the mass of the Z' boson is excluded or 0 otherwise. Currently the latest Z' search in the dilepton final state at $\sqrt{s} = 13$ TeV from ATLAS [?] is implemented. The routine can be used for any Z' uniquely defined by the PDG code 32.

• `LspNlsp_LEP()`
checks the ocmpatibility with the upper limit [?] 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 returns $\sigma \times BR = \sum_i \sigma(e^+e^- \to \tilde{\chi}_1^0\tilde{\chi}_i^0) \times \mathrm{BR}(\tilde{\chi}_i^0 \to \tilde{\chi}_1^0 q\bar{q})$ in pb as well as a flag greater than one if $\sigma \times BR > 0.1(0.5)$ pb if $m_{\mathrm{NLSP}} > (<)100$ GeV [?]. 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.

13 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, UMSSM, IDM` (inert doublet model), `LHM`(little Higgs model), `RHNM` (a Right-handed Neutrino model), SM4 (toy model with 4th generation lepton), and `Z3M` (doublet and singlet model with Z_3 symmetry).

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.

13.1 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. 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* stands for one of the spectrum calculators `suspect`, `isajet`, `spheno`, or `softSusy`. The default spectrum calculator package is `SuSpect`. To work with another package one has to specify the appropriate path in `MSSM/lib/Makefile`. For this, the environment variables `ISAJET`, `SPHENO` or `SOFTSUSY` must be redefined accordingly. Note that we also provide a special interface for `ISAJET` to read a SLHA file. This means that the user must first compile the executable `isajet_slha` which sets up the SLHA interface in `ISAJET`. Specific instructions are provided in the `README` file.

For other MSSM scenarios, the parameters at the electroweak symmetry breaking scale are derived from an input at high scale. The same codes `suspect`, `isajet`, `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.
• `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.

We have added a routine for an interface with `superIso` [?]. This code is not included in micrOMEGAs so one has first to define the global environment variable *superIso* to specify the path to the package.
• `callSuperIsoSLHA()`
launches superIso and downloads the SLHA file which contains the results. The return value is zero when the program was executed successfully. Results for specific observables can be obtained by the command `slhaValFormat` described in section (14.5). Both `superIso` and `callSuperIsoSLHA` use a file interface to exchange data. The `delFiles` flag specifies whether to save or delete the intermediate files.
• `loopGamma(&vcs_gz,&vcs_gg)`
calculates σv for loop induced processes of neutralino annihilation into γZ and into $\gamma\gamma$. The result is given in $\frac{cm^3}{s}$. In case of a problem the function returns a non-zero value.

13.2 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_4.4.1` package [?].
• `nmssmEWSB(void)`
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(),deltaMs()`
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`.
• `loopGamma(&vcs_gz,&vcs_gg)`
calculates σv for loop induced processes of neutralino annihilation into γZ and into $\gamma\gamma$. The result is given in $\frac{cm^3}{s}$. In case of a problem the function returns a non-zero value.

13.3 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.

13.4 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 and Higgs observables as well as the anomalous magnetic moment of the muon were taken from `NMSSMTools_4.7.1` and adapted to the UMSSM [?]. To call these routines `UMSSMTools()` has to be given. 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)`. See the `README` of the UMSSM for further details.

14 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.

14.1 Main steps

- The command `./newProject` *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 both C and FORTRAN 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.

14.2 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 10 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 14.5), the automatic width option will use the widths contained in the SLHA file unless the user chooses the option to ignore this data in the SLHA file, see section 14.5.

14.3 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.

14.4 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 [?]

$$\begin{aligned} M^2_{eff}(Q) &= M(Q)^2\left[1+5.67a+(35.94-1.36n_f)a^2\right. \\ &+ \left.(164.14-n_f(25.77-0.259n_f))a^3\right] \end{aligned} \quad (11)$$

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)`

14.5 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/calchep/models/func2.mdl. Note that in this example the call to slhaRead is done within the function suspectSUGRAc.

14.5.1 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. In Fortran use `call setSysTimeLim(sysTimeLim,sysTimeQuant)` to reset the default time control parameters.

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

14.6 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}.

The function prototypes can be found in
`CalcHEP_src/c_source/SLHAplus/include/SLHAplus.h`

15 Mathematical tools.

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

```
sources/micromegas_aux.h
```

• `simpson(F, x1, x2, eps)`
numerical integration of the function $F(x)$ in the interval $[x1, x2]$ with relative precision *eps*. `simpson` uses an adaptive algorithm for integrand evaluation and increases the number of function calls in the regions where the integrand has peaks.
• `gauss(F,x1,x2,N)`
performs Gauss N-point integration for $N < 8$.
• `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 calculation. 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 routines with a given number of equations `Dim` and to zero for subsequent calls. The flag `first` is used for memory allocation. If `Yscale=NULL` the parameter `eps` defines the absolute precision of calculation ($\delta Y_i < eps$). Otherwise, the precision is defined by the condition $\delta Y_i < epsYscale_i$. 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\cdot 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`.
• `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 y_0 for given point x_0. Here `x` and `y` present grid of function arguments and function values $y_i = Y(x_i)$ Function `spline` fills 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 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 *Dim* array for each of these parameters. $X[i]$ contains the x-grid while $Y[i] = F(X[i])$.
• `bessI0(x), bessK0(x), bessK2(x)` the Bessel functions I_0, K_1, K_2.

• `K1pole(x)=`$K_1(\frac{1}{x})e^{\frac{1}{x}}\sqrt{\frac{2}{\pi x}}$ `;` `K2pole(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 $x = T/M_{cdm}$ they are represented by polynomials in x such that large exponents in ratios of Bessel functions are cancelled symbolically.
• `FeldmanCousins(n0, b,cl)`
is the Feldman-Cousins [?] function for Poisson distribution. Here `n0` is the observed number of events, `b` - expected background, `cl`<1 - the requested confidence level. Assuming that there is some number of signal events in addition to background, this function sets the upper limit on the number of signal events compatible with `n0` and `cl`.
• `ch2pval(Ch2exp,Ch2obs)`
returns the `p-value` assuming a χ^2 distribution with `Ch2exp` degrees of freedom (expected χ^2).

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

• `displayPlot(title,xMin, xMax, xName, Dim, N, ...)`
displays several curves/histograms on one plot. Here `title` contains some text, `xMin,xXax` are the lower and upper limits, and `N` is the number of curves/histograms to display. After the parameter `N` `displayPlot` expects 3N arguments, where each trine contains array of function values, array of function uncertainty, and textual label for function. Tabulated functions should correspond to a grid $x_i = xMin + (i+0.5)(xMax - xMin)/Dim$, where $i = 1, ..., Dim-1$. When plotting a function, the uncertainty should be set to `NULL`.
• `displayFunc(title, F,x1,x2, varName)`
displays a plot of function $F(x)$ in the $[x1, x2]$ interval. `title` is a text which appears as the title of the plot. `varName` is a name of x variable
• `displayFunc10(title, F,x1,x2, varName)`
displays F on $\log_{10}(x)$ axis.

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

$$B(\bar{B} \to X_s\gamma) = B(\bar{B} \to X_c e\bar{\nu}) \left| \frac{V_{ts}^* V_{tb}}{V_{cb}} \right|^2 \frac{6\alpha_{em}}{\pi f(z_0)} K_{NLO}(\delta) \tag{12}$$

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.

$B(\bar{B} \to X_c e\bar{\nu})$	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
$\lambda_2 \approx \frac{1}{4}(m_{B^*}^2 - m_B^2)$	$0.12\mathrm{GeV}^2$ [?]
$\alpha_s(M_Z)$	0.1189

Table 4: Default values in micrOMEGAs

We have also updated the default values for the experimentally determined quantities in Eq. 12, see Table **??**, and we have replaced the factor $f(z_0)$ by C_{sl} where

$$C_{sl} = \left|\frac{V_{ub}}{V_{cb}}\right|^2 \frac{\Gamma(\bar{B} \to X_c e\bar{\nu})}{\Gamma(\bar{B} \to X_u e\bar{\nu})} \tag{13}$$

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

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

$$ckmf = \left|\frac{V_{ts}^* V_{tb}}{V_{cb}}\right|^2 = 1 + \lambda^2(2\bar{\rho} - 1) + \lambda^4(\bar{\rho}^2 + \bar{\eta}^2 - A^2) \tag{14}$$

With these default values the NLO- improved SM contribution is $B(\bar{B} \to X_s\gamma)|_{\mathrm{SM}} = 3.27 \times 10^{-4}$ 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 superIso3.1 which includes the NNLO SM calculation for 10^5 randomly generated MSSM scenarios. The results are presented in Fig. **??** 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.

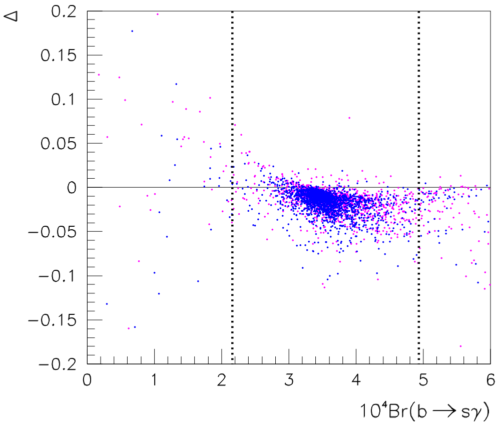

Figure 1: Relative difference for $B(\bar{B} \to s\gamma)$ between micromegas2.4 and superIso3.1. the vertical lines show the 3σ experimentally measured value.Numeri