Date: May 31, 2011

The micrOMEGAs user's manual, version 2.4.1

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 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.

1 Introduction

`micrOMEGAs` is a code to calculate the properties of cold dark matter 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. It is assumed that a discrete symmetry like R-parity ensures the stability of the lightest odd particle (LOP). 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 method used to compute the dark matter properties are described in references [1, 2, 3, 4]. 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 funtions. We always use `double(real*8)` variables for float point numbers and `int(INTEGER)` for integers. In this manual we use *FD* for a 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 output parameters 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 one uses any other type of variable whose length os sufficient to store a computer address (4 or 8 bytes), for example an *INTEGER*4* array with length 2 or a REAL*8 variable.

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.

`micrOMEGAs` assumes that all particles that are odd under the discrete symmetry have a name starting with '`~`', for example `~o1` for the lightest neutralino.

2 Global Parameters

The list of global parameters and their default values are given in Table 1. The use of global parameters is a new feature of version 2.4 of `micrOMEGAs`.

Table 1: Global parameters of micrOMEGAs

Name	default value	units	comments
Mcdm		GeV	Mass of Dark Matter particle
ScalarFFPd	0.03302		
ScalarFFPu	0.02348		Scalar form factor for quark in proton
ScalarFFPs	0.2594		
pVectorFFPd	-0.427		Axial-vector form factor for quark in proton
pVectorFFPu	0.842		
pVectorFFPs	-0.085		
SigmaFFPd	-0.23		Tensor form factor of d-quark in proton
SigmaFFPu	0.84		
SigmaFFPs	-0.046		
ScalarFFNd	0.04241		Scalar form factor of d-quark in neutron
ScalarFFNu	0.01823		
ScalarFFNs	0.2594		
pVectorFFNd	0.842		
pVectorFFNu	-0.427		Axial-vector form factor for quark in proton
pVectorFFNs	-0.085		
SigmaFFNd	0.84		
SigmaFFNu	-0.23		
SigmaFFNs	-0.046		
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 Sun to center of Galaxy
Rdisk	20	kpc	Radius of galactic diffusion disk
rhoDM	0.3	GeV/cm^3	Dark Matter density at Rsun
Vearth	225.2	km/s	Galaxy velocity of Earth
K_dif	0.0112	kpc^2/Myr	The normalized diffusion coefficient
L_dif	4	kpc	Vertical size of Galaxy diffisive halo
Delta_dif	0.7		Slope of diffision coefficient
Tau_dif	10^{16}	s	Electron energy loss time
Vc_dif	0	km/s	Convective Galactic vind

3 Setting of 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)`
assigns 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

```
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. For this one has to 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 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.
• `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` and `massPrt` should be `double*`. The value returned is the `PDG` code. If `pName` does not correspond to any particle of the model then `qNumbers` returns zero.

• `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)`

• `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 numerical value of the particle mass.

• `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 independent and 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.

4 Relic density calculation.

• `vSigma(T,Beps,fast)`
calculates the thermally averaged cross section for DM annihilation times velocity at a temperature T [GeV], see formula (2.6) in [1]. The value for σv is expressed in [pb]. 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].prtc[j]` *(j=0, 4)* define 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,particles)`serves the same purpose.

This function returns 0 if i exceeds the number of annihilation channels and 1 otherwise, $i \geq 1$. *real*8 weight* gives the relative contribution of each annihilation channel. *character*10 particles(4)* contains the names of particles in the annihilation process.

• `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. 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. [2]). 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, [2]. The recommended value is $Beps = 10^{-4} - 10^{-6}$, on the other hand if $Beps = 1$ only annihilation of the lightest odd particle is computed.

• `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 first evaluated 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 $(\Omega h^2)^{-1}$ cohere p1,...,p4 are particle names. To make summation over several channels one can substitute `"*"` instead of a particle name.

• `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,particles)`. These array and function are similar to `vSigmaTCh` described above. The array `omegaCh` if filled after calling either `darkOmegaFO` or `printChannels`.

5 Direct detection.

5.1 Amplitudes for elastic scattering

• `nucleonAmplitudes(qBOX,pAsi,pAsd,nAsi,nAsd)`
calculates the amplitudes for WIMP-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[1]

$$\sigma_{tot} = \frac{4M_\chi^2 M_N^2}{\pi(M_\chi + M_N)^2}(|A^{SI}|^2 + 3|A^{SD}|^2) \qquad (1)$$

[1] All parameters in GeV.

If `qBOX=NULL` (`qBOX=NoLoop` in Fortran) tree level amplitudes are computed. In MSSM-type models with a spin 1/2 WIMP and scalar "squarks", `qBOX=FeScLoop` uses an improved tree-level calculation. `nucleonAmplitudes` returns a value different from zero only when there is an internal problem in calculation.

`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.

• `getScalarFF`$(m_u/m_d, m_s/m_d, \sigma_{\pi N}, \sigma_0)$
computes the scalar coefficients for the quark contents in the nucleon from the mass ratios $m_u/m_d, m_s/m_d$ as well as from $\sigma_{\pi N}$ and σ_0, the amplitudes for πN scattering. $\sigma_{\pi N}$ and σ_0 should be specified in MeV. [4]

5.2 Scattering on nuclei

• `nucleusRecoil(f,A,Z,J,S00,S01,S11,qBOX,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 $J\_\{Name\}\{atomic\_number\}$.

- ◇ `S00, S01, S11` - nucleus form factors for spin-dependent interactions. They depend of 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
```

  where xx is either 00, 11 or 01. The last character is used to distinguish different implementations of the form factor for the same isotope, see details in [4].

- ◇ `qBOX` - a parameter needed by `nucleonAmplitudes`, see the description above.

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,S00Ge73,S01Ge73,S11Ge73,FeScLoop,dNdE);
```

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

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

wich corresponds to the isothermal model. Default values are $\Delta v = 220$km/s, $v_{max} = 700$km/s.V_{Earth} is a global parameter and c_{norm} the normalization factor. This function is an argument of the `nucleusRecoil` function described above.
• `SetfMaxwell(dv,vmax)`
sets parameters Δv and v_{max} for `Maxwell`.
• `nucleusRecoil0(f,A,Z,J,Sp,Sn,qBOX,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` [4]. 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.

5.3 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.

6 Indirect detection

6.1 Interpolation and display of spectra

Various spectra of particles relevant for indirect detection are stored in arrays with **NZ=250** elements. The i^{th} element of an array corresponds to dN/dz_i where $z_i = log(E_i/Mcdm)$. Here N stands for either a number of particles or a particle flux expressed in $(\sec \cdot \mathrm{cm}^2\mathrm{sr})^{-1}$.[2]

Table interpolation can be done by two functions

• `zInterp(z,spectTab)`

or

• `SpectdNdE(E,spectTab)`

interpolates the tabulated spectra and returns the particle distribution `dN/dE` where `E` is the energy in GeV assumimg that $z = 0$ corresponds to `E=Mcdm`.

To display the spectra visualization one can use

• `displaySpectrum(Spectrum,message,Emin,Emax,Units)`

displays the resulting spectrum, `message` is a text string which gives a title to the plot. `Emin` and `Emax` define energy cuts. If `Units=0` the spectrum is written as a function of $z = log_{10}(E/\mathrm{Mcdm})$ otherwise the spectrum is a function of the energy in GeV.

6.2 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`, `Sg`, `Sne`, `Snm`, `Snl`. To remove the calculation of a given spectra, substitute `NULL` for the corresponding argument. `key` is a switch to include polarisation of W,Z bosons (`key=1`) or photon radiation (`key=2`). Photon radiation is added to all subprocesses when computing the photon spectrum while only the 3-body process $\chi\chi \to e^+e^-\gamma$ is included for the positron spectrum. When `key=4` the cross sections for each annihilation channel 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 polarisation 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 kinetically forbidden energies will be produced. A warning is issued for Mcdm< 2GeV.

• `spectrInfo(Xmin,spectrTab,&Ntot,&Xtot)`

provides information on the spectra generated. Here `Xmin` defines the minimum cut for the energy fraction x=E/Mcdm, `Ntot` and `Xtot` are calculated parameters which give on average the total number and the energy fraction of the final particles produced per collision. Note that the upper limit is Xtot=2.

• `vSigmaCh` is an array that contains the relative contribution and particle names for each annihilation channel. In the Fortran version one uses instead the function

[2]The value of z_i can be obtained with the function $Zi(i)$. In the current version $Zi(i) = log\left((10^{-7})^{(i/NZ)^{1.5}}\right)$. This function is in general not needed if one uses the functions described below.

`omegaCh(i,weight,particles)`. These array and function are similar to `vSigmaTCh` described above. The array `vSigmaCh` if filled by `calcSpectrum`.

6.3 Distribution of Dark Matter in Galaxy.

To compute the signal from an indirect detection experiment one has to take into account the dark matter distribution in the Galaxy. Usually the DM density is represented via a *profile* function.

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

Dark Matter annihilation in the Galaxy depends on $< \rho^2 >$ which can be significantly larger than $< \rho >^2$ because of Dark Matter *clumps*. The effect of clumping can be described by another profile

$$< \rho^2 > (r) = \rho_{\odot}^2 F_{halo}^2(r) F_{clump}(r) \tag{2}$$

The DM density at the Sun, $\rho\odot$, is defined in micrOMEGAs by a global variable `rhoDM`.
• `setHaloProfiles(`F_{halo}, F_{clump}`)`
allows to change both the halo and the clump profile. Any sphericaly symmetric DM halo profile can be defined.
• `noClumps(r)`
is a non clumpy profile which is used by default. It returns 1 for any argument.
• `hProfileABG(r)`
is the default halo density profile which is described by the function

$$F_{halo}(r) = \left(\frac{R_{sun}}{r}\right)^{\gamma} \left(\frac{r_c^{\alpha} + R_{sun}^{\alpha}}{r_c^{\alpha} + r^{\alpha}}\right)^{\frac{\beta-\gamma}{\alpha}} \tag{3}$$

• `setProfileABG(alpha,beta,gamma,a)`
resets the parameters $\alpha = 1, \beta = 3, \gamma = 1, rc = 20[pc]$ of the default profile.
• `hProfileEinasto(r)`
is the Einasto halo density profile

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

• `setProfileEinasto(alpha)`
sets the parameter α for the Einasto profile, the default value is $\alpha = 0.17$.

6.4 Photon (neutrino) 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.

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

• `gammaFlux(fi,dfi,vcs)`
is the same function as `gammaFluxTab` above but corresponds to a discrete photon spectrum. `vcs` is the annihilation cross section, for instance in the MSSM it is calculated with the `loopGamma` function. The function returns the number of photons per cm^2 of detector surface per second. Note that for $\chi\chi \to \gamma\gamma$ the result should be multiplied by a factor 2 as each annihilation leads to the production of two photons.

6.5 Propagation of charged particles.

The spectrum of charged particles observed 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 interpolation, in this case the flux returned in (GeV s cm^2sr)$^{-1}$.

• `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.

7 Cross sections and decays.

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

• `pWidth(particleName,&address,&dim)`
returns directly the particle width. If the `1->2` decay channels are kinematically accessible then only these channels are included in the width. If not, `pWidth` compiles all open `1->3` channels and use these for computing the width. An improved routine with a better 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`. The parameter `dim` gives the number of final particles.

• `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,FD)`
Calculates 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 kinematicaly fobidden then $1 \to 3$ decay channels are computed. For models which read an 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 10.5) for details.
• `newProcess(procName, libName)`
compiles the codes for any $2 \to 2$ or $1 \to 2$ reaction. The result of the compilation is stored in the library

`work/so-generated/`*libName*`.so`.

If the library `libName` already exists, it is not recompiled and the correspondence between the contents of the library and the *procName* parameter is not checked. *libName* is also inserted into the names of routines in the *libName*.so library. Thus *libName* can not contain symbols that cannot be used in identifiers, for example the symbols $+, -, *, /,$ .1
The name of a given process, *procName*, has to be specified in the notation of CalcHEP, for example in the MSSM

`"e,E->~1+,~1-"`

stands for the lightest chargino pair production in e^+e^- collisions. Note that *procName* should not contain any blank space. Multi-process generation is also possible by using the symbol `2*x`. For example, `"e,E->2*x"` designates all possible two particle final states for an e^+e^- collision. Note that all library names starting with `omg` or `2width_` are reserved for internal calls of the `darkOmega` routines and cannot be used for new libraries. Although such libraries cannot be created by the user, the ones already compiled in `micrOMEGAs` can be loaded and used to calculate the corresponding matrix elements. In this case the *procName* argument should be left blank. These internal `micrOMEGAs` libraries are named `omg<particle>_<particle>.so` and `2width_<particle>.so`. Here `<particle>` starts with P/A for particle/antiparticle followed by a number whihc designates the particle position in the `CalcHEP`particle table (work/models/prtcls1.mdl) . 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[3]. 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.

• `wimpAnnLib()`
is the name of the library for DM annihilation.

There are two routines which allow to check the library contents.

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

• `cs22(address,nsub,P,c1,c2,&err)`
calculates the cross-section for a given $2 \to 2$ process, *nsub*, with center of mass momentum P(GeV). The differential cross-section is integrated from $c1 < \cos\theta < c2$ and 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,pName1,pName2)` 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. `pName1` and `pName2` are the names of outgoing particles. The value returned is the cross section in [pb]. The QCD scale is fixed to $Q \approx M/3$.

8 Tools for model independent analysis

To facilitate model independent comparisons with data, we provide additional routines to compute the nucleus recoil energy using as input the WIMP mass and the cross sections for SI and SD scattering on nucleons.

• `nucleusRecoilAux(f,A,Z,J,S00,S01,S11,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

[3]In Fortran the format is **call newProcess(procName, libName, address)**

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(pdgN, outN,Spectr)
computes the spectra of outgoing particles and writes the result in an array of dimension 250, Spectr. M is the WIMP mass, pdgN is the PDG code of the particles produced in the annihilation of a pair of WIMPs. To get 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 spectra produced by a polarised 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 output depends on Mcdm. 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.

9 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, LHM(little Higgs model) and RHNM (a Right-handed Neutrino model).

Each model contains a special routine for reading input parameters:
• readVarMSSM, readVarNMSSM, readVarCPVMSSM, readVarlHiggs, readVarRHNM.
These routines are similar to the general readVar routine described in Section 3 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 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 $\tilde{S}$. The sum of the squares of the LSP components should add up to 1.

9.1 MSSM

The MSSM has a long list of independent model parameters, those are specified in the SLHA [10, 6]. With this approach it is possible to work in the context of different supersymmetry breaking scenarios with a unique model implementation. To compute the spectrum one can choose one of the four spectrum calculators suspect, isajet, spheno, or softSusy. The routines that computes the input parameters of the MSSM are

• *spect*SUGRA(tb,MG1,MG2,MG3,Al,At,Ab,sgn,MHu,MHd,Ml1,Ml3,Mr1,Mr3,Mq1,Mq3,
Mu1,Mu3,Md1,Md3)
defines the independent parameters of the MSSM listed in the argument starting from a set of input parameters in the CMSSM or SUGRA model, $m_0, m_{1/2}, A_0, \tan\beta, sgn(\mu)$.
•*spect*AMSB(am0,m32,tb,sng).
does the same as above within the AMSB model.
• *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.

As mentioned the micrOMEGAs interface with spectrum calculators is based on the SLHA file interface. By default all intermediate files are deleted. To keep those files, the user must write in the main file the instruction

```
delFiles=0;
```

In Fortran programs the same result can be obtained by

```
call delFiles(0)
```

We also have an option to directly read a SLHA input file, this uses the function
• lesHinput(file_name}
which returns non-zero number in case of problem.

The routines for computing constraints are (see details in [2]).
• deltarho()
calculates the $\Delta\rho$ parameter which describes the MSSM corrections to electroweak observables. It contains 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()
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 $\tan\beta$.
• 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 NMSSM 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.
• 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 is not implemented and must be added by the user.

We have added a routine for an interface with `superIso` [24]. 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 (10.5). Both `superIso` and `callSuperIsoSLHA` use a file interface to exchange data. The `delFiles` flag specifies whether to save of 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 problem the function returns a non-zero value.

9.2 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` [15] in the `NMSSMTools_2.0` package [21].

• `nmssmEWSB(void)`

calculates the masses of Higgs and supersymmetric particles in the NMSSM starting from weak scale input parameters which can be downloaded by the `readVarNMSSM` routine. [16]

• `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 [3]).

• `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 [17]

• `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`.

9.3 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 potential are read directly from CPsuperH [18, 19], together with masses and mixing matrices of neutralinos, charginos and third generation sfermions. On the other hand, 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, [20]

• `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.

10 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.

10.1 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 [11].

- For odd particles and for the Higgs sector it is recommended to use automatic width calculation with 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 needed 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.

10.2 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 7 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 10.5), 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 10.5.

10.3 Using LanHEP for model file generation.

For complicated models 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 -evl 2 flag

  ```
  lhep  source_file -evl 2
  ```

  Such a flag provides the correct level of optimization for the model's Feynman rules.

- 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.

10.4 QCD functions

Here we describe some QCD functions which can be useful for model construction.
• `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 [14]. Thresholds for 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 [14]

$$\begin{aligned} M^2_{eff}(Q) &= M(Q)^2\left[1+5.67a+(35.94-1.36n_f)a^2\right. \\ &+ \left.(164.14-n_f(25.77-0.259n_f))a^3\right] \end{aligned} \quad (4)$$

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 $Q=$`2Mcdm`. In some special cases one needs a precise treatment of the masses of light quarks. The function
• `MqRun(M2GeV, Q)`
returns the running quark mass defined at the scale 2 GeV. The corresponding effective mass needed for higgs decay width is presented by
• `Mqeff(M2GeV, Q)`

10.5 SLHA reader

Very often calculation of 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 [10, 6]. The `micrOMEGAs` package contains routines for reading files in the SLHA format. Such routines can be very useful for new model implementation.

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 -   read DECAY /do not read   DECAY
  m4 = 0/1 -   read BLOCK/do 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 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 [23]. Unfortunately the stucture 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.

• `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.

The SUSY Les Houches Accord also describes the format of the information given on 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 listed. 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`.

10.5.1 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 default time control parameters.

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

10.6 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 [22]. 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)` routines 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` ($i \le j$), 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`

11 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 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 array Y contains starting variables at `x1` as an input and 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 [22].
• `buildInterpolation(F,x1,x2,eps,&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 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 $Y[i] = F(X[i])$.

• `polint4(x,Dim,X,Y)`
performs cubic interpolation for Dim-dimension arrays X,Y. A similar function, `polint3`, performs quadratic interpolation.

• `bessk0(x)`
The Bessel function of zero order of second kind.

• `displayFunc(F,x1,x2, title)`
displays a plot of function $F(x)$ in the $[x1, x2]$ interval. **title** is a text which appears as the title of the plot.

• `displayFunc10(F,x1,x2, title)`
displays $F(10^x)$.

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

The calculation of $b \to s\gamma$ was described in micromegas1.3 [2]. The branching fraction reads

$$B(\bar{B} \to X_s\gamma) = B(\bar{B} \to X_c e\bar{\nu}) \left|\frac{V_{ts}^* V_{tb}}{V_{cb}}\right|^2 \frac{6\alpha_{em}}{\pi f(z_0)} K_{NLO}(\delta) \tag{5}$$

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 [25] 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 [26, 27].

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

We have also updated the default values for the experimentally determined quantities in Eq. 5, see Table A, and we have replaced the factor $f(z_0)$ by C_{sl} where

$$C_{sl} = \left|\frac{V_{ub}}{V_{cb}}\right|^2 \frac{\Gamma(\bar{B} \to X_c e\bar{\nu})}{\Gamma(\bar{B} \to X_u e\bar{\nu})} \tag{6}$$

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 A are used to compute the central value of $ckmf$ with at order λ^4,

$$ckmf = \left|\frac{V_{ts}^* V_{tb}}{V_{cb}}\right|^2 = 1 + \lambda^2(2\bar{\rho} - 1) + \lambda^4(\bar{\rho}^2 + \bar{\eta}^2 - A^2) \tag{7}$$

$B(\bar{B} \to X_c e \bar{\nu})$	0.1064 [29]
C_{sl}	0.546 [27]
$\lvert V_{ts}^* V_{tb}/V_{cb}\rvert^2$	0.9613 [29]
A	0.808
λ	0.2253
$\bar{\rho}$	0.132
$\bar{\eta}$	0.341
m_b/m_s	50
$\lambda_2 \approx \frac{1}{4}(m_{B^*}^2 - m_B^2)$	0.12GeV^2 [28]
$\alpha_s(M_Z)$	0.1189

Table 2: Default values in micrOMEGAs

With these default values the NLO- improved SM contribution is $B(\bar{B} \to X_s\gamma)|_{\rm SM} = 3.27 \times 10^{-4}$ which corresponds to the result of Gambino and Giordano [27] 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. A 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.

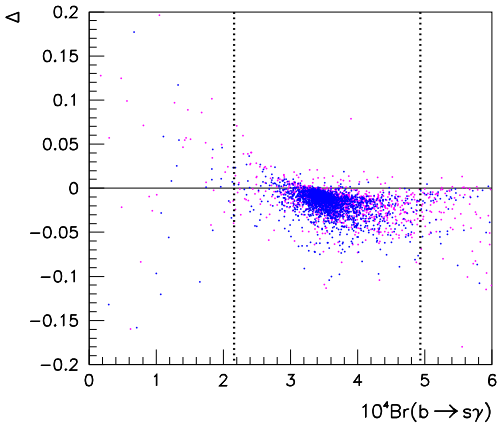

Figure 1: Relative difference for $B(\bar{B} \to s\gamma)$ between micromegas2.4 and superIso3.1. the vertical lines show the 3σ experimentally measured value.

References

[1] G. Belanger, F. Boudjema, A. Pukhov and A. Semenov, Comput. Phys. Commun. **149** (2002) 103 [arXiv:hep-ph/0112278].

[2] G. Belanger, F. Boudjema, A. Pukhov and A. Semenov, Comput. Phys. Commun. **174** (2006) 577 [arXiv:hep-ph/0405253].

[3] G. Belanger, F. Boudjema, A. Pukhov and A. Semenov, Comput. Phys. Commun. **176** (2007) 367 [arXiv:hep-ph/0607059].

[4] G. Belanger, F. Boudjema, A. Pukhov and A. Semenov, Comput. Phys. Commun. **180** (2009) 747 [arXiv:0803.2360 [hep-ph]].

[5] G. Belanger, F. Boudjema, P. Brun, A. Pukhov, S. Rosier-Lees, P. Salati and A. Semenov, arXiv:1004.1092 [hep-ph].

[6] B. Allanach *et al.*, Comput. Phys. Commun. **180** (2009) 8 [arXiv:0801.0045 [hep-ph]].

[7] C. Hugonie, G. Belanger and A. Pukhov, JCAP **0711** (2007) 009 [arXiv:0707.0628 [hep-ph]].

[8] G. Belanger, F. Boudjema, S. Kraml, A. Pukhov and A. Semenov, Phys. Rev. D **73** (2006) 115007 [arXiv:hep-ph/0604150].

[9] G. Belanger, F. Boudjema, C. Hugonie, A. Pukhov and A. Semenov, JCAP **0509** (2005) 001 [arXiv:hep-ph/0505142].

[10] P. Skands *et al.*, JHEP **0407** (2004) 036 [arXiv:hep-ph/0311123].

[11] A. Pukhov, arXiv:hep-ph/0412191.

[12] A. Semenov, Comput. Phys. Commun. **180** (2009) 431 [arXiv:0805.0555 [hep-ph]].

[13] G. Belanger, A. Pukhov and G. Servant, JCAP **0801** (2008) 009 [arXiv:0706.0526 [hep-ph]].

[14] S. Eidelman *et al.* [Particle Data Group], Phys. Lett. B **592** (2004) 1.

[15] U. Ellwanger and C. Hugonie, Comput. Phys. Commun. **177** (2007) 399 [arXiv:hep-ph/0612134].

[16] U. Ellwanger and C. Hugonie, Comput. Phys. Commun. **175** (2006) 290 [arXiv:hep-ph/0508022].

[17] F. Domingo and U. Ellwanger, JHEP **0712** (2007) 090 [arXiv:0710.3714 [hep-ph]].

[18] J. S. Lee, A. Pilaftsis, M. S. Carena, S. Y. Choi, M. Drees, J. R. Ellis and C. E. M. Wagner, Comput. Phys. Commun. **156** (2004) 283 [arXiv:hep-ph/0307377].

[19] J. S. Lee, M. Carena, J. Ellis, A. Pilaftsis and C. E. M. Wagner, Comput. Phys. Commun. **180** (2009) 312 [arXiv:0712.2360 [hep-ph]].

[20] J.S. Lee, A. Pilaftsis, M. Carena, S.Y. Choi, M. Drees, J. Ellis, C. Wagner, `http://www.hep.man.ac.uk/u/jslee/CPsuperH.html`.

[21] U. Ellwanger, J. Gunion, C. Hugonie, `http://www.th.u-psud.fr/NMHDECAY/nmssmtools.html`.

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, "Numerical Recipes: The Art of Scientific Computing", Cambridge University Press (2007).

[23] F. Mahmoudi *et al.*, arXiv:1008.0762 [hep-ph].

[24] A. Arbey and F. Mahmoudi, Comput. Phys. Commun. **182** (2011) 1582.

[25] M. Misiak, H. M. Asatrian, K. Bieri, M. Czakon, A. Czarnecki, T. Ewerth, A. Ferroglia, P. Gambino *et al.*, Phys. Rev. Lett. **98** (2007) 022002. [hep-ph/0609232].

[26] M. Misiak, M. Steinhauser, Nucl. Phys. **B764** (2007) 62-82. [hep-ph/0609241].

[27] P. Gambino, P. Giordano, Phys. Lett. **B669** (2008) 69-73. [arXiv:0805.0271 [hep-ph]].

[28] W. M. Yao *et al.* [Particle Data Group Collaboration], J. Phys. G **G33** (2006) 1-1232.

[29] K. Nakamura *et al.* [Particle Data Group Collaboration], J. Phys. G **G37** (2010) 075021.