SLAM, a Mathematica interface for SUSY spectrum generators

Peter Marquard[a], Nikolai Zerf[b]

[a] *Deutsches Elektronen Synchrotron DESY, Platanenallee 6, D15738 Zeuthen, Germany*
[b] *Department of Physics, University of Alberta, Edmonton AB T6G 2J1, Canada*

Abstract

We present and publish a Mathematica package, which can be used to automatically obtain any numerical MSSM input parameter from SUSY spectrum generators, which follow the SLHA standard, like SPheno, SOFTSUSY or Suspect. The package enables a very comfortable way of numerical evaluations within the MSSM using Mathematica. It implements easy to use predefined high scale and low scale scenarios like mSUGRA or $m_h^{\max}$ and if needed enables the user to directly specify the input required by the spectrum generators. In addition it supports an automatic saving and loading of SUSY spectra to and from a SQL data base, avoiding the rerun of a spectrum generator for a known spectrum.

Program summary

Title of program: `SLAM`

Available from:
`http://www.ttp.kit.edu/Progdata/ttp13/ttp13-024/`

Computer for which the program is designed and others on which it is operable: Any computer where `Mathematica` version 6 or higher is running providing `bash` and `sed`.

Operating system or monitor under which the program has been tested: Linux

No. of bytes in distributed program including test data etc.: 35 207 bytes.

Distribution format: source code

Keywords: SUSY, MSSM, SLHA, SQL, database, Mathematica, particle spectrum generation.

Nature of physical problem: Interfacing published spectrum generators for automated creation, saving and loading of SUSY particle spectra.

Method of solution: `SLAM` automatically writes/reads SLHA spectrum generator input/output and is able to save/load generated data in/from a data base.

Restrictions: No general restrictions, specific restrictions are given in the manuscript.

Typical running time: A single spectrum calculation takes much less than one second on a modern PC.

1. Introduction

Although there has been no experimental evidence for the realization of Supersymmetry (SUSY) yet, the Minimal Supersymmetric Standard Model (MSSM) became very popular among particle physicists. Many calculations of elementary particle processes include the effects of SUSY particles. This applies to cosmological predictions like relic density of dark matter from early universe, predictions of cross sections for the direct production of SUSY particles at colliders like the LHC and ILC, the calculation of radiative corrections due to the presence of SUSY particles for Standard Model (SM) processes, for example in flavour physics, and many more.

Compared to the SM a supersymmetric model is more predictive because supersymmetry imposes many relations between different parameters. However, due to the fact that we do not live in a supersymmetric world SUSY must be softly broken in a realistic model describing our world. Since the breaking mechanism is still unknown many new unknown parameters arise in the broken model.

Once one makes an assumption for those unknown parameters or assumes a certain breaking mechanism of SUSY, many parameters can be determined

directly from the knowledge of the measured SM parameters. For example, the Higgs mass is an independent parameter in the SM. In the MSSM this is no longer true, because after assuming a certain SUSY breaking model it can be calculated from the knowledge of a few parameters. On the one hand this is a nice feature, but on the other hand it leads to the problem of a consistent determination of all relevant parameters like masses, mixing angles, and couplings in such a model. A collection of such a set of parameters will be called spectrum in the following.

Fortunately this problem has already been solved and spectra can be calculated automatically using spectrum generators like `SPheno` [1], `SOFTSUSY` [2], or `Suspect` [3].

In order to manage the identification of particle parameters and couplings as well as option settings for programs in a common way, the Supersymmetry Les Houches Accord (SLHA) has been proposed [4, 5] and is used by many programs including the mentioned ones[1]. Within the SLHA the generation of a consistent spectrum can be specified by providing a single input file (`LesHouches.in`). Providing this file leads, after a run of the generator, to an output file (`LesHouches.out`) containing the complete spectrum in the SLHA notation.

If one is interested in only a few parameters of a single spectrum one can easily extract them via copy and paste by hand. However, this procedure is certainly not feasible if one needs to extract many parameters of more than one spectrum for further numerical evaluation. In fact this is the situation one is facing immediately when trying to create a plot in dependence of high or low scale scenario parameters like $\tan\beta$. That means in order to be able to use the output of spectrum generators or any other program using SLHA, one needs an interface providing an automatic extraction of the relevant parameters from the output file `LesHouches.out`.

There already exist public interfaces written for `C++` [7, 8], `FORTRAN` [9, 10], and `Python` [11].

Due to its vast amount of implemented functions and with increasing computer power and increasing computation speed, `Mathematica` became an attractive alternative for numerical evaluation of analytic expressions obtained in SUSY models. However, to the authors' knowledge there is up to now no public implementation of an interface to spectrum generators, automatically writing and reading SLHA files with the goal to provide simple data sets containing all needed parameters in `Mathematica` for general[2] purpose.

This task is accomplished by `SLAM` (**S**upersymmetry **L**es Houches **A**ccord with **Mathematica**) which will be presented in detail throughout this publication. A preliminary version of this package with name `LHSQLDB` was already used in Refs. [13] and [14].

This paper is organized as follows: For the impatient reader we present in Section 2 typical usage examples which should give a compact overview of what `SLAM` is capable to do. In Section 3 we give detailed information about how to install and configure the package properly before we give a full usage instruction

[1] A list of programs using SLHA can be found in Ref. [6].

[2] The `Mathematica` package `H3.m` [12] comes with such an interface built-in but it enables the automatic extraction of some special parameters needed in the package, only.

in Section 4. Section 5 is dedicated to the internal structure of SLAM and may help interested users to go beyond the "black box" model of this package.

2. Teaser Examples

To demonstrate, how the package should be used, we present some of the implemented scenarios in this section. The full list of predefined scenarios can be found in Tab. 2.

2.1. Predefined m_h^{max}-scenario

Once a spectrum generator is installed on the system and SLAM is configured properly it is very easy to obtain a SLHA spectrum within Mathematica. Install the package into the Mathematica kernel by loading the main file:

```
In[1]:= Get["~/Projects/SLAM/SLAM.m"];

suspect is ready.

spheno is ready.

softsusy is ready.

suseflav is ready.

SLAM v1.0.1 loaded.
```
(m1)

Afterwards, one can generate spectra, e.g. in the predefined $m_h^{\max}$-scenario [15] by using the central function ObtainLesHouchesSpectrum:

```
In[2]:= ObtainLesHouchesSpectrum[UseDataBase → True,
          Model → "mhmax",
          MA → 200., tanbeta → 20.,
          SpectrumGenerator → "spheno"]

Creating new SQL table: SPECTRATABLE

No data match in data base!
Running spectrum generator...

Saving spectrum in data base: /home/zerf/.SLAM/database/db

Out[2]= {alpha → -0.106281, SLHAGen → SPheno, tanbeta → 19.278}
```
(m2)

With the displayed command we have requested a $m_h^{\max}$-spectrum using the spectrum generator SPheno. Such a spectrum depends on M_A and $\tan\beta$. Because we did not switch off the usage of a currently empty data base, SLAM creates a new table in it. Since there is no matching spectrum in the data base the spectrum generator SPheno is run with the given parameters and the spectrum stored in the data base.

Finally it returns a part of the spectrum in a list of replacement rules. Since no parameters have been requested explicitly, SLAM just returned a list of selected parameters, which are defined in a default request list.

Assuming we are only interested in the mass of the lightest Higgs boson M_{h^0} within the same scenario we can use:

```
In[3]:= ObtainLesHouchesSpectrum[UseDataBase → True, Model → "mhmax",
         MA → 200., tanbeta → 20., SpectrumGenerator → "spheno",
         InputRequest → {"MASS" → {25 → {Mh0, Real}}}]

Matching data found in data base...

Out[3]= {Mh0 → 127.573}
```
(m3)

where we have used the SLHA identification for the on-shell Higgs mass, settled in the Block `"MASS"` with key entry 25. The choice of the symbol `Mh0` is completely arbitrary, up to the restriction that it has to be an undefined symbol. This allows the user to use his own notation in `Mathematica`. The `Real` statement tells `SLAM` that it should treat the parameter during the internal processing as real valued number. Note that `SLAM` did not rerun `SPheno` to obtain the Higgs mass, but just retrieved the value directly from the data base, where it has been stored during the previous call of the function `ObtainLesHouchesSpectrum`.

In order to see the default value of the option `InputRequest` one can use:

```
In[4]:= InputRequest /. Options[ObtainLesHouchesSpectrum] // InputForm

Out[4]//InputForm=
{"ALPHA" -> {None -> {alpha, Real}},
 "SPINFO" -> {1 -> {SLHAGen, String}},
 "HMIX" -> {2 -> {tanbeta, Real}}}
```
(m4)

With the given output it is straight forward to customize the `InputRequest` option values for the user's needs.

Beside the $m_h^{\max}$ scenario the no-mixing, gluophobic and small α_{eff} scenario, which were proposed together with the $m_h^{\max}$ scenario in Ref. [15], are already implemented and have the same input variables M_A and $\tan\beta$. They are used when setting the `Model` option to `"nomix"`, `"smallalpha"`, or `"gluophob"`.

2.2. Predefined mSUGRA-scenario

The mSUGRA-scenario exists as predefined model and can be called like follows:

```
In[5]:= ObtainLesHouchesSpectrum[UseDataBase → True, Model → "msugra",
         m0 → 100., m12 → 250., tanbeta → 10.,
         signmu → 1., A0 → -100., SpectrumGenerator → "spheno",
         InputRequest → {"MASS" → {25 → {Mh0, Real}}}]

No data match in data base!
Running spectrum generator...

Saving spectrum in data base: /home/zerf/.SLAM/database/db

Out[5]= {Mh0 → 111.109}
```
(m5)

The shown values for the five input parameters represent their default values. One should be aware that those values are already excluded experimentally and their allowed values are much higher.

2.3. Predefined p19MSSM plane I/II scenario

Since more and more space of the mSUGRA scenario has been excluded at the LHC, there has been a proposal of many scenarios in Ref. [16] taking into account exclusions by this non-observation of SUSY particles. If there is no

interest in a SUSY breaking mechanism motivated by some high scale physics, one can just define all the SUSY breaking parameters at a low scale. With certain assumptions one can reduce the number of these parameters to 19.

Two different planes have been defined in such a phenomenological 19 parameter MSSM [16]:

- p19MSSM plane I (`"p19MSSM1"`) in dependence of M_1(`M1`) and M_3(`M3`).
- p19MSSM plane II (`"p19MSSM2"`) depends on M_1(`M1`) and a common soft slepton mass breaking parameter $m_{\tilde{l}}$(`mslepton`) of the first two generations.

Both scenarios are implemented in `SLAM`. We just show an example for the first one:

```
In[6]:= ObtainLesHouchesSpectrum[UseDataBase → True,
          Model → "p19MSSM1",
          M1 → 1500, M3 → 1800,
          SpectrumGenerator → "spheno",
          InputRequest → {"MASS" → {25 → {Mh0, Real}}}]

No data match in data base!
Running spectrum generator...

Saving spectrum in data base: /home/zerf/.SLAM/database/db

Out[6]= {Mh0 → 124.669}
```
(m6)

To provide a reasonable light Higgs mass, we show the `SPheno` result for the scenario point p19MSSM1.13. The very first points in the sequence (p19MSSM1.1, p19MSSM1.2, ...) seem to be already excluded if one assumes that the new particle discovered at the LHC is the lightest Higgs boson.

2.4. User-defined scenarios

On the one hand predefined scenarios are quite useful, because the user does not need to worry about all the required input parameters, on the other hand they are very restrictive. To remove this disadvantage, `SLAM` can be used to generate spectra for any user-defined scenario one can think of.

This is done by handing all relevant input parameters to `SLAM` directly via the option `Model` of the function `ObtainLesHouchesSpectrum`. In the example below we generate the same mSUGRA spectrum like two sections ago but this time using direct model input for the underlying scenario:

```
MyModel = {"MODSEL" → {11 → {EV → 1},
      12 → {EV → 173.3},
      1 → {EV → 1}},
    "SMINPUTS" → {1 → {EV → 127.934},
      2 → {EV → 0.0000116637},
      3 → {EV → 0.1184},
      4 → {EV → 91.2},
      5 → {EV → 4.25},
      6 → {EV → 173.3},
      7 → {EV → 1.777}},
    "MINPAR" → {1 → {EV → 100.},
      2 → {EV → 250.},
      3 → {EV → 10.},
      4 → {EV → 1.},
      5 → {EV → -100.}}};
ObtainLesHouchesSpectrum[UseDataBase → True, Model → MyModel,
 SpectrumGenerator → "spheno",
 InputRequest → {"MASS" → {25 → {Mh0, Real}}}]

Matching data found in data base...

Out[8]= {Mh0 → 111.109}
```
(m7)

In the first line we defined our own model in variable `MyModel`. In this example, we use a minimal definition of the model, only giving the definition of needed entry values which are indicated by the `EV` symbols. In addition, it is possible to define comments for every entry, which can improve the readability of the SLHA files.

As can be seen from the output, the requested spectrum was already stored in the data base, which is the case since it corresponds to the predefined mSUGRA scenario used before.

We have to add the warning that when using user-defined scenarios, the user has to make sure that the input provided is sufficient for the spectrum generators to run correctly. `SLAM` will not check the provided input for its consistency.

3. Prerequisites & Installation

3.1. Prerequisites

To be able to use `SLAM` one needs:

- `Mathematica` of version 6.0 or higher.
- A UNIX operating system providing `bash` and `sed`.

And at least one of the following spectrum generators:

- `SPheno` (http://spheno.hepforge.org/),
- `SOFTSUSY` (http://softsusy.hepforge.org/),
- `Suspect` (http://www.lpta.univ-montp2.fr/users/kneur/Suspect/),

which can be obtained from the stated websites.

A running `SQL` server may be very convenient for quickly storing and loading spectrum data. But it is not mandatory, because `Mathematica` has a built-in data base server[3].

[3] which however slows down when dealing with a large number of saved spectra of order ~ 1000

3.2. Installation

Installing `SLAM` is quite simple:

- Download the package from [17].
- Unpack the tarball and put the two files:
  - `SLAM.m`
  - `SLAM.config.m`

  in a common folder.
- Open `SLAM.config.m` with any text editor and adjust the given entries appropriately.

To clarify the last step, we give detailed information about every setting contained in `SLAM.config.m`, although the contained comments should be sufficient to adjust the settings without manual.

The configuration file `SLAM.config.m` is – like the extension suggests – obeying `Mathematica` syntax and any change in this file must not break it. In the following we describe the contained settings.

Variable	Symbol Default Value	Description
`invalphaValue`	$\alpha^{-1,\overline{\text{MS}}}(m_Z)$ $1.279340000 \cdot 10^2$	inverse fine structure constant
`GFValue`	G_F $1.16637 \cdot 10^{-5}$	Fermi-constant
`asmzValue`	$\alpha_s^{(5),\overline{\text{MS}}}(m_Z)$ 0.1184	QCD fine structure constant
`MwValue`	m_W 80.399	W boson mass
`MzValue`	m_Z 91.200	Z boson mass
`MbMSbarValue`	$m_b^{\overline{\text{MS}}}(m_b)$ 4.25	b quark mass
`MtOSValue`	M_t 173.3	t quark pole mass
`MtauValue`	m_τ 1.777	τ lepton pole mass
`m0Value`	m_0 $1.0 \cdot 10^2$	common scalar mass ① ③
`m12Value`	$m_{1/2}$ $2.500 \cdot 10^2$	common gaugino mass ①
`signmuValue`	$\text{sgn}(\mu)$ 1.000	sign of bilinear H_1H_2 coupling ① -③
`A0Value`	A $-1.000 \cdot 10^2$	unified trilinear coupling ①
`lambdaValue`	Λ $40. \cdot 10^3$	scale of soft SUSY breaking ②
`mmessValue`	M_{mess} $80. \cdot 10^3$	overall messenger scale ②

Table 1: Default definition of numerical values used in `SLAM` in proper powers of GeV. The circled numbers show the ID of the corresponding model, where the specific default value is used. See Tab. 2 for more details.

Variable	Symbol Default Value	Description
`nmessValue`	N_5 3.	messenger index ②
`m32Value`	$m_{3/2}$ $60. \cdot 10^3$	gravitino mass ③
`MAValue`	m_A 200.	A^0 Higgs boson mass ④ - ⑭
`tanbetaValue`	$\tan\beta$ 10.	v_2/v_1 ① - ⑮
`muValue`	μ $1.5 \cdot 10^3$	bilinear H_1H_2 coupling ⑮
`M1Value`	M_1 300.	gaugino soft breaking mass ⑯ - ⑰
`M3Value`	M_3 360.	gaugino soft breaking mass ⑯
`msleptonValue`	$m_{\tilde{l}}$ 400.	common slepton breaking mass ⑰
`xBFS2013Value`	x 0.	slope parameter x ㉑
`yBFS2013Value`	y 0.	slope parameter y ㉑
`muRenValue`	$\mu_{\rm ren}$ 173.3	renormalization/output scale

Table 1: (continued)

The first set of options has to be adjusted to reflect the user's setup

- `ProgramPath["spheno"]="/.../SPheno-3.2.0/bin/SPheno"`
  `ProgramPath["softsusy"]="/.../softsusy-3.3.4/bin/softpoint.x"`
  `ProgramPath["suspect"]="/.../suspect2/suspect2"`
  define the path to the `SPheno`, `SOFTSUSY`, and `Suspect` executables on the file system as string.

- `SQLDataBaseTypeValue->"hsqldb"`
  sets the data base type to the default data base server included in `Mathematica`. To connect to a dedicated `SQL` server this settings might change to the following:
  `SQLDataBaseTypeValue->"MySQL(Connector/J)"`
  To be more specific this setting is used as argument when connecting to the data base with the command:
  `OpenSQLConnection[JDBC[SQLDataBaseTypeValue,...]...]`.
  One can use the `Mathematica` built-in connection tool
  `Needs["DatabaseLink`"];`
  `OpenSQLConnection[]`
  to find/set the proper definition of the needed values here and below. More details can be found in the documentation center of `Mathematica`.

- `SQLDataBaseOptionsValue->`
  `{"Name" -> "Spectrum", "Username" ->Environment["USER"]}`
  contains a a list of options relevant for the data base.
  When using a `SQL` server the list might look as follows:

`{"Username" -> Environment["USER"], "Password" -> "xxx", "Catalog" -> "..."}`
This list is used when connecting to the data base: `OpenSQLConnection[JDBC[...],SQLDataBaseOptionsValue]`

- `SQLDataBaseValue -> (Environment["HOME"]<>"/.SLAM/database/db")`
  gives the link to the data base as string. When using the built in `Mathematica` data base, this is the location on the system where `Mathematica` will store its data base files. In case a dedicated `SQL` server is used, this string contains its address including port information:
  `SQLDataBaseValue -> "sqlserver.someaddress.de:3306"`
  The option is used as second argument during the connection to the data base:
  `OpenSQLConnection[JDBC[...,SQLDataBaseValue],...]`

The next set of options can be left at their default values, but can be changed if needed

- `SQLVarCharLengthValue -> 50`
  gives the maximum length of strings saved in the data base in the `"VARCHAR"` format.

- `InputFilePathValue-> (Environment["HOME"]<>"/.SLAM/tmp/LesHouches.in")`
  contains the name and location of temporary SLHA input files as string.

- `OutputFilePathValue-> (Environment["HOME"]<>"/.SLAM/tmp/LesHouches.out")`
  contains the name and location of temporary SLHA output files as string.

- The default values of the parameters are defined. All parameters and their default values are listed together with a brief description in Tab. 1.

- After the default numerical values are set, one is able to configure the default input from `SLAM` for further processing. For example:
  `LesHouchesInputRequest = {"MASS"->{25 ->{Mh0, Real}}};`
  will just return the lightest Higgs boson mass by default. A more general example for the list on the right side of the definition can be taken from Mbx. (m4).

- `UseMinimalFlavourViolation=True;`
  Sets the used set of declarations done in the bottom of the configuration file to the minimal flavour violation one. If one works within minimal flavour violating models one should keep this setting. All the predefined scenarios work within this frame work. If one works beyond minimal flavour violation, one may set the variable to `False`.

The following settings should only be changed by experienced users, who intend to extend `SLAM`.

- `LesHouchesInputSQLFormats= { ... }`
  Contains declarations of input parameters, which are required by spectrum generators. These declarations are used to define the data table layout used in the `SQL` data base. Possible data types are `String`, `Integer` and `Real`.

  A change of the declarations requires a reset of the used data base. `SLAM` requires a proper set of declarations in order to be able to use a particular data base table.

- `LesHouchesOutputSQLFormats= { ... }`
  Contains declarations of output parameters, which are generated by spectrum generators. These declarations are used to define the data table layout used in the `SQL` data base. Possible data types are `String`, `Integer` and `Real`.

  A change of the declarations requires a reset of the used data base. `SLAM` requires a proper set of declarations in order to be able to use a particular data base.

If one wants to load `SLAM` in a more convenient way with the command `Needs["SLAM‘"];`, the path to the folder where `SLAM.m` is located can be added to the list of folder names stored in the global `$Path` variable. This can be done for example in the `~/.Mathematica/Kernel/init.m` file by the following command:

```
$Path=Join[$Path,{"/some/path/SLAM"}];
```

Concerning the installation and configuration of a `SQL` server, we recommend to follow the instructions given in the particular server manual. However, we want to add some comments about how to speed up the `SQL` server in the following.

The access to the data base can be greatly improved by adding a key or an index to the most important columns of the table. For example in the case of the $m_h^{\max}$ scenario the most obvious choice would be to at least add a key to the columns containing the input parameters m_A and $\tan\beta$. For other scenarios a different choice might be more suitable[4]. Keys can be defined directly after the initial declaration of the layout of the data base or added at any later time at almost no cost. To define a key using `Mathematica` considering for example the $m_h^{\max}$ scenario one can use the command

```
SQLExecute[$LesHouchesSQLConnectionList[[1]], #] & /@
{"alter table SPECTRATABLE add key (I_EXTPAR_26);" (*MA*),
 "alter table SPECTRATABLE add key (I_MINPAR_3);" (*tanbeta*)};
```

which requires an active data base connection stored in `$LesHouchesSQL-ConnectionList`. An active data base connection can be obtained best by calling `ObtainLesHouchesSpectrum[]` with the option `KeepSQLConnection` set to `True` once, before executing the command above. Since the `Mathematica` function `SQLExecute[]` used in this example simply executes the given `SQL` command, one can easily read off the necessary `SQL` commands enclosed as strings.

[4]See Tab. 2 for examples of free scenario parameters.

If one is interested in the inverse problem, i.e. looking up spectra which e.g. have a Higgs boson in a certain mass range, it is helpful to define a key for the respective column. For a definition of column names in the data base table see Section 5.3 on page 28.

4. Usage Instructions

In this section we try to give detailed instructions about how to use the `SLAM` package. Mainly this comes down to the knowledge how to use the function `ObtainLesHouchesSpectrum`. But also the built-in function `ReadLesHouches-SpectrumFile` might be of general interest because it can automatically import any SLHA file into `Mathematica`. The same holds for the function `Write-LesHouchesFile`, which automatically writes SLHA files according to the input provided in `Mathematica`.

4.1. Using `ObtainLesHouchesSpectrum`

The function `ObtainLesHouchesSpectrum` is completely controlled via options. One might even call the function without giving an explicit option:

```
In[9]:= ObtainLesHouchesSpectrum[]

Matching data found in data base...

Out[9]= {alpha → -0.111941, SLHAGen → SPheno, tanbeta → 9.89694}
```
(m8)

This calls `ObtainLesHouchesSpectrum` with all options set to their default values. A list of all options can be obtained using:

```
In[10]:= First /@ Options[ObtainLesHouchesSpectrum]

Out[10]= {A0, asmz, ClearDataBase, ExtendDataBase, GF, InputFilePath, InputRequest, invalpha, KeepSQLConnection, lambda,
  LoadCompleteSpectra, m0, M1, m12, M3, m32, MA, MaxSQLConnectionAttempts, mb, mmess, Model, mslepton, Mtau, MtOS, mu, muRen, Mw, Mz,
  nmess, OutputFilePath, RefreshDataBase, RemoveTemporaryFiles, signmu, Silent, SpectrumGenerator, SQLDataBase, SQLDataBaseOptions,
  SQLDataBaseType, SQLVarCharLength, TableDeclarationList, TableName, tanbeta, UseDataBase, WorkingFolder, xBFS2013, yBFS2013}
```
(m9)

One can get further information about a single option by the question mark operation as follows:

```
In[11]:= ? asmz

  αs (MZ). Input value for spectrum generators.
```
(m10)

The default option value can be checked with:

```
In[12]:= asmz /. Options[ObtainLesHouchesSpectrum]

Out[12]= 0.1184
```
(m11)

and set via:

Model	Option value	ID/Ref.	Free Parameters
mSUGRA	`"msugra"`	①	`m0`, `m12`, `A0`, `signmu`, `tanbeta`
mGMSB	`"mgmsb"`	②	`lambda`, `mmess`, `tanbeta`, `signmu`, `nmess`
mAMSB	`"mamsb"`	③	`m0`, `m32`, `tanbeta`, `signmu`
$m_h^{\max}$	`"mhmax"`	④ [15]	`MA`, `tanbeta`
no-mixing	`"nomix"`	⑤ [15]	`MA`, `tanbeta`
gluophobic	`"gluophob"`	⑥ [15]	`MA`, `tanbeta`
small α_{eff}	`"smallalpha"`	⑦ [15]	`MA`, `tanbeta`
updated $m_h^{\max}$	`"mhmaxup"`	⑧ [18]	`MA`, `tanbeta`
$m_h^{\text{mod}+}$	`"mhmod+"`	⑨ [18]	`MA`, `tanbeta`
$m_h^{\text{mod}-}$	`"mhmod-"`	⑩ [18]	`MA`, `tanbeta`
light stop	`"lightstop"`	⑪ [18]	`MA`, `tanbeta`
light stau	`"lightstau"`	⑫ [18]	`MA`, `tanbeta`
light stau (Δ_τ)	`"lightstaudeltatau"`	⑬ [18]	`MA`, `tanbeta`
τ-phobic	`"tauphobic"`	⑭ [18]	`MA`, `tanbeta`
low-m_h	`"lowmh"`	⑮ [18]	`mu`, `tanbeta`
p19MSSM I	`"p19MSSM1"`	⑯ [16]	`M1`, `M3`
p19MSSM II	`"p19MSSM2"`	⑰ [16]	`M1`, `mslepton`
374345	`"374345"`	⑱ [19]	
401479	`"401479"`	⑲ [19]	
1046838	`"1046838"`	⑳ [19]	
2342344	`"2342344"`	㉑ [19]	`xBFS2013`, `yBFS2013`
2387564	`"2387564"`	㉒ [19]	
2750334	`"2750334"`	㉓ [19]	

Table 2: List of all predefined models available in `SLAM`. Use the strings listed in the Option value column as option value for the `Model` in order to select the corresponding model. All free parameters of each model are shown in the last column.

```
In[13]:= SetOptions[ObtainLesHouchesSpectrum, asmz → 0.11835];   (m12)
```

In principle one can just go through all options using the given commands for each option in order to get to know all of them.

Since this may be tedious, we just pick out classes of options and discuss them step by step.

The first class of options are part of predefined scenarios. In detail they are just the free parameters left in the corresponding scenario and listed in Tab. 2. Removing these options from the list in Mbx. (m9) the number of unknown options reduces by 16.

The next class of options is built up by the SM parameters which need to be provided to the spectrum generators. They are listed in Tab. **??**

Option	Default Value
`InputFilePath`	see `InputFilePathValue` p10
`InputRequest`	see Mbx. (m4) on p5
`Model`	`"msugra"`
`OutputFilePath`	see `OutputFilePathValue` p10
`RemoveTemporaryFiles`	`True`
`Silent`	`False`
`SpectrumGenerator`	`"spheno"`
`WorkingFolder`	`Automatic` → folder of `InputFilePath`

Table 4: List of options used to control the core operation of `SLAM`.

Variable	Symbol Default Value	Description
`invalpha`	$1/\alpha^{\overline{MS}}(m_Z)$ $1.279340000 \cdot 10^2$	inverse fine structure constant
`GF`	G_F $1.16637 \cdot 10^{-5}$	Fermi-constant
`asmz`	$\alpha_s^{(5),\overline{MS}}(m_Z)$ 0.1184	QCD fine structure constant
`Mw`	m_W 80.399	W boson mass
`Mz`	m_Z 91.200	Z boson mass
`mb`	$m_b^{\overline{MS}}(m_b)$ 4.25	b quark mass
`MtOS`	M_t 173.3	t quark pole mass
`Mtau`	m_τ 1.777	τ lepton pole mass
`muRen`	$\mu_{\rm ren}$ 173.3	renormalization/output scale

Table 3: (continued)

This further reduces the amount of unknown options by 9.

The next class is built up by options used to steer the processing. A list of these options including their default value are given in Tab. 4. In the following we give more detailed information on each of these options:

- `InputFilePath/OutputFilePath`
  The path including the name of SLHA input/output file as string.

  If more than one `SLAM` sessions are running at the same time using a shared file system, one should include the process ID in those paths, in order to avoid different processes writing and reading from the same file.

- `InputRequest`
  Defines the data, which should be returned:

  - `All`
    returns everything that is available in an internal SLHA format, built

up from lists and replacement rules. For more details about the used format see Section 5.2.

In fact the output still depends on the source of the data:

If SLAM got the data from a SLHA-file written by a generator, the entry values are still kept as strings, even though they are integers or real numbers, because SLAM does not know, which format transformation should be applied.

If SLAM loaded the data from the data base, all entry values will have their proper format.

- AllFormated
  Returns only blocks of entry values, which were declared in SLAM.config.m. Moreover SLAM tries to convert all data to the proper format contained in the declaration.
- Direct input
  Can be used to handle direct input, which selects certain data to be returned as list of replacement rules, where the symbol of the left hand side of each rule can be freely chosen. Examples for such selections can be found in Section 2.1 in Mbx. (m3). The default value for this option is printed in Mbx. (m4), where only entry values (EV) are selected. It is also possible to select block comments (BC), entry value comments (EC) and block information (BI) like follows:

```
{"MASS" ->{BC       -> {MASSComment, String},
           {25, EC} -> {Mh0Comment, String}},
 "MSOFT"->{BI       -> {muRen, Real}}};
```

  Note that if data is requested, that was not declared in the data base table, SLAM will print a warning and abort the process, if the data base is in use. If this is not the case, the data will be returned as $MissingData in case the data is missing.

- WorkingFolder
  determines the place where temporary files for and from spectrum generators are created. If one keeps the default option value, given by Automatic, SLAM will extract the directory from the InputFilePath option and use it as working folder.

  When using the default setting, this option has to contain a process ID as well in order to work correctly when running SLAM in parallel on a computer cluster.

- Model
  selects either between different predefined scenarios indicated by a special string or allows direct input, specifying the scenario via an internal SLHA format.
  - "..."
    For predefined scenarios all possible choices of allowed strings can be found in Tab. 2.
  - Direct input
    A minimal example for the direct, or user-defined model input, is

given in Section 2.4. For a better understanding of the used format the user may be referred to Section 5.2. The format enables the user to add block comments or block information, or even entry comments which will be written to the SLHA input file and saved in the data base. However, it is very important that all values should be given in their proper format. That means, if one wants to provide an integer it needs to have the `Head Integer`. If one wants to provide a floating point number, the `Head` of the number should return `Real`. Even if the number happens to be an integer one should append a dot to make its `Head` a `Real`. This is because `SLAM` has to convert the given input to proper `FORTRAN` readable strings. In case one already has a proper SLHA input file, it is possible to automatically generate the direct input using the `ReadLesHouchesSpectrumFile` function (see Section 4.4 on Page 22 for more details), in order to avoid time-consuming retyping.

- `RemoveTemporaryFiles`
  can be `True` or `False`. When set to `True` `SLAM` deletes all temporary generated files automatically after reading them. When set to `False` they are kept.

  When set to `False` one can easily get wrong results because `SLAM` may just load an old file in the case where no new file was generated by a spectrum generator. Thus this setting should only be used when one wants to have a look into the files.

- `Silent`
  - `"True"`
    Will stop `SLAM` to print processing messages.
  - `"False"`
    Processing messages get printed.

- `SpectrumGenerator`
  Specifies the spectrum generator that should be used. Currently the following generators can be used, when installed and configured correctly:
  - `"spheno"`,
  - `"softsusy"`,
  - `"suspect"`.

With the given options one can fully control the call of spectra from generators. However, the options steering the communication with a data base are still missing. They constitute the last class of options which will be introduced now. In Tab. 5 all options can be found in alphabetic order, including their default value and are discussed in this order:

- `ClearDataBase`
  - `False`
    Results in no additional action.

Option	Default Value
ClearDataBase	False
ExtendDataBase	True
KeepSQLConnection	True
LoadCompleteSpectra	False
MaxSQLConnectionAttempts	10
RefreshDataBase	False
SQLDataBase	see SQLDataBaseValue p10
SQLDataBaseOptions	see SQLDataBaseOptionsValue p10
SQLDataBaseType	see SQLDataBaseTypeValue p9
SQLVarCharLength	see SQLVarCharLengthValue p10
TableDeclarationList	see section 5.3
TableName	"SPECTRATABLE"
UseDataBase	True

Table 5: List of options used to control the data base operations of SLAM.

  - Only
    Makes SLAM to only clear the current data base table and nothing else.
  - True
    Does the same as Only but after that normal processing is carried out.

Clearing a data base table leads to a complete loss of all stored data. Use this option carefully!

- ExtendDataBase
  - False
    New spectra are not saved in the data base.
  - True
    New spectra are saved in the data base.
- KeepSQLConnection
  - False
    Makes SLAM disconnect from the SQL server after every transaction.
  - True
    Makes SLAM to hold the connection to the SQL server throughout a Mathematica session.

  This setting may be useful when one has more than one Mathematica kernel running SLAM at the same time (parallel) but using only one SQL server. Depending on the SQL server settings, only a maximum amount of clients may be allowed to connect to the server. That means holding the connection without need may potentially block another client from getting its data. However, establishing and closing a connection after every interaction may increase the network traffic to an unacceptable amount.
- LoadCompleteSpectra

  - `True`
    Makes `SLAM` to load all data values contained in one spectrum and stored inside the data base table, although the user might only have requested some of the data values. However, only the requested data is returned after a filtering.
  - `False`
    Makes `SLAM` to load only the requested data values from the data base. This requires some internal transformation of the `InputRequest` data to a SQL readable form, which might slow down the process for large requested sets of data. However, with this setting the network traffic is reduced in any case and for small number of requested data values this option leads to much faster loading times.

  One should keep the default value (`False`) in case of a small number of data value in a request. For requests close to a full data set the option `True` might speed up the calculation because there is no processing time going into the conversion from `InputRequest` data to a SQL readable form.

- `MaxSQLConnectionAttempts`
  Can be any positive integer and sets the total amount of connection attempts to an SQL server, until `SLAM` stops the operation throwing an error message.

- `RefreshDataBase`
  - `False`
    Makes `SLAM` to load any matching spectra from the data base.
  - `True`
    Makes `SLAM` to overwrite (refresh) matching spectra in the data base with the data of a newly generated spectrum.

- `SQLDataBase`
  Contains the address of the used data base as string. See `SQLDataBaseValue` on Page 10 for examples.

- `SQLDataBaseOptions`
  Contains a list of options which specify mainly the user account on the used data base. See `SQLDataBaseOptionsValue` on Page 10 for examples.

- `SQLDataBaseType`
  Contains a string defining the kind of data base in use. See `SQLDataBaseTypeValue` on Page 9 for examples.

- `SQLVarCharLength`
  Gives the maximum number of characters contained in string saved to the data base as `VARCHAR`. This option is only in use when a new data base is being created.

- `TableDeclarationList`
  Gives a list of declarations for the data base. The default value is generated automatically from the settings given in the file `SLAM.config.m` when the `SLAM` package gets loaded. One should not use any option other than default, except for testing reasons. See Section 5.3 for more details.

- `TableName`
  Holds a string which gives the name of the table inside the data base where all the spectra data is stored. Note that one might use different table names for different scenarios for convenience, especially for later manual analysis which may be independent of `SLAM`. However, `SLAM` can distinguish different scenarios on its own, so saving spectra obtained within different scenarios does not lead to wrong results.

- `UseDataBase`

  - `False`
    Forbids `SLAM` to use the data base. Thus any spectrum request has to be served by a spectrum generator call.

  - `True`
    Allows `SLAM` to use the data base. This means any matching spectrum found in the data base will be used, instead of calling a spectrum generator.

We have now described all relevant options, we need to state some important usage restriction:

Although the SLHA allows for the printout of scale dependent parameters at different scales Q in a single SLHA file, `SLAM` is not able to treat multiple appearances of a block at different scales. That means one should use the spectrum generators in user-defined scenarios in such a way, that each block is appearing only once per SLHA file.

If one needs parameters at different scales, the spectra have to be called separately for each scale.

Since we have now all the basics at hand, we can show some more involved applications of our package.

4.2. Advanced Example

A nice example for the comfortable usage is the calculation of the light Higgs boson mass in a scenario depending on two parameters, like the $m_h^{\max}$ scenario. How easy it is to get the relevant data shows the following code:

```
In[14]:= SetOptions[ObtainLesHouchesSpectrum,
           Model → "mhmax",
           Silent → True,
           SpectrumGenerator → "softsusy",
           InputRequest → {"MASS" → {25 → {Mh0, Real}}}
          ];
         AbsoluteTiming[
          DataList = Flatten[Table[
              {MAx, tanbetay,
               Mh0 /. ObtainLesHouchesSpectrum[
                  MA → MAx, tanbeta → tanbetay]},
              {MAx, 100, 200, 10},
              {tanbetay, 4, 20, 2}], 1];]

Out[15]= {26.414883, Null}                                    (m13)
```

With the first command the default options of `ObtainLesHouchesSpectrum` get changed. We select the $m_h^{\max}$-scenario, make the calculation silent, which

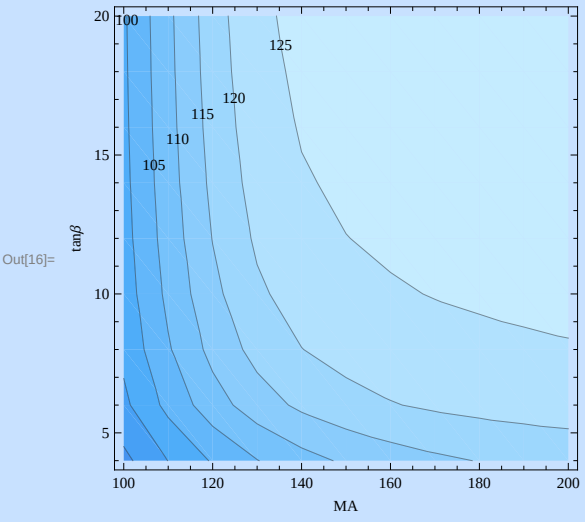
means printouts are disabled, chose `SOFTSUSY` as spectrum generator and request only the light Higgs boson mass to be in the output. Changing the default values has the advantage, that we do not need to specify the options in the call of `ObtainLesHouchesSpectrum` itself, which keeps the next command clear and short.

In the second command, we generate the Higgs mass data in dependence of the two free parameters M_A and $\tan\beta$. This is done with the help of a simple `Table` function, which automatically creates all possible combinations of M_A and $\tan\beta$ values in the stated limits. Note that for each combination `ObtainLesHouchesSpectrum` is called with the current value of the two parameters. Since this function returns just a replacement rule for `Mh0`, one has to use the called function like a replacement rule acting on the symbol `Mh0`. The next to outer-most `Flatten` function reduces the structure to a list of lists, where each of the lists contains three elements: the x-value M_A, the y-value $\tan\beta$ and the z-value M_{h^0}. The outer-most `AbsoluteTiming` function just returns the absolute calculation time in seconds of the command in its argument, which was about 26 seconds, using the `Mathematica` built-in data base. This runtime was consumed to run the spectrum generator and fill the data base with data from 99 spectra.

It is straight forward to display the calculated data, e.g. the following code does this job in a minimal way:

```
In[16]:= ListContourPlot[DataList, FrameLabel → {MA, tanβ},
          ColorFunction → "DeepSeaColors",
          ContourLabels → All]
```

Out[16]=



(m14)

The plot shows the Higgs mass in GeV in dependence of M_A and $\tan\beta$.

In this example we started with an empty data base and the spectrum generator had to run for every combination of input values. We can rerun the command with the already populated data base

```
In[17]:= AbsoluteTiming[
          DataList = Flatten[Table[{MAx, tanbetay,
               Mh0 /. ObtainLesHouchesSpectrum[
                  KeepSQLConnection → True,
                  Silent → True,
                  MA → MAx, tanbeta → tanbetay]},
              {MAx, 100, 200, 10},
              {tanbetay, 4, 20, 2}], 1];]

Out[17]= {2.556519, Null}
```
(m15)

and find that the processing time now is about ten times smaller.

4.3. Using `WriteLesHouchesFile`

The function `WriteLesHouchesFile` is normally called by `ObtainLesHouchesSpectrum`, but because it might be useful to be able to write SLHA files directly from `Mathematica` and independent of the spectrum generator usage, this function can be called independently. The function has only three options, which shall be discussed now:

- `InputFilePath`
  Holds a string which contains the path and the name of the file.
- `InputList`
  Contains the actual data that gets written to the file. An example for the formatting of the data can be displayed with:

  ```
  InputList/.Options[WriteLesHouchesFile]
  ```

  Further information about the format can be found in Section 5.2.
- `AppendToFile`
  - `False`
    `WriteLesHouchesFile` clears already existing content in the target file before writing output to it.
  - `True`
    `WriteLesHouchesFile` appends its output after already existing content in the target file.

With the given function one can write arbitrary data to a SLHA file, like the following example shows:

```
In[18]:= WriteLesHouchesFile[
          InputList → {"XMAS" → {
               BC → "happy event",
               BI → {EV → "at -20 C"},
               1 → {EV → 4, EC → "advents"},
               2 → {EV → 0, EC → "number of easter eggs"}},
             "NEWYEAR" → {
               BC → "New year party",
               BI -> {EV → "at -30 C"},
               1 → {EV → 31, EC → "date"},
               2 → {EV → 1.6, EC → "per mill. alk."},
               {1, 3} → {EV → 1.3}},
             "HOLIDAYS" → {
               None → {EV → 0, EC → "number of days"}}},
          InputFilePath -> "~/.SLAM/LesHouches.test.in"]
```
(m16)

Option	Default Value
AdoptEntryFormats	True
DataStructure	LesHouchesOutputSQLFormats†
Input	File
InputRequest	LesHouchesInputRequest†
RemoveTemporaryFiles	True
OutputFilePath	OutputFilePathValue†

Table 6: List of all options of the function ReadLesHouchesSpectrumFile. †defined in SLAM.config.m.

This example leads to the output file:

```
In[19]:= FilePrint["~/.SLAM/LesHouches.test.in"]

# SLHA input file generated by SLAM.m
Block XMAS at -20 C # happy event
 1   4   # advents
 2   0   # number of easter eggs
Block NEWYEAR at -30 C # New year party
 1   31   # date
 2   1.60000000E+00   # per mill. alk.
 1 3   1.30000000E+00   #
Block HOLIDAYS #
         0   # number of days
```
(m17)

This shows that the application of WriteLesHouchesFile is not restricted to the spectrum generator interface. Note that block information (BI), block comments (BC) and entry value comments (EC) are optional in the InputList. Note further that WriteLesHouchesFile automatically detects the format of the given entry value (EV). All other values and comments have to be strings.

4.4. Using ReadLesHouchesSpectrumFile

The function ReadLesHouchesSpectrumFile, like its name suggests, reads in SLHA files. It has several options which are listed in alphabetic order including their default value in Tab. 6. The options and their descriptions are:

- AdoptEntryFormats
  - False
    Will keep the string format of all entry values read in.
  - True
    Will convert the entry format according to either InputRequest or DataStructure, depending on the value of InputRequest.
- DataStructure
  Gives the default declaration of all entry values. Its default value is given by LesHouchesOutputSQLFormats in the SLAM.config.m file. Note, that this can be changed to what is required for the content of the SLHA file getting loaded.
- Input
  - File
    Tells ReadLesHouchesSpectrumFile to read in the file according to the path and file name provided in OutputFilePath.

  - Direct string input
    Will be read like stemming from a file.

- `InputRequest`
  - `All`
    returns everything without touching the data format, that means everything stays a string.
  - `AllFormated`
    Returns only blocks of entry values which are declared in the `DataStructure` option. Further, all values are converted to their declaration formats, if `AdoptEntryFormats` is set to `True`. Use this option value in order to generate direct input useable as `Model` option value of `ObtainLesHouchesSpectrum`.
  - Direct input
    Only the selected input values will be returned in a replacement rule, working like for `ObtainLesHouchesSpectrum` (See page 12).

- `RemoveTemporaryFiles`
  - `True`
    Tells `ReadLesHouchesSpectrumFile` to delete the file after reading it.
  - `False`
    Will keep the file unchanged after reading it.

- `OutputFilePath`
  Provides a string holding the path and the name of the file that should be read.

Because we already have written out an example SLHA file in the previous section, we can give an usage example of the function `ReadLesHouchesSpectrumFile` by just loading in, what we have written out:

```
In[20]:= SetOptions[ReadLesHouchesSpectrumFile,
           OutputFilePath → "~/.SLAM/LesHouches.test.in",
           RemoveTemporaryFiles → False];
         ReadLesHouchesSpectrumFile[InputRequest → All] // InputForm

Out[21]//InputForm=
{"XMAS" -> {BC -> "happy event ",
   BI -> {EV -> "at -20 C", EC -> None},
   1 -> {EV -> "4", EC -> "advents "},
   2 -> {EV -> "0", EC -> "number of easter eggs "}},
 "NEWYEAR" -> {BC -> "New year party ",
   BI -> {EV -> "at -30 C", EC -> None},
   1 -> {EV -> "31", EC -> "date "},
   2 -> {EV -> "1.60000000E+00", EC -> "per mill. alk. "},
   {1, 3} -> {EV -> "1.30000000E+00", EC -> None}},
 "HOLIDAYS" -> {BC -> None, BI -> {EV -> None, EC -> None},
   None -> {EV -> "0", EC -> "number of days "}}}                    (m18)
```

In the code above we set the default value of `RemoveTemporaryFiles` to `False` to be sure that we can load in the input file again. Further we specify the file which should be loaded. After that we call the `ReadLesHouchesSpectrumFile` function, requesting everything that can be found in the SLHA file by setting the option `InputRequest -> All`. We show the output in the `InputForm` to be able to distinguish strings from regular symbol names. The output

shows everything that is contained in the file, but the entry values (`EV`) are kept as strings.

In order to adopt their original format, we can load the file with the option `InputRequest -> AllFormated`. Because the default option of `DataStructure`, which is given by `LesHouchesOutputSQLFormats`, does not contain any suitable declaration information for our little toy example, we have to provide the proper declaration through the `DataStructure` option on our own. This is done in the following:

```
In[21]:= ReadLesHouchesSpectrumFile[InputRequest → AllFormated,
           DataStructure → {"XMAS" → {1 → Integer, 2 -> Integer},
             "NEWYEAR" → {1 → Integer, {1, 3} → Real}}] // InputForm

Out[21]//InputForm=
{"XMAS" -> {BC -> "happy event ",
   BI -> {EV -> "at -20 C", EC -> None},
   1 -> {EV -> 4, EC -> "advents "},
   2 -> {EV -> 0, EC -> "number of easter eggs "}},
 "NEWYEAR" -> {BC -> "New year party ",
   BI -> {EV -> "at -30 C", EC -> None},
   1 -> {EV -> 31, EC -> "date "},
   2 -> {EV -> "1.60000000E+00", EC -> "per mill. alk. "},
   {1, 3} -> {EV -> 1.3, EC -> None}}}                    (m19)
```

Compared to the output of Mbx. (m18), we see that all entry values which have been declared got converted from string to the declared format. Since we did not declare anything for the block `"HOLIDAYS"`, this block is completely missing.

In the last example we directly request just the number of advents in block `"XMAS"`:

```
In[23]:= ReadLesHouchesSpectrumFile[
           InputRequest → {"XMAS" → {1 → {Advents, Integer}}}] // InputForm

Out[23]//InputForm=
{Advents -> 4}                                            (m20)
```

This results in a list containing a replacement rule, where our choice of symbol for the number of advents has been taken over and the number was properly converted to an integer.

5. Package Internals

in this section we provide information, which enables the user to extend or adjust `SLAM` to very special needs. Before going into details, some words about the general layout of the package and the spirit in which it has been programmed are in order.

The package is purely written in `Mathematica`. Only very little `bash` code is used to manage the spectrum generators and the SLHA files. Throughout the package the code was written in a modular way, heavily relying on the built-in options system of `Mathematica`. A comprehensive introduction to the options interface can be found in the `Mathematica` help centre. It is easy to write argument insensitive code for the call of a sub function inside a normal function, if one uses the options interface. One does not need to worry about which argument has to be put at which position and it is no problem to add further parameters later, without changing the code for the function call itself.

With this feature one can conveniently write modular code, grouping different tasks into different, topic oriented functions. This helps to keep the code local and modular, which is very convenient while debugging. If something is not working properly, one can just check the data exchanged between modules. Finding the place where wrong data appears leads directly to the faulty module. Moreover, since every module serves a very special purpose only, it is simpler to ensure that each module fulfills its task properly than just writing a single code which has to achieve many goals properly at the same time. Thus following the old maxim "divide et impera"[5], one can write complicated code achieving multiple goals properly by just writing small modules doing their job properly and finally connecting all of them.

In the following subsection we are going to give a "map" of all relevant main modules which work inside the `ObtainLesHouchesSpectrum` function. We will walk on this map through the steps which are performed automatically by `Mathematica` in order to carry out commands given by the user. During this subsection the reader should get a basic orientation, needed for any modification of the code.

The second subsection clarifies the `Mathematica` internal representation of the SLHA used in this package.

In the third subsection we explain the used layout of the data base table in more detail.

The fourth subsection should enable the reader to implement new predefined scenarios in the existing code of `SLAM`.

5.1. Internal Structure of `ObtainLesHouchesSpectrum`

In Fig. 1 a simplified "blueprint" of the function `ObtainLesHouches-Spectrum` is shown. The function connects a spectrum generator on the left and a SQL data base on the right with the output of the function at the bottom right, in dependence of input provided by the user, shown at the top left. Inside the function `ObtainLesHouchesSpectrum`, which is indicated by the light gray background, sub functions are displayed in white rectangular boxes. They are connected by red lines indicating the order of sub-function calls. The yellow diamonds represent branchings of function calls in dependence of certain conditions. The rounded boxes stand for data prepared in a certain format indicated by the color of the background. In fact all names inside are actual function and data names used in the `Mathematica` code. Black arrows show the flow of data.

In the following the flowchart is described step by step. The starting point of this description is the place where the user input enters `ObtainLesHouches-Spectrum`:

1. In dependence of the option `ClearDataBase` the SQL data base table will be removed by the sub function `DropSQLLesHouchesTable` when `True` or `Only`.
2. In case of `Only` the evaluation ends after that. ⊠
3. When `True` or `False` was selected it generates the data `GeneratorInput-SQLDefinitions` with either the `ToSQLLesHouchesData` or the `Generate-InputSQLChecklist` function. The first one is used in case a direct input

[5] lat. for "divide and rule"

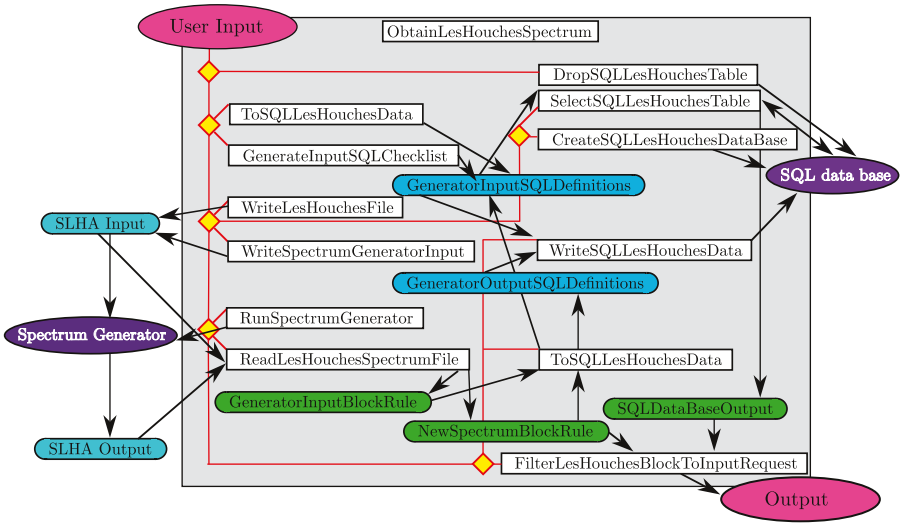


Figure 1: Internal structure of the function `ObtainLesHouchesSpectrum`. The named white rectangular boxes correspond to sub functions used by `ObtainLesHouchesSpectrum`. Arrows indicate the data flow. Rounded boxes stand for data in certain format, depending on their colour. The red lines follow the order of function calls. Yellow diamonds show possible branchings into different function calls in dependence of certain conditions.

model is used, the second function will do the job when a predefined model is considered.

4. When the option `UseDataBase` is `True`, the next step is to check if there is already any data consistent with the defined `GeneratorInputSQL-Definitions` in the data base. This is done by calling the sub function `SelectSQLLesHouchesTable`.
   (a) If there is any spectrum ready, it is stored in `SQLDataBaseOutput` which gets filtered by the sub function `FilterLesHouchesBlock-ToInputRequest` in order to provide the final output and end the evaluation. ⊠
   (b) In case no proper data base table could be found it is created by the sub function `CreateSQLLesHouchesDataBase`. Further steps coincide with the case below.
   (c) If no match was found, the spectrum needs to be obtained from the spectrum generator.
5. In order to obtain a spectrum from the generator, the proper SLHA input file has to be created. This is either done with the sub function `Write-LesHouchesFile`, when a direct input model is considered, or with the sub function `WriteSpectrumGeneratorInput`, when predefined model is in use.
6. Then the spectrum generator is run by the sub function `RunSpectrum-Generator`.
7. If there was no problem in the previous step, the sub function `Read-LesHouchesSpectrumFile` is used to read in the SLHA output file and store the data in the expression `NewSpectrumBlockRule`.
8. In case the options `ExtendDataBase` and `UseDataBase` were set to `True`:

8.1 `ReadLesHouchesSpectrumFile` is used to read in the SLHA input file and store the data in the expressions `GeneratorInputBlockRule`.

8.2 `ToSQLLesHouchesData` transforms the data in `GeneratorInputBlockRule` and `NewSpectrumBlockRule` to the data base writable expressions `GeneratorInputSQLDefinitions` and `GeneratorOutputSQLDefinitions`.

8.3 Then both sets of data are saved together in the `SQL` data base through the sub function `WriteSQLLesHouchesData`.

9. The final step is done by the sub function `FilterLesHouchesBlockToInputRequest` filtering the requested output from the data of `NewSpectrumBlockRule` in order to return it as output. ⊠

We did not discuss any option relevant for the sub functions. For example, the `SpectrumGenerator` option, telling `ObtainLesHouchesSpectrum` which spectrum generator should be used, does not appear in the code of the function at all. This is because `ObtainLesHouchesSpectrum` just trades these options to its sub function via a generic interface. Which turns out to keep the structure of the code very simple and generic.

5.2. Internal Representation for the SLHA

In the following subsection we explain the `Mathematica` internal format of SLHA data, which is in use for the expressions `GeneratorInputBlockRule`, `NewSpectrumBlockRule` and `SQLDataBaseOutput` (see Fig. 1). The easiest way of explaining a certain format is to give just a data example. A SLHA output like:

```
Block MINPAR  # Input parameters
 3   2.00000000E+01  # tanb at m_Z
 4   1.00000000E+00  # Sign(mu)
Block ALPHA  # Effective Higgs mixing angle
    -1.05379330E-01  # alpha
Block YU Q=  1.01490536E+03  # (SUSY scale)
 1 1  8.59529004E-06  # Y_u(Q)^DRbar
```

looks like the following in the `Mathematica` internal notation:

```
 ExampleSLHAData={
 "MINPAR" -> {
      BC -> " Input parameters",
      BI -> {EV -> None, EC -> None},
      3 -> {EV -> "2.00000000E+01", EC -> "tanb at m_Z"},
      4 -> {EV -> "1.00000000E+00", EC -> "Sign(mu)"}},
 "ALPHA" -> {
      BC -> " Effective Higgs mixing angle",
      BI -> {EV -> None, EC -> None},
      None -> {EV -> "-1.05379330E-01", EC -> "alpha"}},
 "YU" -> {
      BC -> " (SUSY scale)",
      BI -> {EV -> "Q=  1.01490536E+03 ", EC -> None},
      {1, 1} -> {EV -> "8.59529004E-06", EC -> "Y_u(Q)^DRbar"}}};
```

As one can see, the SLHA data is ordered with respect to blocks, where each block is just a replacement rule giving the block name in capital letters as string on the left hand side of the rule and the content of the block in a list on the right hand side.

Each of those lists contains a replacement rule for the block comment (`BC`). Further it contains a replacement rule for the block information (`BI`), which has a list of two replacements on its right hand side. The latter contains a replacement for its entry value (`EV`) and entry comment (`EC`), where in case of the block information the last one is always `None`.

In case the entry value is given by a matrix element depending on two entry values, instead of a single integer as entry key, a list of two integers is used, as can be seen in the last line.

In case where there is no entry key for a value, like for the mixing angle α, the variable `None` is used as entry key.

Note that in the example above all entry values are kept as strings. This only happens when SLHA files are loaded from a generator using the option `InputRequest->All`.

In case of `InputRequest->AllFormated SLAM` tries to return converted entry values (e.g. real numbers) using the declarations made in the file `SLAM.config.m`.

However, blocks which were not declared at all will not be displayed in the output.

Entry values are returned in their declared format when loading spectra from the data base.

5.3. Data Base Table Layout

Because one can use the automatically created and built up data base independently from `SLAM`, the layout of the table should be clarified. The default option value for `TableName` leads to the creation of a table with name `"SPECTRATABLE"`.

Any table is declared and initialized with the information provided in the variables `LesHouchesInputSQLFormats` and `LesHouchesOutputSQLFormats` defined in the file `SLAM.config.m`. These variables are read in and the contained information is processed once, when the `SLAM` Package gets loaded, to give the full declaration list of the table. The full list of declarations can be printed using the command:

```
(TableDeclarationList /. Options[ObtainLesHouchesSpectrum]) // TableForm
```

We do not give the full default output here because it is quite lengthy, but pick only a selection for demonstration:

```
Out[24]//TableForm=
I_SMINPUTS_COMMENT      VARCHAR
I_SMINPUTS_INFO         VARCHAR
I_MINPAR_COMMENT        VARCHAR
I_MINPAR_INFO           VARCHAR
I_EXTPAR_COMMENT        VARCHAR
I_EXTPAR_INFO           VARCHAR
O_SPHENOINFO_1          INTEGER
O_SPHENOINFO_1_COMMENT  VARCHAR
O_SPHENOINFO_2          INTEGER
O_SPHENOINFO_2_COMMENT  VARCHAR                                    (m21)
```

Note that all terms appearing in this output are in fact strings, because `TableForm` does not display the `"` character.

The output shows pairs of strings where the first string gives the name of the corresponding column and encodes mainly the place where the data is located in the SLHA file. The second string defines the type of data which will be stored in the corresponding column. So far there are only three different data types in use: `"INTEGER"`, `"DOUBLE"` and `"VARCHAR"`.

In the selection shown in Mbx. (m21) one can distinguish two different kinds of column names:

- Column names starting with a "`I`" reserve space for data stemming from SLHA input files.
- Column names starting with a "`O`" reserve space for data generated by the spectrum generators, so stemming from the SLHA output files.

Further the column names ending with `COMMENT` lead to columns, which do only save comments in the `"VARCHAR"` format. This is because one can optionally place comments behind the hash symbol (`#`) in a SLHA file in each line.

After the initial "`I`" or "`O`" comes a block name in capital letters separated by an underscore.

- For columns which do not save a string one of the following holds then:
  a) At least one further underscore separated integer value is required. This value corresponds to the key value defined in the SLHA. See line seven and nine in the displayed output above for example.
  b) If the value represents a matrix element there may be two underscore separated integers. Those integers correspond to two valued key entries defined in the SLHA.
  c) If the corresponding SLHA quantity has no key value, that means it comes alone in its own data block, like it is the case for the mixing angle α, the string `NONE` has to follow the block name after one underscore.
- Every data value entry can have its own comment. The name of the comment column is obtained by simply appending the string `_COMMENT` to the name of the corresponding data value column. See line ten and eight in the output above for example.
- Every block can have its own comment which is saved in the column name given by the block location appending the string `_COMMENT`. Line one, three and five of the displayed output above give a examples for this case.
- Every block can have an additional information entry which is mainly used to hold the scale information. For example a block definition in a SLHA file might look like the following:

```
BLOCK GAUGE Q=  1.01490536E+03  # (SUSY scale)
```

Everything after the space behind GAUGE and before the hash symbol (#) gets stored in the block information (BI). The information is automatically saved in the "VARCHAR" format, as can be seen e.g. from line two, four and six in the printout above where the name of the column is ending with the string _INFO. Note that requesting this data in the output as "Real" will force SLAM to return just the real number without the Q=.

With the given information it is easy to search directly in the data base for certain parameter limits. An example is to search for Higgs boson masses between 123 GeV and 129 GeV:

```
select I_EXTPAR_26 as MA,
       I_MINPAR_3 as tanbeta,
       O_MASS_25 as MH
       from SPECTRATABLE
       where (O_MASS_25 > 123 && O_MASS_25 < 129)
       order by MH;
```

5.4. Creating Additional Predefined Scenarios

Adding a new predefined scenario requires 6 steps but may help to speed up the generation of spectra, because less calculation steps have to be done for predefined scenarios in Mathematica.

1. If the new predefined scenario requires new option values because is depends on parameters which have not been used yet, one should add a new usage instruction before the Private section in the file SLAM.m for every symbol which will become an option. For example, the usage instruction for the nmess parameter used in the mGMSB-scenario looks like:

   ```
   nmess::usage="Number of minimal copies of the messenger
   sector in the gauge mediated symmetry breaking model.";
   ```

   This enables the user to get information for that option via a question mark operation like: ?nmess.
2. Add the option name and description of the new scenario to the usage instruction of the Model variable:

   ```
   Model::usage="..."
   ```

   For example the additional line should look like:

   ```
   \"mgmsb\" (gauge mediated symmetry breaking).
   ```

3. Add the definitions for the default option value symbols related to the new option value in the SLAM.config.m file. The names of the default option value symbols are just obtained from the option name itself by appending Value. See for example the nmessValue in Tab. 1. That way, the file SLAM.m will stay clean of any default numerical value definition and any user has a nice collection of the default numerical values in the config file.
4. Add the new options to the options of WriteSpectrumGeneratorInput. For our nmess example the corresponding line, which would have to be added, looks like:

   ```
   nmess -> Global‘nmessValue,
   ```

5. The most difficult part is to add the new scenario consistently to the subfuction `WriteSpectrumGeneratorInput`. There are multiple `Switch` instructions where new code for the new scenario has to be added. This can be done by just following the given examples in the code itself. Once having finished the modification of this function, one should check, after reloading the package, that the string written to the input SLHA file is correct when calling the new scenario. An easy way to do that is to call the modified function directly via:

   ```
   SLAM‘Private‘WriteSpectrumGeneratorInput[Model -> "mgmsb",
   InputFilePath -> String]
   ```

   where one of course has to replace `"mgmsb"` by the option value of the new scenario.
6. In the final step the new scenario has to be added to the body of the sub function `GenerateInputSQLChecklist`. Note that this works pretty much the same like in the previous step.
   In fact all numerical values entered directly *have* to be equal to those entered in the previous step! If this is not the case `SLAM` may not find anything in the data base although it already saved a requested spectrum. Please note that one should apply the function `StableNumber` to real numbers in order to make them stable under the conversion from and to their `FORTRAN` form. If a number is not stable under these conversions, very small rounding errors occur, which spoil the detection of matching spectra in the data base.
   Each numerical value follows an identification string in a list. This string can be built up from the position of the numerical value in a SLHA file as follows: Conventionally the first letter is a “`I`” which indicates that the value is entering the input value part of the `SQL` data base. Since white spaces are not allowed in `SQL` to be part of a column name, the separation to the following block name in capital letters is done by an underscore. After the block name the underscore separated key value follows as integer. If it is a double key value, like used for matrix elements, there may be two integer numbers separated by one underscore. If there is no entry key at all, like it is the case for the output value of the mixing angle α, which somehow got its own block, `NONE` replaces the usual integer number. One can check the adjustments by directly calling the function via:

   ```
   SLAM‘Private‘GenerateInputSQLChecklist[Model -> "mgmsb"]
   ```

   where `"mgmsb"` has to be replaced by the option value of the new scenario. The output should be just a list of pairs, where each pair is a list with an identification string as first element and a numerical value as second element.

Once the given steps have been completed, the new predefined scenario should work without any problems.

6. Summary

We presented and published the package `SLAM`, which provides a convenient interface for SLHA spectrum generators in `Mathematica`.

The package enables the user to obtain spectrum data from generators in a fully automatic way. Results of different spectrum generators can be compared without any effort and it allows the user to use his own notation in `Mathematica`. `SLAM` comes with a large number of built-in benchmark scenarios. Furthermore, it allows the user to freely define any desired scenario following the SLHA standard.

Moreover, it can store and recall all acquired data to and from a data base in order to avoid a recalculation of known spectra. Storing spectra in a data base allows the examination of parameter spaces by simply using powerful data base functionalities. A parallel use of `SLAM` is possible and helps to reduce the time needed for possible parameter scans and builds of data bases.

Besides the pure usage documentation including examples, we provided more details about the internal structure of the package which may help in case a modification of the program code is needed due to special user requirements.

Acknowledgements

We would like to thank M. Iskrzynski and A. Kurz for beta testing the application, and J. Hoff, A. Kurz and M. Steinhauser for reading of the manuscript. This work has been supported in part by the EU Network LHCPHENOnet PITN-GA-2010-264564.

[1] W. Porod, Comput. Phys. Commun. **153** (2003) 275 [hep-ph/0301101].

[2] B. C. Allanach, Comput. Phys. Commun. **143** (2002) 305 [arXiv:hep-ph/0104145].

[3] A. Djouadi, J. -L. Kneur and G. Moultaka, Comput. Phys. Commun. **176** (2007) 426 [hep-ph/0211331].

[4] P. Z. Skands, B. C. Allanach, H. Baer, C. Balazs, G. Belanger, F. Boudjema, A. Djouadi and R. Godbole *et al.*, JHEP **0407** (2004) 036 [hep-ph/0311123].

[5] B. C. Allanach, C. Balazs, G. Belanger, M. Bernhardt, F. Boudjema, D. Choudhury, K. Desch and U. Ellwanger *et al.*, Comput. Phys. Commun. **180** (2009) 8 [arXiv:0801.0045 [hep-ph]].

[6] http://home.fnal.gov/ skands/slha/

[7] http://fthomas.github.com/slhaea/

[8] http://www.svenkreiss.com/SLHAio

[9] T. Hahn, hep-ph/0408283.

[10] T. Hahn, Comput. Phys. Commun. **180** (2009) 1681 [hep-ph/0605049].

[11] http://www.insectnation.org/projects/pyslha

[12] P. Kant, R. V. Harlander, L. Mihaila and M. Steinhauser, JHEP **1008** (2010) 104 [arXiv:1005.5709 [hep-ph]].

[13] A. Pak, M. Steinhauser and N. Zerf, JHEP **1209** (2012) 118 [arXiv:1208.1588 [hep-ph]].

[14] A. Kurz, M. Steinhauser and N. Zerf, JHEP **1207** (2012) 138 [arXiv:1206.6675 [hep-ph]].

[15] M. S. Carena, S. Heinemeyer, C. E. M. Wagner and G. Weiglein, Eur. Phys. J. C **26** (2003) 601 [hep-ph/0202167].

[16] S. S. AbdusSalam, B. C. Allanach, H. K. Dreiner, J. Ellis, U. Ellwanger, J. Gunion, S. Heinemeyer and M. Kraemer *et al.*, Eur. Phys. J. C **71** (2011) 1835 [arXiv:1109.3859 [hep-ph]].

[17] http://www.ttp.kit.edu/Progdata/ttp13/ttp13-024/

[18] M. Carena, S. Heinemeyer, O. Stl, C. E. M. Wagner and G. Weiglein, arXiv:1302.7033 [hep-ph].

[19] M. W. Cahill-Rowley, J. L. Hewett, A. Ismail, M. E. Peskin and T. G. Rizzo,