Eur. Phys. J. C (2018) 78:98
https://doi.org/10.1140/epjc/s10052-017-5513-2

THE EUROPEAN PHYSICAL JOURNAL C

Addendum

GAMBIT: the global and modular beyond-the-standard-model inference tool

Addendum for GAMBIT 1.1: Mathematica backends, SUSYHD interface and updated likelihoods

The GAMBIT Collaboration: Peter Athron[1,2], Csaba Balazs[1,2], Torsten Bringmann[3], Andy Buckley[4], Marcin Chrząszcz[5,6], Jan Conrad[7,8], Jonathan M. Cornell[9], Lars A. Dal[3], Hugh Dickinson[10], Joakim Edsjö[7,8], Ben Farmer[7,8,a], Tomás E. Gonzalo[3], Paul Jackson[2,11], Abram Krislock[3], Anders Kvellestad[12,b], Johan Lundberg[7,8], James McKay[13], Farvah Mahmoudi[14,15,24], Gregory D. Martinez[16], Antje Putze[17], Are Raklev[3], Joachim Ripken[18], Christopher Rogan[19], Aldo Saavedra[2,20], Christopher Savage[12], Pat Scott[13,c], Seon-Hee Seo[21], Nicola Serra[5], Christoph Weniger[22,d], Martin White[2,11], Sebastian Wild[23]

[1] School of Physics and Astronomy, Monash University, Melbourne, VIC 3800, Australia
[2] Australian Research Council Centre of Excellence for Particle Physics at the Tera-scale, Australia, http://www.coepp.org.au/
[3] Department of Physics, University of Oslo, 0316 Oslo, Norway
[4] SUPA, School of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, UK
[5] Physik-Institut, Universität Zürich, Winterthurerstrasse 190, 8057 Zurich, Switzerland
[6] H. Niewodniczański Institute of Nuclear Physics, Polish Academy of Sciences, 31-342 Kraków, Poland
[7] Oskar Klein Centre for Cosmoparticle Physics, AlbaNova University Centre, 10691 Stockholm, Sweden
[8] Department of Physics, Stockholm University, 10691 Stockholm, Sweden
[9] Department of Physics, McGill University, 3600 rue University, Montreal, QC H3A 2T8, Canada
[10] Minnesota Institute for Astrophysics, University of Minnesota, Minneapolis, MN 55455, USA
[11] Department of Physics, University of Adelaide, Adelaide, SA 5005, Australia
[12] NORDITA, Roslagstullsbacken 23, 10691 Stockholm, Sweden
[13] Department of Physics, Blackett Laboratory, Imperial College London, Prince Consort Road, London SW7 2AZ, UK
[14] Univ Lyon, Univ Lyon 1, ENS de Lyon, CNRS, Centre de Recherche Astrophysique de Lyon UMR5574, 69230 Saint-Genis-Laval, France
[15] Theoretical Physics Department, CERN, 1211 Geneva 23, Switzerland
[16] Physics and Astronomy Department, University of California, Los Angeles, CA 90095, USA
[17] LAPTh, Université de Savoie, CNRS, 9 chemin de Bellevue B.P.110, 74941 Annecy-le-Vieux, France
[18] Max Planck Institute for Solar System Research, Justus-von-Liebig-Weg 3, 37077 Göttingen, Germany
[19] Department of Physics, Harvard University, Cambridge, MA 02138, USA
[20] Centre for Translational Data Science, Faculty of Engineering and Information Technologies, School of Physics, The University of Sydney, Sydney, NSW 2006, Australia
[21] Department of Physics and Astronomy, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Korea
[22] GRAPPA, Institute of Physics, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands
[23] DESY, Notkestraße 85, 22607 Hamburg, Germany
[24] Institut Universitaire de France, 103 boulevard Saint-Michel, 75005 Paris, France

Received: 3 November 2017 / Accepted: 28 December 2017 / Published online: 2 February 2018

Abstract In Ref. (GAMBIT Collaboration: Athron et. al., Eur. Phys. J. C. arXiv:1705.07908, 2017) we introduced the global-fitting framework GAMBIT. In this addendum, we describe a new minor version increment of this package. GAMBIT 1.1 includes full support for Mathematica backends, which we describe in some detail here. As an example, we backend SUSYHD (Vega and Villadoro, JHEP 07:159, 2015), which calculates the mass of the Higgs boson in the MSSM from effective field theory. We also describe updated likelihoods in PrecisionBit and DarkBit, and updated decay data included in DecayBit.

1 Using Mathematica backends in GAMBIT

For decades, Wolfram Mathematica[1] has been the symbolic computing framework of choice for many physicists. GAM-

[a] e-mail: benjamin.farmer@fysik.su.se
[b] e-mail: anders.kvellestad@nordita.org
[c] e-mail: p.scott@imperial.ac.uk
[d] e-mail: c.weniger@uva.nl

[1] http://www.wolfram.com/mathematica/.

BIT[2] users can now access both public and private Mathematica packages from within GAMBIT modules. This new feature provides a seamless interface to Mathematica, indistinguishable from the existing GAMBIT backend interfaces to codes written in C, C++ and Fortran.

Mathematica is proprietary software. It is the responsibility of the user to acquire a license in order to use the Mathematica framework. The number of Mathematica Kernel instances and subprocesses that can be launched per Mathematica license is limited, and can be found by evaluating the variables `$MaxLicenseProcesses` and `$MaxLicenseSubprocesses` (typically 8 and 16 respectively for a standard Mathematica license).

This addendum describes our implementation of the Mathematica interface. Section 1.1 describes the mechanism used to communicate with the Mathematica Kernel from GAMBIT. In Sects. 1.2–1.4 we explain the different aspects of the implementation, consisting of configuration (1.2), registration of backend functions and variables (1.3), and updates to the GAMBIT diagnostics to accommodate Mathematica backends (1.4). In Sect. 1.5, we give a realistic example of a Mathematica backend interfaced to GAMBIT, namely SUSYHD [2], which calculates the mass of the Higgs boson using methods from effective field theory (EFT).

1.1 The Wolfram symbolic transfer protocol

The Wolfram Symbolic Transfer Protocol (WSTP[3]), is the communication standard used by Mathematica to communicate with external programs. WSTP provides a two-way communication system, allowing the user to both call external codes from within Mathematica, and to call Mathematica routines from external programs written in many languages (C, C++, Java, etc). GAMBIT 1.1 uses the communication pathway between Mathematica and C++. This allows GAMBIT users to directly call Mathematica functions and variables from GAMBIT modules, in the same way as functions and variables are accessed from C, C++ and Fortran backends in GAMBIT 1.0.

The GAMBIT-Mathematica interface uses the WSTP messaging functions to launch and establish a link to a Mathematica Kernel session, and to send and receive information from that Kernel. Almost all the messaging functions that we employ use a WSTP link object, of type `WSLINK`, as a handle for communicating with the Kernel. One retrieves such a handle by opening a link to the Kernel with the messaging functions `WSOpenArgcArgv` or `WSOpenString`. The most important messaging functions for our purposes are

```
WSPutFunction(WSlink, function_name, n_args);
WSPutString(WSlink, string_name);
WSPutSymbol(WSlink, symbol_name);
WSPutInteger(WSlink, integer_number);
WSPutReal32(WSlink, float_number);
WSPutReal64(WSlink, double_precision_number);

WSGetString(WSlink, &string_variable);
WSGetInteger(WSlink, &integer_variable);
WSGetReal32(WSlink, &float_variable);
WSGetReal64(WSlink, &double_precision_variable);
```

where `WSlink` is an object of type `WSLINK`. The first argument in each of these must be a C++ object with the appropriate type, e.g. for symbols and functions this is a C++ string, and the argument *n_args* corresponds to the number of arguments of the Mathematica function *function_name*.

To interpret data received from the `WSGet` functions, one must also receive the packets sent by the Kernel. We use `WSNextPacket(WSlink)` to find the next packet head, `WSNewPacket(WSlink)` to skip the rest of the current packet, and `WSError(WSlink)` to check for errors during packet reception.

All these messaging functions live in the header `wstp.h`, native to the Mathematica installation, and will be used by GAMBIT for different purposes, as described below.

1.2 Preparing and configuring Mathematica backends

1.2.1 Configuration

At `cmake` time, GAMBIT searches for an installation of Mathematica on the user's system, locating the installation directory and defining several `cmake` variables, including the paths to the Mathematica header files and executables. The GAMBIT `cmake` system also searches for the `libuuid` library, required by WSTP. It writes a new header file containing a series of variables relating to the system's Mathematica configuration, including the preprocessor macro `HAVE_MATHEMATICA`, which is later used to enable or disable all code associated with Mathematica backends, according to whether or not the user has Mathematica installed. In order to manually switch off Mathematica support (from GAMBIT 1.1.2 onwards), one may run the `cmake` command as

```
cmake -Ditch=Mathematica ..
```

Mathematica backends are obtained and configured in a similar manner to other backends, by adding relevant entries to `cmake/backends.cmake` and `config/backend_locations.yaml.default`, and then running `make` *backend_name*. Mathematica backends require neither a configuration nor a build step, only a download link and an installation directory.

[2] http://gambit.hepforge.org.

[3] More information on WSTP can be found at http://reference.wolfram.com/language/.

1.2.2 Frontend header

As described in Sect. 4 of Ref. [1], GAMBIT loads backends at runtime as dynamic libraries, regardless of whether they are written in C, C++ or Fortran, using the POSIX-standard `dl` library. The macro `LOAD_LIBRARY`, used in the corresponding frontend header in GAMBIT, activates the backend library and loads its symbols.

GAMBIT communicates with Mathematica backends in a fundamentally different way to backends written in compiled languages. To redirect compilation flow in the GAMBIT backend system to use the preprocessor directives relevant for Mathematica backends, we define the new macro `BACKENDLANG`. This specifies the language of the backend in question, and can take the values `CC`, `CXX`, `FORTRAN` or `MATHEMATICA`. For example, in GAMBIT 1.1, the frontend header for version 1.2 of a C++ backend called *backend_name* would begin with

```
#define BACKENDNAME backend_name
#define BACKENDLANG CXX
#define VERSION 1.2
#define SAFE_VERSION 1_2
LOAD_LIBRARY
```

1.2.3 Backend types

In analogy with the definition of new types and typedefs given for Fortran backends in Sect. 4.4 of Ref. [1], here we provide a list of new types defined in GAMBIT 1.1 for use with Mathematica backends. These have clear correspondences with equivalent types in C and C++ types, and should be adopted whenever one is working with Mathematica backends. These are:

```
MVoid
MInteger
MReal
MBool
MChar
MString
MList<TYPE>
```

where `TYPE` can take any of the other defined types as its template argument. All these types are just convenient type redefinitions of native C++ types and thus can be used in exactly the same manner.

1.2.4 Link to the Kernel

Prior to loading the backend library, the connection with the Mathematica Kernel must be established. For this, the macro `LOAD_LIBRARY` initializes a WSTP environment with a call to `WSInitialize`, which returns a handle to the WSTP environment of type `WSENV`. After retrieving this handle, GAMBIT opens a new WSTP connection using the function `WSOpenString`, which launches the Mathematica Kernel and finishes the intialization phase of the connection. These functions are employed in the following way:

```
WSenv = WSInitialize();
WSlink = WSOpenString(WSenv, WSTPflags,
 &WSerrno);
```

The variable `WSerrno` captures any error that might have ocurred during the establishment of the link and `WSTPflags` is a string containing flags that point to the Kernel executable and give the order to initiate the link, e.g. `WSTPflags = "-linkname math -mathlink"` for Linux systems.

If the link is successfully established, the pointer `WSlink` is set to point to the new link between the main program and the Kernel. This handle is used in every subsequent communication with the Kernel.

1.2.5 Importing the package

After defining the relevant macros and backend types, loading the Mathematica environment and Kernel, the final task of the `LOAD_LIBRARY` macro is to import the backend package into the Mathematica Kernel. This is achieved by using the pointer to the WSTP link `WSlink` via the following load sequence:

```
WSPutFunction(WSlink, "Once", 1);
WSPutFunction(WSlink, "Get", 1);
WSPutString(WSlink, path);
```

which is equivalent to the call `Get[`*path*`]` within a notebook session.

Note that before we load the package, we call the Mathematica function `Once`, which makes sure that the following function will be executed only once in a Kernel session. Here *path* is a string containing the path to the `.m` file of the backend package; the actual value used is taken from the default GAMBIT backend location file `config/backend_locations.yaml.default` or a user-supplied override.

When loading a package in Mathematica the `Get` function will return information regarding its success. The exact return value will always be package dependent, so is difficult to interpret in general from the GAMBIT interface. Therefore we use the standard library functions of C++ to check if the package and all required functions and variables exist. This is done via the GAMBIT diagnostic system (see Sect. 1.4).

1.3 Backend functions and variables

The backend system in GAMBIT, as described in Sect. 4 of Ref. [1], provides a set of macros for registering functions and variables from backends. In a nutshell, GAMBIT extracts a pointer to the function/variable living in the shared library of

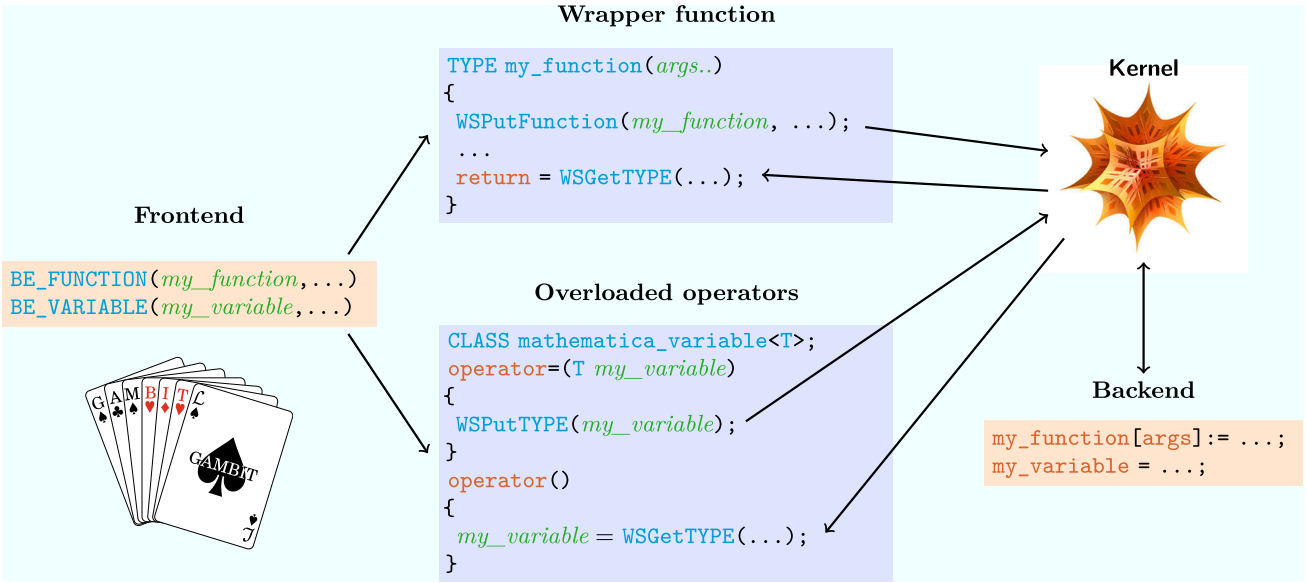

Fig. 1 Diagrammatic representation of the communication link between GAMBIT and the Mathematica Kernel, showing the wrappers for backend functions and the variable classes for backend variables

the backend, and wraps it into a functor object to be handled by the dependency resolver.

When dealing with backends written in Mathematica, the above procedure must be modified, as there is no shared library from which to extract the pointer to the function or variable. Instead, we must interface with the functions and variables via the Mathematica Kernel. The general strategy for doing this is shown in Fig. 1.

1.3.1 Backend functions

Backend functions in GAMBIT are defined with the macro `BE_FUNCTION`. This macro creates a pointer to the function in the namespace of the backend, and assigns it a name and capability within GAMBIT of the frontend author's choosing. In order to communicate with the Mathematica Kernel and provide the same sort of handle to the dependency resolver for Mathematica backends, we must define a wrapper function around message calls to the Kernel, in such a way as to have the wrapper function operate like a regular C/C++ backend function.

We achieve this by redirecting the macro `BE_FUNCTION` for backends written in Mathematica, according to the value of `BACKENDLANG`. The version of `BE_FUNCTION` that gets used for Mathematica backends therefore constructs a wrapper function around each Mathematica function that the user wishes to access from within GAMBIT. The wrapper function handles the submission and receipt of all messages to and from the Kernel relating to the function. This includes all function arguments, and the eventual return value of the function. The sequence of messages is:

```
WSPutFunction(WSlink, symbol_name, n);
WSPutVariable(WSlink, arg_1);
WSPutVariable(WSlink, arg_2);
...
WSPutVariable(WSlink, arg_n);
WSGetVariable(WSlink, &val);
```

where *symbol_name* is the name of the function within the Mathematica package, *n* is the number of arguments it accepts, *arg_1*, *arg_2*, etc are the arguments themselves, and `val` is the return value. Here `WSPutVariable` and `WSGetVariable` are overloaded functions, which expand to different type-specific setter and getter functions intrinsic to WSTP (mentioned in Sect. 1.1).

There is an important subtlety regarding the names of functions in Mathematica backends. Mathematica permits non-alphanumeric characters in function names. Many of the backends that are interesting for GAMBIT include functions with such names. However, function names in GAMBIT are treated as regular strings, so there is no easy way to communicate the actual name of the function to the Kernel. To circumvent this issue, names of Mathematica backend functions should be declared to GAMBIT with all special characters written in terms of their "full names", e.g. α = `\[Alpha]`, but making sure to escape the slash "\" so that C++ can parse it. To illustrate this, we show the signature used for the function `ΔMHiggs` from the backend SUSYHD [2], which calculates uncertainties on the Higgs mass:

```
BE_FUNCTION(DeltaMHiggs, MReal,
 (const MList<MReal>&),
 "\\[CapitalDelta]MHiggs",
  "SUSYHD_DeltaMHiggs")
```

where the first argument is the name of the function within GAMBIT with its return type and signature as the second and third arguments. The last two arguments of the BE_FUNCTION macro are the symbol associated to the backend function, the Mathematica function in our case, and the capability that the function provides.

The generic function call is modified in the case that the symbol name contains non-alphanumeric characters. The symbol name (for example "\\[CapitalDelta]MHiggs") must be sent as a string to the Kernel, preceded by a message that transforms it into a valid Mathematica expression. This expression matches the string version of the name to the name of the actual backend function, including the correct non-alphanumeric characters. The sequence of messages is

```
WSPutFunction(WSlink, "Apply", 2);
WSPutFunction(WSlink, "ToExpression",1);
WSPutString(WSlink, symbol_name);
WSPutFunction(WSlink, "List", n_args);
WSPutVariable(WSlink, arg_1);
WSPutVariable(WSlink, arg_2);
...
WSPutVariable(WSlink, arg_n);
WSGetVariable(WSlink, &val);
```

which is equivalent to WSPutFunction(WSlink, *symbol_name*, *n_args*), in the case that the *symbol_name* contains only alpha-numeric characters.

1.3.2 Backend variables

Backend variables in GAMBIT are registered with the preprocessor macro BE_VARIABLE. For backends written in C, C++ or Fortran, this macro retrieves a pointer to the variable from the shared library. Through this pointer, module functions can extract and modify values of global variables in backends as needed. However, such an interface cannot be constructed if the backend is written in Mathematica. The WSTP messaging system can only access variables from the Kernel by value, so it is not possible to obtain a pointer via which GAMBIT can modify the value inside the Kernel.

To work around this limitation, the GAMBIT 1.1 backend system creates a templated class mathematica_variable <TYPE>, with a private member variable of type TYPE. The templated class acts as a wrapper for the required calls to the Mathematica Kernel. As with backend functions, the macro BE_VARIABLE is redefined for Mathematica backends. This macro creates an instance of the class mathematica_variable<TYPE>, with TYPE equal to the type of the variable in the Kernel (see Sect. 1.2.3). Like backend functions, backend variable names are sent to the Kernel as a string preceded by the ToExpression function, instead of simply sending the symbol name via WSPutSymbol. This allows access to variables with special characters in their names.

As we show schematically in Fig. 1, the mathematica_variable<TYPE> class allows variables in the Kernel to be seamlessly read, modified and implicitly cast to variables of type TYPE, allowing them to be used directly in C++ expressions. These functionalities are achieved by overloading two operators that act on the mathematica_variable<TYPE> class. The first of these is the assignment operator mathematica_variable& operator=(const TYPE &), enabling modification of the variables in the Kernel by regular assignment expressions in module functions. The other is the cast operator operator TYPE const(), which converts an object of type mathematica_variable<TYPE> into its member variable of type TYPE, allowing the variable to be used in all C++ expressions where TYPE would be permitted.

1.4 Diagnostics

GAMBIT already ships with extensive diagnostic tools, providing the user with information about the status of backends, modules, capabilities, scanners and more. We have added a few additional diagnostic checks for Mathematica backends in GAMBIT 1.1.

The first of these simply assesses the status of a Mathematica backend, as typically viewed when using the gambit backends diagnostic. GAMBIT 1.1 adds an additional possible status in this readout, triggered in the case that a specific backend requires Mathematica but no acceptable version of it has been found in the system. When backends are loaded at runtime, they each register in the Backends::backendInfo() object (described in Sec. 4.6 of Ref. [1]) whether or not they require Mathematica, in the new BackendInfo member variable

```
std::map<str, bool> needsMathematica;
```

At runtime, this information is checked against the actual presence (or absence) of Mathematica, and the status Mathematica absent is displayed if it is needed but not found.

GAMBIT will also inform the user of the status of each individual function and global variable in a backend. For Mathematica backends, we achieve this by calling the Mathematica function NameQ. This function takes the symbol name (whether a function or a variable) as a string argument, and returns a boolean result indicating if the symbol is currently present in the Kernel. The diagnostic system then assigns a numerical flag to the status of the function or variable:

```
 1: This function is available, but the
backend
    version does not match your request.
 0: This function is not compatible with
any
    model you are scanning.
-1: The backend that provides this
function is
```

```
    missing.
-2: The backend is present, but the
 function is
    absent or broken.
-5: The backend requires Mathematica, but
    Mathematica is absent.
```

1.5 Higgs mass calculations with SUSYHD

GAMBIT 1.0 offers three options for calculating the mass of the Higgs boson: FlexibleSUSY (1.51) [3], SPheno (3.3.8) [4,5] and FeynHiggs (2.11.2, 2.11.3, 2.12.0) [6–11]. The first two are fixed order $\overline{DR}$ calculations of the Higgs pole mass. The third is an on-shell calculation that can be performed at fixed order, or can include some resummed logarithmic corrections to give a hybrid EFT/fixed order calculation [10–12].[4] Here we provide a different approach to computing the Higgs mass, for any MSSM model,[5] via the Mathematica package SUSYHD [2].

SUSYHD uses the pure EFT calculation, which resums logarithms through the matching and running procedure, using three-loop Standard Model (SM) renormalisation group equations and two-loop matching, providing a more precise result than fixed-order calculations when $M_{\rm SUSY} \gg m_t$. SUSYHD is the first pure EFT Higgs calculation in GAMBIT, and the calculation remains state-of-the-art. This therefore represents an important new option for calculating the Higgs mass. This calculation is most appropriate when $M_{\rm SUSY} \gg m_t$ because the pure EFT approach neglects terms of order $\mathcal{O}(v_0^2/M_{\rm SUSY}^2)$ (where v_0 is the SM Higgs vacuum expectation value), which can be important at the TeV scale and below.

SUSYHD includes three main functions: `MHiggs`, `ΔMHiggs` and `SetSMparameters` with the following signatures

```
MHiggs[shortList_]
ΔMHiggs[shortList_]
SetSMparameters[QMTorgt_,Qα3MZ_]
```

The first two calculate the mass of the lightest Higgs boson and its theoretical uncertainty in the minimal supersymmetric SM (MSSM), starting from a list of $\overline{DR}$ masses at scale $M_{\rm SUSY}$. The third function sets the values of the SM parameters to be used in the calculation.

We use these functions by declaring them to GAMBIT as usual in a frontend header. This SUSYHD frontend header contains the declarations

```
BE_FUNCTION(MHiggs, MReal,
 (const MList<MReal>&),  "MHiggs",
 "SUSYHD_MHiggs")
BE_FUNCTION(DeltaMHiggs, MReal,
 (const MList<MReal>&),
 "\\[CapitalDelta]MHiggs",
 "SUSYHD_DeltaMHiggs")
BE_FUNCTION(SetSMparameters, MVoid,
 (const MReal&, const MReal&),
 "SetSMparameters",
 "SUSYHD_SetSMparameters")
```

GAMBIT does not support alternative interface options for the input parameters of these functions, as it is already guaranteed that the correct input will be provided by the spectrum object. Additional performance options will be supported in future releases.

To use SUSYHD from SpecBit to compute the Higgs mass and uncertainty, we define a new module function `SHD_HiggsMass`, which provides the capability `prec_mh`. The declaration of this function in the rollcall header for SpecBit is:

```
#define FUNCTION SHD_HiggsMass
START_FUNCTION(triplet<double>)
DEPENDENCY(unimproved_MSSM_spectrum,
 Spectrum)
BACKEND_REQ(SUSYHD_MHiggs, (), MReal,
 (const MList<MReal>&))
BACKEND_REQ(SUSYHD_DeltaMHiggs, (), MReal,
 (const MList<MReal>&))
ALLOW_MODELS(MSSM63atQ, MSSM63atMGUT)
#undef FUNCTION
```

2 SpecBit, DecayBit and PrecisionBit updates

GAMBIT 1.1 also includes a simultaneous version update to SpecBit, DecayBit and PrecisionBit [15]. In this section we briefly discuss the changes made to the module functions and structure of the code.

PrecisionBit 1.1 features updated likelihood functions for the first and second generation quark masses, the strong and electromagnetic couplings, the anomalous magnetic moment of the muon, and $\Delta\rho$ (the departure from unity of the ratio of the Fermi couplings associated with W and Z bosons). DecayBit 1.1 contains updates to all SM particle widths, branching fractions and uncertainties. These updates bring the data contained in PrecisionBit and DecayBit up to date with the central values and uncertainties of the 2017 Particle Data Book [16,17].

The module functions of SpecBit and PrecisionBit have undergone some rearrangement and expansion in order to better accommodate alternative spectrum generators and precision calculators. The updated functions are detailed in Tables 1 and 2.

[4] Although both FlexibleSUSY [13] and SPheno [14] also include the option of employing a hybrid $\overline{DR}$ calculation, implementing the algorithm in Ref. [13], these options are not yet included in GAMBIT. They will be added in further updates, as will the pure EFT FlexibleSUSY calculation, HSSUSY.

[5] See Sect. 5.4 of [1] for a list of MSSM models in GAMBIT.

Table 1 New or modified module functions of note in SpecBit 1.1. Functions that use FlexibleSUSY to generate an `unimproved_MSSM_spectrum` have been appropriately renamed, a new precision SM-like Higgs mass calculator using SUSYHD has been added, and the precision Higgs mass calculation with FeynHiggs has been split into separate functions for the SM-like Higgs and the other three MSSM Higgs bosons

Capability	Function (return type): brief description	Dependencies	Backend requirements
`unimproved_MSSM_spectrum`	`get_CMSSM_spectrum_FS(Spectrum)`: Renamed from `get_CMSSM_spectrum`. Make an MSSM spectrum object with FlexibleSUSY using CMSSM boundary conditions	`SMINPUTS`	FlexibleSUSY
	`get_MSSMatMGUT_spectrum_FS(Spectrum)`: Renamed from `get_MSSMatMGUT_spectrum`. Make an MSSM spectrum object with FlexibleSUSY using soft breaking masses input at the GUT scale	`SMINPUTS`	FlexibleSUSY
	`get_MSSMatQ_spectrum_FS (Spectrum)`: Renamed from `get_MSSMatQ_spectrum`. Make an MSSM spectrum object with FlexibleSUSY from soft breaking masses input at (user-defined) scale Q	`SMINPUTS`	FlexibleSUSY
`FH_HiggsMasses`	`FH_AllHiggsMasses (fh_HiggsMassObs)`: Renamed from `FH_HiggsMasses`; capabiliity reassigned from `prec_HiggsMasses`. Higgs masses and mixings with theoretical uncertainties, as computed by FeynHiggs	–	FeynHiggs
`prec_mh`	`FH_HiggsMass (triplet<double>)`: Mass of the most SM-like neutral Higgs boson and associated uncertainty, as computed by FeynHiggs	`FH_HiggsMasses` `unimproved_MSSM_spectrum`	–
	`SHD_HiggsMass (triplet<double>)`: Mass of the most SM-like neutral Higgs boson and associated uncertainty, as computed by SUSYHD	`unimproved_MSSM_spectrum`	SUSYHD
`prec_HeavyHiggsMasses`	`FH_HeavyHiggsMasses (std::map<int,triplet<double>>)`: Masses of the three non-SM Higgs bosons and associated uncertainties in a model with two Higgs doublets, as computed by FeynHiggs	`FH_HiggsMasses` `unimproved_MSSM_spectrum`	–

Table 2 New or modified module functions of note in PrecisionBit 1.1. Precision spectra can now be computed using precision values for 1) the W mass only (as in PrecisionBit 1.0; not shown), 2) the SM-like Higgs mass only, 3) the W mass and the SM-like Higgs, or 4) the W mass and the masses of all four MSSM Higgs bosons. In the latter case, the precision value for the SM-like Higgs mass can be taken from a different source to the precision masses of the other three Higgs bosons

Capability	Function (return type): brief description	Dependencies	Backend requirements
MSSM_spectrum	make_MSSM_precision_spectrum_H(Spectrum): Function to provide an updated MSSM spectrum with precision mass for the most SM-like Higgs boson	unimproved_MSSM_spectrum prec_mh	–
	make_MSSM_precision_spectrum_H_W(Spectrum): Function to provide an updated MSSM spectrum with precision masses for the W boson and the most SM-like Higgs boson	unimproved_MSSM_spectrum prec_mw prec_mh	–
	make_MSSM_precision_spectrum_4H_W(Spectrum): Function to provide an updated MSSM spectrum with precision masses for the W boson and all four Higgs bosons	unimproved_MSSM_spectrum prec_mw prec_mh prec_HeavyHiggsMasses	–

Functions get_CMSSM_spectrum, get_MSSMatQ_spectrum and get_MSSMatMGUT_spectrum of SpecBit 1.0 have been renamed to get_CMSSM_spectrum_FS, get_MSSMatQ_spectrum_FS and get_MSSMatMGUT_spectrum_FS in SpecBit 1.1. This is to explicitly reflect the fact that they make use of FlexibleSUSY for spectrum generation, in contrast to other equivalent functions that use e.g. SPheno (or, in the future, other spectrum generators).

As described in Sect. 4.2.3 of Ref. [15], the Spectrum object computed by SpecBit using a spectrum generator (FlexibleSUSY, SPheno, etc) can be supplemented with more precise calculations of the masses of the W and Higgs bosons. Initial spectrum generation and subsequent precision mass calculations all take place in SpecBit, but PrecisionBit is responsible for combining them to make the final 'precision Spectrum' object.

The computation of the precision mass of the most SM-like Higgs boson in SpecBit is now separated from the computation of the precision masses of the three other MSSM Higgs bosons (Table 1). For the SM-like Higgs (capability prec_mh), we now provide two module functions: FH_HiggsMass and SHD_HiggsMass, which provide the prediction from FeynHiggs and SUSYHD, respectively, including uncertainties. The precision calculation of the masses of the three other MSSM Higgs bosons and their uncertainties (capability prec_HeavyHiggsMasses) is now provided by the function FH_HeavyHiggsMasses, which makes use of results from FeynHiggs.

The precision spectrum functions in PrecisionBit now allow the mass of the SM-like Higgs boson to be improved independently of the masses of the other three MSSM Higgs bosons (Table 2). This makes it possible to use e.g. SUSYHD for the mass of the SM-like Higgs boson, but FeynHiggs, FlexibleSUSY or SPheno for the other Higgses. Each of the precision spectrum functions in PrecisionBit now also recognises a YAML option allow_fallback_to_unimproved_masses, which specifies the preferred behaviour when one or more of the requested precision mass calculations returns an invalid value (zero, a negative mass or not-a-number). With this option set to false (the default), the parameter combination will be flagged as invalid; if it is set to true, failures will instead cause the original masses from the spectrum generator to be retained for the parameter point under investigation. This can be useful for e.g. defaulting to the Higgs mass from FlexibleSUSY or SPheno for models with Lagrangian mass parameters below the top mass, where the expansion used in SUSYHD breaks down and the backend returns zero for the Higgs mass.

GAMBIT 1.1 also marks the removal of any explicit module function for indicating the PDG code of the most SM-like Higgs boson. This functionality is now provided instead by the internal function SMlike_higgs_PDG_code, which takes a high-scale SubSpectrum object as input, and can be called from any module function by including the header gambit/Elements/smlike_higgs.hpp.

3 Updated DarkBit likelihoods

DarkBit 1.1 includes an interface to DDCalc 1.1, along with corresponding likelihood functions for the XENON1T [18] and 2017 PICO-60 [19] experiments. GAMBIT 1.1 also offers an additional option for turning on one-loop corrections to direct detection cross-sections computed with DarkSUSY (loop; enabled by default), and switches the default behaviour of the micrOMEGAs option internal_decays to false, causing decay widths and branching fractions to be passed from DecayBit by default. More details can be found in the DarkBit paper [20].

Funded by SCOAP[3].

References

1. GAMBIT Collaboration: P. Athron, C. Balazs, et. al., GAMBIT: the global and modular beyond-the-standard-model inference tool. Eur. Phys. J. C (2017). arXiv:1705.07908
2. J.P. Vega, G. Villadoro, SusyHD: Higgs mass determination in supersymmetry. JHEP **07**, 159 (2015). arXiv:1504.05200
3. P. Athron, J.-H. Park, D. Stöckinger, A. Voigt, FlexibleSUSY—a spectrum generator for supersymmetric models. Comput. Phys. Commun. **190**, 139–172 (2015). arXiv:1406.2319
4. W. Porod, SPheno, a program for calculating supersymmetric spectra, SUSY particle decays and SUSY particle production at e^+e^- colliders. Comput. Phys. Commun. **153**, 275–315 (2003). arXiv:hep-ph/0301101
5. W. Porod, F. Staub, SPheno 3.1: extensions including flavour, CP-phases and models beyond the MSSM. Comput. Phys. Commun. **183**, 2458–2469 (2012). arXiv:1104.1573
6. S. Heinemeyer, W. Hollik, G. Weiglein, FeynHiggs: a program for the calculation of the masses of the neutral CP even Higgs bosons in the MSSM. Comput. Phys. Commun. **124**, 76–89 (2000). arXiv:hep-ph/9812320
7. S. Heinemeyer, W. Hollik, G. Weiglein, The Masses of the neutral CP-even Higgs bosons in the MSSM: accurate analysis at the two loop level. Eur. Phys. J. C **9**, 343–366 (1999). arXiv:hep-ph/9812472
8. G. Degrassi, S. Heinemeyer, W. Hollik, P. Slavich, G. Weiglein, Towards high precision predictions for the MSSM Higgs sector. Eur. Phys. J. C **28**, 133–143 (2003). arXiv:hep-ph/0212020
9. M. Frank, T. Hahn et al., The Higgs boson masses and mixings of the complex MSSM in the Feynman-diagrammatic approach. JHEP **02**, 047 (2007). arXiv:hep-ph/0611326
10. T. Hahn, S. Heinemeyer, W. Hollik, H. Rzehak, G. Weiglein, High-precision predictions for the light CP-even Higgs boson mass of the minimal supersymmetric standard model. Phys. Rev. Lett. **112**, 141801 (2014). arXiv:1312.4937
11. H. Bahl, W. Hollik, Precise prediction for the light MSSM Higgs boson mass combining effective field theory and fixed-order calculations. Eur. Phys. J. C **76**, 499 (2016). arXiv:1608.01880
12. H. Bahl, S. Heinemeyer, W. Hollik, G. Weiglein, Reconciling EFT and hybrid calculations of the light MSSM Higgs-boson mass. arXiv:1706.00346
13. P. Athron, J.-H. Park, T. Steudtner, D. Stöckinger, A. Voigt, Precise Higgs mass calculations in (non-)minimal supersymmetry at both high and low scales. JHEP **01**, 079 (2017). arXiv:1609.00371
14. F. Staub, W. Porod, Improved predictions for intermediate and heavy supersymmetry in the MSSM and beyond. arXiv:1703.03267
15. GAMBIT Models Workgroup: P. Athron, C. Balázs, et. al., SpecBit, DecayBit and PrecisionBit: GAMBIT modules for computing mass spectra, particle decay rates and precision observables. Eur. Phys. J. C (2017). arXiv:1705.07936
16. Particle Data Group: K.A. Olive et. al., Review of Particle Physics. Update to Ref. [17] (2017)
17. Particle Data Group: C. Patrignani et. al., Review of Particle Physics. Chin. Phys. C **40**, 100001 (2016)
18. XENON: E. Aprile et. al., First dark matter search results from the XENON1T experiment. arXiv:1705.06655
19. PICO: C. Amole et. al., Dark matter search results from the PICO-60 C_3F_8 bubble chamber. Phys. Rev. Lett. **118**, 251301 (2017). arXiv:1702.07666
20. GAMBIT Dark Matter Workgroup: T. Bringmann, J. Conrad, et. al., DarkBit: a GAMBIT module for computing dark matter observables and likelihoods. Eur. Phys. J. C (2017). arXiv:1705.07920