Ben Farmer
(on behalf of the GAMBIT collaboration)
Overview

- The GAMBIT project
- Why global fits?
- GAMBIT overview
  - Design philosophy
  - Model hierarchy
  - Backend system
  - Generic container objects
  - Generic scanner interface (ScannerBit)
  - (For recasting LHC limits, see Martin White’s talk)
  - Comparing tools and preliminary results
- Summary
GAMBIT: The Global And Modular BSM Inference Tool

gambit.hepforge.org

- Fast definition of new datasets and theoretical models
- Plug and play scanning, physics and likelihood packages
- Extensive model database – not just SUSY
- Extensive observable/data libraries
- Many statistical and scanning options (Bayesian & frequentist)
- Fast LHC likelihood calculator
- Massively parallel
- Fully open-source

ATLAS
A. Buckley, P. Jackson, C. Rogan, M. White,
LHCb
M. Chrząszcz, N. Serra
Belle-II
F. Bernlochner, P. Jackson
Fermi-LAT
J. Conrad, J. Edsjö, G. Martinez, P. Scott
CTA
C. Balázs, T. Bringmann, J. Conrad, M. White
HESS
J. Conrad
IceCube
J. Edsjö, P. Scott
XENON/DARWIN
J. Conrad, R. Trotta
Theory
P. Athron, C. Balázs, T. Bringmann,
J. Cornell, J. Edsjö, B. Farmer, T. Gonzalo, S. Hoof,
F. Kahlhoefer, A. Krislock, A. Kvellerstad, M. Pato,
F. Mahmoudi, J. McKay, A. Raklev, R. Ruiz, P. Scott,
R. Trotta, C. Weniger, M. White

27 Members, 9 Experiments, 4 major theory codes, 10 countries

Oskar Klein Centre, Stockholm University

Ben Farmer
Why global fits?

- Lots of theories of TeV scale physics

- For each theory, a parameter space of varying phenomenology

- What new physics scenarios are preferred / ruled out? - Compare to data!
Why global fits?

• In/out cuts based on 95% confidence limits are useful to get a quick heuristic picture.

• But ultimately, we want to make valid statistical inferences.

• To do this, we must perform full likelihood calculations, and sample parameter spaces in a statistically valid way.
But it is a hard problem

- Even for a comprehensive in/out analysis, the problem is multi-faceted:
  - Pick a model
  - Compute predictions for all physical observables of interest
  - Compare these predictions to experimental limits
    - Sometimes limits already exist directly on the relevant quantities for a model.
    - Usually need to recast limits
      - May involve numerically intensive simulations of specific experiments, e.g. simulating LHC collisions, neutrino events (IceCube), gamma ray events (Fermi-LAT), WIMP-nucleon scattering.
      - Limits often dependent on background model assumptions (e.g. dark matter halo models, cosmology, simplified model assumptions)
But it is a hard problem

• **For full statistically valid global fit, even harder**
  
  • Need likelihood calculations for many experiments – can be numerically intensive
  
  • Need computations at many (millions - hundreds of millions) of model points, for moderately complex models (e.g. MSSM7-30)

A lot of effort just for one model family.
Organisation is essential

- It requires a huge amount of manpower to prepare analyses of this kind even for one model class (e.g. MSSM).

- Need to reuse as many calculations and as much “book-keeping” code as possible.
GAMBIT overview
Key design points

- Reuse calculations to the maximum theoretically permissible extent

- Graph-based dependency resolution for run-time “plug and play”
  - Keep calculations as modular as possible so that any piece can be easily “swapped out” for an alternate calculation.
  - Seamless integration of new calculations as they become available (minimise “hacking” of existing code).
Physics modules
- **DarkBit** – dark matter observables (relic density, direct + indirect detection)
- **ColliderBit** – collider observables inc. Higgs + SUSY searches from ATLAS, CMS + LEP
- **FlavBit** – flavour physics inc. $g - 2$, $b \rightarrow s\gamma$, $B$ decays (new channels, angular obs., theory uncerts, LHCb likelihoods)
- **SpecBit** – generic BSM spectrum object, providing RGE running, masses, mixings, etc via interchangeable interfaces to different RGE codes
- **DecayBit** – decay widths for all relevant SM & BSM particles
- **PrecisionBit** – SM likelihoods, precision BSM tests ($W$ mass, $\Delta\rho$ etc)

Each consists of a number of **module functions** that can have **dependencies** on each other.

+ **ScannerBit**: manages stats, sampling and optimisation
YAML file

Model Database

Model α
Model β
Model γ
Model δ
Model ε

User requests scan of model δ using
1. capability of A2
2. capability of B1

Physics Modules

Module functions
(Feed upwards and horizontally only)

Backend functions and variables
(Feed upwards only)

External Library C
C2
C1
C3
C4

Module A
A1 → A2

Module B
B1 → B2

Core

Printers

ScannerBit

Core runs
1. A1
2. A2
3. B2
4. B1

Oskar Klein Centre, Stockholm University
Ben Farmer
Module function declaration

#define CAPABILITY MSSM_spectrum
START_CAPABILITY

// FlexibleSUSY compatible maximal CMSSM generalisation (MSSM with GUT boundary conditions)
#define FUNCTION get_MSSMatMGUT_spectrum
START_FUNCTION(Spectrum)
ALLOW_MODELS(MSSM63atMGUT)
DEPENDENCY(SNINPUTS, SWInputs) // Need SLHA2 SNINPUTS to set up spectrum generator
#undef FUNCTION

#undef CAPABILITY

// FeynHiggs SUSY masses and mixings
#define CAPABILITY FH_WSSMMasses
START_CAPABILITY
#define FUNCTION FH_WSSMMasses
START_FUNCTION(fh_WSSMMass0bs)
BACKEND_REQ(FHGetPara, (libfeynhiggs), void, (int&, int&,
   Farray<fh_real, 1,2, 1,5, 1,3>&, Farray<fh_complex, 1,2, 1,2, 1,5, 1,3>&,
   Farray<fh_real, 1,6, 1,5>&, Farray<fh_complex, 1,36, 1,5>&,
   Farray< fh_real,1,2>&, Farray< fh_complex,1,4>&,
   Farray< fh_complex,1,4>&, Farray< fh_real,1,4>&,
   Farray< fh_complex,1,16>&, fh_complex&,, fh_real&,
   Farray< fh_real,1,4>&, fh_real&))
BACKEND_OPTION( (FeynHiggs. 2.11.2, 2.11.3), (libfeynhiggs) )
ALLOW_MODELS(MSSM63atQ, MSSM63atMGUT)
#undef FUNCTION
#undef CAPABILITY
Example dependency graphs

CMSSM:

MSSM7:

Red: Model parameter translations
Blue: Precision calculations
Green: LEP rates+likelihoods
Purple: Decays
Orange: LHC observables and likelihoods
Grey: DM direct, indirect and relic density
Pink: Flavour physics
Hierarchical Model Database

void MODEL_NAMESPACE::MSSM19atQ_to_MSSM24atQ (const ModelParameters &myP, ModelParameters &targetP) {
    // Send all parameter values upstream to matching parameters in parent.
    // Ignore that some parameters don’t exist in the parent, as these are set below.
    targetP.setValues(myP, false);
    // RH squark soft masses, gen 1 and 2
    targetP.setValue("mq2_1", myP["nq2_12"]); // mq2_11 in MSSM63
    targetP.setValue("mq2_2", myP["nq2_12"]); // mq2_22 ""
    // RH slepton soft masses, gen 1 and 2
    targetP.setValue("ml2_1", myP["nl2_12"]); // ml2_11 in MSSM63
    targetP.setValue("ml2_2", myP["nl2_12"]); // ml2_22 ""
    // LH down-type squark soft masses
    targetP.setValue("md2_1", myP["nd2_12"]); // md2_11 in MSSM63
    targetP.setValue("md2_2", myP["nd2_12"]); // md2_22 ""
    // LH up-type squark soft masses
    targetP.setValue("mu2_1", myP["nu2_12"]); // mu2_11 in MSSM63
    targetP.setValue("mu2_2", myP["nu2_12"]); // mu2_22 ""
    // LH charged slepton soft masses
    targetP.setValue("me2_1", myP["ne2_12"]); // me2_11 in MSSM63
    targetP.setValue("me2_2", myP["ne2_12"]); // me2_22 ""
}

#define MODEL MSSM19atQ
#define PARENT MSSM24atQ

START_MODEL
DEFINEPARS(Qin,TanBeta,SignMu,
           mHd2, mHd2, U1, U2, U3)
DEFINEPARS(mq2_12, mq2_3)
DEFINEPARS(ml2_12, ml2_3)
DEFINEPARS(md2_12, md2_3)
DEFINEPARS(mu2_12, mu2_3)
DEFINEPARS(me2_12, me2_3)
DEFINEPARS(Ad_3)
DEFINEPARS(Au_3)
INTERPRET_AS_PARENT_FUNCTION(MSSM19atQ_to_MSSM24atQ)
INTERPRET_AS_X_FUNCTION(MSSM24atQ, MSSM19atQ_to_MSSM24atQ)
#undef PARENT
#undef MODEL
Mix and match backends

<table>
<thead>
<tr>
<th>BACKENDS</th>
<th>VERSION</th>
<th>PATH TO LIB</th>
<th>STATUS</th>
<th>#FUNC</th>
<th>#TYPES</th>
<th>#CTORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDCalc0</td>
<td>0.0</td>
<td>Backends/installed/DDCalc/0.0/libDDCalc0.so</td>
<td>OK</td>
<td>62</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DarkSUSY</td>
<td>5.1.1</td>
<td>Backends/installed/DarkSUSY/5.1.1/lib/darksusy.so</td>
<td>OK</td>
<td>68</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FastSim</td>
<td>1.0</td>
<td>Backends/installed/fastsim/1.0/libfastsim.so</td>
<td>absent/broken</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FeynHiggs</td>
<td>2.11</td>
<td>Backends/installed/FeynHiggs/2.11.2/lib/libFH.so</td>
<td>OK</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HiggsBounds</td>
<td>4.2.1</td>
<td>Backends/installed/HiggsBounds/4.2.1/lib/libhiggsbounds.so</td>
<td>OK</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HiggsSignals</td>
<td>1.4</td>
<td>Backends/installed/HiggsSignals/1.4.0/lib/libhiggs signals.so</td>
<td>OK</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LibFarrayTest</td>
<td>1.0</td>
<td>Backends/examples/libFarrayTest.so</td>
<td>OK</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LibFirst</td>
<td>1.0</td>
<td>Backends/examples/libfirst.so</td>
<td>OK</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>Backends/examples/libfirst.so</td>
<td>OK</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LibFortran</td>
<td>1.0</td>
<td>Backends/examples/libfortran.so</td>
<td>OK</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MicrOmegs</td>
<td>3.5.5</td>
<td>Backends/installed/micromegas/3.5.5/MSSM/MSM/libmicromegas.so</td>
<td>OK</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MicrOmegsSingletDM</td>
<td>3.5.5</td>
<td>Backends/installed/micromegas/3.5.5/SingletDM/SingletDM/libmicromegas.so</td>
<td>OK</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pythia</td>
<td>8.186</td>
<td>Backends/installed/Pythia/8.186/lib/pythia8.so</td>
<td>absent/broken</td>
<td>0</td>
<td>27</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>8.209</td>
<td>Backends/installed/Pythia/8.209/lib/pythia8.so</td>
<td>OK</td>
<td>0</td>
<td>28</td>
<td>107</td>
</tr>
<tr>
<td>SUSYPOPE</td>
<td>0.2</td>
<td>no path in config/backend_locations.yaml</td>
<td>absent/broken</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SUSY_HIT</td>
<td>1.5</td>
<td>Backends/installed/SUSY-HIT/1.5/libsusyhit.so</td>
<td>OK</td>
<td>55</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SuperIso</td>
<td>3.4</td>
<td>Backends/installed/SuperIso/3.4/libsuperiso.so</td>
<td>OK</td>
<td>32</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gamLike</td>
<td>1.0.0</td>
<td>Backends/installed/gamLike/1.0.0/lib/gamLike.so</td>
<td>OK</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nulike</td>
<td>1.0.0</td>
<td>Backends/installed/nulike/1.0.0/lib/nulike.so</td>
<td>OK</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Generic container objects

Mass spectra:

Generic interface functions:

```c
double MZ = spec.get(Pole_Mass, "Z0");
spec.RunToScale(MZ);
```

Backend wrapper:

```c
map_collection[Par::Pole_Mass].map1 = tmp_map;
```

Decays:

```c
// Declare a set of decays for the W+

DecayTable::Entry Wplus;

// Set the total width
Wplus.width_inGeV = 2.085;

// Set the branching fraction for W \to e \nu_e
Wplus.setBF(0.1071, 0.0016, "e", "nu_e");

// Find the partial width for W \to e \nu_e
double pw = Wplus.width_inGeV*Wplus.BF("e", "nu_e");
```

PDG code convenience overloads:

```c
spectrum.get(Pole_Mass, "e-", 1);
spectrum.get(Pole_Mass, "e-", 1);
spectrum.get(Pole_Mass, std::make_pair("e-", 1));
spectrum.get(Pole_Mass, 1000011, 0);
spectrum.get(Pole_Mass, std::make_pair(1000011, 0));
spectrum.get(Pole_Mass, "e+", 1);
spectrum.get(Pole_Mass, "e+", 1);
spectrum.get(Pole_Mass, std::make_pair("e+", 1));
spectrum.get(Pole_Mass, -1000011, 0);
spectrum.get(Pole_Mass, std::make_pair(-1000011, 0));
```

ProcessCatalogue (DarkBit) – Decay and annihilation processes (used for dark matter annihilation rates, gamma-ray and neutrino yields for indirect searches)
Multi-format output system

Current plugins: ASCII and HDF5

Results of module function calculations automatically streamed to output system, in format selected at runtime.

HDF5 support added to Pippi (“parse it, plot it” https://github.com/patscott/pippi)

But HDF5 interface exists in many languages (C, Fortran, Java, C++, Python)

e.g. Plotting in Python is easy:

```python
import h5py
import numpy as np

f = h5py.File("runs/spartan_multinest_hdf5/samples/results.hdf5",'r')
group = f["/gambit_data"]

mu = group["#NormalDist_parameters @NormalDist::primary_parameters::mu"]
```
Uniform interface to scanning algorithms and priors (ScannerBit)

- **External algorithms (or native C++) wrapped into plugins:**

```cpp
scanner_plugin(random_sampler, version(1, 0, 0))
{
    like_ptr LogLike;
    int num, dim;

    plugin_constructor
    {
        LogLike = get_purpose(get_inifile_value<std::string>("like"));
        num = get_inifile_value<int>("point_number", 10);
        dim = get_dimension();
    }

    int plugin_main ()
    {
        std::vector<double> a(dim);
        std::cout << "Entering random sampler." << "\n\nnumber of points to calculate: " << num << std::endl;
        for (int k = 0; k < num; k++)
        {
            for (int i = 0; i < dim; i++)
            {
                a[i] = Gambit::Random::draw();
                LogLike(a);
                if (k%1000 == 0)
                    std::cout << "points: " << k << " / " << num << std::endl;
            }
            return 0;
        }
    }
}
Other tools:

**CMake** build system to organise compilation of many tools and modules.

**Scanners:** MultiNest, Diver (diff. evolution), twalk (population MC), GreAT (MCMC)

**Parallelisation:** Mixed mode MPI + openMP, mostly automated

**Backends:** dynamic loading of C++ classes from backends (BOSS)

**POSIX signal handling:** for safe early shutdown/resuming
LHC likelihoods

- **MC generation**: Pythia8 parallelised with OpenMP + other speed tweaks
- **Detector simulation**: fast simulation based on 4-vector smearing → matches DELPHES results very closely (but much faster)
- **Cross-sections**: LO + LL from MC generator by default (fast NLO in works for SUSY)
- **Analysis framework**: custom event-level, independent of experiment or simulation
- **Likelihood**: inline systematic error marginalisation (via nulike)
- **Initially shipping with**:
  - ATLAS SUSY searches ($0\ell$, $0/1/2\ell\tilde{t}$, $b$ jets + MET, $2/3\ell$ EW)
  - CMS multi-$\ell$ SUSY
  - CMS DM ($t$ pair + MET, mono-$b$, monojet)

For more details, see talk by Martin White
(1630 Thursday, Experimental and Collider Aspects of SUSY session)
## Comparison of tools

<table>
<thead>
<tr>
<th>Model point description</th>
<th>$\Omega h^2$ DarkSUSY</th>
<th>$\Omega h^2$ micrOMEGAs</th>
<th>$\sigma_{SI,p}$ [10^{-46}cm²] DarkSUSY</th>
<th>$\sigma_{SI,p}$ [10^{-46}cm²] micrOMEGAs</th>
<th>$\sigma_{SD,p}$ [10^{-40}cm²] DarkSUSY</th>
<th>$\sigma_{SD,p}$ [10^{-40}cm²] micrOMEGAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resonant annihilation via $A^0$, gaugino-like neutralino</td>
<td>0.08110</td>
<td>0.07877</td>
<td>4.317</td>
<td>4.778</td>
<td>10.20 × 10^{-6}</td>
<td>9.381 × 10^{-6}</td>
</tr>
<tr>
<td>Resonant annihilation via $A^0$, mixed neutralino</td>
<td>0.08067</td>
<td>0.08521</td>
<td>1.827</td>
<td>1.937</td>
<td>5.713</td>
<td>5.422</td>
</tr>
<tr>
<td>Resonant annihilation via $A^0$, Higgsino-like neutralino</td>
<td>0.08063</td>
<td>0.08543</td>
<td>0.7372</td>
<td>0.6826</td>
<td>1.564 × 10^{-4}</td>
<td>1.485 × 10^{-4}</td>
</tr>
<tr>
<td>Sfermion coannihilations, gaugino-like neutralino</td>
<td>0.1775</td>
<td>0.2461</td>
<td>1.667</td>
<td>1.716</td>
<td>4.209 × 10^{-3}</td>
<td>24.59 × 10^{-3}</td>
</tr>
<tr>
<td>Sfermion coannihilations, mixed neutralino</td>
<td>0.09823</td>
<td>0.1112</td>
<td>404.1</td>
<td>378.5</td>
<td>0.2476</td>
<td>0.2125</td>
</tr>
<tr>
<td>Sfermion coannihilations, Higgsino-like neutralino</td>
<td>7.549 × 10^{-3}</td>
<td>0.1095</td>
<td>18.74</td>
<td>29.87</td>
<td>1.686 × 10^{-3}</td>
<td>1.464 × 10^{-3}</td>
</tr>
<tr>
<td>Chargino coannihilations</td>
<td>0.09655</td>
<td>0.09997</td>
<td>0.5142</td>
<td>0.4848</td>
<td>16.92</td>
<td>16.05</td>
</tr>
<tr>
<td>Chargino coannihilations</td>
<td>0.08138</td>
<td>0.08720</td>
<td>4.147</td>
<td>3.954</td>
<td>9.122</td>
<td>8.657</td>
</tr>
</tbody>
</table>

**Table 20:** A table showing the dark matter relic density and proton scattering cross-sections, both spin-independent and spin-dependent, for a range of MSSM model points. The points were chosen to have different types of processes contribute to the relic density calculation as shown in the model point description column. All quantities were calculated through DarkBit_standalone_MSSM using both the DarkSUSY and micrOMEGAs backends.
Comparison of scanning algorithms
Scalar singlet dark matter
(Relic density + direct detection + Multinest scanner)
Comparison of scanning algorithms
Scalar singlet dark matter
(Relic density + direct detection + Diver scanner)

Oskar Klein Centre, Stockholm University
Ben Farmer

Cline et al. 2013
Summary

- GAMBIT to be released later this year
- As well as likelihoods and physics calculations for popular models, offers a robust framework and numerous helpful utilities for systematically including more and more models, likelihoods for new experiments, and scanning algorithms over time (we want to backend your codes!)
- We are very happy to talk to prospective users!

(some other GAMBITers here at SUSY2016 to talk to:
- Peter Athron
- Csaba Balazs
- Paul Jackson
- Chris Rogan
- Martin White)