https://sites.google.com/view/mark-d-goodsell

The W mass in SARAH

Mark Goodsell

@PitifulRed

https://realselfenergy.blogspot.com

LPTHE
LABORATOIRE DE PHYSIQUE THEORIQUE ET HAUTES ENERGIES

SORBONNE UNIVERSITÉ

PROJET FINANCÉ PAR L'ANR - PROJECT FUNDED BY THE ANR

cnrs

Overview

- W mass calculation in SARAH
- Other ongoing developments in SARAH
- Advertising for BSMArt

Part 1: W mass in SARAH

Based largely on arXiv:2208.05867 with Benakli, Ke and Slavich

Pre 2022 SARAH W mass

Originally the computation was along the lines of BPMZ:

Extract EW parameters:

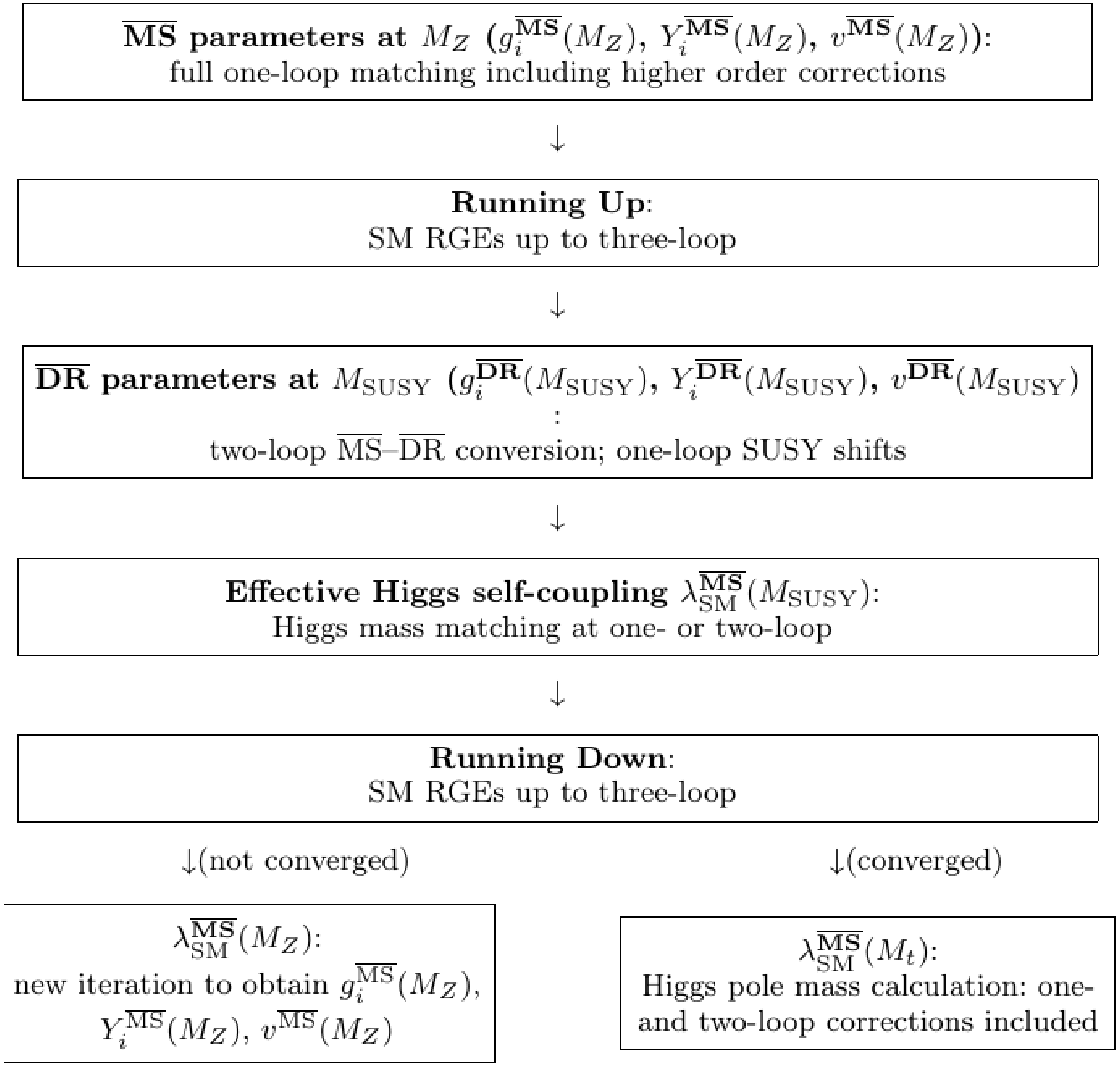
$$\hat{\alpha} = \frac{\alpha_{em}}{1 - \Delta\hat{\alpha}} \qquad \hat{c}^2 \hat{s}^2 = \frac{\pi \hat{\alpha}}{\sqrt{2}\, M_Z^2\, G_\mu\, (1 - \Delta\hat{r})}$$

$$\Delta\hat{r} = \hat{\rho}\, \frac{\Pi^T_{WW}(0)}{M_W^2} - \mathcal{R}e\, \frac{\Pi^T_{ZZ}(M_Z^2)}{M_Z^2} + \delta_{\rm VB}$$

Compute the pole mass:

$$M_W^2 = \hat{M}_W^2(Q) - \mathcal{R}e\, \Pi^T_{WW}(M_W^2)$$

In 2017 in Staub & Porod, arXiv:1703.03267 the spectrum calculation was upgraded to employ pole-mass matching, from version 4.10.0:



$\overline{\textbf{MS}}$ **parameters at** M_Z $(g_i^{\overline{\mathbf{MS}}}(M_Z),\ Y_i^{\overline{\mathbf{MS}}}(M_Z),\ v^{\overline{\mathbf{MS}}}(M_Z))$:
full one-loop matching including higher order corrections

$\downarrow$

Running Up:
SM RGEs up to three-loop

$\downarrow$

$\overline{\textbf{DR}}$ **parameters at** M_{SUSY} $(g_i^{\overline{\mathbf{DR}}}(M_{\text{SUSY}}),\ Y_i^{\overline{\mathbf{DR}}}(M_{\text{SUSY}}),\ v^{\overline{\mathbf{DR}}}(M_{\text{SUSY}})$
:
two-loop $\overline{\text{MS}}$–$\overline{\text{DR}}$ conversion; one-loop SUSY shifts

$\downarrow$

Effective Higgs self-coupling $\lambda_{\text{SM}}^{\overline{\mathbf{MS}}}(M_{\text{SUSY}})$:
Higgs mass matching at one- or two-loop

$\downarrow$

Running Down:
SM RGEs up to three-loop

$\downarrow$(not converged)

$\lambda_{\text{SM}}^{\overline{\mathbf{MS}}}(M_Z)$:
new iteration to obtain $g_i^{\overline{\text{MS}}}(M_Z)$, $Y_i^{\overline{\text{MS}}}(M_Z)$, $v^{\overline{\text{MS}}}(M_Z)$

$\downarrow$(converged)

$\lambda_{\text{SM}}^{\overline{\mathbf{MS}}}(M_t)$:
Higgs pole mass calculation: one- and two-loop corrections included

Unfortunately the W mass computation was not upgraded at the same time

New strategy

If we are matching to an effective SM and want to modify MW, need to use the SMEFT:

$$\begin{aligned}\delta m_W^2 &= \hat{m}_W^2(\sqrt{2}\delta G_F + 2\frac{\delta g_2}{\hat{g}_2}) \\ &= -\frac{s_{2\hat{\theta}}\overline{v}_T^2}{4c_{2\hat{\theta}}}\left(\frac{c_{\hat{\theta}}}{s_{\hat{\theta}}}C_{HD} + \frac{s_{\hat{\theta}}}{c_{\hat{\theta}}}(4C_{H\ell}^{(3)} - 2C_{\ell\ell}) + 4C_{HWB}\right)\end{aligned}$$

Ideally, would extract all the operators at the matching scale and run, this would resum large logs of form

$$\log M^2/Q^2$$

This should be the approach when the matching is actually available!

c.f.

Athron *et al*, 2204.05285

$$M_W^2 = (M_W^2)_{SM}\left(1 + \frac{s_W^2}{c_W^2 - s_W^2}\left[\frac{c_W^2}{s_W^2}(\Delta\rho_{\rm tree} + \Delta\rho) - \Delta r_W - \Delta\alpha\right]\right)$$

Can make a 1-1 mapping between this and combinations of SMEFT coefficients at the matching scale!

Tree-level part

e.g.

$$C_{HD} = \frac{2}{v^2}\left[-\Delta\rho_0(1 + \frac{\Pi_{WW}(0)}{M_W^2}) + \frac{\Pi_{WW}(0)}{M_W^2} - \frac{\Pi_{ZZ}(0)}{M_Z^2}\right]$$

Just need to separate out

$$\delta_{VB} = \frac{1}{16\pi^2}\left[\frac{V}{g_2} - \frac{2c_W^2 M_Z^2}{g_2^2}\mathrm{Re}(B)\right]$$

$C^{(3)}_{H\ell}$

$C_{\ell\ell}$

For now, we just match the total combination and neglect running in the SMEFT

New procedure (stopgap):

Run up SM parameters to matching scale

Guess parameters in high-energy theory based on tree-level matching plus gauge thresholds on first run. On subsequent runs use stored Yukawas

Compute loop masses in HET and “effective SM” subset of theory. Compute

$$c_W^2 s_W^2 = (c_W^2 s_W^2)_{\mathrm{SM}} \times \frac{(1+\Delta\alpha+\Delta\hat{r})}{1+\Delta\rho_{\mathrm{tree}}}$$

Match quantities in the SM from the high-energy ones via pole matching. Compute

$$\left(1+\frac{s_W^2}{c_W^2-s_W^2}\left[\frac{c_W^2}{s_W^2}(\Delta\rho_{\mathrm{tree}}+\Delta\rho)-\Delta r_W-\Delta\alpha\right]\right)$$

Once converged: run down to top mass and compute Higgs and W masses in the SM wih the above correction

We use

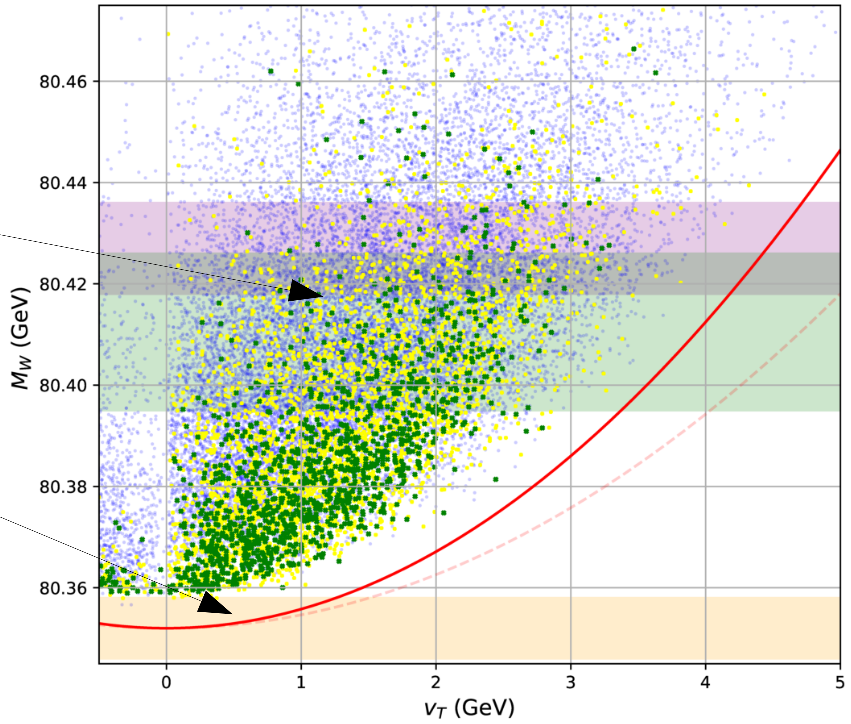
$$\Delta\rho_0 = \frac{M_W^2(\mathrm{tree})}{M_Z^2(\mathrm{tree})}\frac{g_1^2+g_2^2}{g_2^2}$$

The tree-level relations are determined in the high-energy theory

Can incorporate tree-level shifts, which occur in models with extra vevs, e.g. triplets in Dirac Gaugino models

$$\rho = 1 + 4\frac{v_T^2}{v^2} \longrightarrow M_W^2 = \rho c_W^2 M_Z^2$$

Or shifts from Z' mixing with the Z boson … such as from heterotic-inspired Z' models (work in progress with A. Faraggi)



Part 2: ongoing directions for SARAH

Ongoing developments in SARAH

- Unitarity for fermions
- Improvements to the W mass calculation
- Charged Higgs masses @ 2 loops
- Genuine pole mass matching at two loops
- Improvements to the muon g-2 calculation, EDMs, …
- WET matching
- Full SMEFT matching (longer term goal)

To be made public soon

I would love to talk in detail about some of these … needs a little more time ...

Muon g-2 and EDMs

Only a handful of SMEFT operators are important for lepton g-2:

Nowadays if we can compute the SMEFT coefficients, can include running effects equivalent to the leading logs of 2-loop fixed-order … but more precise because we resum the logs

Implementation of (most of) these in SARAH will be available soon

Dipoles, contribute at leading order (one loop)

$$
\begin{aligned}
\Delta a_\ell^{250\text{ GeV}} = \frac{m_\ell}{m_\mu}\mathrm{Re}\Bigg[& \begin{matrix}2.9_\mu\\ 2.8_e\end{matrix}\times 10^{-3}\widetilde{C}_{\substack{eB\\ \ell\ell}} - \begin{matrix}1.6_\mu\\ 1.5_e\end{matrix}\times 10^{-3}\widetilde{C}_{\substack{eW\\ \ell\ell}} \\
& - \begin{matrix}4.3_\mu\\ 4.1_e\end{matrix}\times 10^{-5}\widetilde{C}^{(3)}_{\substack{\ell equ\\ \ell\ell 33}} - \left(2.6 + 0.37c_T^{(c)}\right)\times 10^{-6}\widetilde{C}^{(3)}_{\substack{\ell equ\\ \ell\ell 22}} \\
& - 7.9\times 10^{-8}\widetilde{C}_{\substack{\ell e\\ \ell 33\ell}} + \left(5.7c_T - \begin{matrix}0.49_\mu\\ 0.48_e\end{matrix}\right)\times 10^{-8}\widetilde{C}^{(3)}_{\substack{\ell equ\\ \ell\ell 11}} + 1.4\times 10^{-8}\widetilde{C}^{(1)}_{\substack{\ell equ\\ \ell\ell 33}} \\
& + \left(\begin{matrix}10_\mu\\ 9.8_e\end{matrix} + 2.5c_T^{(c)}\right)\times 10^{-9}\widetilde{C}^{(1)}_{\substack{\ell equ\\ \ell\ell 22}} - \begin{matrix}4.6_\mu\\ 4.7_e\end{matrix}\times 10^{-9}\widetilde{C}_{\substack{\ell e\\ \ell 22\ell}} \\
& + \frac{m_\ell}{m_\mu}\Bigg\{\begin{matrix}2.5_\mu\\ 2.4_e\end{matrix}\times 10^{-8}\left(\widetilde{C}_{HWB} + i\widetilde{C}_{H\widetilde{W}B}\right) - \begin{matrix}1.8_\mu\\ 1.7_e\end{matrix}\times 10^{-8}\left(\widetilde{C}_{HB} + i\widetilde{C}_{H\widetilde{B}}\right) \\
& - \begin{matrix}6.0_\mu\\ 5.7_e\end{matrix}\times 10^{-9}\left(\widetilde{C}_{HW} + i\widetilde{C}_{H\widetilde{W}}\right) + 3.8\times 10^{-9}\widetilde{C}_{\substack{He\\ \ell\ell}} - \begin{matrix}3.7_\mu\\ 3.6_e\end{matrix}\times 10^{-9}\widetilde{C}^{(1)}_{\substack{Hl\\ \ell\ell}} \\
& + \begin{matrix}3.6_\mu\\ 3.3_e\end{matrix}\times 10^{-9}\widetilde{C}^{(3)}_{\substack{Hl\\ \ell\ell}} + \begin{matrix}1.8_\mu\\ 1.7_e\end{matrix}\times 10^{-9}\widetilde{C}_{HD} + \begin{matrix}2.1_\mu\\ 2.0_e\end{matrix}\times 10^{-9}\widetilde{C}_{W} \\
& + 1.1\times 10^{-9}i\widetilde{C}_{\widetilde{W}}\Bigg\}\Bigg]
\end{aligned}
$$

From Aebischer *et al*, 2102.08954

SMEFT vs FlavorKit

For getting flavour constraints from SARAH, in principle the workflow can be

$$\begin{tikzcd} \boxed{\text{FlavorKit}} \arrow[r] & \boxed{\text{WCXF file}} \arrow[r] & \boxed{\text{flavio}} \end{tikzcd}$$

However, there are issues:

- FlavorKit basis is not a complete basis of the WET (this can be got around)
- It cannot be made to be as there is no wavefunction renormalisation …
- Generating new operators is probably not possible any more as PreSARAH no longer works with newer versions of FormCalc
- If we wanted to upgrade to the SMEFT we need new classes of operators not included in FlavorKit anyway

So it was necessary to build an entirely new module to handle these calculations … this is still work in progress. An early step might actually just be a complete WET basis!

Part 3: BSMArt

For your new favourite model:

- Specify some set of fields and pattern of masses at high energy scale (maybe just a TeV)
- Potentially have a constrained set of UV parameters (e.g. mSUGRA ~ 5 parameters) but large set of observables we want to check

- EWPT
- Higgs mass
- Higgs couplings
- Rare decays

→ Compute spectrum and decays with SARAH → Check with HiggsBounds, HiggsSignals, Lilleth, flavio, smelli, ...

- Dark matter

→ Produce CalcHEP /UFO from SARAH, compute with MicrOmegas/MadDM/...

- LHC searches

→ Produce UFO from SARAH, events with MadGraph (or maybe Pythia) → MadAnalysis, SModelS, ColliderBit, Hackanalysis...

- ...

From the **bottom up:**

- May have *many* parameters not fixed by observations, e.g.
  - e.g. > 100 in MSSM
  - 2499 in SMEFT
- Don’t necessarily have good priors
- Need to apply experimental and ***theoretical*** constraints:

- EWPT
- Higgs mass
- Higgs couplings
- Rare decays
- Dark matter
- LHC searches
- ...

- Vacuum stability
- Landau pole vs cutoff
- Unitarity / Positivity bounds
- ???

Not very well exploited

Parameter space exploration

Suppose I have some model to confront to data with a few parameters. Simplest (old school) approach is to run the codes on a grid or do a random selection

$\longrightarrow$

Usually people have their own codes (reinventing the wheel each time), but there existed SARAH Scan and Plot (Mathematica package).

Not much else AFAIK

- This is massively inefficient if the number of parameters is large! E.g. scan 10 points per variable. So no good for even pMSSM, don't even think about full MSSM.
- Also there is the problem of how to combine constraints.

But: it's simple to understand and very useful for making line plots & simple models.

Likelihood sampling

A solution to the problem of combining constraints is to construct a likelihood function:

$$\mathcal{L} = \mathcal{L}_{\text{collider}} \times \mathcal{L}_{\text{DM}} \times \mathcal{L}_{\text{Higgs}} \times \mathcal{L}_{\text{Flavour}} \cdots$$

This can be a simple gaussian:

$$\mathcal{L}_i = \exp\left[-\frac{(\mathcal{O}_i - \overline{\mathcal{O}}_i)^2}{2\sigma^2}\right]$$

Or we can take log likelihoods and add.

We can also take correlations into account

Markov-Chain-Monte-Carlo techniques give a sample of points where, after a long enough time $\longrightarrow$ $\text{density} \propto \mathcal{L}$

There now exist sophisticated generalisations of this (e.g. multinest, diver, …) developed for other fields, but the aim is always to have points distributed proportional to the likelihood

The advantages claimed for this are that there is a rigorous statistical interpretation in a Bayesian/frequentist approach:

- Can make statements about statistics
- Can find ‘most likely point’
- Can use likelihoods to compare models

As a result the community has mostly adopted this strategy.

Numerous papers using this; e.g. Mastercode collaboration scanning the MSSM

GAMBIT now includes **GUM** (Gambit Universal Machine) which generates the model files from SARAH or FeynRules for any model.

Suppose we aren’t interested in having points around the most ‘likely’ regions, e.g.:

- If we ignore low-energy anomalies, why should new physics be very light? The likelihood function should be flat at high energies.
- What if we are interested instead in the exclusion boundary? By definition these are unlikely regions, all standard algorithms give only very few points there – need a supercomputer/long time to map it out.

I argue that this is the sort of question many people want to ask instead – especially from string models where the overall mass scale may vary by orders of magnitude.

Want something simple that runs on a laptop that tells us what the allowed values are.

BSMArt

BSMArt is a new python code that fills this gap:

- Designed for SARAH family of tools
- Replaces BSMToolbox: shipped with scripts to install all necessary tools and build the code for your model:

```
./installHEPtools.py --All
```

```
./prepareModel.py --All <ModelName>
```

```
.<BSMArt path>/bin/BSMArt <input json file>
```

Compatible Tools

- SPheno
- MicrOMEGAs
- HiggsBounds
- HiggsSignals
- HiggsTools
- Vevacious++
- HackAnalysis
- MadGraph
- flavio

Via effC approach and python interface (thanks!)

Python anyway … very easy to interface

SModelS and Lilith should be available in the next version … probably also NMSSMCalc!

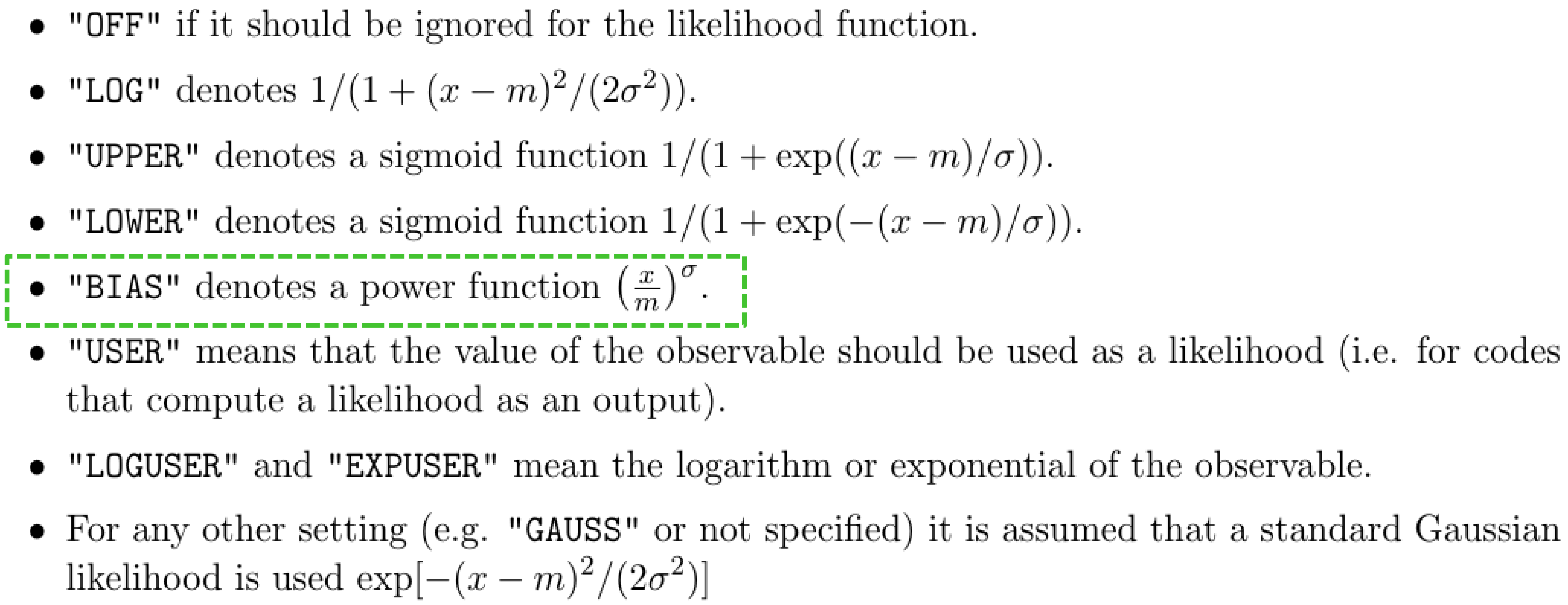
Included Scans

Scans can be written as standalone python scripts using the physics engine as a black box

Or external scanning programs can be used in the same way (even with MPI)

- Grid
- Random
- read_dir
- read_csv
- MCMC (with many options for likelihoods)
- AL
- MultiNest
- Diver

E.g. in the likelihood-based scans (MCMC, MultiNest, Diver, ...), have fine control over the likelihood function for each variable, including ‘cheats’:



- `"OFF"` if it should be ignored for the likelihood function.
- `"LOG"` denotes $1/(1+(x-m)^2/(2\sigma^2))$.
- `"UPPER"` denotes a sigmoid function $1/(1+\exp((x-m)/\sigma))$.
- `"LOWER"` denotes a sigmoid function $1/(1+\exp(-(x-m)/\sigma))$.
- `"BIAS"` denotes a power function $\left(\frac{x}{m}\right)^{\sigma}$.
- `"USER"` means that the value of the observable should be used as a likelihood (i.e. for codes that compute a likelihood as an output).
- `"LOGUSER"` and `"EXPUSER"` mean the logarithm or exponential of the observable.
- For any other setting (e.g. `"GAUSS"` or not specified) it is assumed that a standard Gaussian likelihood is used $\exp[-(x-m)^2/(2\sigma^2)]$

Setting up a scan

Inputs are handled by a json file.

This contains info about the codes to run, the parameters of the scan, plots to generate

For SARAH-based scans, usually an slha template file (like the one automatically generated by SARAH) is needed

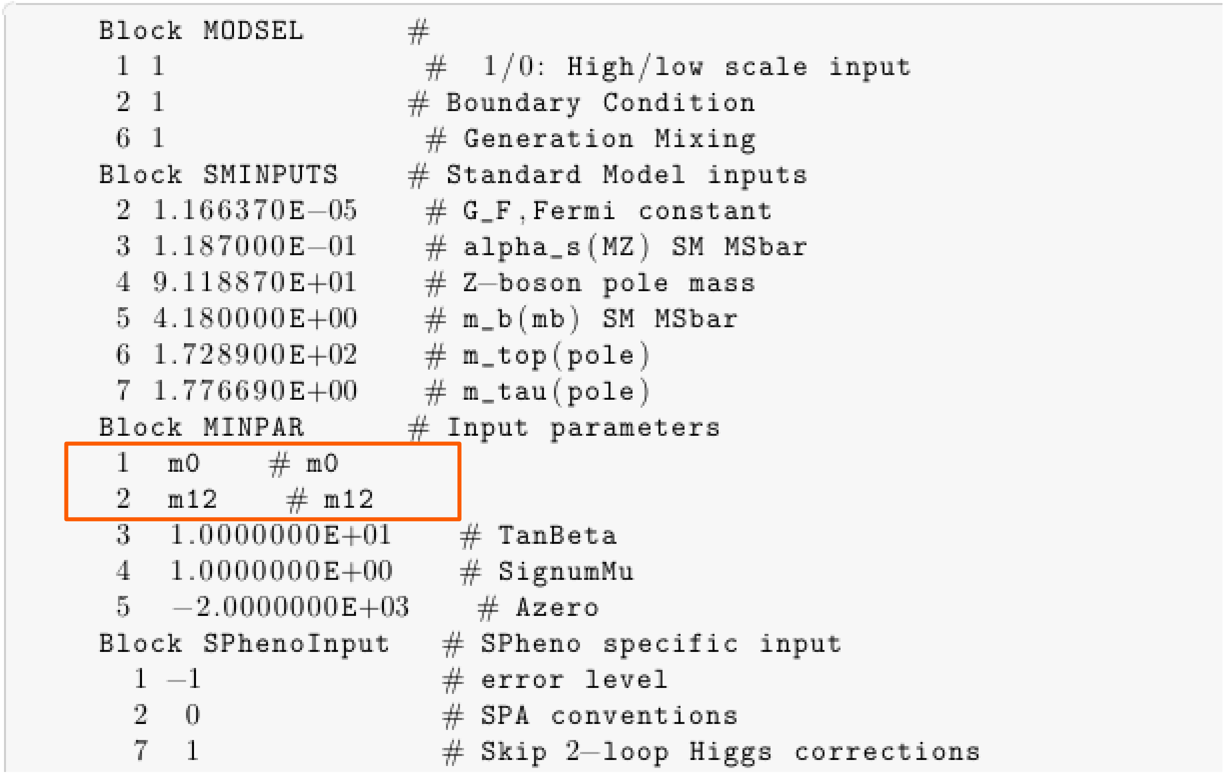
Very fast to setup simple scan for a new model with minimal editing of files

The prepareModel.py script will setup a template file to make this simpler ...

Variables can be specified directly in the template file as strings, along with constant values

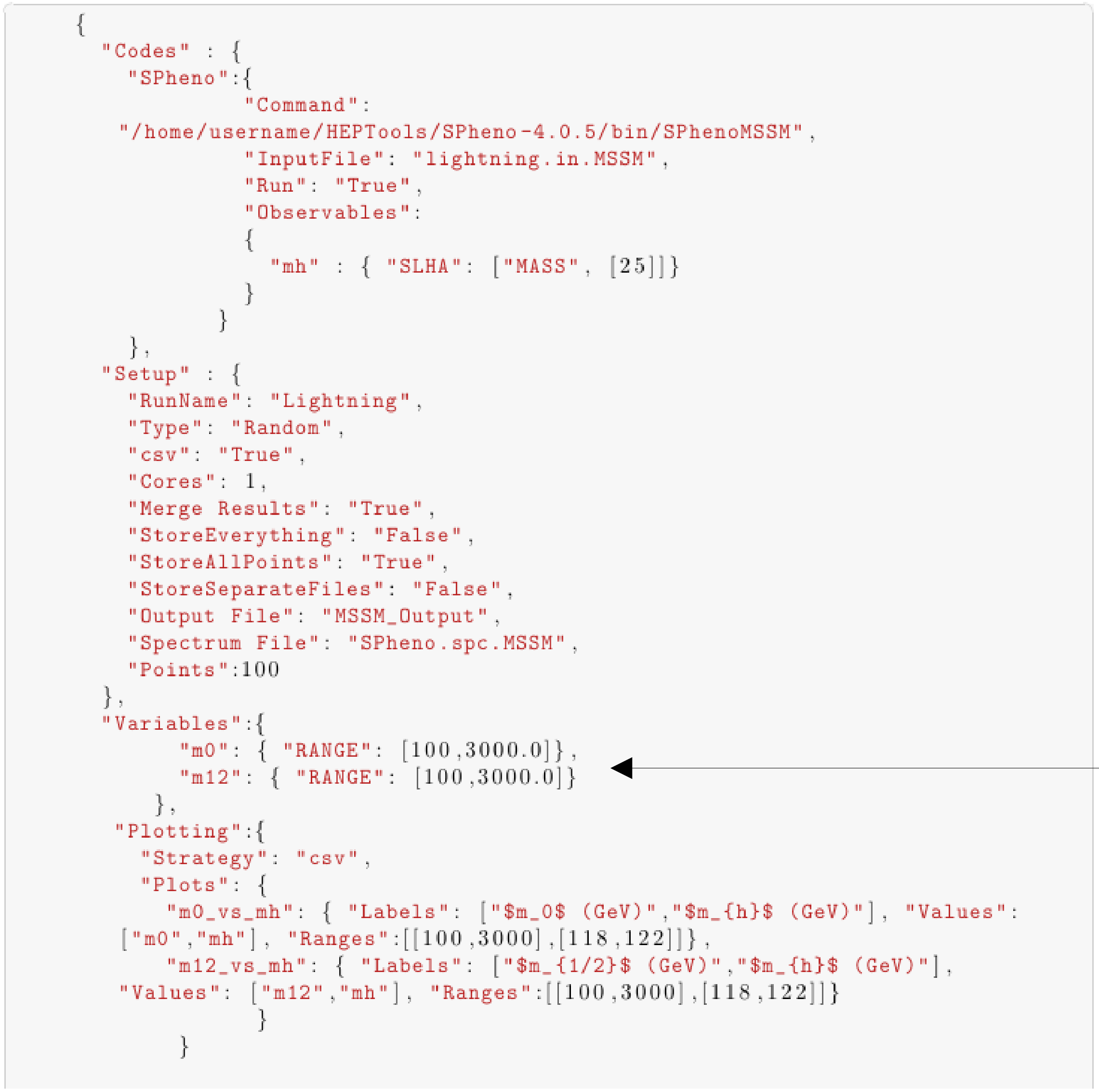
Can even use formulas here!

lightning.in.MSSM



```
Block MODSEL        #
 1 1                 #  1/0: High/low scale input
 2 1                # Boundary Condition
 6 1                 # Generation Mixing
Block SMINPUTS      # Standard Model inputs
 2 1.166370E-05      # G_F,Fermi constant
 3 1.187000E-01      # alpha_s(MZ) SM MSbar
 4 9.118870E+01      # Z-boson pole mass
 5 4.180000E+00      # m_b(mb) SM MSbar
 6 1.728900E+02      # m_top(pole)
 7 1.776690E+00      # m_tau(pole)
Block MINPAR        # Input parameters
 1  m0     # m0
 2  m12     # m12
 3  1.0000000E+01     # TanBeta
 4  1.0000000E+00     # SignumMu
 5  -2.0000000E+03     # Azero
Block SPhenoInput    # SPheno specific input
  1 -1               # error level
  2  0               # SPA conventions
  7  1               # Skip 2-loop Higgs corrections
```

Example lightning scan



```
{
  "Codes" : {
    "SPheno":{
              "Command":
   "/home/username/HEPTools/SPheno-4.0.5/bin/SPhenoMSSM",
              "InputFile": "lightning.in.MSSM",
              "Run": "True",
              "Observables":
              {
                "mh" : { "SLHA": ["MASS", [25]]}
              }
            }
    },
  "Setup" : {
    "RunName": "Lightning",
    "Type": "Random",
    "csv": "True",
    "Cores": 1,
    "Merge Results": "True",
    "StoreEverything": "False",
    "StoreAllPoints": "True",
    "StoreSeparateFiles": "False",
    "Output File": "MSSM_Output",
    "Spectrum File": "SPheno.spc.MSSM",
    "Points":100
  },
  "Variables":{
        "m0": { "RANGE": [100,3000.0]},
        "m12": { "RANGE": [100,3000.0]}
      },
   "Plotting":{
     "Strategy": "csv",
     "Plots": {
       "m0_vs_mh": { "Labels": ["$m_0$ (GeV)","$m_{h}$ (GeV)"], "Values":
   ["m0","mh"], "Ranges":[[100,3000],[118,122]]},
       "m12_vs_mh": { "Labels": ["$m_{1/2}$ (GeV)","$m_{h}$ (GeV)"],
   "Values": ["m12","mh"], "Ranges":[[100,3000],[118,122]]}
             }
       }
```

These correspond to variables in the template file and stored in the outputs

Example for using HiggsTools

```
     "Codes":{
       "HiggsTools":{
                   "HiggsBounds Dataset": "/home/username/BSMArt/hbdataset-master",
                   "HiggsSignals Dataset": "/home/username/BSMArt/hsdataset-main",
                   "Neutral Higgs": [25, 35, 36],
                   "Charged Higgs": [37],
                   "Observables":{
                                  "HBresult" : {"SLHA": ["HIGGSTOOLS",[1]],"SCALING":"OFF"},
                                  "HBobsr" : {"SLHA": ["HIGGSTOOLS",[2]],
         "SCALING":"UPPER","MEAN":1.0,"VARIANCE":0.1},
                                  "HSchi2" : {"SLHA": ["HIGGSTOOLS",[11]],"SCALING":"OFF"},
                                  "HSnobs" : {"SLHA": ["HIGGSTOOLS",[12]],"SCALING":"OFF"},
                                  "HSpval" : {"SLHA": ["HIGGSTOOLS",[13]],
         "SCALING":"LOWER","MEAN":0.001,"VARIANCE":0.1}
                                 },

                   "Run": "True"
                }
     }
```

Creating your own scan

Can have as much or as little control over running of codes as desired

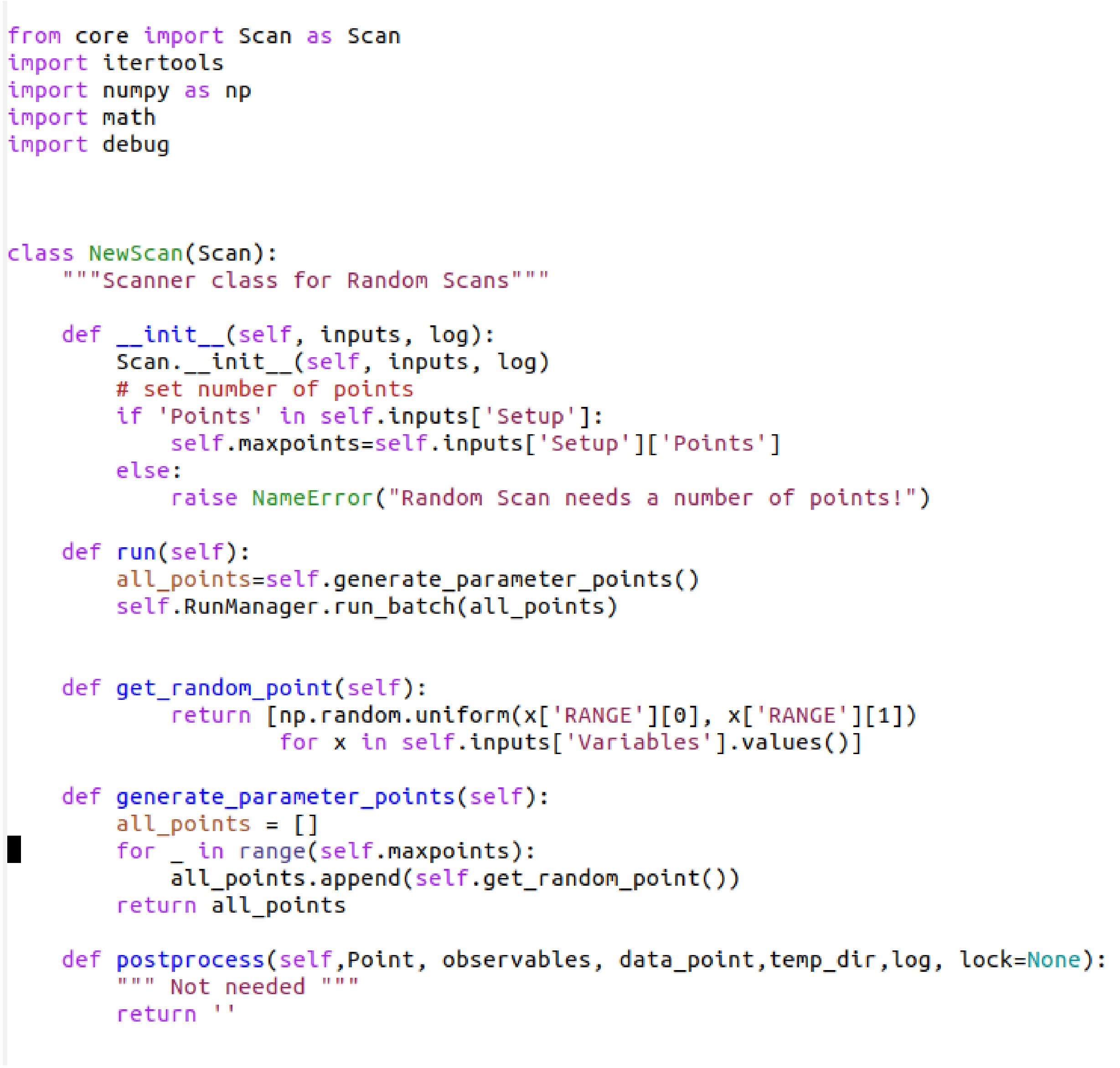
E.g. can call the running of points (treated as a list of floats) via

```
self.RunManager.run_batch(points)
```

Will automatically spread over number of cores specified in the json file

The scan is a NewScan class inherited from Scan, and you only need to specify the Run method (although you can overload __init__ and PostProcess)

E.g. this is the random scan:



```
from core import Scan as Scan
import itertools
import numpy as np
import math
import debug



class NewScan(Scan):
    """Scanner class for Random Scans"""

    def __init__(self, inputs, log):
        Scan.__init__(self, inputs, log)
        # set number of points
        if 'Points' in self.inputs['Setup']:
            self.maxpoints=self.inputs['Setup']['Points']
        else:
            raise NameError("Random Scan needs a number of points!")

    def run(self):
        all_points=self.generate_parameter_points()
        self.RunManager.run_batch(all_points)


    def get_random_point(self):
            return [np.random.uniform(x['RANGE'][0], x['RANGE'][1])
                    for x in self.inputs['Variables'].values()]

    def generate_parameter_points(self):
        all_points = []
        for _ in range(self.maxpoints):
            all_points.append(self.get_random_point())
        return all_points

    def postprocess(self,Point, observables, data_point,temp_dir,log, lock=None):
        """ Not needed """
        return ''
```