SuSpect 3: status and roadmap

A.Djouadi, J-L.Kneur, G.Moultaka, M.Ughetto, D.Zerwas

LPT (Orsay), L2C (Montpellier), CPPM (Marseille), LAL (Orsay)

March 28, 2012

Centre de Physique des Particules de Marseille CPPM

LABORATOIRE CHARLES COULOMB MONTPELLIER

SuSpect 2 & Co.

SuSpect 2

- SuSy spectrum calculator
- Former authors: A.Djouadi, J-L.Kneur and G.Moultaka
- Fortran code
- MSSM spectrum, supports “mSUGRA”, AMSB and GMSB (custom versions exists for heavy scalars, no-scale, right-handed sneutrinos, . . .)

SuSpect 3

- SuSpect 2 has reached his limits in terms of flexibility
- C++ code, OOP conception
- ROOT output option, SLHA IO support
- Flexible structure through polymorphism use, inheritance properties and interfaces
- Started one year ago

Standard Case: *MSSM

MSSM = SM-gauged susy invariant theory with minimal field content

- 105 parameters
- 22 parameters when:
  - SuSy breaking terms are real
  - Trilinear couplings are diagonal
  - Differences between first and second families are negligibles
- 5 parameters in "mSUGRA": m_0, $m_{1/2}$, A_0, $\tan\beta$ and $\text{sgn}(\mu)$

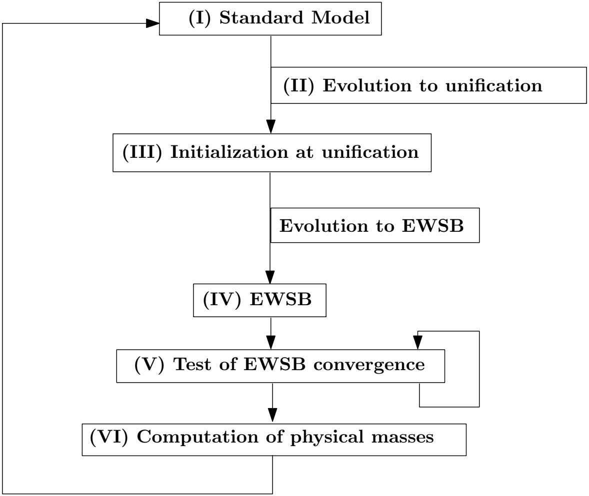
Let's review the spectrum calculation in this specific case

Ingredients

In the "mSUGRA" case:

- A way to run parameters through energy scales: RGEs
- Boundary conditions for these (very non-linear) ODEs
  - MZ scale: SM inputs
  - GUT scale: assumptions on soft breaking terms, ie universality in "mSUGRA" case
  - EWSB scale: minimization equations for the scalar potential, tadpole contributions
- Mass matrices, radiative corrections

Typical Algorithm



Typical algorithm

Step 1: Low energy input

$\alpha(M_Z)$,$\alpha_S(M_Z)$, $M_t^{\rm pole}$, $M_\tau^{\rm pole}$,$m_b^{\overline{\rm MS}}(m_b)$, $M_Z^{\rm pole}$,etc.
Translation to $\overline{\rm DR}$

Step 2: One– or two–loop RGEs running

RGEs with choice: $g_1 = g_2 \cdot \sqrt{3/5}$
$M_{\rm GUT} \sim 2 \cdot 10^{16}$ GeV

Step 3: Choice of SUSY-breaking model

mSUGRA, GMSB, AMSB, or pMSSM. Choice of high-energy input, eg:
mSUGRA: m_0, $m_{1/2}$, A_0, sign(μ) and $\tan\beta$

Step 4: EWSB

Run down all parameters to m_Z and $M_{\rm EWSB}$ scales
Calculate μ^2, $\mu B = F(m_{H_u}, m_{H_d}, \tan\beta, V_{\rm loop})$

Step 5: Testing EWSB

Check of consistent EWSB (μ convergence, no tachyons, simple CCB/UFB, etc.)

Step 6: Masses and corrections

Diagonalization of mass matrices and calculation of masses/couplings
Radiative corrections to the physical Higgs, sfermions, gauginos masses

Code Example

main.cxx

- `SUSPECT::suspect aSuspectCalculation;`
- `aSuspectCalculation.Initialize(SLHAstructure);`
  - Read inSLHAfile and fill a SLHA object
  - Initialize the model according to MODSEL
    - `m_model = new SUSPECT::ModelmSUGRA(m_SLHAblock);`
    - `m_model = new SUSPECT::ModelLowScaleMSSM(m_SLHAblock);`
    - `m_model = new SUSPECT::ModelGMSB(m_SLHAblock);`
    - `m_model = new SUSPECT::ModelmAMSB(m_SLHAblock);`
    - ...
- `aSuspectCalculation.Execute();`
  - `m_model->Execute();`
    - `m_DRparam.Execute();`
    - `m_RGErunner.Initialize(log(m_scaleMZ),log(m_s...`
    - `m_RGErunner.Execute();`
    - ...
    - `FinalizeMasses(m_scaleEWSB);`
- `aSuspectCalculation.Finalize(verbose,outSLHAfile);`

SLHA

The common data storage structure

The Model Structure

ModelBase

public:

- `virtual void Initialize();`
- `virtual void Execute();`
- and so on for low energy inputs, boundary conditions, rad.corr., EWSB conditions,. . .

Model3Scales

public:

- `virtual void Initialize();` ⇒ Preparing SLHA blocks at GUT, EWSB and MZ scales
- `virtual void Execute();` ⇒ Main loop implementation (between 3 scales)
- `virtual void ApplyBoundaryConditions();` ⇒ Dumb Boundary Conditions for security

protected:

- `double m_scaleMZ;`
- `double m_scaleEWSB;`
- `double m_scaleGUT;`

⇒ Storage of the 3 scales of interests for 3 scale scenarios (mSUGRA, High scale pMSSM, AMSB, . . .)

ModelmSUGRA

public:

- `void Initialize();` ⇒ 3 Scales Initialization
- `void ApplyBoundaryConditions();` ⇒ Universality in mSUGRA case

Right now. . .

SuSpect 3 supports:

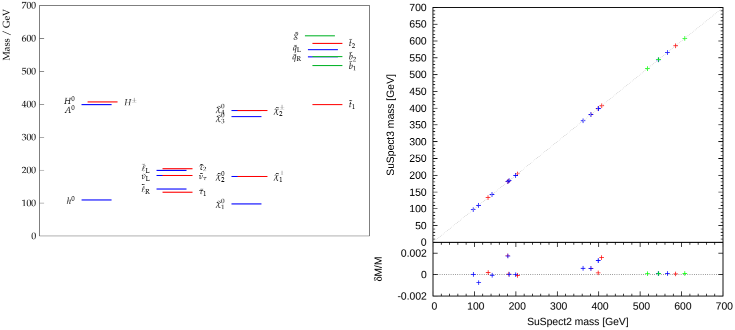
- mSUGRA (3 scales)
- AMSB (3 scales)
- GMSB (4 scales)

} Implemented but need more tests (AMSB, GMSB)

- Low-Scale pMSSM (2 scales) (no running to GUT, boundary conditions given at EWSB)
- Bottom-up pMSSM (3 scales) (Running to GUT, boundary conditions given at EWSB)
- High-Scale MSSM (3 scales) (Non-universal boundary conditions at GUT scale)
- Compressed-SuSy (3 scales) (example of non-universal gauginos soft-breaking terms)

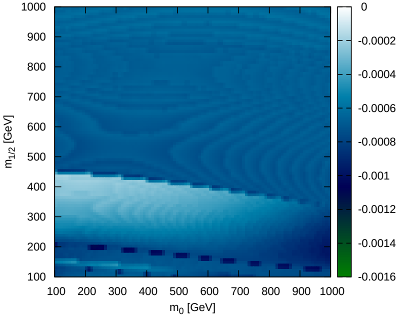
SuSpect 2 used as a validator, and things are going pretty well. . .

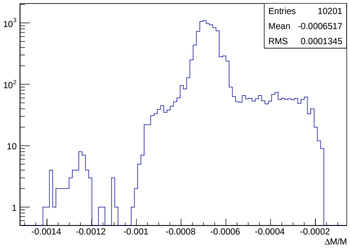
SPS1A



$$m_0 = 100,\ m_{\text{half}} = 250,\ A_0 = -100,\ \tan\beta = 10 \text{ and } \mu > 0$$

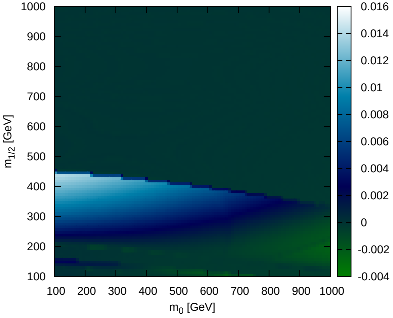
Around SPS1A

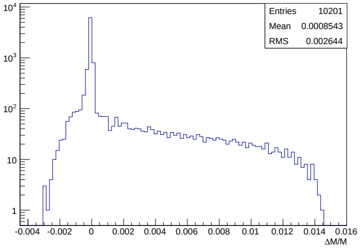




Relative differences between SuSpect 2 and SuSpect 3 for the light higgs mass. Left plot is this difference on the m_0/m_{half} plane. Right plots is her distribution.

Around SPS1A





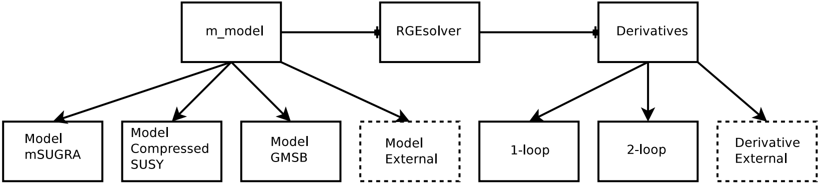
Relative differences between SuSpect 2 and SuSpect 3 for the lightest stop mass. Left plot is this difference on the m_0/m_{half} plane. Right plots is her distribution.

1. New RGEs
   - Full-MSSM (FV, RPV but CPC)
2. New boundary conditions
   - true-mSUGRA
   - No-scale type
   - Yukawa unification
3. New EWSB algorithm
   - NMSSM
   - No-scale

On a longer timescale:

- CPV
- higher/multi thresholds effects in RG running
- New particles: Dirac neutralinos, N=1, N=2 hybride,...

Principles



SuSpect 3 can pick up an external definition of the following elements:

- Model (Initialization, Boundary conditions, . . .)
- RGEs
- Particle content (eigenstates, +RC)
- EWSB (way to know if EWSB is realized and consistent)

Feynrules and SARAH as spectrum generator generators

The interfaces allows automated generation of some brick of the code.

There is currently two projects:

- Feynrules ⇒ SuSpect3
- SARAH ⇒ SuSpect3

The first goal is automated RGEs computation

SARAH2SuSpect3 command

After proper SARAH initialization (`Start[‘‘MSSM’’]` for this example):

```
RGEsarah2suspect    [complex-> yes OR no,
                    {"diagonalYukawa"} OR {"diagonalmsoft"} OR {"diagonalYukawa", "diagonalmsoft"} OR {}
                    looplevel-> 1 OR 2,
                    excludegenerations-> {} OR {generation number to exclude},
                    BetaGauge, BetaYijk, BetaMi, ...  OR {BetaGauge, i}, ...,
                    "filename"] ;
```

So for example, generation of gauge/Yukawa subset of RGEs:

```
RGEsarah2suspect[complex->no, {"diagonalYukawa"}, looplevel-> 1, excludegenerations-> {},

BetaGauge, BetaYijk, "all-gauge-Yukawa-Ydiag.cpp"];
```

Example of SARAH generated RGEs

```
if (RGEon[RGE::g1])
    dydx[RGE::g1] = (33 * m_cpi * pow(yVector[RGE::g1], 3)) / 5.;
if (RGEon[RGE::g2])
    dydx[RGE::g2] = m_cpi * pow(yVector[RGE::g2], 3);
if (RGEon[RGE::g3])
    dydx[RGE::g3] = -3 * m_cpi * pow(yVector[RGE::g3], 3);
if (RGEon[RGE::Yu11])
    dydx[RGE::Yu11] =
        m_cpi * (pow(yVector[RGE::Yd11],
            2) * yVector[RGE::Yu11] + 3 * pow(yVector[RGE::Yu11],
            3) - (yVector[RGE::Yu11] * (13 * pow(yVector[RGE::g1],
                    2) + 45 * pow(yVector[RGE::g2],
                    2) + 80 * pow(yVector[RGE::g3],
                    2) - 45 * (pow(yVector[RGE::Yt],
                        2) + pow(yVector[RGE::Yu11],
                        2) + pow(yVector[RGE::Yu22], 2)))) / 15.);
if (RGEon[RGE::Yu22])
    dydx[RGE::Yu22] =
        m_cpi * (pow(yVector[RGE::Yd22],
            2) * yVector[RGE::Yu22] + 3 * pow(yVector[RGE::Yu22],
            3) - (yVector[RGE::Yu22] * (13 * pow(yVector[RGE::g1],
                    2) + 45 * pow(yVector[RGE::g2],
                    2) + 80 * pow(yVector[RGE::g3],
                    2) - 45 * (pow(yVector[RGE::Yt],
                        2) + pow(yVector[RGE::Yu11],
                        2) + pow(yVector[RGE::Yu22], 2)))) / 15.);
if (RGEon[RGE::Yt])
    dydx[RGE::Yt] =
        m_cpi * (pow(yVector[RGE::Yb],
            2) * yVector[RGE::Yt] + 3 * pow(yVector[RGE::Yt],
            3) - (yVector[RGE::Yt] * (13 * pow(yVector[RGE::g1],
                    2) + 45 * pow(yVector[RGE::g2],
                    2) + 80 * pow(yVector[RGE::g3],
                    2) - 45 * (pow(yVector[RGE::Yt],
                        2) + pow(yVector[RGE::Yu11],
                        2) + pow(yVector[RGE::Yu22], 2)))) / 15.);
if (RGEon[RGE::Yd11])
    dydx[RGE::Yd11] =
        m_cpi * (3 * pow(yVector[RGE::Yd11],
            3) - (yVector[RGE::Yd11] * (7 * pow(yVector[RGE::g1],
                    2) + 45 * pow(yVector[RGE::g2],
                    2) + 80 * pow(yVector[RGE::g3],
```

Tests have been performed and it works perfectly (for 1–loop and 2–loop)

A more elaborated case: EWSB

Usually, one uses minimization conditions of the scalar potential to evaluate μ at EWSB scale.
But:

- very specific to MSSM
- recursive algorithm
- the equations are constrained along the “minimization” process by an *a priori* knowledge of M_Z and gauge couplings

Interfaces allow:

- customization of the scalar potential
- customization of the minimization algorithm

For example, we are currently studying the possiblity to use the Newton-Raphson method to improve minimization process. In principle, it only needs the jacobian of the minimization's system of equations.

Conclusions

- Finish validation of "S2 supported" models
- mSUGRA starting to become robust, GMSB, AMSB, general MSSM to be done
- Alpha release after robustness tests
- On the FeynRules side, we're working with Adam and Benjamin on the automated production of RGEs
- This work on automatization is a perfect exercise to see where are the actual flaws in flexibility
- On a longer timescale we would like to work on EWSB realization