A new software to compute MSSM squared amplitudes for particle physics and relic density calculations

Marco Palmiotto

Université Claude Bernard Lyon 1, France
Institut de Physique des 2 infinis

28/07/2022

Motivations

- Extension of the features of the software `SuperIso Relic`

Motivations

- Extension of the features of the software `SuperIso Relic`
- The first goal is improving the relic density calculation:

Motivations

- Extension of the features of the software `SuperIso Relic`
- The first goal is improving the relic density calculation:

Now

- providing the **total** density of the BSM particles, in freeze-out scenarios
- only MSSM and NMSSM

Motivations

- Extension of the features of the software `SuperIso Relic`
- The first goal is improving the relic density calculation:

Now

- providing the **total** density of the BSM particles, in freeze-out scenarios
- only MSSM and NMSSM

After

- following the evolution of the density of particles different from the LSP in the MSSM for freeze-out scenarios
- allowing models with **multiple stable DM particles**
- allowing **freeze-in** scenarios
- allowing **user-defined models** from the Lagrangian

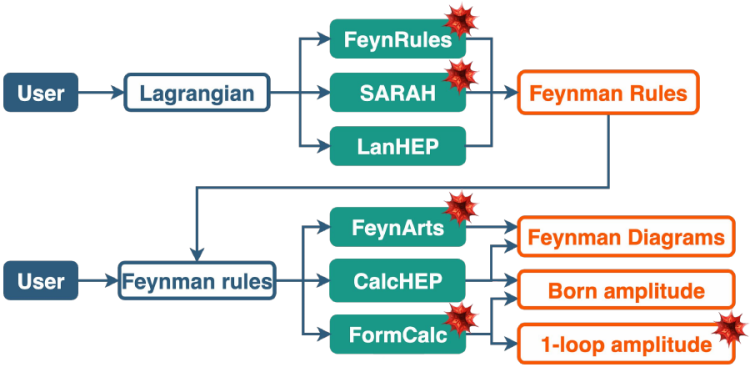
The current goal: relic density

- Follow the evolution of the densities of **more than one particle**
  - Better explore the **parameter space** of each viable model
  - Compute relic densities for **more DM candidates**

The current goal: relic density

- Follow the evolution of the densities of **more than one particle**
  - Better explore the **parameter space** of each viable model
  - Compute relic densities for **more DM candidates**
- New setting to compute $\langle \sigma v \rangle$ and W_{eff}
- The current setup relies on self-generated `FormCalc` code,
  - Only for MSSM and NMSSM
  - Hard to separate different contributions to $\langle \sigma v \rangle$ and W_{eff}

Why these limitations?



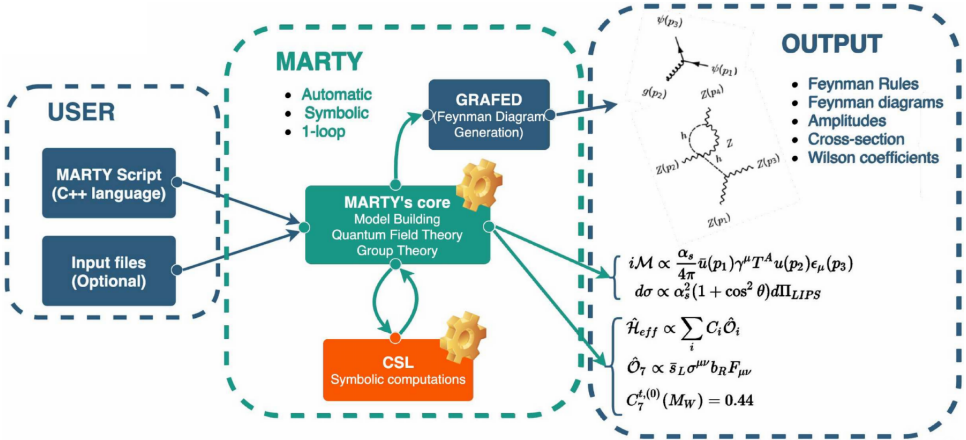
In `SuperIso Relic v4`:

- Many codes are required
- Several passages of input
- Mathematica dependencies

MARTY

website: `https://marty.in2p3.fr`

manual: 2011.02478



MARTY

With MARTY the user can

- Write a Lagrangian symbolically in a C++ source file
  - By defining the **gauge symmetries** of the model

MARTY

With MARTY the user can

- Write a Lagrangian symbolically in a C++ source file
  - By defining the **gauge symmetries** of the model
  - By defining the **fields** of the model

MARTY

With MARTY the user can

- Write a Lagrangian symbolically in a C++ source file
  - By defining the **gauge symmetries** of the model
  - By defining the **fields** of the model
  - By adding **potential** terms

MARTY

With MARTY the user can

- Write a Lagrangian symbolically in a C++ source file
  - By defining the **gauge symmetries** of the model
  - By defining the **fields** of the model
  - By adding **potential** terms
  - By performing **SSB** if that's in the model

MARTY

With MARTY the user can

- Write a Lagrangian symbolically in a C++ source file
  - By defining the **gauge symmetries** of the model
  - By defining the **fields** of the model
  - By adding **potential** terms
  - By performing **SSB** if that's in the model
- **Symbolically** get quantities such as
  - Sum of the squared amplitudes
  - Wilson coefficients
  - Feyman diagrams
- Output those results in a **numerical** C++ library

Our improvements

Making easy the passage

$$\text{Lagrangian} \rightarrow \text{numerical library}$$

Our improvements

Making easy the passage

$$\text{Lagrangian} \rightarrow \text{numerical library}$$

Wrapping the numerical library in an interactive structure

Our improvements

Making easy the passage

$$\text{Lagrangian} \rightarrow \text{numerical library}$$

Wrapping the numerical library in an interactive structure
$\rightarrow$ this will make also the library easier to link with other software

Our improvements

Making easy the passage

$$\text{Lagrangian} \rightarrow \text{numerical library}$$

Wrapping the numerical library in an interactive structure
$\rightarrow$ this will make also the library easier to link with other software

And soon improving the passage

$$\text{numerical library} \rightarrow \text{SuperIso}$$

The content of the package

The package can be downloaded at
`https://gitlab.in2p3.fr/marco.palmiotto/mssm-public.git`

It contains:

- A file `MSSM.cpp` containing the code that uses `MARTY` to generate a numerical library
- The auxiliary files we wrote to add functionalities to `MARTY`'s self-generate libraries
- Files with examples of programs that the user can write
- Some setup scripts

You need to have `MARTY` installed, and define the environmental variable `INSTALLMARTYPATH` as the path where it is built

The setup of the package

- To configure and automatically compile the `mssm2to2` library, run

  ```
  ./lib_setup.sh -nomake
  cd mssm2to2
  make
  ```

  This is automatic by executing `./lib_setup.sh` (with no flags)
- To generate the library with the numerical functions present in the `MSSM.cpp` file,

  ```
  ./lib_generate.sh
  ```
- The example files for the executables can be found in the `mssm2to2/script` directory
- The executables are generated in the directory `mssm2to2/bin`

Example: giving the inputs

Let us show how to read input from a SLHA file:

```
struct Param_t input;
int err;
ReadLHA(input, "example.lha", &err);
if(err != 0) return err;
input.Print();
```

Example: definition of a process

Let us show how to define $N_1, N_1 \to Z, Z$:

```
vector<Insertion>v={corr::N_1, corr::N_1, corr::Z, corr::Z};
Process2to2 proc(v);
if(!proc.checkExistance()){
  cerr « "Warning! The process "  «
   proc.getName() « " is not present in the library!\n";
  return 1;
}
string proc_name = proc.getName();
cout « "We created the process " « proc_name « endl;
```

Example: some calculations - 1

Let us show how to compute quantities:

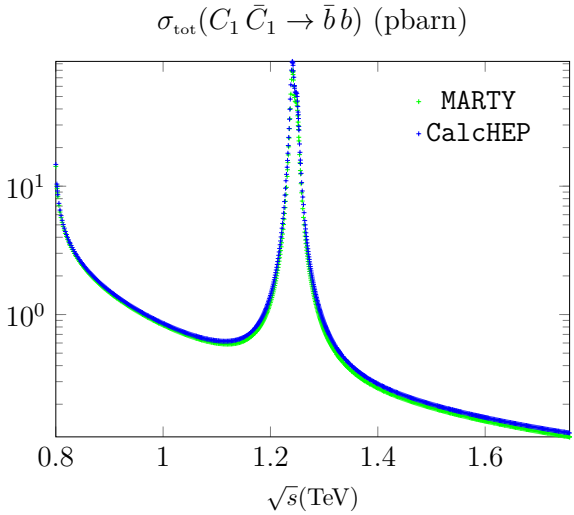
```
double sqrts = 3000.;
double ctheta = 0.5;
double degrees_of_freedom = proc.getDof();
double squared_amplitude = proc.getSumSquaredAmpl(input, sqrts, ctheta);
double weff_contrib = proc.getDiffWeffContrib(input, sqrts, ctheta);
double diff_xsec = proc.getDiffCrossSection(input, sqrts, ctheta);
double total_xsec = proc.getTotalCrossSection(input, sqrts);
```

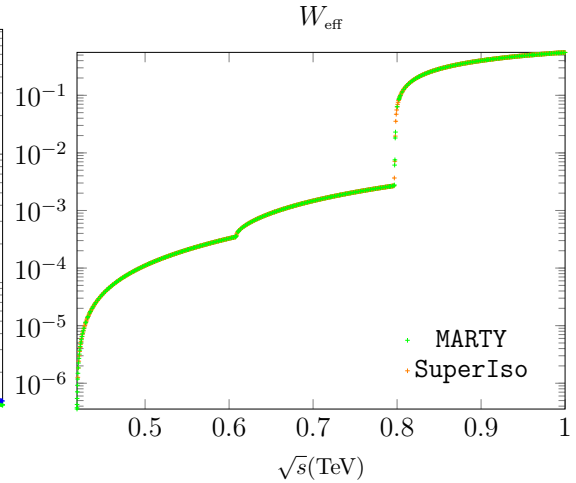
Example: some calculations - 2

Let us show how to compute the total W_{eff}:

```
SetOfProc allprocsptr(input);
double dweff = allprocsptr.getdWeff_dcos(input, ctheta);
double weff = allprocsptr.getWeff(sqrts);
```

Some output


$\sigma_{\rm tot}(C_1\,\bar{C}_1 \to \bar{b}\,b)$ (pbarn)
MARTY
CalcHEP
10^1
10^0
0.8
1
1.2
1.4
1.6
$\sqrt{s}$(TeV)


$W_{\rm eff}$
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
MARTY
SuperIso
0.5
0.6
0.7
0.8
0.9
1
$\sqrt{s}$(TeV)

Future goals

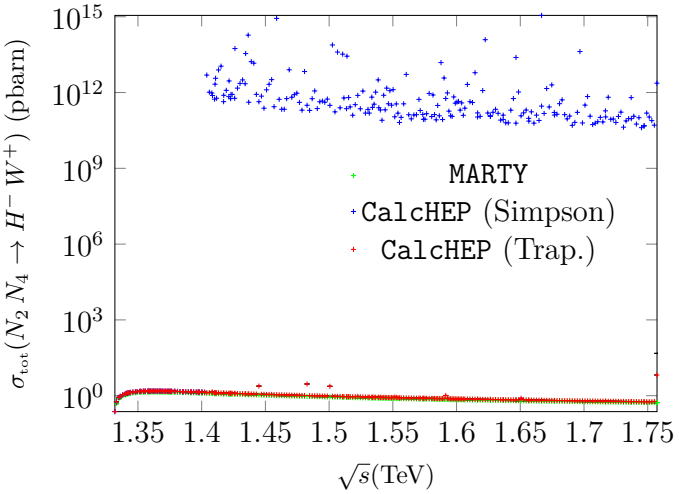
- Improving performance
- Integration within `SuperIso`
- Solving multiple coupled Boltzmann equations
- Ideas to improve `MARTY` and have more portability and integration
- Adding more pre-defined models
- Creating a more general interface for treating user-defined BSM models
- Upgrading `SuperIso` to study freeze-in
- Improving direct and indirect DM detection in `SuperIso`
- Adding the NMSSM

Conclusions

- A new way of dealing with 2 to 2 sum of the squared amplitudes is provided in the MSSM at the LO
- It is possible to use the ideas behind our algorithms to generalise the features of `MARTY`'s numerical libraries
- This package provides a library easy to use and to integrate with other software
- Calculations are on average faster than other software we tested
- We validated our results with other software

Thanks for your attention!

Simpson rule vs trapezoidal rule pt. 1



Simpson rule vs trapezoidal rule pt. 2