# Darkpack: a modular software to compute BSM squared amplitudes for particle physics and relic density calculations

Marco Palmiotto

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

04/05/2023

# Motivations

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

# Motivations

- Extension of the features of the software `SuperIso Relic`
- It is a software that tests the parameter space of some BSM models, providing observables related to **flavour** and **dark matter detection and density**

# Motivations

- Extension of the features of the software `SuperIso Relic`
- It is a software that tests the parameter space of some BSM models, providing observables related to **flavour** and **dark matter detection and density**
- The first goal is improving the relic **density** calculation:

# Motivations

- Extension of the features of the software `SuperIso Relic`
- It is a software that tests the parameter space of some BSM models, providing observables related to **flavour** and **dark matter detection and density**
- The first goal is improving the relic **density** calculation:

**`SuperIso Relic`**

  - 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`
- It is a software that tests the parameter space of some BSM models, providing observables related to **flavour** and **dark matter detection and density**
- The first goal is improving the relic **density** calculation:

**`SuperIso Relic`**

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

**`Darkpack`**

- 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

# Our improvements

Making easy the passage

Lagrangian $\rightarrow$ numerical library

# Our improvements

Making easy the passage

Lagrangian → 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

Lagrangian $\rightarrow$ numerical library

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

numerical library $\rightarrow$ Relic density, DM observables, other software

Our goal is taking care of

- Minimising the passages the user has to do from writing the Lagrangian to getting an easy to use numerical library

# 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

$$\text{numerical library} \rightarrow \text{Relic density, DM observables, other software}$$

Our goal is taking care of

- Minimising the passages the user has to do from writing the Lagrangian to getting an easy to use numerical library
- Providing an interactive structure to the numerical library that helps the linking with other software

# Our improvements

Making easy the passage

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

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

$$\text{numerical library} \rightarrow \text{Relic density, DM observables, other software}$$

Our goal is taking care of

- Minimising the passages the user has to do from writing the Lagrangian to getting an easy to use numerical library
- Providing an interactive structure to the numerical library that helps the linking with other software
- Minimising the passages from the generated libary to the ready to use linking with external tools

# Creation of the model file

A MARTY model file has the following structure

- The **model-independent headers**: i.e. all the necessary libraries to be included, both from the C++ standard ones, the MARTY ones and the custom functions we provide
- The **model-dependent headers**: where you can define custom types and macros, according to your needs
- The definition of the model: it can be done in the previous part (as e.g. if you want to do it via inheritance from some model already in the library) or at the beginning of the main
- The coherent (re)definition of **particles' names** and their masses
- The calculation of some particle-related quantities (including **widths**)
- The definition of the **process list**
- The call of `computeAndAddToLib` function
- The writing of the library on disk

# Library preparation

You need to write your own programs, but you may want to define some custom functions first, as for instance to handle the input.
For how to fo that you can look at the files in `auxiliary_library/mssm2to2` or `auxiliary_library/scalar2to2`

# 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();
```

Note that

- you can give the inputs setting by hand the elements of the `Param_t` variable
- For a custom model, you need to adjust the read function

# 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 diff_xsec = proc.getDiffCrossSection(input, sqrts, ctheta);
double total_xsec = proc.getTotalCrossSection(input, sqrts);
```

# Example: some calculations - 2

$$\langle \sigma v \rangle \propto \frac{\int W_{\text{eff}}(\sqrt{s}) f(s)\, \mathrm{d}s}{g(s)}$$

Let us show how to compute the $g^2_{\text{LSP}} W_{\text{eff}}$ and its contributions:

```
AvgSvCalculator allprocsptr(input);
double T = input.getLightestBSMmass()/20.;
double dweff = allprocsptr.getdWeff_dcos(sqrts, ctheta);
double weff = allprocsptr.getWeff(sqrts);
double avgsv = allprocsptr.getAverageSigmav(T);
```

# Example: some output

Let us show a part of the output of `example_1_single_process.cpp`:

```
We created the process N_1 N_1 -> Z Z and we calculated
 - Symmetry factor = 8.00000e+00
 - Sum|M|^2 (sqrts = 3.00000e+03, ctheta = 5.00000e-01)
     = 2.09380e-03
 - contribution to g^2 dWeff/dcos(theta) (sqrts = 3.00000e+03,
     ctheta = 5.00000e-01) = 5.22483e-04
 - dsigma/dcos(theta) (sqrts = 3.00000e+03,
     ctheta = 5.00000e-01) = 1.13540e-04 pbarn
 - sigma_tot (sqrts = 3.00000e+03) = 3.56877e-04 pbarn
```