

# MiMeS: Misalignment Mechanism Solver

Dimitrios Karamitros

Manchester U.

24/11/2021

*Computational Tools for High Energy Physics and Cosmology*

*Institut de Physique des 2 Infinis Lyon, France*

Detailed documentation: **arXiv:2110.12253**

Github: **github.com/dkaramit/MiMeS**

HEPForge: **mimes.hepforge.org**

## 1 Calculating the Relic Abundance

- The axion EOM
- How hard can it be?

## 2 MiMeS

- MiMeS
- MiMeS under the hood
- How to get MiMeS
- Configure (and make)
- Classes
- Template arguments
- MiMeS from python
- Assumptions
- What MiMeS expects from you

## 3 Examples

- python
- C++

## 4 Outlook

# Calculating the Relic Abundance

1

## Calculating the Relic Abundance

- The axion EOM
- How hard can it be?

2

## MiMeS

- MiMeS
- MiMeS under the hood
- How to get MiMeS
- Configure (and make)
- Classes
- Template arguments
- MiMeS from python
- Assumptions
- What MiMeS expects from you

3

## Examples

- python
- C++

4

## Outlook

Axions and ALPs follow a similar equation of motion (EOM):

$$\left( \frac{d^2}{dt^2} + 3H(t) \frac{d}{dt} \right) \theta(t) + \tilde{m}_a^2(t) \sin \theta(t) = 0 ,$$

where  $\theta = A f_a$ , with  $A$  the axion field, and  $f_a$  some energy scale that characterises the potential (Peccei-Quinn breaking scale).

# How hard can it be?

**Hard** (in general).

---

<sup>1</sup>There is the adiabatic invariant,  $J$ , but it only helps us if the Hamiltonian varies slowly.

# How hard can it be?

**Hard** (in general).

The classical analogue is the damped pendulum with both frequency (length) and friction being time-dependent:

- There is no closed form solution.
- There are no constants of motion. <sup>1</sup>
- No package/library/program available!

---

<sup>1</sup>There is the adiabatic invariant,  $J$ , but it only helps us if the Hamiltonian varies slowly.

# How hard can it be?

**Hard** (in general).

The classical analogue is the damped pendulum with both frequency (length) and friction being time-dependent:

- There is no closed form solution.
- There are no constants of motion. <sup>1</sup>
- No package/library/program available!

MiMeS *simulates the evolution of the axion/ALP, for (virtually) any cosmological scenario and axion/ALP (thermal) mass.*

---

<sup>1</sup>There is the adiabatic invariant,  $J$ , but it only helps us if the Hamiltonian varies slowly.

## 1 Calculating the Relic Abundance

- The axion EOM
- How hard can it be?

## 2

### MiMeS

- MiMeS
- MiMeS under the hood
- How to get MiMeS
- Configure (and make)
- Classes
- Template arguments
- MiMeS from python
- Assumptions
- What MiMeS expects from you

## 3

### Examples

- python
- C++

## 4

### Outlook



We need accurate code that solves the EOM, but most importantly we need *reproducible* results!

First of all:

- MiMeS is a C++ header-only library that contains various templated classes; there is no "installation" and no special procedures, just include the header files.
- MiMeS comes with a `python` interface so that everybody can use it.
- MiMeS is distributed under the MIT license; you can do whatever you want with it, and I am *not* responsible.

MiMeS also

- Is *easy* to use; anyone can run it and see if their model can work or check against the literature.
- Is reasonably fast; less than 0.05 s for the scenarios tested.
- Provides full access to results and their errors, which can help determine if the results are accurate.
- Asks the user to decide when to start, stop, and when adiabaticity is reached; *no guesses anywhere!*

MiMeS relies on `NaBBODES`<sup>2</sup> for the numerical integration, and `SimpleSplines`<sup>3</sup> for the various interpolations.

## Advantages:

- You only need to have the standard C++ library.
- The two libraries are developed by myself, so their integration with MiMeS is seamless.
- There is always going to be a compatible version of these libraries that works with MiMeS.

## Disadvantages:

- These are not well tested libraries.
- No community of contributors; if it doesn't work, I have to fix it.
- Slow development.

---

<sup>2</sup> <https://github.com/dkaramit/NaBBODES>.

<sup>3</sup> <https://github.com/dkaramit/SimpleSplines>.

# How to get MiMeS

There are several ways you can get a stable version of MiMeS:

1

`git clone -b stable https://github.com/dkaramit/MiMeS.git`.  
This is the preferred way, as it is guaranteed to be the latest stable version.

2

Go to [mimes.hepforge.org/downloads](https://mimes.hepforge.org/downloads), and download it.

3

Go to [github.com/dkaramit/MiMeS/releases](https://github.com/dkaramit/MiMeS/releases), and download a released version.

You can get the most up-to-date code – not always the most stable one – including the latest version of NaBBODES and SimpleSplines, by running

---

```
1 git clone https://github.com/dkaramit/MiMeS.git
2 cd MiMeS
3 git submodule init
4 git submodule update --remote
```

---

# Configure (and make)

There is no need to install anything if you are going to use MiMeS in a C++ program. The only thing you *must* do is run

---

```
1 bash configure.sh
```

---

Alter that, you can include the header file `MiMeS/MiMeS.hpp`, and you are good to go.

However, you can also run

- `make lib`, in order to produce the (shared) libraries. This is needed in order to run the `python` interface.
- `make examples`, in order to compile the examples in `MiMeS/UserSpace/Cpp`.
- `make exec`, in order to produce some test executables (in `MiMeS/exec`). You just need to run them in order to see if you get any segfaults.

There are three classes useful to the user. <sup>4</sup>

---

<sup>4</sup> There are various arguments that need to be passed to the constructors, and they are all listed and explained in the Appendix of the documentation.

<sup>5</sup> K. Saikawa and S. Shirai, JCAP **08** (2020), 011 [arXiv:2005.03544 [hep-ph]].

<sup>6</sup> S. Borsanyi, Z. Fodor, J. Guenther, K. H. Kampert, S. D. Katz, T. Kawanai, T. G. Kovacs, S. W. Mages, A. Pasztor and F. Pittler, *et al.* Nature **539** (2016) no.7627, 69-71 [arXiv:1606.07494 [hep-lat]].

There are three classes useful to the user. <sup>4</sup>

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020 <sup>5</sup> data. The user can choose another file easily.

---

<sup>4</sup> There are various arguments that need to be passed to the constructors, and they are all listed and explained in the Appendix of the documentation.

<sup>5</sup> K. Saikawa and S. Shirai, JCAP **08** (2020), 011 [arXiv:2005.03544 [hep-ph]].

<sup>6</sup> S. Borsanyi, Z. Fodor, J. Guenther, K. H. Kampert, S. D. Katz, T. Kawanai, T. G. Kovacs, S. W. Mages, A. Pasztor and F. Pittler, *et al.* Nature **539** (2016) no.7627, 69-71 [arXiv:1606.07494 [hep-lat]].

There are three classes useful to the user. <sup>4</sup>

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020 <sup>5</sup> data. The user can choose another file easily.
- `mimes::AxionMass<LD>`: definition of axion/ALP mass as a function of the temperature and  $f_a$ . MiMeS is shipped with data from Lattice calculation <sup>6</sup> of the QCD axion mass.

---

<sup>4</sup> There are various arguments that need to be passed to the constructors, and they are all listed and explained in the Appendix of the documentation.

<sup>5</sup> K. Saikawa and S. Shirai, JCAP **08** (2020), 011 [arXiv:2005.03544 [hep-ph]].

<sup>6</sup> S. Borsanyi, Z. Fodor, J. Guenther, K. H. Kampert, S. D. Katz, T. Kawanai, T. G. Kovacs, S. W. Mages, A. Pasztor and F. Pittler, *et al.* Nature **539** (2016) no.7627, 69-71 [arXiv:1606.07494 [hep-lat]].

There are three classes useful to the user. <sup>4</sup>

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020 <sup>5</sup> data. The user can choose another file easily.
- `mimes::AxionMass<LD>`: definition of axion/ALP mass as a function of the temperature and  $f_a$ . MiMeS is shipped with data from Lattice calculation <sup>6</sup> of the QCD axion mass.
- `mimes::Axion<LD, Solver, Method>`: This is responsible for actually solving the EOM.

---

<sup>4</sup> There are various arguments that need to be passed to the constructors, and they are all listed and explained in the Appendix of the documentation.

<sup>5</sup> K. Saikawa and S. Shirai, JCAP **08** (2020), 011 [arXiv:2005.03544 [hep-ph]].

<sup>6</sup> S. Borsanyi, Z. Fodor, J. Guenther, K. H. Kampert, S. D. Katz, T. Kawanai, T. G. Kovacs, S. W. Mages, A. Pasztor and F. Pittler, *et al.* Nature **539** (2016) no.7627, 69-71 [arXiv:1606.07494 [hep-lat]].



# Template arguments

You need to choose what numeric type to use. This is done by the template argument `LD` which should be `double` (fast) or `long double` (accurate).<sup>7</sup>

You also need to tell `MiMeS` which integration strategy to use. This is done by choosing template arguments:

- `Solver` can be set to `1` for Rosenbrock (semi-implicit Runge-Kutta). The `Method` argument in this case can be:
  - `RODASPR2<LD>` (4th order).
  - `ROS34PW2<LD>` (3rd order).
  - `ROS3W<LD>` (2rd order, *very bad*).
- `Solver` can be set to `2` for explicit RK. The `Method` argument can be:
  - `DormandPrince<LD>` (7th order)
  - `CashKarpRK45<LD>` (5th order, *very bad*).
  - `RK45<LD>` (5th order, *very bad*).

---

<sup>7</sup> You could choose `float`, but we live in 2021.

In order to call the `python` interface of MiMeS, we need to first call `make lib` in the root directory of MiMeS.

Before that, we can take some time to decide what the template arguments and compilation options should be. In the file `MiMeS/Definitions.mk`, you can change the variables:

- `LONGpy=long` will compile the library with `long double` numeric types. `LONGpy=` will compile the library with `double` numeric types.
- `SOLVER` and `METHOD`, as in the template arguments.

Also, in the same file, you can change compilation options:

- **Compiler:**
  - `CC=g++` in order to use the GNU C++ compiler.
  - `CC=clang -lstdc++` in order to use the `clang` C++ compiler.
- **Optimization level:**
  - `OPT=O0`: No optimization.
  - `OPT=O1`, `O2`, or `O3`: all these perform mostly the same (read the compiler documentation for more information on the optimization).
  - `OPT=Ofast`: full optimization (fast, but dangerous).

# Assumptions

MiMeS is designed to make as few assumptions as possible.  
However, it still assumes that:

# Assumptions

MiMeS is designed to make as few assumptions as possible. However, it still assumes that:

- 1  $H/\tilde{m}_a$  increases monotonically with the temperature.

MiMeS is designed to make as few assumptions as possible. However, it still assumes that:

- 1  $H/\tilde{m}_a$  increases monotonically with the temperature.
- 2  $\dot{\theta}(0) = 0$ . This will be changed in the future.

MiMeS is designed to make as few assumptions as possible. However, it still assumes that:

- 1  $H/\tilde{m}_a$  increases monotonically with the temperature.
- 2  $\dot{\theta}(0) = 0$ . This will be changed in the future.
- 3 The energy density of the axion/ALP is always subdominant.

MiMeS is designed to make as few assumptions as possible. However, it still assumes that:

- 1  $H/\tilde{m}_a$  increases monotonically with the temperature.
- 2  $\dot{\theta}(0) = 0$ . This will be changed in the future.
- 3 The energy density of the axion/ALP is always subdominant.
- 4 Only the EOM determines the energy density (no annihilations, no strings, etc.).

# What MiMeS expects from you

Apart from  $\theta_{\text{ini}}$  and  $f_a$ , keep in mind that MiMeS needs:



# What MiMeS expects from you

Apart from  $\theta_{\text{ini}}$  and  $f_a$ , keep in mind that MiMeS needs:

- 1 The mass of the axion/ALP. A data file or an actual function.

# What MiMeS expects from you

Apart from  $\theta_{\text{ini}}$  and  $f_a$ , keep in mind that MiMeS needs:

- 1 The mass of the axion/ALP. A data file or an actual function.
- 2 Data file with  $\log a/a_i$  ( $a_i$  is some arbitrary value; MiMeS rescales it appropriately),  $T$ , and  $\log H$  of the underlying cosmology.

# What MiMeS expects from you

Apart from  $\theta_{\text{ini}}$  and  $f_a$ , keep in mind that MiMeS needs:

- 1 The mass of the axion/ALP. A data file or an actual function.
- 2 Data file with  $\log a/a_i$  ( $a_i$  is some arbitrary value; MiMeS rescales it appropriately),  $T$ , and  $\log H$  of the underlying cosmology.
- 3 Value for  $3H/\tilde{m}_a \gg 1$ , which defines the point where integration begins.

# What MiMeS expects from you

Apart from  $\theta_{\text{ini}}$  and  $f_a$ , keep in mind that MiMeS needs:

- 1 The mass of the axion/ALP. A data file or an actual function.
- 2 Data file with  $\log a/a_i$  ( $a_i$  is some arbitrary value; MiMeS rescales it appropriately),  $T$ , and  $\log H$  of the underlying cosmology.
- 3 Value for  $3H/\tilde{m}_a \gg 1$ , which defines the point where integration begins.
- 4 At which point the adiabatic invariant is constant enough so that numerical integration can stop.

# What MiMeS expects from you

Apart from  $\theta_{\text{ini}}$  and  $f_a$ , keep in mind that MiMeS needs:

- 1 The mass of the axion/ALP. A data file or an actual function.
- 2 Data file with  $\log a/a_i$  ( $a_i$  is some arbitrary value; MiMeS rescales it appropriately),  $T$ , and  $\log H$  of the underlying cosmology.
- 3 Value for  $3H/\tilde{m}_a \gg 1$ , which defines the point where integration begins.
- 4 At which point the adiabatic invariant is constant enough so that numerical integration can stop.
- 5 Other input, related to the algorithm, that might confuse you; e.g. temperature at which integration stops no *matter what!*

# Examples

## 1 Calculating the Relic Abundance

- The axion EOM
- How hard can it be?

## 2 MiMeS

- MiMeS
- MiMeS under the hood
- How to get MiMeS
- Configure (and make)
- Classes
- Template arguments
- MiMeS from python
- Assumptions
- What MiMeS expects from you

## 3 Examples

- python
- C++

## 4 Outlook

## Define everything and solve in just a few lines of code!

---

```

1 from time import time; from sys import stderr #you need these in order to print the time in stderr
2
3 #add the relative path for MiMeS/src
4 from sys import path as sysPath; sysPath.append('../src')
5
6 from interfacePy.AxionMass import AxionMass #import the AxionMass class
7 from interfacePy.Axion import Axion #import the Axion class
8 from interfacePy.Cosmo import mP #import the Planck mass
9
10 def main():
11     # AxionMass instance
12     axionMass = AxionMass(r'../src/data/chi.dat',0,mP)
13
14     # define  $\tilde{m}_a^2$  for  $T \leq T_{\min}$ 
15     TMin, chiMin=axionMass.getTMin(), axionMass.getChiMin()
16     axionMass.set_ma2_MIN( lambda T,fa: chiMin/fa/fa )
17
18     # define  $\tilde{m}_a^2$  for  $T > T_{\max}$ 
19     TMax, chiMax=axionMass.getTMax(), axionMass.getChiMax()
20     axionMass.set_ma2_MAX( lambda T,fa: chiMax/fa/fa*pow(TMax/T,8.16))
21
22     #in python it is more convenient to use relative paths
23     inputFile="..UserSpace/InputExamples/MatterInput.dat"
24
25     ax = Axion(0.1, 1e16, 500, 1e-4, 1e3, 10, 1e-2, inputFile, axionMass,
26               1e-2, 1e-8, 1e-2, 1e-10, 1e-10, 0.85, 1.5, int(1e7))
27
28     ax.solveAxion()
29
30     print("theta_i=",ax.theta_i,"f_a=",ax.fa,"GeV\n","theta_osc~=",
31           ax.theta_osc,"T_osc~=",ax.T_osc,"GeV\n","Omega_h^2=",ax.relic)
32
33     #once we are done we should run the destructor
34     del ax,axionMass
35
36 if __name__ == '__main__':
37     _=time()
38     main()
39     print(round(time()-_,3),file=stderr)
40

```

---

## Notice: C++ and python are quite similar!

---

```

1 #include<iomanip>
2 #include"MiMeS.hpp"
3
4 using numeric = long double;//make life easier if you want to change to double
5
6 int main(){
7     mimes::util::Timer _timer_;//use this to time it!
8
9     // use chi_PATH to interpolate the axion mass.
10    mimes::AxionMass<numeric> axionMass(chi_PATH,0,mimes::Cosmo<numeric>::mP);
11
12    /*set  $\tilde{m}_a^2$  for  $T \geq T_{\max}$ */
13    numeric TMax=axionMass.getTMax(), chiMax=axionMass.getChiMax();
14
15    axionMass.set_ma2_MAX(
16        [&chiMax,&TMax](numeric T, numeric fa){ return chiMax/fa/fa*std::pow(T/TMax, -8.16);}
17    );
18
19    /*set  $\tilde{m}_a^2$  for  $T \leq T_{\min}$ */
20    numeric TMin=axionMass.getTMin(), chiMin=axionMass.getChiMin();
21
22    axionMass.set_ma2_MIN(
23        [&chiMin,&TMin](numeric T, numeric fa){ return chiMin/fa/fa;}
24    );
25
26    /*this path contains the cosmology*/
27    std::string inputFile = std::string(rootDir)+
28        std::string("/UserSpace/InputExamples/MatterInput.dat");
29
30    /*declare an instance of Axion*/
31    mimes::Axion<numeric, 1, RODASPR2<numeric> > ax(0.1, 1e16, 500, 1e-4, 1e3, 10, 1e-2,
32        inputFile, &axionMass, 1e-2, 1e-8, 1e-2, 1e-10, 1e-10, 0.85, 1.5, 0.85,
33        int(1e7) );
34    /*solve the EOM!*/
35    ax.solveAxion();
36
37    std::cout<<std::setprecision(5)
38    <<"theta_i="<<ax.theta_i<<std::setw(25)<<"f_a="<<ax.fa<<"_GeV\n"<<"theta_osc~="<<ax.theta_osc
39    <<std::setw(20)<<"T_osc~="<<ax.T_osc<<"GeV_\n"<<"Omega_h^2="<<ax.relic<<"n";
40
41    return 0;
42 }

```



## 1 Calculating the Relic Abundance

- The axion EOM
- How hard can it be?

## 2 MiMeS

- MiMeS
- MiMeS under the hood
- How to get MiMeS
- Configure (and make)
- Classes
- Template arguments
- MiMeS from python
- Assumptions
- What MiMeS expects from you

## 3 Examples

- python
- C++

## 4 Outlook

## What we saw:

- MiMeS solves the axion/ALP EOM.
- MiMeS treats both the mass and the underlying cosmology as user inputs.
- MiMeS allows the user to change a number of other things, from the plasma RDOFs to the convergence conditions.

## MiMeS **may be amended in the future because:**

- MiMeS should allow the user to consider different initial value of  $\dot{\theta}$ ; the "kinematic" MiMeS might come soon.
- MiMeS should be able to handle non-vanishing RHS; *i.e.* solve the "driven" dumped time-dependent pendulum.
- MiMeS should be able to compare against searches on the fly.

# Thank you!

## Breakdown of MiMeS:

Language	files	comment	code
C/C++ Header	35	444	1595
C++	20	198	1106
Python	22	367	1000
SUM:	77	1009	3701



# Backup

(equations, derivations, tables)

$$\left( \frac{d^2}{dt^2} + 3H(t) \frac{d}{dt} + \tilde{m}_a^2(t) \right) \theta(t) = 0 .$$

Reparametrize by introducing

$$\theta_{\text{trial}} = \exp \left[ i \int dt \left( \psi(t) + 3/2 i H(t) \right) \right] .$$

The Eome, then becomes just

$$\psi^2 = \Omega^2 + i \dot{\psi} ,$$

with  $\Omega^2 = \tilde{m}_a^2 - \frac{9}{4}H^2 - \frac{3}{2}\dot{H}$ . The solution takes the form

$\psi = \pm \sqrt{\Omega^2 + i\dot{\psi}}$ . However, for  $\dot{\psi} \ll \Omega^2$  and  $\dot{\Omega} \ll \Omega^2$ , it can be approximated as

$$\psi \approx \pm \Omega + \frac{i}{2} \frac{d \log \Omega}{dt} .$$

So, after applying the initial conditions, the EOM is solved by

$$\theta(t) \approx \theta_{\text{ini}} \sqrt{\frac{\Omega_{\text{ini}}}{\Omega(t)}} \left(\frac{a}{a_{\text{ini}}}\right)^{-3/2} \cos\left(\int_{t_{\text{ini}}}^t dt' \Omega(t')\right).$$

Taking  $t_{\text{ini}} = t_{\text{osc}}$  (*i.e.*  $\dot{\theta}(t_{\text{osc}}) = 0$ , which is not generally good), have

$$\theta(t) \approx \theta_{\text{osc}} \left(\frac{3}{4}\right)^{1/4} \sqrt{\frac{\tilde{m}_a|_{t=t_{\text{osc}}}}{\tilde{m}_a(t)}} \left(\frac{a}{a_{\text{osc}}}\right)^{-3/2} \cos\left(\int_{t_{\text{osc}}}^t dt' \tilde{m}_a(t')\right),$$

where  $\theta_{\text{osc}} = \theta|_{t=t_{\text{osc}}}$ . This equation is further simplified if we assume that  $\theta_{\text{osc}} \approx \theta_{\text{ini}}$  (again not really good), *i.e.*

$$\theta(t) \approx \theta_{\text{ini}} \left(\frac{3}{4}\right)^{1/4} \sqrt{\frac{\tilde{m}_a|_{t=t_{\text{osc}}}}{\tilde{m}_a(t)}} \left(\frac{a}{a_{\text{osc}}}\right)^{-3/2} \cos\left(\int_{t_{\text{osc}}}^t dt' \tilde{m}_a(t')\right).$$

# Adiabatic invariant – I

Given a system with Hamiltonian  $\mathcal{H}(\theta, p; t)$ , the equations of motion are

$$\dot{p} = -\frac{\partial \mathcal{H}}{\partial \theta}, \quad \dot{\theta} = \frac{\partial \mathcal{H}}{\partial p}.$$

Also,

$$d\mathcal{H} = \dot{\theta} dp - \dot{p} d\theta + \frac{\partial \mathcal{H}}{\partial t} dt.$$

If this system exhibits closed orbits (e.g. if it oscillates), we define

$$J \equiv C \oint p d\theta,$$

where the integral is over a closed path (e.g. a period,  $T$ ), and  $C$  indicates that  $J$  can always be rescaled with a constant. If the Hamiltonian varies slowly during a cycle,

$$\frac{dJ}{dt} = C \oint (\dot{p} d\theta + p d\dot{\theta}) = C \int_t^{t+T} \frac{\partial \mathcal{H}}{\partial t'} dt' \approx T \left. \frac{\partial \mathcal{H}(t')}{\partial t'} \right|_{t'=t} \approx 0.$$

So,  $J$  is an adiabatic invariant!

# Adiabatic invariant – II

The Hamiltonian that results in the EOM

$$\mathcal{H} = \frac{1}{2} \frac{p^2}{f_a^2 a^3} + V(\theta) a^3 ,$$

with

$$p = f_a^2 a^3 \dot{\theta}$$
$$V(\theta) = \tilde{m}_a^2 f_a^2 (1 - \cos \theta) .$$

If  $\mathcal{H}$  varies slowly –  $\dot{\tilde{m}}_a(T)/\tilde{m}_a \ll \tilde{m}_a$  and  $H \ll \tilde{m}_a$ , then

$$\begin{aligned} J &= \frac{\oint p d\theta}{\pi f_a^2} = \frac{1}{\pi f_a^2} \oint \sqrt{2(\mathcal{H}(\theta) - V(\theta) a^3) f_a^2 a^3} d\theta \\ &= \frac{2}{\pi f_a^2} \int_{-\theta_{\text{peak}}}^{\theta_{\text{peak}}} \sqrt{2(\mathcal{H}(\theta_{\text{peak}}) - V(\theta) a^3) f_a^2 a^3} d\theta \\ &= \frac{2\sqrt{2}}{\pi f_a} \int_{-\theta_{\text{peak}}}^{\theta_{\text{peak}}} \sqrt{V(\theta_{\text{peak}}) - V(\theta) a^3} d\theta \\ &= \frac{2\sqrt{2}}{\pi} \tilde{m}_a a^3 \int_{-\theta_{\text{peak}}}^{\theta_{\text{peak}}} \sqrt{\cos \theta - \cos \theta_{\text{peak}}} d\theta , \end{aligned}$$

is the adiabatic invariant – up to a multiplication with a constant.



# C++ Input

# AxionMass class – Definition via file

In order to define an instance of the `AxionMass` class that interpolates the  $\tilde{m}_a$ , use the constructor:

```
1 template<class LD>  
2 mimes::AxionMass<LD>(std::string chi_PATH, LD minT=0, LD maxT=mimes::Cosmo::mP)
```

The arguments are:

- 1 `chi_Path`: Relative or absolute path to data file with  $T$  (in GeV),  $\chi(T)$  (in  $\text{GeV}^4$ ).
- 2 `minT, maxT`: Interpolation limits. These are used in order to stop the interpolation at the closest temperatures that exist in the data file. This means that the actual interpolation limits are  $T_{\min} \geq \text{minT}$  and  $T_{\max} \leq \text{maxT}$ . Beyond these limits the axion mass is assumed to be constant.

The definition of  $\tilde{m}_a^2$  beyond  $T_{\min}$  and  $T_{\max}$  can be changed to realistic function, using `mimes::AxionMass<LD>::set_ma2_MIN(std::function<LD(LD,LD)> ma2_MIN)` and `mimes::AxionMass<LD>::set_ma2_MAX(std::function<LD(LD,LD)> ma2_MAX)`. These definitions may need the actual values of  $T_{\min, \max}$  and  $\chi(T_{\min, \max})$ . These are obtained from

- `template<class LD> LD mimes::AxionMass<LD>::getTMin()`: This function returns the minimum interpolation temperature,  $T_{\min}$ .
- `template<class LD> LD mimes::AxionMass<LD>::getTMax()`: This function returns the maximum interpolation temperature,  $T_{\max}$ .
- `template<class LD> LD mimes::AxionMass<LD>::getChiMin()`: This function returns  $\chi(T_{\min})$ .
- `template<class LD> LD mimes::AxionMass<LD>::getChiMax()`: This function returns  $\chi(T_{\max})$ .

Note that all `std::function<LD(LD,LD)>` can be any callable object that takes  $T$  and  $f_a$  and returns  $\tilde{m}_a^2$ .

# AxionMass class – Definition via function

In order to define an instance of the `AxionMass` class via a function, use the constructor:

---

```
1 template<class LD>  
2 mimes::AxionMass<LD>(std::function<LD(LD,LD)> ma2)
```

---

Here, `ma2` can be any callable object that takes  $T$  and  $f_a$  and returns  $\tilde{m}_a^2$ .

# Axion class – Expected input

The constructor of the `Axion` class is

---

```
1 template<class LD, const int Solver, class Method>
2 mimes::Axion<LD, Solver, Method>(LD theta_i, LD fa, LD umax, LD TSTOP,
3     LD ratio_ini, unsigned int N_convergence_max, LD convergence_lim,
4     std::string inputFile, AxionMass<LD> *axionMass, LD initial_step_size=1e-2,
5     LD minimum_step_size=1e-8, LD maximum_step_size=1e-2,
6     LD absolute_tolerance=1e-8, LD relative_tolerance=1e-8, LD beta=0.9,
7     LD fac_max=1.2, LD fac_min=0.8, unsigned int maximum_No_steps=10000000)
```

---

The input that `MiMeS` expects is:

- 1 `theta_i`: Initial angle.
- 2 `fa`: The PQ scale.
- 3 `umax`: Once  $u = \log a/a_i > umax$ , the integration stops. Typical value:  $\sim 500$ .
- 4 `TSTOP`: Once  $T < TSTOP$ , integration stops. Typical value:  $10^{-4}$  GeV.
- 5 `ratio_ini`: Integration starts at  $u$  with  $3H/\tilde{m}_a \approx ratio\_ini$ . Typical value:  $\sim 10^3$ .
- 6 `N_convergence_max`, `convergence_lim`: Integration stops when the relative difference between two consecutive peaks is less than `convergence_lim` for `N_convergence_max` consecutive peaks.
- 7 `inputFile`: Relative (or absolute) path to a file that describes the cosmology. The columns should be:  $u$   $T$  [GeV]  $\log H$ , with acceding  $u$ . Entropy injection should have stopped before the lowest temperature of given in `inputFile`.
- 8 `axionMass`: Instance of `mimes::AxionMass<LD>` class. In C++ this instance is passed as a pointer to the constructor of the `mimes::Axion<LD, Solver, Method>` class, while in python it is simply passed as a variable.

# Axion class – Optional input

The optional input, relative to the RK algorithm, is:

- 1 `initial_stepsize`: Initial step-size of the solver. Default value:  $10^{-2}$ .
- 2 `minimum_stepsize`: Lower limit of the step-size. Default value:  $10^{-8}$ .
- 3 `maximum_stepsize`: Upper limit of the step-size. Default value:  $10^{-2}$ .
- 4 `absolute_tolerance`: Absolute tolerance of the RK solver. Default value:  $10^{-8}$ .
- 5 `relative_tolerance`: Relative tolerance of the RK solver. Default value:  $10^{-8}$ .
- 6 `beta`: Aggressiveness of the adaptation strategy. Default value: 0.9.
- 7 `fac_max, fac_min`: The step-size does not change more than `fac_max` and less than `fac_min` within a trial step. Default values: 1.2 and 0.8, respectively.
- 8 `maximum_No_steps`: If integration needs more than `maximum_No_steps` integration stops. Default value:  $10^7$ .

python **Input**

# AxionMass class – Definition via file

The actual constructor of the `AxionMass` in the `python` interface is `AxionMass(*args)`. However, it is intended to be used in *only* two ways. In order to define an instance of the `AxionMass` class that interpolates the  $\tilde{m}_a$ , use the constructor as:

---

1 `AxionMass(chi_PATH, minT=0, maxT=Cosmo.mP)`

---

The arguments are the same as in the `C++` case.

The definition of  $\tilde{m}_a^2$  beyond  $T_{\min}$  and  $T_{\max}$  can be changed using `AxionMass.set_ma2_MIN(ma2_MIN)` and `AxionMass.set_ma2_MAX(ma2_MAX)`. These definitions may need the actual values of  $T_{\min, \max}$  and  $\chi(T_{\min, \max})$ . These are obtained from

- `AxionMass.getTMin()`: This function returns the minimum interpolation temperature,  $T_{\min}$ .
- `AxionMass.getTMax()`: This function returns the maximum interpolation temperature,  $T_{\max}$ .
- `AxionMass.getChiMin()`: This function returns  $\chi(T_{\min})$ .
- `AxionMass.getChiMax()`: This function returns  $\chi(T_{\max})$ .

The difference between the `C++` case is that `ma2` cannot be any callable object; it has to be a regular function that takes  $T$  and  $f_a$  and returns  $\tilde{m}_a^2$ .

# AxionMass class – Definition via function

In order to define an instance of the `AxionMass` class via a function, use the constructor as:

---

1 `AxionMass`(ma2)

---

The difference between the C++ case is that `ma2` cannot be any callable object; it has to be a regular function that takes  $T$  and  $f_a$  and returns  $\tilde{m}_a^2$ .



The constructor of the `Axion` class is

---

```
1 Axion(theta_i, fa, umax, TSTOP, ratio_ini, N_convergence_max, convergence_lim, inputFile,  
2   axionMass, initial_step_size=1e-2, minimum_step_size=1e-8, maximum_step_size=1e-2,  
3   absolute_tolerance=1e-8, relative_tolerance=1e-8, beta=0.9, fac_max=1.2, fac_min=0.8,  
4   maximum_No_steps=10000000)
```

---

All the arguments are the same as in the C++ case. The only difference is that the `AxionMass` instance ( `axionMass`) is not passed as a pointer, as there is no direct way to do it in `python`. However, the underlying object is the same, as it is converted internally using `ctypes`.

# Files and compilation variables

There are some paths to file that the user can provide in order to use different data for the RDOF, anharmonic factor, and  $\chi$  (optional).

These paths are stored as strings in `MiMeS/src/misc_dir/path.hpp` when `bash configure.sh` is run.

These paths can be changed by changing the following variables in `MiMeS/Paths.mk`:

- `cosmoDat`: Relative path to data file with  $T$  (in GeV),  $h_{\text{eff}}$ ,  $g_{\text{eff}}$ .
- `axMDat`: Relative path to data file with  $T$  (in GeV),  $h_{\text{eff}}$ ,  $g_{\text{eff}}$ . This variable can be omitted if the user intends to define all masses via functions.
- `anFDat`: Relative path to data file with  $\theta_{\text{peak}}$ ,  $f(\theta_{\text{peak}})$ .

It is advisable that if the paths change `bash configure.sh` and `make` should be run.

# Template arguments

You need to choose what numeric type to use. This is done by the template argument `LD` which should be `double` (fast) or `long double` (accurate).<sup>8</sup>

You also need to tell `MiMeS` which integration strategy to use. This is done by choosing template arguments:

- `Solver` can be set to `1` for Rosenbrock (semi-implicit Runge-Kutta). The `Method` argument in this case can be:
  - `RODASPR2<LD>` (4th order).
  - `ROS34PW2<LD>` (3rd order).
  - `ROS3W<LD>` (2rd order, *very bad*).
- `Solver` can be set to `2` for explicit RK. The `Method` argument can be:
  - `DormandPrinceRK45<LD>` (7th order)
  - `CashKarpRK45<LD>` (5th order, *very bad*).
  - `RK45<LD>` (5th order, *very bad*).

---

<sup>8</sup> You could choose `float`, but we live in 2021.

# Definitions.mk

In order to call the `python` interface of MiMeS, we need to first call `make lib` in the root directory of MiMeS.

Before that, we can take some time to decide what the template arguments and compilation options should be. In the file `MiMeS/Definitions.mk`, you can change the variables:

- `LONGpy=long` will compile the library with `long double` numeric types. `LONGpy=` will compile the library with `double` numeric types.
- `SOLVER` and `METHOD`, as in the template arguments.

Also, in the same file, you can change compilation options:

- **Compiler:**
  - `CC=g++` in order to use the GNU C++ compiler.
  - `CC=clang -lstdc++` in order to use the clang C++ compiler.
- **Optimization level:**
  - `OPT=O0`: No optimization.
  - `O=O1, O2, or O3`: all these perform mostly the same (read the compiler documentation for more information on the optimization).
  - `OPT=Ofast`: full optimization (fast, but dangerous).