

Misalignment Mechanism Solver

Dimitrios Karamitros

Manchester U.

Recent Progress in Axion Theory and Experiment
IPPP, Durham
05/09/2022

Comput. Phys. Commun. **275**, 108311 (2022)

[arXiv:2110.12253 \[hep-ph\]](https://arxiv.org/abs/2110.12253)

github.com/dkaramit/MiMeS

mimes.hepforge.org

Outline

1

Axion Dark Matter

- Why particle dark matter
- The dark matter particle
- The axion (like) particle

2

Calculating the Relic Abundance

- The axion EOM
- How hard can it be?
- Initial conditions
- (Bad) Approximations
- Need for speed, accuracy, and automation

3

MiMeS

- MiMeS: Why?
- MiMeS: Under the hood
- Sidenote: adiabatic invariant – I
- Sidenote: adiabatic invariant – II
- Using adiabatic invariant
- MiMeS: the *solver* part

4

Using MiMeS

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

5

Examples

- python
- C++

6

Outlook

Axion Dark Matter

1

Axion Dark Matter

- Why particle dark matter
- The dark matter particle
- The axion (like) particle

2

Calculating the Relic Abundance

- The axion EOM
- How hard can it be?
- Initial conditions
- (Bad) Approximations
- Need for speed, accuracy, and automation

3

MiMeS

- MiMeS: Why?
- MiMeS: Under the hood
- Sidenote: adiabatic invariant – I
- Sidenote: adiabatic invariant – II
- Using adiabatic invariant
- MiMeS: the *solver* part

4

Using MiMeS

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

5

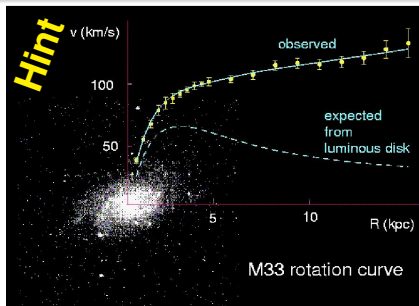
Examples

- python
- C++

6

Outlook

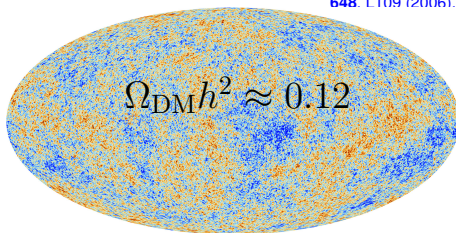
Why particle dark matter



E. Corbelli and P. Salucci, *Mon. Not. Roy. Astron. Soc.*
311 441 (2000), [arXiv:astro-ph/9909252](#).



M. Markevitch, *ESA Spec. Publ.* **604** (2006) 723,
[astro-ph/0511345](#). Clowe, Bradac, *et. al.* *Astrophys. J.*
648. L109 (2006). [astro-ph/0608407](#)



N. Aghanim *et al.* [Planck Collaboration], [arXiv:1807.06209 \[astro-ph.CO\]](#).

The dark matter particle

“I know one thing, that I know nothing.”

—Socrates

- Gravitational interactions.
- Mostly electrically neutral.
- Stable or very slow decay rate.
- Non-Baryonic.
- Cold/Warm and non-relativistic today.

The axion (like) particle

Notably, the original axion was originally introduced in order to solve the *strong-CP problem* of the SM. Axion-like-particles (ALPs) arise in a number of new physics models, beyond the SM.

Axions and ALPs generally:

- Have suppressed interactions with photons.
- Are (mostly) stable.
- Were non-relativistic around the epoch of structure formation.
- Non-baryonic (by definition), and of-course interact gravitationally.

The axion (like) particle

Notably, the original axion was originally introduced in order to solve the *strong-CP problem* of the SM. Axion-like-particles (ALPs) arise in a number of new physics models, beyond the SM.

Axions and ALPs generally:

- Have suppressed interactions with photons.
- Are (mostly) stable.
- Were non-relativistic around the epoch of structure formation.
- Non-baryonic (by definition), and of-course interact gravitationally.

Maybe DM has axionic nature!

Calculating the Relic Abundance

1

Axion Dark Matter

- Why particle dark matter
- The dark matter particle
- The axion (like) particle

2

Calculating the Relic Abundance

- The axion EOM
- How hard can it be?
- Initial conditions
- (Bad) Approximations
- Need for speed, accuracy, and automation

3

MiMeS

- MiMeS: Why?
- MiMeS: Under the hood
- Sidenote: adiabatic invariant – I
- Sidenote: adiabatic invariant – II
- Using adiabatic invariant
- MiMeS: the *solver* part

4

Using MiMeS

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

5

Examples

- python
- C++

6

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).

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 (wait a minute...).
- No package/library/program available!

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 (wait a minute...).
- No package/library/program available!

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

Some time at the very early Universe, $\tilde{m}_a \ll H(T)$,¹ with

$$\ddot{\theta} + 3H \dot{\theta} \approx 0 .$$

The solution is

$$\theta = \theta_{\text{ini}} + C \int_0^t dt' \left(\frac{a(t' = 0)}{a(t')} \right)^3 .$$

So, $\dot{\theta} \sim a^{-3}$. Since we are interested in θ at much later times (once the potential becomes relevant), $\dot{\theta} \approx 0$. Therefore, we can begin integration at some point with $3H \gg \tilde{m}_a$, and set $\theta(t = t_{\text{ini}}) = \theta_{\text{ini}}$ and $\dot{\theta}(t = t_{\text{ini}}) = 0$.

¹ This is an assumption that MiMeS has to make for the sake of generality.

(Bad) Approximations

“If a man knows not which port he sails, no wind is favourable.”

–Seneca

- Assume $\theta \ll 1$, and linearise the EOM. *Not general.*
- Assume that at $\tilde{m}_a(T_{\text{osc}}) \approx 3H(T_{\text{osc}})$ we have $\dot{\theta}(T_{\text{osc}}) = 0$. *Not very precise.*
- Assume that $\theta_{\text{osc}} \approx \theta_{\text{ini}}$. *Generally quite bad.*

(Bad) Approximations

“If a man knows not which port he sails, no wind is favourable.”

–Seneca

- Assume $\theta \ll 1$, and linearise the EOM. *Not general.*
- Assume that at $\tilde{m}_a(T_{\text{osc}}) \approx 3H(T_{\text{osc}})$ we have $\dot{\theta}(T_{\text{osc}}) = 0$. *Not very precise.*
- Assume that $\theta_{\text{osc}} \approx \theta_{\text{ini}}$. *Generally quite bad.*

These result in “WKB”-approximate solution

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

which gives us:

$$\rho_{a,0} = \gamma^{-1} \frac{s_0}{s_{\text{osc}}} \frac{1}{2} f_a^2 m_a \tilde{m}_{a,\text{osc}} \theta_{\text{ini}}^2,$$

where γ the amount of entropy injection between T_{osc} and today.

Need for speed, accuracy, and automation

Life before MiMeS:

Life before MiMeS:

- No available tool that can help us reproduce published results obtained by numerical integration. *Reproducing results means reproducing effort.*

Life before MiMeS:

- No available tool that can help us reproduce published results obtained by numerical integration. *Reproducing results means reproducing effort.*
- The approximations can be tested against numerical results in a case-by-case basis. *No measure of accuracy.*

Need for speed, accuracy, and automation

Life before MiMeS:

- No available tool that can help us reproduce published results obtained by numerical integration. *Reproducing results means reproducing effort.*
- The approximations can be tested against numerical results in a case-by-case basis. *No measure of accuracy.*
- Simply checking if an ALP model is compatible with a cosmological scenario is *slow* or *inaccurate*.

1

Axion Dark Matter

- Why particle dark matter
- The dark matter particle
- The axion (like) particle

2

Calculating the Relic Abundance

- The axion EOM
- How hard can it be?
- Initial conditions
- (Bad) Approximations
- Need for speed, accuracy, and automation

3

MiMeS

- MiMeS: Why?
- MiMeS: Under the hood
- Sidenote: adiabatic invariant – I
- Sidenote: adiabatic invariant – II
- Using adiabatic invariant
- MiMeS: the *solver* part

4

Using MiMeS

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

5

Examples

- python
- C++

6

Outlook

MiMeS: Why?

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

MiMeS is:

- MiMeS is a C++ header-only library that contains various templated classes.
- MiMeS comes with a `python` interface.
- 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.

“It is the empty space which makes a bowl useful.”

–Laozi

MiMeS is built as minimally as possible:

- MiMeS relies on `NaBBODES`² `SimpleSplines`³.
- 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.

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

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

Sidenote: adiabatic invariant – J

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,

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

So, J is an adiabatic invariant!

Sidenote: 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

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

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

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

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

Sidenote: 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

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

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

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

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

Important: θ_{peak} is the peak of the oscillation.

Using adiabatic invariant

We rewrite the adiabatic invariant as

$$J = a^3 \tilde{m}_a \theta_{\text{peak}}^2 f(\theta_{\text{peak}}) ,$$

with

$$f(\theta_{\text{peak}}) = \frac{2\sqrt{2}}{\pi \theta_{\text{peak}}^2} \int_{-\theta_{\text{peak}}}^{\theta_{\text{peak}}} d\theta \sqrt{\cos \theta - \cos \theta_{\text{peak}}} ,$$

the so-called anharmonic factor.

Using adiabatic invariant

We rewrite the adiabatic invariant as

$$J = a^3 \tilde{m}_a \theta_{\text{peak}}^2 f(\theta_{\text{peak}}) ,$$

with

$$f(\theta_{\text{peak}}) = \frac{2\sqrt{2}}{\pi \theta_{\text{peak}}^2} \int_{-\theta_{\text{peak}}}^{\theta_{\text{peak}}} d\theta \sqrt{\cos \theta - \cos \theta_{\text{peak}}} ,$$

the so-called anharmonic factor.

J can be used to determine how θ_{peak} changes with time. By definition, at $\theta = \theta_{\text{peak}}$, $p \sim \dot{\theta} = 0$. This means that we can find $\rho_{a,0}$ on the peak of today's θ , as

$$\rho_{a,0} = \gamma^{-1} \frac{s_0}{s_*} m_a \tilde{m}_{a,*} \frac{1}{2} f_a^2 \theta_{\text{peak},*}^2 f(\theta_{\text{peak},*}) ,$$

where T_* the temperature at which adiabaticity was reached, and γ the entropy injection between T_* and today (*i.e.* $s_0 = \gamma a_*^3 s_*$).

MiMeS: the *solver* part

MiMeS solves the transformed EOM: ⁴

$$\frac{d\zeta}{du} + \left[\frac{1}{2} \frac{d \log H^2}{du} + 3 \right] \zeta + \left(\frac{\tilde{m}_a}{H} \right)^2 \sin \theta = 0 .$$
$$\frac{d\theta}{du} - \zeta = 0 .$$

The parameter u is “time” $u \equiv \log (a/a_{\text{ini}})$. ⁵

⁴Suitable for Runge-Kutta.

⁵ With a_{ini} the initial value of the scale factor. It corresponds to T_{ini} .

MiMeS solves the transformed EOM: ⁴

$$\frac{d\zeta}{du} + \left[\frac{1}{2} \frac{d \log H^2}{du} + 3 \right] \zeta + \left(\frac{\tilde{m}_a}{H} \right)^2 \sin \theta = 0 .$$
$$\frac{d\theta}{du} - \zeta = 0 .$$

The parameter u is “time” $u \equiv \log(a/a_{\text{ini}})$. ⁵

- Integration starts at T_{ini} such that $3H(T_{\text{ini}})/m_a(T_{\text{ini}}) = x \gg 1$, with a user defined x (T_{ini} is automatically determined).
- $H(u)$, $m_a(u)$, and plasma parameters are interpolated from T_{ini} .
- Initial conditions, set at T_{ini} , $\zeta(T_{\text{ini}}) = 0$ and $\theta(T_{\text{ini}}) = \theta_{\text{ini}}$. The initial angle θ_{ini} is user defined.
- MiMeS stops integrating when J becomes slowly varying over enough cycles. This is a user defined condition.

⁴Suitable for Runge-Kutta.

⁵ With a_{ini} the initial value of the scale factor. It corresponds to T_{ini} .

Using MiMeS

1

Axion Dark Matter

- Why particle dark matter
- The dark matter particle
- The axion (like) particle

2

Calculating the Relic Abundance

- The axion EOM
- How hard can it be?
- Initial conditions
- (Bad) Approximations
- Need for speed, accuracy, and automation

3

MiMeS

- MiMeS: Why?
- MiMeS: Under the hood
- Sidenote: adiabatic invariant – I
- Sidenote: adiabatic invariant – II
- Using adiabatic invariant
- MiMeS: the *solver* part

4

Using MiMeS

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

5

Examples

- python
- C++

6

Outlook

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, and download it.

3

Go to 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 then in order to see if you get any segfaults.

There are three classes useful to the user.⁶

⁶ 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.

⁷ K. Saikawa and S. Shirai, JCAP **08** (2020), 011 [arXiv:2005.03544 [hep-ph]].

⁸ 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.⁶

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020⁷ data. The user can choose another file easily.

⁶ 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.

⁷ K. Saikawa and S. Shirai, JCAP **08** (2020), 011 [arXiv:2005.03544 [hep-ph]].

⁸ 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. ⁶

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020 ⁷ 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 ⁸ of the QCD axion mass.

⁶ 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.

⁷ K. Saikawa and S. Shirai, JCAP **08** (2020), 011 [arXiv:2005.03544 [hep-ph]].

⁸ 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.⁶

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020⁷ 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 calculation⁸ of the QCD axion mass.
- `mimes::Axion<LD, Solver, Method>`: This is responsible for actually solving the EOM.

⁶ 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.

⁷ K. Saikawa and S. Shirai, JCAP **08** (2020), 011 [arXiv:2005.03544 [hep-ph]].

⁸ 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).⁹

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*).

⁹ 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.
 - `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).

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 $\zeta(0) = 0$. 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 $\zeta(0) = 0$. 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 $\zeta(0) = 0$. 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 θ_{ini} and f_a , keep in mind that MiMeS needs:

What MiMeS expects from you

Apart from θ_{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 θ_{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 θ_{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 θ_{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 Relative difference of J between a given number of peaks at which we consider adiabaticity to have been reached.

What MiMeS expects from you

Apart from θ_{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 Relative difference of J between a given number of peaks at which we consider adiabaticity to have been reached.
- 5 Other input, such as the temperature at which integration exits.

Examples

1

Axion Dark Matter

- Why particle dark matter
- The dark matter particle
- The axion (like) particle

2

Calculating the Relic Abundance

- The axion EOM
- How hard can it be?
- Initial conditions
- (Bad) Approximations
- Need for speed, accuracy, and automation

3

MiMeS

- MiMeS: Why?
- MiMeS: Under the hood
- Sidenote: adiabatic invariant – I
- Sidenote: adiabatic invariant – II
- Using adiabatic invariant
- MiMeS: the *solver* part

4

Using MiMeS

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

5

Examples

- python
- C++

6

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
17     axionMass.set_ma2_MIN( lambda T,fa: chiMin/fa/fa )
18
19     # define  $\tilde{m}_a^2$  for  $T > T_{\max}$ 
20     TMax, chiMax=axionMass.getTMax(), axionMass.getChiMax()
21
22     axionMass.set_ma2_MAX( lambda T,fa: chiMax/fa/fa*pow(TMax/T,8.16))
23
24     #in python it is more convenient to use relative paths
25     inputFile="..UserSpace/InputExamples/MatterInput.dat"
26
27     ax = Axion(0.1, 1e16, 500, 1e-4, 1e3, 10, 1e-2, inputFile, axionMass,
28               1e-2, 1e-8, 1e-2, 1e-10, 1e-10, 0.85, 1.5, 0.85, int(1e7))
29
30     ax.solveAxion()
31
32     print("theta_i=",ax.theta_i,"t\t\t\t","f_a=",ax.fa,"GeV\n","theta_osc~=",
33          ax.theta_osc,"t","T_osc~=",ax.T_osc,"GeV_\n","Omega_h^2=",ax.relic)
34
35     #once we are done we should run the destructor
36     del ax,axionMass
37
38
39 if __name__ == '__main__':
40     _=time()
41     main()
42     print(round(time()-_,3),file=stderr)

```

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  $T > T_{\max}$ */
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
35    /*solve the EOM!*/
36    ax.solveAxion();
37
38    std::cout<<std::setprecision(5)
39    <<"theta_i="<<ax.theta_i<<std::setw(25)<<"f_a="<<ax.fa<<"_GeV\n"<<"theta_osc~="<<ax.theta_osc
40    <<std::setw(20)<<"T_osc~="<<ax.T_osc<<"GeV_\n"<<"Omega_h^2="<<ax.relic<<"\n";
41
42    return 0;
43 }
```

Outlook

1

Axion Dark Matter

- Why particle dark matter
- The dark matter particle
- The axion (like) particle

2

Calculating the Relic Abundance

- The axion EOM
- How hard can it be?
- Initial conditions
- (Bad) Approximations
- Need for speed, accuracy, and automation

3

MiMeS

- MiMeS: Why?
- MiMeS: Under the hood
- Sidenote: adiabatic invariant – I
- Sidenote: adiabatic invariant – II
- Using adiabatic invariant
- MiMeS: the *solver* part

4

Using MiMeS

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

5

Examples

- python
- C++

6

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 ζ ; the "kinematic" MiMeS might come soon.
- MiMeS should be able to handle non-vanishing RHS; *i.e.* solve the "driven" damped 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.*

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

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 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 $u = \log a/a_i > \text{umax}$, the integration stops. Typical value: ~ 500 .
- 4 `TSTOP`: Once $T < \text{TSTOP}$, integration stops. Typical value: 10^{-4} GeV.
- 5 `ratio_ini`: Integration starts at u with $3H/\tilde{m}_a \approx \text{ratio_ini}$. 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 χ (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_{eff} , g_{eff} .
- `axMDat`: Relative path to data file with T (in GeV), h_{eff} , g_{eff} . This variable can be omitted if the user intends to define all masses via functions.
- `anFDat`: Relative path to data file with θ_{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).¹⁰

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*).

¹⁰ 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).