`MiMeS`: Misalignment Mechanism Solver

Dimitrios Karamitros

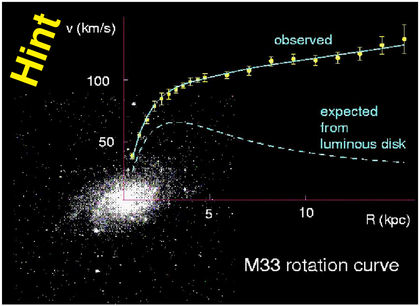
Manchester U.

23/11/2021

El Journal Club más Sabroso

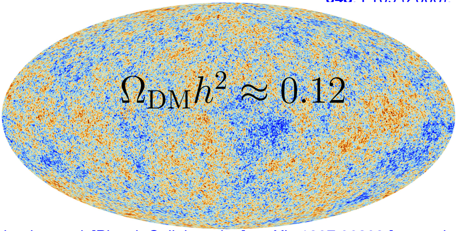
Outline

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

"Ἓν οἶδα, ὅτι οὐδὲν οἶδα."
"I know one thing, that I know nothing."

–Socrates

The dark matter particle

“Ἓν οἶδα, ὅτι οὐδὲν οἶδα.”
“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!

The axion EOM

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 filed, and f_a some energy scale that characterises the potential (Peccei-Quinn breaking scale).

How hard can it be?

Hard (in general).

How hard can it be?

Hard (in general).

The classical analogue is the dumped 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 dumped 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.*

Initial conditions

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

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

So, $\dot{\theta} \sim a^{-3}$. Since we are interested in θ at much later times (once the potential becomes relevant), $\dot{\theta} \approx 0$. [2] Therefore, we can start integration at some point ($t = t_{\rm ini}$) with $3H \gg \tilde{m}_a$, and set $\theta(t = t_{\rm ini}) = \theta_{\rm ini}$ and $\dot{\theta}(t = t_{\rm ini}) = 0$.

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

[2] Standard misalignment mechanism. For the kinetic one see R. T. Co, L. J. Hall and K. Harigaya, Phys. Rev. Lett. **124** (2020) no.25, 251802 [arXiv:1910.14152 [hep-ph]], C. F. Chang and Y. Cui, Phys. Rev. D **102** (2020) no.1, 015003 [arXiv:1911.11885 [hep-ph]], or B. Barman, N. Bernal, N. Ramberg and L. Visinelli, [arXiv:2111.03677 [hep-ph]] .

(Bad) Analytical approximations

Once we agree on the initial conditions, we move to the next important things:

[3] Defined from $s(T_0) = \gamma \, {a_{\rm osc}}^3 \, s_{\rm osc}$.

(Bad) Analytical approximations

Once we agree on the initial conditions, we move to the next important things:

- Assume $\theta \ll 1$, and linearise the EOM. *Not always that bad.*

[3] Defined from $s(T_0) = \gamma \, {a_{\mathrm{osc}}}^3 \, s_{\mathrm{osc}}$.

(Bad) Analytical approximations

Once we agree on the initial conditions, we move to the next important things:

- Assume $\theta \ll 1$, and linearise the EOM. *Not always that bad.*
- Assume that at $\tilde{m}_a(T_{\rm osc}) \approx 3H(T_{\rm osc})$, the oscillations begin with $\dot{\theta}(T_{\rm osc}) = 0$. *This is in general quite bad.*

[3] Defined from $s(T_0) = \gamma \; {a_{\rm osc}}^3 \; s_{\rm osc}$.

(Bad) Analytical approximations

Once we agree on the initial conditions, we move to the next important things:

- Assume $\theta \ll 1$, and linearise the EOM. *Not always that bad.*
- Assume that at $\tilde{m}_a(T_{\rm osc}) \approx 3H(T_{\rm osc})$, the oscillations begin with $\dot{\theta}(T_{\rm osc}) = 0$. *This is in general quite bad.*
- Assume that $\theta_{\rm osc} \approx \theta_{\rm ini}$. *Generally quite bad.*

[3] Defined from $s(T_0) = \gamma \; {a_{\rm osc}}^3 \; s_{\rm osc}$.

(Bad) Analytical approximations

Once we agree on the initial conditions, we move to the next important things:

- Assume $\theta \ll 1$, and linearise the EOM. *Not always that bad.*
- Assume that at $\tilde{m}_a(T_{\rm osc}) \approx 3H(T_{\rm osc})$, the oscillations begin with $\dot{\theta}(T_{\rm osc}) = 0$. *This is in general quite bad.*
- Assume that $\theta_{\rm osc} \approx \theta_{\rm ini}$. *Generally quite bad.*

Then, we get the “WKB"-approximate solution

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

The advantage of this approximation is that we get an easy formula for the axion/ALP energy density today:

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

where γ the amount of entropy injection between $T_{\rm osc}$ and today. [3]

[3] Defined from $s(T_0) = \gamma \; {a_{\rm osc}}^3 \; s_{\rm osc}$.

Need for accuracy, speed, and automation

Serious disadvantages of the approximate results:

- The approximations can be tested against numerical results in a case-by-case basis; there is no way to tell if they will work in new models and cosmological scenarios.
- There is no available tool that can help us reproduce published results obtained by numerical integration; people use their own ***private*** code.
- If someone wants to simply see if an ALP model is compatible with a cosmological scenario, they have to develop their own private code; the overall effort of the community increases.

Ingredients for numerical integration: Adiabatic invariant

If a system exhibits closed orbits, the quantity

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

is the adiabatic invariant. In this case, it becomes

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

with

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

the so-called anharmonic factor.

Ingredients for numerical integration: Adiabatic invariant

If a system exhibits closed orbits, the quantity

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

is the adiabatic invariant. In this case, it becomes

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

with

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

the so-called anharmonic factor.
Important: $\theta_{\rm peak}$ is the peak of the oscillation. So, J can be used to determine how $\theta_{\rm peak}$ changes with time. By definition, at $\theta = \theta_{\rm peak}$, $p \sim \dot{\theta} = 0$. This means that we can find $\rho_{a,0}$ on the peak of today's θ, as

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

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

Ingredients for numerical integration: Integration limits

The starting condition should be:

- At a point with $\dot{\theta} = 0$.
- At some $T_{\rm ini}$ with $3H(T_{\rm ini})/\tilde{m}_a(T_{\rm ini}) \gg 1$.
- Such that the evolution of θ is the same for any other starting point with $T > T_{\rm ini}$.
- *User defined, so that we can change it the cosmology.*

The stopping condition should be:

- At a point, $T = T_*$, with $\Delta J/J \leq \epsilon \ll 1$.
- Such that the evolution of θ is the same for any other end point with $T < T_*$.
- *User defined, so that we can change it the cosmology.*

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; 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

MiMeS: Under the hood

MiMeS relies on NaBBODES [4] for the numerical integration, and SimpleSplines [5] 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.

[4] https://github.com/dkaramit/NaBBODES.
[5] https://github.com/dkaramit/SimpleSplines.

MiMeS uses a notation suitable (any) underlying cosmology, since it is up to the user to define the cosmological evolution.
First, we define

$$u \equiv \log\left(a/a_{\rm ini}\right) \;,$$

with $a_{\rm ini}$ some initial value of the scale factor. [6] Then, the EOM becomes

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

The initial conditions are $\zeta(0) = 0$ and $\theta(0) = \theta_{\rm ini}$.

[6] Only the ratios $a/a_{\rm ini}$ appear in the calculations.

MiMeS: Notation

MiMeS uses a notation suitable (any) underlying cosmology, since it is up to the user to define the cosmological evolution.
First, we define

$$u \equiv \log\left(a/a_{\rm ini}\right) \;,$$

with $a_{\rm ini}$ some initial value of the scale factor. [6] Then, the EOM becomes

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

The initial conditions are $\zeta(0) = 0$ and $\theta(0) = \theta_{\rm ini}$.

This is the same notation as in the code $\Rightarrow$ *You can change it easily.*

[6] Only the ratios $a/a_{\rm ini}$ appear in the calculations.

Using MiMeS

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

```
git clone https://github.com/dkaramit/MiMeS.git
cd MiMeS
git submodule init
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

```
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.

Classes

There are three classes useful to the user. [7]

[7] There are various arguments that need to be passed to the constructors, and the are all listed and explained in the Appendix of the documentation.

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

[9] 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]].

Classes

There are three classes useful to the user. [7]

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020 [8] data. The user can choose another file easily.

[7] There are various arguments that need to be passed to the constructors, and the are all listed and explained in the Appendix of the documentation.

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

[9] 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]].

Classes

There are three classes useful to the user. [7]

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020 [8] 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 [9] of the QCD axion mass.

[7] There are various arguments that need to be passed to the constructors, and the are all listed and explained in the Appendix of the documentation.

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

[9] 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]].

Classes

There are three classes useful to the user. [7]

- `mimes::Cosmo<LD>`: interpolation of relativistic degrees of freedom of the plasma. By default it uses the EOS2020 [8] 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 [9] of the QCD axion mass.
- `mimes::Axion<LD,Solver,Method>`: This is responsible for actually solving the EOM.

---

[7] There are various arguments that need to be passed to the constructors, and the are all listed and explained in the Appendix of the documentation.

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

[9] 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]].

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.

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.
2. $\zeta(0) = 0$. This will be changed in the future.

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.
2. $\zeta(0) = 0$. This will be changed in the future.
3. The energy density of the axion/ALP is always subdominant.

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.
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 $\theta_{\rm ini}$ and f_a, keep in mind that `MiMeS` needs:

What `MiMeS` expects from you

Apart from $\theta_{\rm ini}$ 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 $\theta_{\rm ini}$ 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 $\theta_{\rm ini}$ 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 $\theta_{\rm ini}$ 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 $\theta_{\rm ini}$ 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, related to the algorithm, that might confuse you; *e.g.* temperature at which integration stops no *matter what*!

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:
  - `DormandPrince<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.

MiMeS from python

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).

Examples

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

```python
from time import time; from sys import stderr #you need these in order to print the time in stderr

#add the relative path for MiMeS/src
from sys import path as sysPath; sysPath.append('../src')

from interfacePy.AxionMass import AxionMass #import the AxionMass class
from interfacePy.Axion import Axion #import the Axion class
from interfacePy.Cosmo import mP #import the Planck mass

def main():

    # AxionMass instance
    axionMass = AxionMass(r'../src/data/chi.dat',0,mP)

    # define $\tilde{m}_a{}^2$ for $T \leq T_{\rm min}$
    TMin, chiMin=axionMass.getTMin(), axionMass.getChiMin()

    axionMass.set_ma2_MIN( lambda T,fa: chiMin/fa/fa )

    # define $\tilde{m}_a{}^2$ for $T \geq T_{\rm max}$
    TMax, chiMax=axionMass.getTMax(), axionMass.getChiMax()

    axionMass.set_ma2_MAX( lambda T,fa: chiMax/fa/fa*pow(TMax/T,8.16))

    #in python it is more convenient to use relative paths
    inputFile="../UserSpace/InputExamples/MatterInput.dat"

    ax = Axion(0.1, 1e16, 500, 1e-4, 1e3, 10, 1e-2, inputFile, axionMass,
        1e-2, 1e-8, 1e-2, 1e-10, 1e-10, 0.85, 1.5, 0.85, int(1e7))

    ax.solveAxion()

    print("theta_i=",ax.theta_i,"\t\t\t\t","f_a=",ax.fa,"GeV\n","theta_osc~=",
            ax.theta_osc,"\t","T_osc~=",ax.T_osc,"GeV␣\n","Omega␣h^2=",ax.relic)

    #once we are done we should run the destructor
    del ax,axionMass

if __name__ == '__main__':
    _=time()
    main()
    print(round(time()-_,3),file=stderr)
```

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

```cpp
#include<iomanip>
#include"MiMeS.hpp"

using numeric = long double;//make life easier if you want to change to double

int main(){
    mimes::util::Timer _timer_;//use this to time it!

    // use chi_PATH to interpolate the axion mass.
    mimes::AxionMass<numeric> axionMass(chi_PATH,0,mimes::Cosmo<numeric>::mP);

    /*set \tilde{m}_a^2 for T >= T_max*/
    numeric TMax=axionMass.getTMax(), chiMax=axionMass.getChiMax();

    axionMass.set_ma2_MAX(
        [&chiMax,&TMax](numeric T, numeric fa){ return chiMax/fa/fa*std::pow(T/TMax,-8.16);}
    );

    /*set \tilde{m}_a^2 for T <= T_min*/
    numeric TMin=axionMass.getTMin(), chiMin=axionMass.getChiMin();

    axionMass.set_ma2_MIN(
        [&chiMin,&TMin](numeric T, numeric fa){ return chiMin/fa/fa;}
    );

    /*this path contains the cosmology*/
    std::string inputFile = std::string(rootDir)+
        std::string("/UserSpace/InputExamples/MatterInput.dat");

    /*declare an instance of Axion*/
    mimes::Axion<numeric, 1, RODASPR2<numeric> > ax(0.1, 1e16, 500, 1e-4, 1e3, 10, 1e-2,
                        inputFile, &axionMass, 1e-2, 1e-8, 1e-2, 1e-10, 1e-10, 0.85, 1.5, 0.85,
                        int(1e7) );
    /*solve the EOM!*/
    ax.solveAxion();

    std::cout<<std::setprecision(5)
    <<"theta_i="<<ax.theta_i<<std::setw(25)<<"f_a="<<ax.fa<<" GeV\n"<<"theta_osc~="<<ax.theta_osc
    <<std::setw(20)<<"T_osc~="<<ax.T_osc<<"GeV \n"<<"Omega h^2="<<ax.relic<<"\n";

    return 0;
}
```

Outlook

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" dumped time-dependent pendulum.
- Would be nice if `MiMeS` could handle case of freeze-out/in.
- `MiMeS` should be able to compare against searches on the fly.

Thank you!

Language	files	blank	comment	code
C/C++ Header	35	677	444	1595
C++	20	483	198	1106
Python	22	483	367	1000
SUM:	77	1643	1009	3701

Backup

(equations, derivations, tables)

WKB – I

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

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

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

WKB – II

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

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

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

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

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

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

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,

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

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

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

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:

```
template<class LD>
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 $T_{\rm min} \geq$`minT` and $T_{\rm max} \leq$`maxT`. Beyond these limits hat axion mass is assumed to be constant.

The definition of $\tilde{m}_a{}^2$ beyond $T_{\rm min}$ and $T_{\rm max}$ 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 $T_{\rm min,max}$ and $\chi(T_{\rm min,max})$. These are obtained from

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

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:

```
template<class LD>
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

```
template<class LD, const int Solver, class Method>
mimes::Axion<LD, Solver, Method>(LD theta_i, LD fa, LD umax, LD TSTOP,
            LD ratio_ini, unsigned int N_convergence_max, LD convergence_lim,
            std::string inputFile, AxionMass<LD> *axionMass, LD initial_step_size=1e-2,
            LD minimum_step_size=1e-8, LD maximum_step_size=1e-2,
            LD absolute_tolerance=1e-8, LD relative_tolerance=1e-8, LD beta=0.9,
            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: ~ 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 $[\mathrm{GeV}]$ $\log H$, 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:

```
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 $T_{\rm min}$ and $T_{\rm max}$ 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 $T_{\rm min,max}$ and $\chi(T_{\rm min,max})$. These are obtained from

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

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:

```
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 $\tilde{m}_a{}^2$.

Axion class

The constructor of the `Axion` class is

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

Paths.mk

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

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). [11]

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).

[11] 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).