PROCEEDINGS OF SCIENCE

MiMeS, the Misalignment Mechanism Solver

Dimitrios Karamitros

School of Physics and Astronomy, The University of Manchester, Manchester M13 9PL, United Kingdom

E-mail: dimitrios.karamitros@manchester.ac.uk

We briefly discuss MiMeS, a C++ library that can solve the axion and axion-like equation of motion in various axion models and cosmological histories.

Computational Tools for High Energy Physics and Cosmology (CompTools2021)
22-26 November 2021
Institut de Physique des 2 Infinis (IP2I), Lyon, France

https://pos.sissa.it/

PoS(CompTools2021)018

1. Motivation

One class of promising dark matter candidates are the Axions [1–11] as well as axion-like paricles (ALPs) [12–14], which follow an equation of motion (EOM)

$$\left(\frac{d^2}{dt^2} + 3H(t)\,\frac{d}{dt}\right)\theta(t) + \tilde{m}_a{}^2(t)\;\sin\theta(t) = 0\,, \tag{1}$$

where $\theta = A\,f_a$, with A the axion field, and f_a some energy scale that characterises the potential (Peccei-Quinn breaking scale). The classical analogue is the dumped pendulum with both frequency (length) and friction being time-dependent. That is, there is no closed form solution, no constants of motion until the Hamiltonian starts to vary slowly.

In this proceedings, we present the basic aspects of `MiMeS` [15]. `MiMeS` is designed to solve the EOM (1) numerically; *i.e.* `MiMeS` simulates the evolution of the axion/ALP, for many cosmological scenarios and axion/ALP (thermal) masses. In short, `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. It comes with a `python` interface making it accessible to everyone. `MiMeS` also is easy to use; anyone can run it and see if their model can work or check against the literature. Moreover, the user is free to decide when to start, stop, and when adiabaticity is reached. So, almost every aspect of the numerical solution can be tuned by the user.

2. About `MiMeS`

2.1 Under the hood

`MiMeS` relies on `NaBBODES` [16] for the numerical integration, and `SimpleSplines` [17] for the various interpolations. These are header-only libraries developed and maintained by the author. They are included in the source code of `MiMeS`, which means that `MiMeS` only needs the standard `C++` library. Moreover, their integration with `MiMeS` is guaranteed, since there will always be suitable versions of these libraries that work with the current version of `MiMeS`.

2.2 How to get `MiMeS`

There are several ways one can get a stable version of `MiMeS`. First, via github, by running `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. Moreover, a copy can be downloaded from mimes.hepforge.org/downloads or github.com/dkaramit/MiMeS/releases. In order to get the most up-to-date code – not always the most stable one – including the latest version of `NaBBODES` and `SimpleSplines`, one needs to run the following commands

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

2.3 Configure (and make)

If the user intents to use MiMeS directly from C++ they do not need to install anything in particular. The only requirement is to run

```
bash configure.sh
```

in the root directory of MiMeS. After that, the classes and functions of MiMeS can be used by including the header file MiMeS/MiMeS.hpp. However, there are several examples in MiMeS/UserSpace/Cpp, which can be modified in order to meet the needs of the user. These examples can be compiled by running make examples in the root directory of MiMeS, or simply make in the sub directories inside MiMeS/UserSpace/Cpp.

In order to call MiMeS from python scripts, one needs to compile the shared libraries that translate the C++ classes to C-style functions that can be loaded to python. This can be done simply by running make lib in the root directory of MiMeS.

2.4 Classes

There are three classes useful to the user. The first one that is responsible for the interpolation of relativistic degrees of freedom of the plasma, mimes::Cosmo<LD>. Although this is called internally by others, the user should be aware of its existence, since the file that contains the relevant data (by default, these are taken from [18]) can be changed freely. The class used to define the axion mass as a function of f_a and the temperature is mimes::AxionMass<LD>. MiMeS comes with data from Lattice calculation [19] of the QCD axion mass, but the user can use other data or a simple function. The class that is responsible for actually solving the EOM is mimes::Axion<LD,Solver,Method>. Here, the user is free to use a number of different Runge-Kutta methods, as well control various aspects of how integration happens. The details on how exactly these classes work can be found in ref. [15].

All classes take a template argument labelled LD, which should be double (fast) or long double (accurate). For most cases, the latter is recommended, since θ can be very small especially in cosmologies with entropy injection. Moreover, the other template arguments are

- Solver can be set to 1 for Rosenbrock (semi-implicit Runge-Kutta). The Method argument in this case can be: [1]
  - RODASPR2<LD> (4th order). This is the most accurate one, and the one that is used in all the examples.
  - ROS34PW2<LD> (3rd order). This is generally less accurate, but it can be used for quick estimates.
  - ROS3W<LD> (2rd order). This, generally should be avoided since it is inaccurate.
- Solver can be set to 2 for explicit RK. The Method argument can be:
  - DormandPrince<LD> (7th order). This is an explicit RK, and not very good for this EOM. However, it can still be used.

[1] We should note that the user is free to build their own RK method, as illustrated in [15].

- CashKarpRK45<LD> (5th order). This method should be avoided as it fails to converge in most cases.
- RK45<LD> (5th order). Similarly to the previous one, this also seems to have difficulty converging.

2.5 Calling 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. This compiles the shared libraries that python needs. Before that, the user can change the template arguments and various compilation options. This can be done by changing the various variables in the file MiMeS/Definitions.mk

- LONGpy=long will compile the library with long double numeric types. LONGpy= will compile the library with double numeric types.
- SOLVER and METHOD, are in the template arguments for the C++ case.
- Compiler:
  - CC=g++ in order to use the GNU C++ compiler. This is the compiler used to test MiMeS; *i.e.* should be the preferred choice.
  - CC=clang -lstdc++ in order to use the clang C++ compiler. Sometimes this is faster than g++, but should be used carefully.
- Optimization level:
  - OPT=O0: No optimization. The executables compiled using this option are generally slow compared to the following choices.
  - O=O1, O2, or O3: all these perform mostly the same (read the compiler documentation for more information on the optimization). Generally, the latter is preferred.
  - OPT=Ofast: full optimization. This choice produces the fastest executables. Although it is generally considered dangerous, we have not observed any negative side-effects.

2.6 Assumptions

MiMeS is designed to make as few assumptions as possible. The basic assumption it makes are:

1. $\dot{\theta}(0) = 0$. This means that at high temperatures the kinetic energy dominates, $\dot{\theta}$ is a conserved current; *i.e.* it falls as a^{-3} (a is the scale factor). Therefore, at low temperatures, $\dot{\theta}(0)$ can be assumed to be negligible.
2. $H/\tilde{m}_a$ increases monotonically with the temperature. This is related to the previous assumption. Practically, it is also used to find an appropriate starting integration point.
3. The energy density of the axion/ALP is always subdominant, since the axion/ALP is assumed to be at least a DM component. This ensures that the axion/ALP evolves in some cosmological background. Otherwise, one would need to solve a system of equations that take into account the effect of the axion to the evolution of the rest of Universe (*e.g.* contribution to plasma energy density).

4. Only eq. (1) determines the axion/ALP energy density. That is, there are other sources of axion production.

3. Examples

There are several examples in both C++ and python that can be found in MiMeS/UserSpace. However, it would be useful to append two simple ones here.

3.1 C++ example

In this example, we show how to solve the QCD axion EOM for a matter dominated universe, using $\theta_{\rm ini} = 0.1$ and $f_a = 10^{16}$ GeV. The data for the mass used is given in [19]. The path of this file is stored by in the global variable chi_PATH that is created when we run the configure.sh script. The evolution of the Hubble parameter is solved separately (see *e.g.* ref. [20]), and the corresponding file is MiMeS/UserSpace/InputExamples/MatterInput.dat.

```
#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 m̃_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 m̃_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();

```

```
36    std::cout<<std::setprecision(5)
37    <<"theta_i="<<ax.theta_i<<std::setw(25)<<"f_a="<<ax.fa<<" GeV\n"
38    <<"theta_osc~="<<ax.theta_osc <<std::setw(20)
39    <<"T_osc~="<<ax.T_osc<<"GeV \n"<<"Omega h^2="<<ax.relic<<"\n";
40
41    return 0;
42    }
```

We note here that the axion mass is interpolated between the temperatures that exist in the corresponding file. Beyond these points, we set the mass to follow the functions that are passed as arguments in `set_ma2_MAX` and `set_ma2_MIN`. Here, we choose

$$\tilde{m}_a = \begin{cases} \dfrac{\chi(T_{\rm min})}{{f_a}^2} & \text{for } T < T_{\rm min} \\ \dfrac{\chi(T_{\rm max})}{{f_a}^2}\left(\dfrac{T}{T_{\rm max}}\right)^{-8.16} & \text{for } T > T_{\rm max} \end{cases} . \tag{2}$$

Once the mass is defined according to our model, we move to setting-up and solving the EOM. This is done in lines 30-34. Observe that the constructor of the `mimes::Axion` class takes a lot of arguments. The role of all these arguments is explained in detail in the documentation of `MiMeS` [15]. However, it is important to note that we choose as a starting point the temperature where $H/m_a = 1000$. One can detemine if this is a valid value by taking a larger one and confirming that the end result does not change. Also, we consider the axion to start evolving adiabatically at the point where the relative difference of its adiabatic invariant is less that 10^{-2} between peaks of the oscillation for 10 consecutive times.

Assuming that the code is saved in a subdirectory of the root of `MiMeS`, we can compile this code as

```
1     g++ -O3 -std=c++17 -lm -I../ -o axion example.cpp
```

Then, the executable (`axion`) can be called as `./axion`.

3.2 `python` example

`MiMeS` can also be called from `python` using a script with similar structure. The `python` interface is made as close to the `C++` code as possible. This can be seen by the following example, where we show a `python` script that performs the same calculation as the `C++` example, in a very similar way.

```
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
12      # AxionMass instance
13      axionMass = AxionMass(r'../src/data/chi.dat',0,mP)
14
15      # define m̃_a^2 for T ≤ T_min
16      TMin, chiMin=axionMass.getTMin(), axionMass.getChiMin()
17
18      axionMass.set_ma2_MIN( lambda T,fa: chiMin/fa/fa )
19
20      # define m̃_a^2 for T ≥ T_max
21      TMax, chiMax=axionMass.getTMax(), axionMass.getChiMax()
22
23      axionMass.set_ma2_MAX( lambda T,fa: chiMax/fa/fa*pow(TMax/T,8.16))
24
25      #in python it is more convenient to use relative paths
26      inputFile="../UserSpace/InputExamples/MatterInput.dat"
27
28      ax = Axion(0.1, 1e16, 500, 1e-4, 1e3, 10, 1e-2, inputFile, axionMass,
29            1e-2, 1e-8, 1e-2, 1e-10, 1e-10, 0.85, 1.5, 0.85, int(1e7))
30
31      ax.solveAxion()
32
33      print("theta_i=",ax.theta_i,"\t\t\t\t","f_a=",ax.fa,"GeV\n","theta_osc~=",
34                  ax.theta_osc,"\t","T_osc~=",ax.T_osc,"GeV␣\n","Omega␣h^2=",ax.relic)
35
36      #once we are done we should run the destructor
37      del ax,axionMass
38
39  if __name__ == '__main__':
40      _=time()
41      main()
42      print(round(time()-_,3),file=stderr)
```

In order for this script to work, we should place it in a subdirectory os the root of `MiMeS`. This is the only requirement, assuming that we have already compiled the shared libraries.

4. Outlook

We briefly introduced the basics of `MiMeS`. We have discussed that it can be used to solve the axion/ALP EOM for various underline cosmologies and axion/ALP masses. We have also shown that `MiMeS` is flexible and the user can change almost everything that it needs; *e.g.* the plasma relativistic degrees of freedom, the convergence conditions of the integration, and underline the Runge-Kutta methods used.

However, `MiMeS` can be extended in the future, in order to make it even more general. For example, `MiMeS` can be generalized to allow the user to consider different initial value of $\dot{\theta}$. `MiMeS` can, conceivably, handle non-vanishing RHS; *i.e.* solve the "driven" dumped time-dependent pendulum. Another interesting addition would be to be able to compare against searches on the fly, by introducing lists of constraints from various experiments and observations.

References

[1] R. D. Peccei and H. R. Quinn, *CP Conservation in the Presence of Instantons*, *Phys. Rev. Lett.* **38** (1977) 1440–1443.

[2] S. Weinberg, *A New Light Boson?*, *Phys. Rev. Lett.* **40** (1978) 223–226.

[3] F. Wilczek, *Problem of Strong P and T Invariance in the Presence of Instantons*, *Phys. Rev. Lett.* **40** (1978) 279–282.

[4] J. Preskill, M. B. Wise, and F. Wilczek, *Cosmology of the Invisible Axion*, *Phys. Lett. B* **120** (1983) 127–132.

[5] M. Dine and W. Fischler, *The Not So Harmless Axion*, *Phys. Lett. B* **120** (1983) 137–141.

[6] L. F. Abbott and P. Sikivie, *A Cosmological Bound on the Invisible Axion*, *Phys. Lett. B* **120** (1983) 133–136.

[7] Z. G. Berezhiani and M. Y. Khlopov, *Cosmology of Spontaneously Broken Gauge Family Symmetry*, *Z. Phys. C* **49** (1991) 73–78.

[8] Z. G. Berezhiani, A. S. Sakharov, and M. Y. Khlopov, *Primordial background of cosmological axions*, *Sov. J. Nucl. Phys.* **55** (1992) 1063–1071.

[9] A. S. Sakharov and M. Y. Khlopov, *The Nonhomogeneity problem for the primordial axion field*, *Phys. Atom. Nucl.* **57** (1994) 485–487.

[10] A. S. Sakharov, D. D. Sokoloff, and M. Y. Khlopov, *Large scale modulation of the distribution of coherent oscillations of a primordial axion field in the universe*, *Phys. Atom. Nucl.* **59** (1996) 1005–1010.

[11] M. Y. Khlopov, A. S. Sakharov, and D. D. Sokoloff, *The nonlinear modulation of the density distribution in standard axionic CDM and its cosmological impact*, *Nucl. Phys. B Proc. Suppl.* **72** (1999) 105–109.

[12] Y. Chikashige, R. N. Mohapatra, and R. D. Peccei, *Are There Real Goldstone Bosons Associated with Broken Lepton Number?*, *Phys. Lett. B* **98** (1981) 265–268.

[13] H. M. Georgi, L. J. Hall, and M. B. Wise, *Grand Unified Models With an Automatic Peccei-Quinn Symmetry*, *Nucl. Phys. B* **192** (1981) 409–416.

[14] A. Ringwald, *Axions and Axion-Like Particles*, in *49th Rencontres de Moriond on Electroweak Interactions and Unified Theories*, pp. 223–230, 2014. arXiv:1407.0546.

[15] D. Karamitros, *MiMeS: Misalignment Mechanism Solver*, arXiv:2110.12253.

[16] D. Karamitros, *NaBBODES: Not a black box ordinary differential equation solver in C++*, 2019–.

[17] D. Karamitros, *SimpleSplines: A header only library for linear and cubic spline interpolation in C++.*, 2021–.

[18] K. Saikawa and S. Shirai, *Precise WIMP Dark Matter Abundance and Standard Model Thermodynamics*, *JCAP* **08** (2020) 011, [arXiv:2005.03544].

[19] S. Borsanyi et al., *Calculation of the axion mass based on high-temperature lattice quantum chromodynamics*, *Nature* **539** (2016), no. 7627 69–71, [arXiv:1606.07494].

[20] P. Arias, D. Karamitros, and L. Roszkowski, *Frozen-in fermionic singlet dark matter in non-standard cosmology with a decaying fluid*, *JCAP* **05** (2021) 041, [arXiv:2012.07202].