Muesr Documentation

Release 0.1.2

Pietro Bonfa'

Mar 15, 2018

Contents

MuESR (Magnetic structure and mUon Embedding Site Refinement) is a tool to identify muon sites and analize local fields for a given magnetic structure quickly and effectively.

The code is distributed as a library. Basic python skills are required to proficiently use the code.

Contents:

CHAPTER 1

Basic theory

1.1 Local fields in MuSR

Muon spin rotation and relaxation spectroscopy is mainly used to probe magnetic materials. We briefly describe here the interactions between the magnetic moments of the muon and the electrons in the host system that produce a local magnetic field at the muon site in magnetically ordered samples.

1.1.1 Dipolar Field

The dipolar field is produced by the magnetic dipolar interaction between the spin polarized electronic orbitals and the muon spin. Even though the interaction is best described with quantum mechanics, for the sake of simplicity, here we approximate the spin polarized electronic orbitals with classical dipoles centered at the nuclei of the magnetic atoms. This approximation is also implicit in the code and works rather well in many cases.

The dipolar field is given by

$$\mathbf{B}'_{\mathrm{dip}} = \frac{\mu_0}{4\pi} \sum_{i=1}^{N} \left(-\frac{\mathbf{m}}{r_i^3} + \frac{3(\mathbf{m}_i \cdot \mathbf{r}_i)\mathbf{r}_i}{r_i^5} \right)$$

where, from a quantum perspective, $\mathbf{m}_i = -g_i \mu_{\mathrm{B}} \mathbf{J}_i$ and $\mathbf{J}_i$ is the total angular momentum of the i-th atom. Finally, the radius $\mathbf{r}_i$ is the distance between the muon and the i-th atom of N magnetic ions in the sample.

When the above sum is performed in real space, it is customary to select a spherical portion of the sample (smaller than a magnetic domain) centered at the muon site and subdivide $\mathbf{B}_{\mathrm{dip}}$ in three terms:

$$\mathbf{B}'_{\mathrm{dip}} = \mathbf{B}_{\mathrm{dip}} + \mathbf{B}_{\mathrm{Lor}} + \mathbf{B}_{\mathrm{dem}}$$

The first term originates from the magnetic moments inside the sphere of radius R_{sphere}, i.e.:

$$\mathbf{B}_{\mathrm{dip}} = \frac{\mu_0}{4\pi} \sum_{r_i < R_{\mathrm{sphere}}} \left(-\frac{\mathbf{m}}{r_i^3} + \frac{3(\mathbf{m}_i \cdot \mathbf{r}_i)\mathbf{r}_i}{r_i^5} \right)$$

The second and the term originate from magnetic moments outside the sphere and are evaluated in the continuum approximation. They are

$$\mathbf{B}_{\mathrm{Lor}} = \frac{\mu_0}{3}\mathbf{M}_{\mathrm{Lor}} = \frac{\mu_0}{3V_{\mathrm{sphere}}} \sum_{r_i < R_{\mathrm{sphere}}} \mathbf{m}_i$$

$$\mathbf{B}_{\mathrm{dem}} = -\mu_0 \mathbf{N}\mathbf{M}_{\mathrm{meas}}$$

where $\mathbf{N}$ is the demagnetization tensor and $\mathbf{M}_{\mathrm{meas}}$ is the **bulk** magnetization of the sample.

Note: *muesr* only estimates $\mathbf{B}_{\mathrm{dip}}$ and $\mathbf{B}_{\mathrm{Lor}}$. The demagnetisation field depends on both the sample shape and the experiment conditions and it must be evaluated case by case.

1.1.2 Contact Hyperfine field

A distinct contribution to the local magnetic field at the muon site is referred to as Fermi contact hyperfine field. It accounts for the finite probability for the quantum electron with wavefunction $\psi_s(\mathbf{r})$ to share the classical muon position $\mathbf{r}_\mu$ and it amounts to

$$\mathbf{B}_{\mathrm{cont}} = \frac{2\mu_0}{3}|\psi_s(\mathbf{r}_\mu)|^2 \mathbf{m}_e^s$$

In *muesr*, only a scalar coupling between $\mathbf{B}_{\mathrm{cont}}$ and $\mathbf{m}_e$ is allowed, proportional to $|\psi_s(\mathbf{r}_\mu)|^2$.

In principle the quantum nature of the contact hyperfine interaction requires the knowledge of the electronic distribution around the muon site for an accurate description. Each magnetic atom that is a muon neighbor may contribute with a different coupling value, while the current version of *muesr* allows for just one average value. Furthermore the coupling may produce contributions from one or more neighbor magnetic atoms. They add up differently, depending to the magnetic structure.

At the moment this is only implemented by varying the number of nearest neighbours considered in the above sum. It is **very badly** approximated by considerig that each magnetic atom within a given radius contributes to the total hyperfine field by an amount inversely proportional to the cube of its distance from the muon. The total is then scaled by the common factor ACont.

[TODO]

Improve the implementation of an effective contact interaction in muesr!!!!

1.2 Description of Magnetic Structures

There are two possibilities to describe a magnetic structure: by using the color (Shubnikov) group theory or by defining one (or more) propagation vector(s) and using the Fourier coefficients formalism. *muesr* opts for the latter, limited to single wavevector (1-k) structures for the time being. A magnetic structure is defined as

$$\mu_{\mathbf{n}\nu} = \sum_{\mathbf{k}} \mathbf{m}_{\nu\mathbf{k}} e^{-2\pi i \mathbf{k}\cdot\mathbf{R}_n}$$

where ν runs over the atoms of the unit cell and n identifies the n-th cell where atomic positions $\mathbf{R}_{n\nu}$ are obtained according to

$$\mathbf{R}_{n\nu} = \mathbf{R}_n + \mathbf{r}_\nu$$

with $\mathbf{R}_n = n_a\mathbf{a} + n_b\mathbf{b} + n_c\mathbf{c}$ and $\mathbf{r}_\nu = x_\nu\mathbf{a} + y_\nu\mathbf{b} + z_\nu\mathbf{c}$.

The fourier coefficients $m_{\nu\mathbf{k}}$ are three dimensional complex vectors. They are related to the irreducible representations of the so called "little groups" i.e. the subgroup of the crystallographic space group formed by the operators leaving invariant the propagation vector.

[TODO] Discuss the phase!

As we said *muesr* can only handle 1-k magnetic structures. However, since local field are linear in the magnetic moment, the results for multiple-k magnetic orders can be obtained by performing multiple simulations for each of the k vectors and Fourier components which describe the system and summing the results.

1.3 Implementation details

muesr is a tool to analyze muon sites and local field contributions generated by a known magnetic structure. It is intended to be used in an interactive python environment such as IPython or Jupyter notebooks.

Internally, muesr uses Tesla units and Angstrom for lengths if not specified. Magnetic moments are specified in units of Bohr magnetons.

CHAPTER 2

Installing Muesr

This section provides an overview and guidance for installing Muesr on various target platforms.

2.1 Prerequisites

Muesr relies on some 3rd party packages to be fully usable and to provide you full access to all of its features.

You must have at least the following Python packages installed:

- Python 2.7, 3.1+ (http://www.python.org)
- Numpy 1.6.0+ (http://www.numpy.org)
- mulfc (http://github.com/bonfus/muLFC)

Other Python versions or Python implementations might work, but are (currently) not officially tested or supported.

Additionally, you will need the following packages and libraries to be installed to use *all* features of Muesr:

Package	Version	Required for	Package URL
YAML	>= 2.0.0	*muesr.i_o.sampleIO*	http://pyyaml.org/
Spglib	>= 1.6	*muesr.utilities.symsearch*	http://atztogo.github.io/spglib/
Sympy	>= 1.0	`muesr.core.magmodel.SMM`	http://sympy.org
appdirs	>= 1.1	`muesr.settings`	
XCrysDen	>= 1.0	`muesr.i_o.xsf.xsf.show_cell` , `muesr.i_o.xsf.xsf.show_supercell`	http://www.xcrysden.org

Note: The muesr distribution ships with a internal version of appdirs which, however, may not be up to date.

2.2 System-wide installation

The installation with *pip* is as symple as

```
pip install -r requirements.txt muesr
```

2.3 Installation in virtualenv

Virtualenv offers a simple way of virtualizing the Python environment. This means that you can have a separate collection of python packages for running Muesr (and install Muesr itself) without affecting the Python installation system-wide.

To install Muesr in a virtualenv, first make sure that the command *virtualenv* is available on your system. If not, please check online what is the recommended way of installing virtualenv for your operation system.

To create the virualenv run in a terminal:

```
virtualenv muesr-env
```

and to activate the environment (Linux and OsX)

```
cd muesr-env
source bin/activate
```

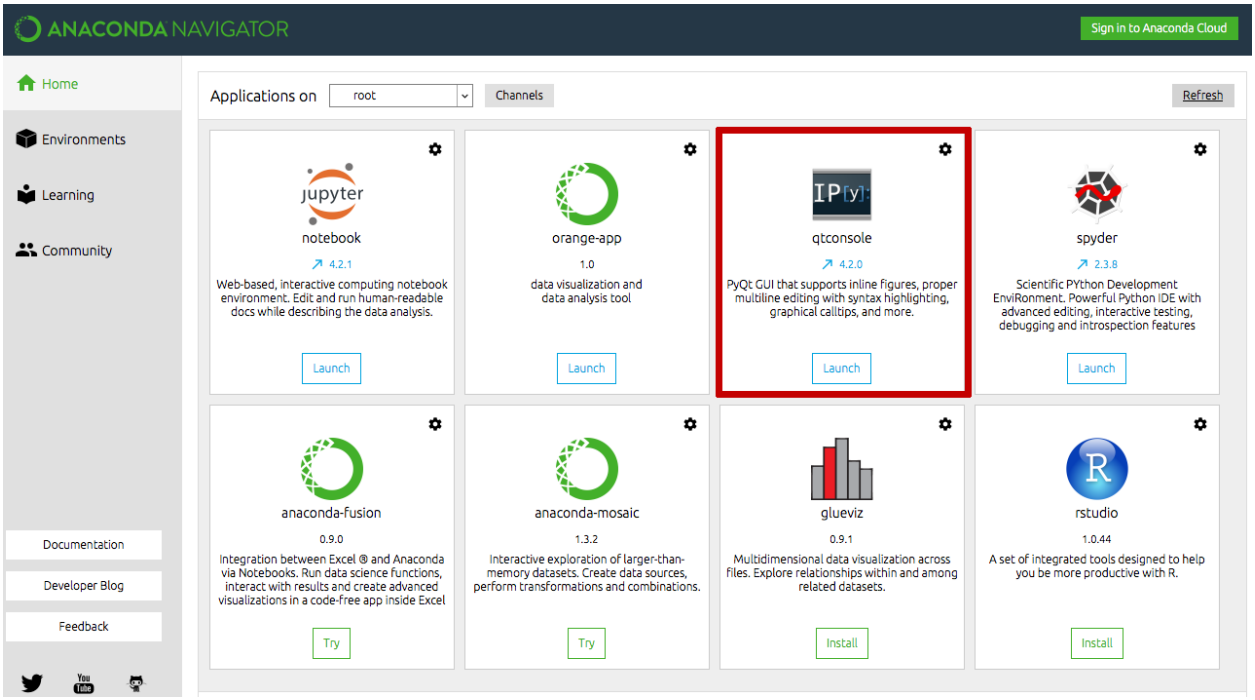
now you can install mulfc and Muesr in the virtualenv with the same commands reported above

```
pip install -r requirements.txt muesr
```

2.4 A few notes for Windows users

In order to install *muesr* on Windows you need a working python environment. The best user experience is probably provided by Anaconda, which is a complete Python distribution for scientific data analysis. The following steps assume that a working version of Anaconda is available on the target system.

Start Anaconda navigator and open an interactive python terminal:



From within the interactive terminal do:

```
import pip
pip.main("install mulfc spglib pyyaml muesr".split())
```

Now you are ready to go! Why not start with a look at the first paragraph of the *Tutorial* and then move directly to the Muesr *Examples*?

2.5 Compilation from source and system-wide installation

This is the hard way, but allows to custmize some parts of the installation. In order to compile the python extension you also need the build tools appropriate for your system (gcc on Linux, XCode on OS X, Visual Studio or gcc on Windows). To install the packages you'll need to be superuser.

Use git to clone Muesr and muLFC projects. First install *muLFC*

```
git clone https://github.com/bonfus/muLFC.git
cd muLFC
python setup.py install
```

Next install *muesr* (and possibly optional requirements)

```
# optional, but suggested:
pip install spglib pyyaml
#
pip install muesr
```

CHAPTER 3

Tutorial

This tutorial describes how to use the *muesr* API to obtained the local magnetic field at a specific interstitial muon site in a magnetic sample.

To proficiently use *muesr* a basic knowledge of python is strongly suggested. There are plenty of tutorial out there in the web, pick one and get familiar with the basic python syntax before going forward.

To use *muesr* you must be familiar with python. An interactive shell like ipython or jupyter can help a lot but it is not needed.

To be pedantic, you can

1. Run the commands listed below in an ipython console
2. Write these commands in a example.py file and run it with python
3. Use the muesr gui (which implies having what?)
4. Use Mantid, that contains a muesr library

3.1 First steps with muesr

Find below a tutorial description of the main functions. An alternative way to familiarise with muer is to run directly the *Examples* first and then come back here for a more systematic introduction.

3.1.1 Defining the sample

The fundamental component of Muesr is the *Sample* object. You can import and initialize it like this:

```
>>> from muesr.core import Sample
>>>
>>> mysample = Sample()
```

3.1.2 Specifying the lattice structure

The first thing that must be defined in a sample object is the lattice structure and the atomic positions. Muesr uses a custom version of the ASE *Atoms* class to do so. The following code defines and add the the lattice structure of simple cubic Iron to the sample object:

```
>>> import numpy
>>> from muesr.core.atoms import Atoms
>>>
>>> atms = Atoms(symbols=['Fe'], scaled_positions=[[0.,0.,0.]], cell=numpy.diag([3.,3.
↪,3.]), pbc=True)
>>>
>>> mysample.cell = atms
```

However this procedure is quite tedious and error prone so it is much better to use builtin functions to parse crystallographic files.

At the moment muesr can parse crystallographic data from CIF (cif) files and XCrysDen (xsf) files. Here's an example:

```
>>> # load data from XCrysden *.xsf file
>>> from muesr.i_o import load_xsf
>>>
>>> load_xsf(mysample, "/path/to/file.xsf")
>>>
>>>
>>> # load data from *.cif file
>>> from muesr.i_o import load_cif
>>>
>>> load_cif(mysample, "/path/to/file.cif")
>>>
>>>
```

The *load_cif()* function will also load symmetry information. Please note that **only a single lattice structure at a time** can be defined so each load function will remove the previous lattice structure definition.

3.1.3 Setting muon positions

When the lattice structure is defined it is possible to specify the muon position and the magnetic orders.

To specify the muon position, just do:

```
>>> mysample.add_muon([0.1,0,0])
```

positions are assumed to be in fractional coordinates. If Cartesian coordinates are needed, they can be specified as

```
>>> mysample.add_muon([0.3,0,0], cartesian=True)
```

You can verify that the two positions are equivalent by printing them with the command

```
>>> print(mysample.muons)
[array([ 0.1,  0. ,  0. ]), array([ 0.1,  0. ,  0. ])]
```

If symmetry information are present in the sample definition, it symmetry equivalent muon sites can be obtained. This can be done with the utility function *muon_find_equiv()*. In our case we did not load any symmetry information so the following command will raise an error. You can check that by doing

```
>>> from muesr.utilities import muon_find_equiv
>>> muon_find_equiv(mysample)
[...]
SymmetryError: Symmetry is not defined.
```

3.1.4 Defining a magnetic structure

The next step is the definition of a magnetic structure. To do so one must specify the propagation vector and the Fourier components and, optionally, the phases. A quick way to do that is using the helper function *mago_add()* from *ms*.

```
>>> from muesr.utilities.ms import mago_add
>>>
>>> mago_add(mysample)
```

You will be asked the propagation vector and the Fourier coefficients for the specified atomic symbol. By default the Fourier components are specified in **Cartesian** coordinates. You can use the keyword argument *inputConvention* to change this behavior (see *mago_add()* documentation for more info). Here's an example:

```
>>> mago_add(a)
   Propagation vector (w.r.t. conv. rec. cell): 0 0 0
   Magnetic moments in Bohr magnetons and Cartesian coordinates.
   Which atom? (enter for all)Fe
   Lattice vectors:
       a    3.000000000000000    0.000000000000000    0.000000000000000
       b    0.000000000000000    3.000000000000000    0.000000000000000
       c    0.000000000000000    0.000000000000000    3.000000000000000
   Atomic positions (fractional):
       1 Fe  0.00000000000000  0.00000000000000  0.00000000000000  63.546
   FC for atom 1 Fe (3 real, [3 imag]): 0 0 1
```

The same can be achieved without interactive input like this:

```
>>> mysample.new_mm()
>>> mysample.mm.k = numpy.array([ 0.,  0.,  0.])
>>> mysample.mm.fc = numpy.array([[ 0.+0.j,  0.+0.j,  1.+0.j]])
>>> mysample.mm.desc = "FM m//c"
```

Note: In this method each atom must have a Fourier component! For a 8 atoms unit cell the numpy array specifying the value must be a 8 x 3 complex array!

It is possible to specify multiple magnetic structure for the same lattice structure. **Each time a new magnetic structure is added to the sample object it is immediately selected for the later operations**. The currently selected magnetic order can be checked with the following command:

```
>>> print(mysample)
Sample status:

Crystal structure:           Yes
Magnetic structure:          Yes
Muon position(s):            2 site(s)
Symmetry data:               No

Magnetic orders available ('*' means selected)
```

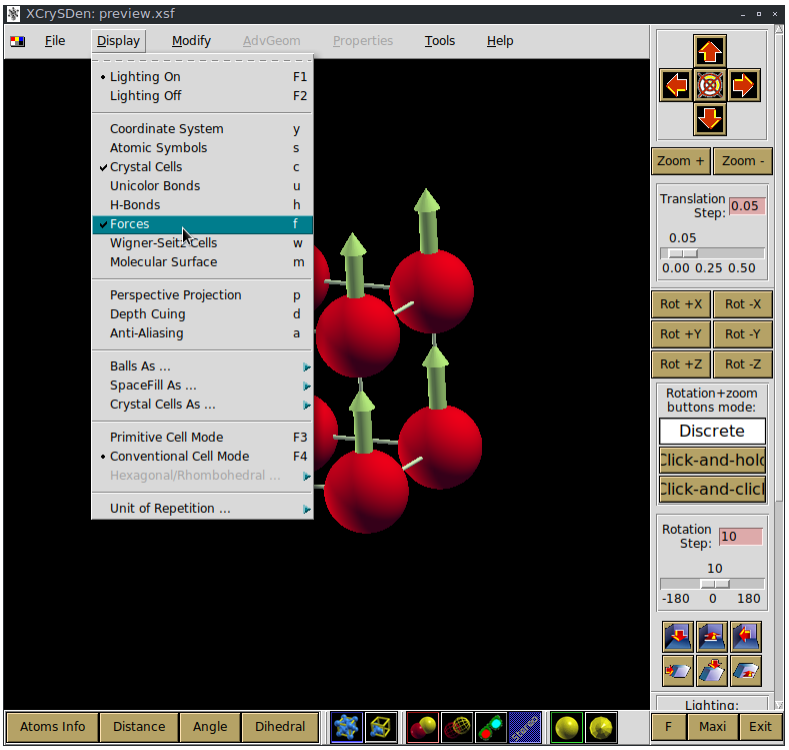
```
Idx  | Sel  | Desc.
 0   |      | No title
 1   |  *   | FM m//c
```

3.1.5 Checking the magnetic structure

The magnetic structures already defined can be visualized with the XCrysDen software.

```
>>> from muesr.utilities import show_structure
>>> show_structure(mysample)
```

the interactive session will block until XCrysDen is in execution. To show the local moments on Iron atoms press the 'f' key or 'Display -> Forces'.



To procede with the tutorial close the XCrysDen Window.

3.1.6 Evaluating the local field

Once you are done with the definition of the sample details it's time to crunch some numbers! To evaluate the local fields at the muon site *muesr* uses a python extension written in C in order to get decent performances. You can load a simple wrapper to the extension as providing local fields with the following command

```
>>> from muesr.engines.clfc import locfield
```

A detailed description of the possible computations is given in the muLFC documentation.

Let's go straight to the local field evaluation which is obtained by running the command:

```
>>> results = locfield(mysample, 'sum', [30, 30, 30] , 40)
```

The first argument is just the sample object that was just defined. The second and third argument respectively specify that a simple *sum* of all magnetic moments should be performed using a supercell obtained replicating *30x30x30 times* the unit cell along the lattice vectors. The fourth argument is the radius of the Lorentz sphere considered. All magnetic moments outside the Lorentz sphere are ignored and the muon is automatically placed in the center of the supercell.

Note: To get an estimate of the largest radius that you can use to avoid sampling outside the supercell size you can use the python function *find_largest_sphere* in the LFC python package.

Warning: If the Lorentz sphere does not fit into the supercell, the results obtained with this function are not accurate!

The *results* variable now contains a list of `LocalField` objects. However, if you print the *results* variable you'll see something that looks like a numpy array:

```
>>> print(results)
[array([  3.83028907e-18,  -3.37919319e-18,  -3.42111893e+01]),
 array([  3.83028907e-18,  -3.37919319e-18,  -3.42111893e+01])]
```

these are the **total field** for the muon positions and the magnetic structure defined above. To access the various components you do:

```
>>> results[0].Lorentz
array([ 0.        ,  0.        ,  0.14355877])

>>> results[0].Dipolar
array([  3.83028907e-18,  -3.37919319e-18,  -3.43547481e+01])

>>> results[0].Contact
array([ 0.,  0.,  0.])
```

And you are done! Remember that all results are in Tesla units.

3.1.7 Saving for later use

The current sample definition can be stored in a file with the following command:

```
>>> from muesr.i_o import save_sample
>>> save_sample(mysample, '/path/to/mysample.yaml')
```

and later loaded with

```
>>> from muesr.i_o import load_sample
>>> mysample_again = load_sample('/path/to/mysample.yaml')
```

CHAPTER 4

Examples

The following examples provide a broad introduction to MuESR.

4.1 LiFePO4

In this example we will go through the relevant lines of the script *run_example.py* in the *LiFePO4* directory of the examples. This example shows how to calculate the local fields at the muon sites in LiFePO4 and loosely follows the description of Ref. *[Sugiyama2011]*.

4.1.1 Interactive magnetic order definition

In this section we will prepare a script that asks the user to specify the magnetic order at runtime and prints the output on screen.

Let's first import the necessary python objects and functions of `muesr`.

```
11 import numpy as np # to calculate the norm
12 import os          # join path on windows.
13
14 from muesr.core import Sample            # this object contains all the info on our␣
   ↪sample
15 from muesr.i_o import load_cif            # loads lattice and symmetry from CIF file
16 from muesr.utilities import  mago_add    # this function provides a CLI for inserting␣
   ↪the MAGnetic Order
17 from muesr.engines.clfc import locfield  # this is the function which actually␣
   ↪performs the sum and returns
```

To create a new sample definition and initialize it, just do:

```
28 smpl = Sample()
```

The sample object holds the following information:

- Lattice structure
- Magnetic Orders
- Muon positions
- Symmetry (optional)

The lattice structure is orthorhombic, with *Pnma* symmetry and lattice parameters $a = 10.3244(2), b = 6.0064(3), c = 4.6901(5)$. We can import these data from a CIF file using the function `load_cif()`:

```
load_cif(smpl, os.path.join('.','cifs','4001848.cif'))
```

The next step is the definition of the magnetic order. This can be done interactively during the script execution or programmatically.

Let's discuss the former case first. The function `mago_add()` will let you specify the Fourier components and the propagation vector.

We want to describe LiFePo anti-ferromagnetic order in which the iron moments, 4.19 μ_B in magnitude, lie along the y axis. We do this by defining a *ferromagnetic order* (propagation vector **k** = 0), with a basis of four moments. If you run the code the full output below allows you to recognize the input that you have to provide fromthe function prompts.

```
mago_add(smpl)

# the interactive session of the above command should be like this:
#     Propagation vector (w.r.t. conv. rec. cell): 0 0 0
#     Magnetic moments in bohr magnetons and cartesian coordinates.
#     Which atom? (enter for all)Fe
#     Lattice vectors:
#       a   10.324400000000001    0.000000000000000    0.000000000000000
#       b    0.000000000000000    6.006400000000000    0.000000000000000
#       c    0.000000000000000    0.000000000000000    4.690100000000000
#     Atomic positions (fractional):
#         1 Fe  0.28220100000000  0.25000000000000  0.97474000000000  55.845
#         2 Fe  0.21779900000000  0.75000000000000  0.47474000000000  55.845
#         3 Fe  0.71779900000000  0.75000000000000  0.02526000000000  55.845
#         4 Fe  0.78220100000000  0.25000000000000  0.52526000000000  55.845
#     FC for atom 1 Fe (3 real, [3 imag]): 0 4.19 0
#     FC for atom 2 Fe (3 real, [3 imag]): 0 -4.19 0
#     FC for atom 3 Fe (3 real, [3 imag]): 0 -4.19 0
#     FC for atom 4 Fe (3 real, [3 imag]): 0 4.19 0
```

In order to complete the setup we specify the muon positions. The method `add_muon` inputs lattice (fractional) coordinates by default. Four muon sites are discussed by Jun Sugiyama *et al.* *[Sugiyama2011]*.

```
smpl.add_muon([0.1225,0.3772,0.8679])
smpl.add_muon([0.0416,0.2500,0.9172])
smpl.add_muon([0.3901,0.2500,0.3599])
smpl.add_muon([0.8146,0.0404,0.8914])
```

Note: The muon sites will be automatically placed in the central unit cell of the supercell that will be used for the subsequent calculations.

The local fields are finally calculated with the `locfield()` command.

```
#            sample   type of calculation     supercell    Lorentz radius
r = locfield( smpl,          's',           [100,100,100],       40)
```

Warning:

- Always check convergence against the supercell size.
- Always use a Lorentz sphere that can be inscribed in the selected supercell.

Please see the documentation of the function `locfield()` to see a description of the input parameters. As it is immediately evident, the supercell size and the Lorentz radius are not the ideal choices. Improving this parameters is left as an exercise to the reader.

The results are stored in a list of `LocalField` objects which, for each muon site, contain the total, dipolar, Lorentz and Contact contributions in Tesla (in the present case, an antiferromagnet in zero external field, with no Fermi contact term, only the dipolar field is obtained).

The next four lines of code print the results of the simulation in T/μ_B

```
print(r[0].T/4.19, "Norm {: 0.4f}".format(np.linalg.norm(r[0].T/4.19)))
print(r[1].T/4.19, "Norm {: 0.4f}".format(np.linalg.norm(r[1].T/4.19)))
print(r[2].T/4.19, "Norm {: 0.4f}".format(np.linalg.norm(r[2].T/4.19)))
print(r[3].T/4.19, "Norm {: 0.4f}".format(np.linalg.norm(r[3].T/4.19)))
```

You should now see the following self explaining result

```
(array([-0.15540174, -0.12234256, -0.02399385]), 'Norm  0.1992')
(array([ -1.32075572e-17,  -1.24059244e-01,  -1.10237957e-19]), 'Norm  0.1241')
(array([ -5.22880379e-18,  -1.80946551e-01,   3.16831013e-18]), 'Norm  0.1809')
(array([-0.1333684 , -0.11733706, -0.03497624]), 'Norm  0.1810')"
```

These are the Cartesian components and modulus of the local field at the four Sugiyama sites, in Tesla.

Note: **The results are always** reported in the **Cartesian** coordinate system defined by the lattice vectors of the crystal.

4.1.2 Programmatic magnetic order definition

As already mentioned above, it's also possible to specify a magnetic order programmatically. This can be done with the help of the methods *new_mm*, *k* and *fc*.

```
# create a new magnetic model for the sample
smpl.new_mm()

# The new magnetic order is automatically selected.
# One can obtain the current magnetic model with the
#  property "mm". E.g. smpl.mm.k prints the k-vector, etc.

# Set a description for the magnetic order
smpl.mm.desc = "Ferromagnetic"
# Remember: anti-ferro = ferro with a bassis

# The smpl.mm.k property can also set the propagation vector,
```

```
#  in reciprocal lattice units (r.l.u.).
smpl.mm.k=np.array([0.,0.,0.])

# Now define Fourier components (in CARTESIAN coordinates and Bohr magnetons);
#  the components must be set for all the atoms in a cell (28 in the present case).
FCs = np.array([[ 0.00+0.j,  4.19+0.j,  0.00+0.j],
        [ 0.00+0.j, -4.19+0.j,  0.00+0.j],
        [ 0.00+0.j, -4.19+0.j,  0.00+0.j],
        [ 0.00+0.j,  4.19+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j],
        [ 0.00+0.j,  0.00+0.j,  0.00+0.j]])

# Set the Fourier components, by default in Cartesian coordinates.
```

Please remember to specify the FC in a 2D array with 3 columns and N_{Atoms} rows of complex values.

Note: The **propagation vector** is **always** specified in **reciprocal lattice units**. On the other hand, the **Fourier components** can be specified with **three different coordinate system and units**:

1. Bohr magnetons in Cartesian coordinates (Cartesian vector notation)
2. Bohr magnetons/Angstrom along the lattice vectors (Lattice vector notation)
3. Modulus (in Bohr magnetons) along the lattice vectors.

The results can be retrieved as described above.

4.2 Fe BCC

In this example the local field at the tetrahedral site(s) in bcc-Fe is calculated.

```
In [1]: import numpy as np
        from muesr.core.sample import Sample             # Retains all the sample info.
        from muesr.i_o.cif.cif import load_cif           # For loading the structure from cif
```

```
from muesr.utilities import mago_add, show_structure   # For magnetic structure description a
from muesr.utilities import muon_find_equiv            # For finding and including the symmet

from muesr.engines.clfc import locfield, find_largest_sphere # Does the sum and returns the

np.set_printoptions(suppress=True,precision=5)
```

Create a `Sample` instance and load lattice structure from a CIF file.

```
In [2]: fe=Sample()
        load_cif(fe,"./Fe.cif");
```

Add the muon position

```
In [3]: fe.add_muon([0.50,0.25,0.0]);
```

and find the symmetry equivalent positions. In BCC Fe there are 12 symmetry equivalent sites for the position provided above. These are automatically added with the following command.

```
In [4]: muon_find_equiv(fe);
        fe

Out[4]: Sample status:

         Crystal structure:           [92mYes[0m
         Magnetic structure:          [93mNo[0m
         Muon position(s):            [92m12 site(s)[0m
         Symmetry data:               [92mYes[0m
```

The definition of the magnetic structure of Fe is created with `fe.new_mm()` (new magnetic model). The experimental magnetic moment at the Fe atoms is 2.22 Bohr magneton and the propagation vector is **0** (ferromagnetic order).

```
In [5]: fe.new_mm()
        fe.mm.k=np.array([0.0,0.0,0.0])
        fe.mm.fc= np.array([[0.0+0.j, 0.0+0.j, 2.22+0.j],
                            [0.0+0.j, 0.0+0.j, 2.22+0.j]])
```

There is another way to define the magnetic structure especially useful when using the interactive session:

```
mago_add(fe)
```

It should appear like this on the interactive screen

```
>>> mago_add(fe)
... Propagation vector (w.r.t. conv. rec. cell): 0.0 0.0 0.0
... Magnetic moments in bohr magnetons and cartesian coordinates.
... Which atom? (enter for all)Fe
... Lattice vectors:
...    a    2.868018200000000    0.000000000000000    0.000000000000000
...    b    0.000000000000000    2.868018200000000    0.000000000000000
...    c    0.000000000000000    0.000000000000000    2.868018200000000
... Atomic positions (fractional):
...      1 Fe  0.00000000000000  0.00000000000000  0.00000000000000  55.845
...      2 Fe  0.50000000000000  0.50000000000000  0.50000000000000  55.845
... FC for atom 1 Fe (3 real, [3 imag]): 0.00 0.00 2.22
... FC for atom 2 Fe (3 real, [3 imag]): 0.00 0.00 2.22
```

The following commands may be used to visualize the lattice structure, the muon position and the magnetic order with XCrysDen.

```
In [6]: #show_structure(fe, [2,2,2]);
```

The function `find_largest_sphere` can be used to find the largest sphere with center at the muon site(s) that can be inscribed in a 100x100x100 supercell.

```
In [7]: radius=find_largest_sphere(fe,[100, 100, 100])
```

With the following call, the local field at the muon sites is finally evaluated. The first argument is the sample object, the second argument spcifies that a simple sum should be performed, the third argument is the supercell dimension along the three lattice vectors and the last argument is the radius of the Lorentz sphere.

```
In [8]: r=locfield(fe, 's', [100, 100, 100] ,radius)
```

All the local fied contributions are contained in r, the dipolar contribution in `r[i].D`, the Lorentz contribution in `r[i]`.L, the Fermi contact contribution in `r[i].C`, and the sum of all contributions in `r[i].T` where `i` runs on the muon sites.

If `r[i].ACont` (isotropic contact coupling term) is not defined `r[i].C` is zero. For the details on the contact coupling term see the documentation (http://muesr.readthedocs.io/en/latest/ContactTerm.html)

```
In [9]: B_dip=np.zeros([len(fe.muons),3])
        B_Lor=np.zeros([len(fe.muons),3])
        B_Cont=np.zeros([len(fe.muons),3])
        B_Tot=np.zeros([len(fe.muons),3])

        for i in range(len(fe.muons)):
            B_dip[i]=r[i].D
            B_Lor[i]=r[i].L
            r[i].ACont = 0.0644
            B_Cont[i]=r[i].C
            B_Tot[i]=r[i].T
```

As discussed in M. Schmolz et.al (Hyperfine Interactions 31 (1986) 199-204), in BCC Fe the muon jumps between the tetrahedral sites and "the field contribution at each equivalent site is either parallel or antiparallel to the magnetization of the domains" such that B_dip(parallel)=-2B_dip(antiparallel) [...] the average of the dipolar field at these three sites vanishes"

```
In [10]: print("Dipolar Field for all the 12 tetrahedral equivalent sites")
         print(B_dip)

         # This is and should be same for all the equivalent sites
         print("The Lorentz field is {:4.3f} {:4.3f} {:4.3f}".format(*tuple(B_Lor[0])))

         print("The contact field is {:4.3f} T".format(np.linalg.norm(B_Cont[0])))

         print("Dipolar average of 1 parallel site and 2 antiparallel sites is {:4.5f} T".format(np.
```

```
Dipolar Field for all the 12 tetrahedral equivalent sites
[[ 0.       0.       0.26498]
 [-0.      -0.       0.26498]
 [-0.       0.      -0.52995]
 [ 0.       0.      -0.52995]
 [ 0.       0.       0.26498]
 [-0.      -0.       0.26498]
 [ 0.      -0.       0.26498]
 [-0.      -0.       0.26498]
 [ 0.       0.      -0.52995]
 [ 0.       0.      -0.52995]
 [-0.      -0.       0.26498]
 [-0.      -0.       0.26498]]
The Lorentz field is 0.000 0.000 0.731
The contact field is 1.111 T
Dipolar average of 1 parallel site and 2 antiparallel sites is 0.00000 T
```

4.3 MnSi

This example will guide you through the experimental investigation conducted by Amato et al. and Dalmas de Réotier et. al reported in the following journal articles *[Amato2014]*, *[DalmasdeReotier2016]*.

For the complete code see *run_example.py* in the *MnSi* the examples directory of the muesr package (or open it on github).

Note: This is an advanced example. It is assumed that you already familiarized with Muesr by following one of the other examples or the tutorial.

4.3.1 Scientific background

MnSi has a cubic lattice structure with lattice constant 4.558 Å. It has P213 (No. 198) space group symmetry and the Mn-ion occupy the position (0.138,0.138,0.138), while the Si-ion the position (0.845,0.845,0.845).

At T ~ 0 K, the magnetic structure of MnSi is characterized by spins forming a left-handed incommensurate helix with a propagation vector k0.036 Å^1 in the [111] direction [5–7]. The static Mn moments (0.4μB for T→0 K) point in a plane perpendicular to the propagation vector.

4.3.2 Dipolar Tensor

Given the number of oscillations that are found in the experiment, only a Wyckoff site of type 4a is possible. We therefore proceed by inspecting the dipolar tensor for all points the points of type 4a which happen to be in the 111 (and equivalent) direction of the cubic cell. As a first step, we will identify the number of parameters that characterize the dipolar tensor numerically by visually checking the dipolar tensor elements for all the equivalent sites. This can be done much more accurately analytically but a numerical check can come in handy.

```
# this is a general position along the 111,
# just to identify the form of the dipolar tensor for the sites
# along the 111
s.add_muon([0.45,0.45,0.45])

# we find the remainig eq muon sites
muon_find_equiv(s)

# apply an arbitrary small field to select magnetic atoms.
# the abslute value is not used, but it must be different from 0.
APP_FCs = 0.001*np.array([[0,0,1],
                          [0,0,1],
                          [0,0,1],
                          [0,0,1],
                          [0,0,0],
                          [0,0,0],
                          [0,0,0],
                          [0,0,0]], dtype=np.complex)


s.new_mm()
s.mm.desc = "Applied field"
s.mm.k = np.array([0,0,0])
s.mm.fc = APP_FCs

```

```
62 # Calculate the dipolar tensor. Result is in Ang^-3
63 dts = dipten(s, [30,30,30],50) # supercell size set to 30 unit cells,
64                                 # a 50 Ang sphere si certainly contained.
65
66 # Print the muon site for all the 4 equivalent site.
67 for i, pos in enumerate(s.muons):
68     print(("\nFrac. muon position: {:2.3f} {:2.3f} {:2.3f}\n" + \
69             "Dipolar Tensor: {:2.3f} {:2.3f} {:2.3f}\n" + \
70             "                {:2.3f} {:2.3f} {:2.3f}\n" + \
71             "                {:2.3f} {:2.3f} {:2.3f}\n").format( \
72               *(pos.tolist() + (dts[i] * 6.022E24/1E24/4).flatten().tolist()) \
73             )
74           )
75
76
77 # We will now identify the position of the muon site using the TF data
78 # remove all defined positions for the search
79 muon_reset(s)
```

In line 40 we add one of the four symmetry equivalent positions for a muon in the *4a* Wyckoff site and we let the code find the other sites in line 42 with the function *muon_find_equiv*. In line 46 we define arbitrary small Fourier components which are used to specify which atoms are magnetic (Mn in this case). We finally add this sort of magnetic order to the sample description in lines 56-59 The value of the propagation vector given in line 58 can be omitted (resulting in 0 as default)as it is not used anywhere in this initial estimation of the dipolar tensor. The non-zero values of the dipolar tensor, shown at standard output, are

```
Frac. muon position: 0.450 0.450 0.450
Dipolar Tensor: 0.000 0.092 0.092
                0.092 0.000 0.092
                0.092 0.092 0.000


Frac. muon position: 0.050 0.550 0.950
Dipolar Tensor: 0.000 0.092 -0.092
                0.092 -0.000 -0.092
                -0.092 -0.092 0.000


Frac. muon position: 0.550 0.950 0.050
Dipolar Tensor: -0.000 -0.092 0.092
                -0.092 0.000 -0.092
                0.092 -0.092 0.000


Frac. muon position: 0.950 0.050 0.550
Dipolar Tensor: 0.000 -0.092 -0.092
                -0.092 0.000 0.092
                -0.092 0.092 -0.000
```

In order to compare with the experiment, we will evaluate the dipolar tensor for 100 values along the 111 direction.

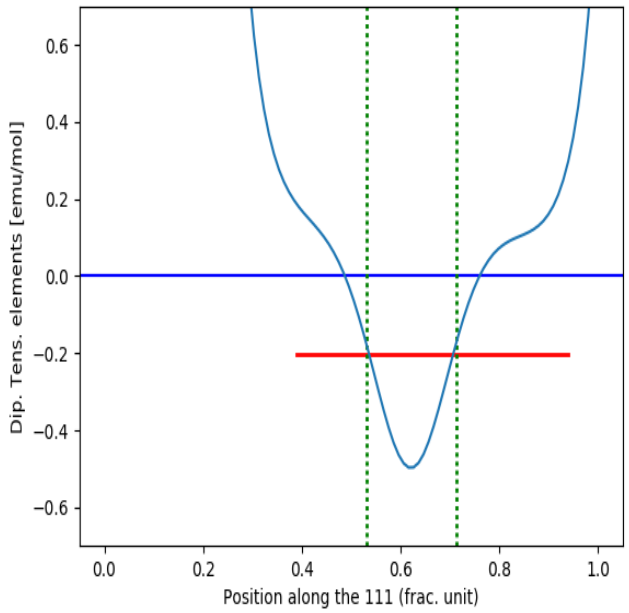
```
84 positions = np.linspace(0,1,100)
85 for pos in positions:
86     s.add_muon([pos,pos,pos])
87
88
89 dts = dipten(s, [30,30,30],50) # supercell size set to 30 unit cells,
```

```
                                    # a 50 Ang sphere si certainly contained.

values_for_plot = np.zeros_like(positions)
for i, dt in enumerate(dts):
    # to reproduce the plot of Amato et. al, we convert to emu/mol
    values_for_plot[i] = dt[0,1] * 6.022E24/1E24/4 # (cm^3/angstrom^3) = 1E24

# remove all defined positions for the search
muon_reset(s)
```

Lines 99-110 produce the following figure:



which is essentially what is reported in *[Amato2014]*.

4.3.3 Local Fields

In order to calculate the local fields at the muon site in the helical state, we will first define the magnetic order and then evaluate the local fields with a optimized algorithm for incommensurate magnetic structures.

Let us first create a few useful variables for the definition of the Fourier components (FC).

```
scaled_pos = s.cell.get_scaled_positions()
pos_Mn1 = scaled_pos[0]
pos_Mn2 = scaled_pos[1]
pos_Mn3 = scaled_pos[2]
pos_Mn4 = scaled_pos[3]

# let's do some simple math

a = 4.558 # Ang
a_star = 2*np.pi/a #Ang^-1

```

```
norm_k = 0.035 # Å 1

k_astar = k_bstar = k_cstar = (1/np.sqrt(3))*norm_k

k_rlu = np.array([k_astar,k_bstar,k_cstar])/a_star  # this only works for a cubic␣
↪lattic


# rotation plane must be perpendicular to k // [111]
#  let's chose one direction to be in the [1,-1,0] direction and find the other

def uv(vec):
    return vec/np.linalg.norm(vec)

k_u = uv(k_rlu)      # unitary vector in k direction
a = uv([1,-1,0])     # one of the two directions which define the plane of
                     #  the rotation

b = np.cross(k_u,a) # the other vector (perp. to a) defining the plane
                     #  where the moments lie.
                     # There are two choices here: left handed or right
                     # handed spiral. We will do both.
```

We can now define both a left handed and a right handed helix.

```

# assuming that, at x=0, a Mn moment is // a, the spiral can be obtained as
RH_FCs = 0.385 * np.array([(a+1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn1)),
                           (a+1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn2)),
                           (a+1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn3)),
                           (a+1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn4)),
                           [0,0,0],
                           [0,0,0],
                           [0,0,0],
                           [0,0,0]
                          ])
# Notice the minus sign.
LH_FCs = 0.385 * np.array([(a-1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn1)),
                           (a-1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn2)),
                           (a-1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn3)),
                           (a-1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn4)),
                           [0,0,0],
                           [0,0,0],
                           [0,0,0],
                           [0,0,0]
                          ])
```

We finally add the muon positions and the two magnetic orders with the commands

```
s.new_mm()
s.mm.desc = "Right handed spiral"
s.mm.k = k_rlu
s.mm.fc = RH_FCs

s.new_mm()
s.mm.desc = "Left handed spiral"
s.mm.k = k_rlu
s.mm.fc = LH_FCs
```

```
s.add_muon([0.532, 0.532, 0.532])
muon_find_equiv(s)
```

Note: When a new magnetic order is added, the index is automatically incremented and the new entry is immediately selected

In order to get the local contributions to the magnetic field at the muon site we use the function locfield function and specify the 'i' sum type.

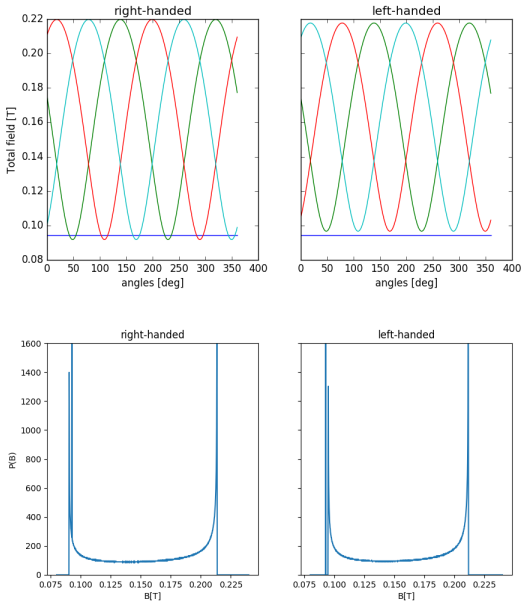
```
# For Reft Handed
s.current_mm_idx = 1;       # N.B.: indexes start from 0 but idx=0 is the transverse␣
↪field!
r_RH = locfield(s, 'i',[50,50,50],100,nnn=3,nangles=360)

s.current_mm_idx = 2;
r_LH = locfield(s, 'i',[50,50,50],100,nnn=3,nangles=360)
```

By default, the contact coupling is 0 for all sites. In order to have a contact term different from 0 we have to set the parameter ACont for all the muon sites that we defined.

```
for i in range(4):
    r_RH[i].ACont = ContatExp
    r_LH[i].ACont = ContatExp
```

The lines 205-2624produce the following pictures:



Interestingly, left-handed and right-handed orders produce different local field. This is due to the lack of inversion symmetry.

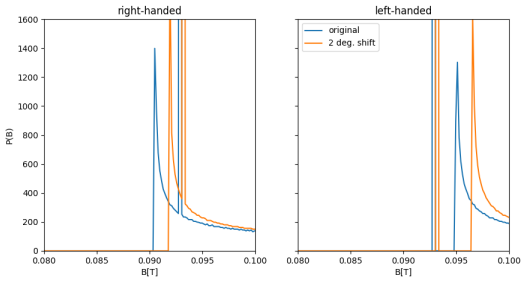
4.3.4 The phase between Mn atoms in the unit cell

Dalmas De and Reotier colleagues pointed out that the complete description of the spiral phase requires three additional degrees of freedom: the two angles describing the misalignmente between the rotation plane of the Mn moments and the plane perpendicular to the propagation vector k and the phase between the helices in the Mn sites belonging to the two crystallographic orbits (see *[DalmasdeReotier2016]*). This last parameter strongly influence the results.

Here we define a phase shift of $\phi = 2$ degrees as reported in the article mentioned above.

```
RH_FCs = 0.385 * np.array([(a+1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn1)-np.pi*2./
↪180.),
                           (a+1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn2)),
                           (a+1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn3)),
                           (a+1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn4)),
                           [0,0,0],
                           [0,0,0],
                           [0,0,0],
                           [0,0,0]
                          ])
# Notice the minus sign.
LH_FCs = 0.385 * np.array([(a-1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn1)-np.pi*2./
↪180.),
                           (a-1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn2)),
                           (a-1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn3)),
                           (a-1j*b)*np.exp(-2*np.pi*1j*np.dot(k_rlu,pos_Mn4)),
                           [0,0,0],
                           [0,0,0],
                           [0,0,0],
                           [0,0,0]
                          ])
```

Repeating the same procedure discussed above leads to the following comparison between the magnetic structures with $\phi = 0$ (blue) and and $\phi = 2$ deg (orange).



4.3.5 Bibliography

CHAPTER 5

Usage

5.1 Defining a sample

This is very easy! You just do:

```
>>> from muesr.core.sample import Sample
>>> smp = Sample()
>>> smp.name = "My very interesting experiment on ..."
```

the *name* property is optional.

5.2 Defining a lattice structure

This can be done at code level by using the *cell* property. For example

```
>>> from muesr.core.sample import Sample
>>> from muesr.core.atoms import Atoms
>>> smp = Sample()
>>> smp.cell = Atoms(...)
```

where the dots replace the *Atoms* class initialization arguments.

However this can be tedious and error prone. Therefore you are strongly suggested to load the lattice information from a file using use one of these functions:

- *load_cif()* for Crystallographic Information Files
- *load_mcif()* for Magnetic Crystallographic Information Files
- *load_xsf()* for XCrysDen files.

Note: For CIF files the symmetry is automatically parsed and set from the file. For MCIF and XSF files it is not set and must be defined by hand or with the *symsearch()* function (only available if *spglib* is installed).

5.3 Defining a magnetic structure

In *muesr* the magnetic structure is defined by the propagation vector *k*, the Fourier components and the phases (see *Description of Magnetic Structures*)

5.3.1 Propagation vector

In *muesr*, the propagation vector is always specified in **reciprocal lattice units** with the *k* property.

5.3.2 Fourier components

There are many conventions to specify the Fourier components of a magnetic structures. At the current stage *muesr* supports the following coordinate systems:

0. **Cartesian system** the same used to define the lattice parameters which is implicitly defined by the lattice vectors.
1. **Bohr Magneton/Angstrom units, with x||a, y||b and z||c** This is the reduced lattice coordinate system, where the magnetic metric tensor (M) is the same metric used for inter-atomic distances (G).
2. **Bohr Magneton units, with x||a, y||b and z||c** This is the crystal-axis coordinate system, where components of the moment are defined by their projections along the lattice basis vectors. If we define L = {{a,0,0},{0,b,0},{0,0,c}}, then the magnetic metric tensor is M = L.G.L^(-1), which is unit-less.

Here's a table connecting the three possible input and related functions

coordinate system number	*MM* property	String version (used in YAML files and in helper functions)
0	*fc fcCart*	'bohr-cartesian' or 'b-c' (case insensitive)
1	*fcLattBMA*	'bohr/angstrom-lattice' or 'b/a-l' (case insensitive)
2	*fcLattBM*	'bohr-lattice' or 'b-l' (case insensitive)

5.3.3 Quick overview

To define a new magnetic structure just do

```
>>> from muesr.core.sample import Sample
>>> from muesr.core.magmodel import MM
>>> smp = Sample()
>>>
>>> # load a lattice structure!
>>>
>>> smp.new_mm()
```

The newly created magnetic structure is automatically selected as the current magnetic model and can be obtained with the *mm* property. From that you can access and define all the properties of the magnetic definition

```
>>> smp.mm.k
... array([0, 0, 0])
```

The three fundamental properties of a magnetic model are:

- *fc*
- *k*
- *phi*

Please see the *muesr.core.magmodel.MM* documentation for the details.

To simplify the definition of the magnetic structure, the *mago_add()* helper function is available in the *muesr.utilities.ms* module.

It prompts an interactive interface like the one shown below (for a Ti2O3 structure):

```
>>> from muesr.utilities.ms import mago_add
>>> mago_add(smp,coordinates='bohr-lattice')
...      Propagation vector (w.r.t. conv. rec. cell): 0 0 0
...          Magnetic moments in bohr magnetons and lattice coordinates.
...          Which atom? (enter for all)Ti
...          Lattice vectors:
...            a    5.149000000000000    0.000000000000000    0.000000000000000
...            b   -2.574499999999999    4.459164804086075    0.000000000000000
...            c    0.000000000000001    0.000000000000001   13.641999999999999
...          Atomic positions (fractional):
...              1 Ti  0.00000000000000  0.00000000000000  0.34500000000000  47.867
...              2 Ti  0.66666666666667  0.33333333333333  0.67833333333333  47.867
...              3 Ti  0.33333333333333  0.66666666666667  0.01166666666667  47.867
...              4 Ti  0.00000000000000  0.00000000000000  0.84500000000000  47.867
...              5 Ti  0.66666666666667  0.33333333333333  0.17833333333333  47.867
...              6 Ti  0.33333333333333  0.66666666666667  0.51166666666667  47.867
...              7 Ti  0.00000000000000  0.00000000000000  0.15500000000000  47.867
...              8 Ti  0.66666666666667  0.33333333333333  0.48833333333333  47.867
...              9 Ti  0.33333333333333  0.66666666666667  0.82166666666667  47.867
...             10 Ti  0.00000000000000  0.00000000000000  0.65500000000000  47.867
...             11 Ti  0.66666666666667  0.33333333333333  0.98833333333333  47.867
...             12 Ti  0.33333333333333  0.66666666666667  0.32166666666667  47.867
...          FC for atom 1 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 2 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 3 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 4 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 5 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 6 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 7 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 8 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 9 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 10 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 11 Ti (3 real, [3 imag]): 0 1 0
...          FC for atom 12 Ti (3 real, [3 imag]): 0 1 0
...
```

This produces the following Fourier components in Cartesian coordinates

```
>>> smp.mm.fc
... array([[-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...        [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...        [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
```

```
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [-0.5000000+0.j,  0.8660254+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j],
...         [ 0.0000000+0.j,  0.0000000+0.j,  0.0000000+0.j]])
```

Which are indeed:

```
>>> smp.mm.fcLattBM
... array([[ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  1.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
```

```
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j],
...         [ 0.+0.j,  0.+0.j,  0.+0.j]])
```

The zeros in the Fourier components are from the atoms different from *Ti*.

Note: The phases can only be set with the *phi* property.

5.3.4 Useful readings

- http://www.neutron-sciences.org/articles/sfn/pdf/2014/01/sfn201402001.pdf

5.4 Setting the muon position

The muon position can be easily set with the *add_muon* method.

If symmetry is defined, equivalent muon positions can be obtained with the function *muon_find_equiv()* in the *muesr.utilities.muon* module.

5.5 Calculate local fields

The function simulating local fields at the muon site is `locfield()`.

There are three type of simulations which are targeted to different types of problems:

- *sum*: a simple sum of all the magnetic moments in the Lorentz sphere.
- *rotate*: rotates the local moments around a given axis and perform the sum. This function offer great flexibility in the way local moments are rotated but is not computationally efficient. For incommensurate magnetic orders the following function is much more efficient.
- *incommensurate*: Fast version of 'rotate' which exploits the method discussed in Phys. Rev. B **93**, 174405 (2016).

5.6 Calculate the dipolar tensor

The function providing the dipolar tensor at the muon site is `dipten()`.

To use it you have to specify a (arbitrary) value for the Fourier components of the magnetic atoms that you want to include in the sum. The specified value has no meaning only the 0 vs different from zero has.

Note: Results are provided in Angstrom^-3 !

5.7 Generate grid of interstitial points for DFT simulations

A useful function to prepare the input for DFT simulations is `build()`.

The function provides a set of symmetry inequivalent interstitial positions with the additional constraint of being sufficiently separated from the atoms of the hosting system.

5.8 Understanding errors

muesr raises the conventional python exceptions (mainly ValueError and TypeError) or other 4 specific Exceptions:

- *CellError*
- *MuonError*
- *MagDefError*
- *SymmetryError*

To see their meaning follow the links above.

N.B.: the utility functions are mainly intended for interactive usage and therefore report problems by printing error messages on the screen. Exceptions are only raised in core components.

5.9 Saving and loading sample details to/from file

To save a sample use *save_sample()*. To load a saved sample use *load_sample()*.

Data is stored in an YAML file. It is possible (but error prone) to write an input file by hand. When loaded, the file will undergo a minimal validation. Identifying the errors is not so easy so the best method to specify the sample details is probably using the various functions discussed in this manual.

5.10 Symmetry

muesr tries to grab symmetry information from either the CIF files or using the *spglib* routines. If a magnetic cif (*.mcif) is loaded, the symmetry is disregarded. The symmetry equivalent sites depend indeed only on the symmetry of the parent cell. If one consider the symmetry of the magnetic structure and searches for the equivalent muon sites, many of them will be missing. The symmetry of every cell can always be identified with the help of the *symsearch()* command. Note however that, **when dealing with supercells**, some of the symmetry **equivalent muon positions must be specified by hand**.

CHAPTER 6

Contact Hyperfine Fields - Some notes

Muesr assumes that hyperfine fields are isotropic. The contact term is obtained by specifying two terms: 1) the number of nearest neighbors magnetic atoms to the muon 2) a rcont parameters which governs the maximum radius of the interaction.

Finally, the LocalField object has a ACont property which is an effective hyperfine contact coupling term. The total field is therefore obtained as

$$\mathbf{B_T} = \mathbf{B_D} + \mathbf{B_L} + ACont \cdot \frac{2\mu_0}{3} \sum_i^N \frac{r_i^{-3}}{\sum_i^N r_i^{-3}} \mathbf{m}_i$$

The value of N can strongly impact on the results. The best approximation strongly depend on the simulated system and must be considered case by case.

6.1 From CGS to SI

Hyperfine couplings are often reported in mol/emu while Muesr uses $\mathring{\text{A}}^{-3}$. Here's how to convert the former into the latter.

$$B_c = -\frac{8}{3}\pi|\Psi(0)|^2\boldsymbol{\mu}_e \qquad \text{(c.g.i)}$$

$$B_c = -\frac{2}{3}\mu_0|\Psi(0)|^2\boldsymbol{\mu}_e \qquad \text{(S.I.)}$$

Assuming the following formula for the hyperfine contact field produced by the nearest neighboring magnetic atom

$$B_c = N_A A_{cont}\boldsymbol{\mu}_e$$

where A_{cont} is expressed in mol/emu and $\boldsymbol{\mu}_e$ is in emu. As a consequence:

$$A_{cont} = \frac{8\pi|\Psi(0)|^2}{3N_A}$$

Assuming 1 mol/emu, the value for A in $\mathring{\text{A}}^{-3}$ is

$$A_{cont} = \frac{1\text{mol/emu} * 3N_A}{8\pi} = 7.188E22cm^{-3} = 0.071884019\mathring{\text{A}}^{-3}$$

CHAPTER 7

Frequently Asked Questions

Symmetry is not correctly recognized by spglib! Try lowering the numerical precision threshold by running, for example,

```
>>> symsearch(yoursample, precision=1e-3)
```

If that does not work I'm keen to think that there is an error in your input structure since spglib is a well tested piece of code. If your input is correct file a bug at spglib.sf.net.

How do I convert the hyperfine filed to reasonable units? See ContactTerm

CHAPTER 8

Source documentation

8.1 muesr – Main package

8.1.1 muesr.core – Core functionalities

muesr.core.sample – The Sample definition

class muesr.core.sample.**Sample**
Bases: object

This object contains all the information about the sample under investigation. It includes a definition of the lattice parameters and the atomic positions, the symmetry, the positions of the muons and the magnetic structure.

In each sample there can only be a single lattice structure. Therefore, also a single symmetry is defined.

Multiple magnetic structures can be defined instead. Every time a magnetic structure is defined, it automatically becomes the current magnetic structure. To select a different magnetic structure the user can use the property *current_mm_idx*.

```
>>> yoursample = Sample()  #initialize the sample
>>> yoursample.name = "Test"
```

add_muon (*position*, *cartesian=False*)
Adds a muon position.

Parameters

- **position** (*list*) – list of three float or numpy array of shape (3,) describing the position of the muon.
- **cartesian** (*bool*) – if True, the position os assumed to be in Cartesian coordinates. If False (default) the position is assumed to be in fractional coordinates.

Returns None

Return type None

Raises MuonError

cell
Returns the atomic structure definition, i.e. an Atoms object.

Getter returns a Atoms Object.

Setter Sets the atomic structure definition from a Atoms object.

Type int

Raises TypeError, CellError

check_status(*cell=False*, *magdefs=False*, *muon=False*, *sym=False*)

current_mm_idx
Index of the currently selected magnetic model (starting from 0)

Getter returns the index of the currently selected Magnetic Model

Setter Sets the current Magnetic Model from the index.

Type int

Raises MagDefError

mm
Current Magnetic Model.

Getter Returns the current Magnetic Model (a MM object)

Setter Adds and select a new Magnetic Model (MM object)

Type MM object

Raises TypeError, MagDefError

mm_count
Count the loaded Magnetic Model(s).

Getter Returns the current Magnetic Model (a MM object)

Setter Read-only

Type MM object

Raises None

muons
Muon positions in fractional coordinates.

Getter Returns a list of numpy array of shape (3,).

name
The sample name.

Getter Returns the sample name

Setter Sets the sample name

Type str

new_mm()
Creates a new empty magnetic model (everything is set to zero)

Returns None

Return type None

new_smm(*symbolic_parameters*)
Creates a new empty Symbolic Magnetic Model (everything is set to zero)

Parameters **symbolic_parameters** (*str*) – String containing a list of symbolic parameters. Example "x,y,z"

Returns None

Return type None

Raises ImportError if sympy is not installed. CellError if lattice is not defined.

sym
Symmetry of the system, i.e. a Spacegroup Object.

Getter returns a Spacegroup Object.

Setter Sets the symmetry of the system from a Spacegroup Object.

Type int

Raises TypeError, SymError

muesr.core.sampleErrors – Sample related exceptions

exception muesr.core.sampleErrors.**CellError**
Bases: *muesr.core.sampleErrors.SampleException*

Lattice structure needed but not defined.

exception muesr.core.sampleErrors.**MagDefError**
Bases: *muesr.core.sampleErrors.SampleException*

Magnetic structure not defined or incompatible.

exception muesr.core.sampleErrors.**MuonError**
Bases: *muesr.core.sampleErrors.SampleException*

Muon position needed but not defined.

exception muesr.core.sampleErrors.**SampleException**
Bases: Exception

Exceptions connected with problems with the Sample definition.

exception muesr.core.sampleErrors.**SymmetryError**
Bases: *muesr.core.sampleErrors.SampleException*

Symmetry needed but not defined.

muesr.core.atoms – Defined the unit cell

class muesr.core.atoms.**Atoms**(*symbols=None*, *positions=None*, *numbers=None*, *masses=None*, *magmoms=None*, *scaled_positions=None*, *cell=None*, *pbc=None*)
Bases: object

Atoms class compatible with the ASE Atoms class Only stuff needed by muesr is implemented.

```
>>> a = 4.05  # Gold lattice constant
>>> b = a / 2
>>> fcc = Atoms('Au',
...             cell=[(0, b, b), (b, 0, b), (b, b, 0)],
...             pbc=True)
```

del_atom(*i*)
Removes one atom

edit_atom(*i*, *symbol=None*, *number=None*, *magmom=None*, *mass=None*)

extend(*symbol=None*, *position=None*, *number=None*, *mass=None*, *magmom=None*, *scaled_position=None*)
Extends the current atomic structure by one atom.

get_atomic_numbers()

get_cell()

get_chemical_symbols()

get_magnetic_moments()
Get magnetic moments if set. None if nothing set.

get_masses()

get_number_of_atoms()

get_positions()

get_scaled_positions()

get_volume()

numbers_to_symbols()

set_cell(*cell*)
Set lattice vectors parameters.

Parameters **cell** – numpy array of the form

[[v1(1), v1(2), v1(3)],
v2(1), v2(2), v2(3)],
v3(1), v3(2), v3(3)]]

where v1, v2, v3 are the lattice vectors.

set_chemical_symbols(*symbols*)

set_magnetic_moments(*magmoms=None*)

set_masses(*masses*)

set_positions(*cart_positions*)

set_scaled_positions(*scaled_positions*)

symbols_to_masses()

symbols_to_numbers()

muesr.core.cells – Functions for lattice

muesr.core.cells.**R2S**(*x*, *y*, *z*, *H*, *K*, *L*)
Given reciprocal-space coordinates of a vecotre, calculate its coordinates in the Cartesian space.

muesr.core.cells.**S2R**(*qx*, *qy*, *qz*, *cell*)
Given cartesian coordinates of a vector in the S System, calculate its Miller indexes.

muesr.core.cells.**get_Delaunay_reduction**(*lattice*, *tolerance*)

muesr.core.cells.**get_angles**(*lattice*)
Get alpha, beta and gamma angles from lattice vectors.

```
>>> get_angles( np.diag([1,2,3]) )
(90.0, 90.0, 90.0)
```

muesr.core.cells.**get_atom_distance**(*scaled_pos*, *center*, *cell*, *tolerance=1e-05*)
Return the shortest distance to atom from specified position

muesr.core.cells.**get_atom_vec**(*scaled_pos*, *center*, *cell*, *tolerance=1e-05*)
Return the shortest distance to atom from specified position

muesr.core.cells.**get_cell_matrix**(*a*, *b*, *c*, *alpha*, *beta*, *gamma*)

muesr.core.cells.**get_cell_parameters**(*lattice*)

muesr.core.cells.**get_distance**(*atoms*, *a0*, *a1*, *tolerance=1e-05*)
Return the shortest distance between a pair of atoms in PBC

muesr.core.cells.**get_distance_with_center**(*atoms*, *center*, *atom_num*, *tolerance=1e-05*)
Return the shortest distance to atom from specified position

muesr.core.cells.**get_reciprocal_lattice**(*lattice*)

muesr.core.cells.**get_reduced_bases**(*cell*, *tolerance=1e-05*)
This is an implementation of Delaunay reduction. Some information is found in International table.

muesr.core.cells.**get_shortest_bases_from_extented_bases**(*extended_bases*, *tolerance*)

muesr.core.cells.**get_simple_supercell**(*sample*, *multi*)
This function creates a simple supercell by expanding the unit cell in a, b and c directions.

muesr.core.cells.**gtensor**(*lattice*)
calculates the metric tensor of a lattice

muesr.core.cells.**print_cell**(*cell*, *mapping=None*)
Print lattice structure.

cell : Aroms instance, the atomic structure to be printed mapping : list or None, can be used to show relations between atoms.

muesr.core.cells.**reciprocate**(*x*, *y*, *z*, *cell*)
calculate miller indexes of a vector defined by its fractional cell coords

muesr.core.cells.**reduce_bases**(*extended_bases*, *tolerance*)

muesr.core.magmodel – Magnetic Model

class muesr.core.magmodel.**MM**(*cell_size*, *latt_vects=None*)
Bases: object

Magnetic model class.

The magnetic structures are defined in terms of a single propagation vector, complex Fourier components and phases.

Parameters

- **cell_size** (*int*) – number of Fourier components=number of atoms
- **latt_vects** – lattice vectors as numpy 3x3 ndarray. If None, the Fourier componentscan only be specified in cartesian coordinates.

Raises TypeError

desc
Description of the magnetic structure

Getter Returns the description

Setter Sets the description

Type str

fc
Fourier components in Cartesian coordinates. Same as *fcCart*

fcCart
Get Fourier components in Cartesian coordinates.

Getter Returns a numpy array of size (*size*,3) with the fourier components.

Setter Sets the fourier compinents

Type numpy ndarray of size (*size*,3)

fcLattBM
Fourier components in Bohr Magneton units, with x||a, y||b and z||c

Getter Returns a numpy array of size (*size*,3) with the fourier components.

Setter Sets the fourier compinents

Type numpy ndarray of size (*size*,3)

fcLattBMA
Get Fourier components in Bohr Magneton/Angstrom units, with x||a, y||b and z||c

Getter Returns a numpy array of size (*size*,3) with the fourier components.

Setter Sets the fourier compinents

Type numpy ndarray of size (*size*,3)

fc_get (*coord_system=0*)
Retrives Fourier components.

Params int coord_system requested coordinate system.

- 0 means Cartesian coordinates
- 1 means Lattice coordinates (values in units of Bohr Magneton/Angstrom)
- 2 means Bohr magnetons in each lattice direction (values in units Bohr Magneton)

See http://magcryst.org/resources/magnetic-coordinates/

Raises ValueError

fc_set (*value*, *coord_system=0*)
Sets Fourier components.

Parameters

- **value** – numpy array containing the fourier components for all the atoms.
- **coord_system** (*int*) – requested coordinate system.

- 0 means Cartesian coordinates
- 1 means Lattice coordinates (values in units of Bohr Magneton/Angstrom)
- 2 means Bohr magnetons in each lattice direction (values in units Bohr Magneton)

See http://magcryst.org/resources/magnetic-coordinates/

Raises ValueError, TypeError

isSymbolic
True if Fourier components are defined as sympy symbols, False otherwise.

ALWAYS False for MM objects.

k
The propagation vector in reciprocal lattice units.

Getter Returns a numpy array of shape (3,) with the propagation vector.

Setter Sets the propagation vector.

Type numpy ndarray of shape (3,)

lattice_params
Lattice parameters used to convert the Fourier components to the various coordinates systems.

phi
The phase for each fourier component in units of 2 PI.

Getter Returns a numpy array of size *size* with the phases.

Setter Sets the phases. Type can be list of numpy array.

Type numpy ndarray

size
Number of Fourier components.

muesr.core.spg – Spacegroups (from ASE)

Definition of the Spacegroup class.

This module only depends on NumPy and the space group database.

class muesr.core.spg.**Spacegroup**(*spacegroup*, *setting=1*, *datafile=None*)
Bases: object

A space group class.

The instances of Spacegroup describes the symmetry operations for the given space group.

Example:

```
>>> from muesr.core.spg import Spacegroup
>>>
>>> sg = Spacegroup(225)
>>> print('Space group', sg.no, sg.symbol)
Space group 225 F m -3 m
>>> sg.scaled_primitive_cell
array([[ 0. ,  0.5,  0.5],
       [ 0.5,  0. ,  0.5],
       [ 0.5,  0.5,  0. ]])
>>> sites, kinds = sg.equivalent_sites([[0,0,0]])
>>> sites
```

```
array([[ 0. ,  0. ,  0. ],
       [ 0. ,  0.5,  0.5],
       [ 0.5,  0. ,  0.5],
       [ 0.5,  0.5,  0. ]])
```

centrosymmetric
Whether a center of symmetry exists.

equivalent_lattice_points(*uvw*)
Return all lattice points equivalent to any of the lattice points in *uvw* with respect to rotations only.

Only equivalent lattice points that conserves the distance to origo are included in the output (making this a kind of real space version of the equivalent_reflections() method).

Example:

```
>>> from ase.spacegroup import Spacegroup
>>> sg = Spacegroup(225)  # fcc
>>> sg.equivalent_lattice_points([[0, 0, 2]])
array([[ 0,  0, -2],
       [ 0, -2,  0],
       [-2,  0,  0],
       [ 2,  0,  0],
       [ 0,  2,  0],
       [ 0,  0,  2]])
```

equivalent_reflections(*hkl*)
Return all equivalent reflections to the list of Miller indices in hkl.

Example:

```
>>> from muesr.core.spg import Spacegroup
>>> sg = Spacegroup(225)  # fcc
>>> sg.equivalent_reflections([[0, 0, 2]])
array([[ 0,  0, -2],
       [ 0, -2,  0],
       [-2,  0,  0],
       [ 2,  0,  0],
       [ 0,  2,  0],
       [ 0,  0,  2]])
```

equivalent_sites(*scaled_positions*, *onduplicates='error'*, *symprec=0.001*)
Returns the scaled positions and all their equivalent sites.

Parameters:

scaled_positions: list | array List of non-equivalent sites given in unit cell coordinates.

onduplicates ['keep' | 'replace' | 'warn' | 'error'] Action if *scaled_positions* contain symmetry-equivalent positions:

'keep' ignore additional symmetry-equivalent positions

'replace' replace

'warn' like 'keep', but issue an UserWarning

'error' raises a SpacegroupValueError

symprec: float Minimum "distance" betweed two sites in scaled coordinates before they are counted as the same site.

Returns:

sites: array A NumPy array of equivalent sites.

kinds: list A list of integer indices specifying which input site is equivalent to the corresponding returned site.

Example:

```
>>> from muesr.core.spg import Spacegroup
>>> sg = Spacegroup(225)  # fcc
>>> sites, kinds = sg.equivalent_sites([[0, 0, 0], [0.5, 0.0, 0.0]])
>>> sites
array([[ 0. ,  0. ,  0. ],
       [ 0. ,  0.5,  0.5],
       [ 0.5,  0. ,  0.5],
       [ 0.5,  0.5,  0. ],
       [ 0.5,  0. ,  0. ],
       [ 0. ,  0.5,  0. ],
       [ 0. ,  0. ,  0.5],
       [ 0.5,  0.5,  0.5]])
>>> kinds
[0, 0, 0, 0, 1, 1, 1, 1]
```

get_op()
Returns all symmetry operations (including inversions and subtranslations), but unlike get_symop(), they are returned as two ndarrays.

get_rotations()
Return all rotations, including inversions for centrosymmetric crystals.

get_symop()
Returns all symmetry operations (including inversions and subtranslations) as a sequence of (rotation, translation) tuples.

lattice
Lattice type:

P primitive I body centering, h+k+l=2n F face centering, h,k,l all odd or even A,B,C single face centering, k+l=2n, h+l=2n, h+k=2n R rhombohedral centering, -h+k+l=3n (obverse); h-k+l=3n (reverse)

no
Space group number in International Tables of Crystallography.

nsubtrans
Number of cell-subtranslation vectors.

nsymop
Total number of symmetry operations.

reciprocal_cell
Tree Miller indices that span all kinematically non-forbidden reflections as a matrix with the Miller indices along the rows.

rotations
Symmetry rotation matrices. The invertions are not included for centrosymmetrical crystals.

scaled_primitive_cell
Primitive cell in scaled coordinates as a matrix with the primitive vectors along the rows.

setting
Space group setting. Either one or two.

subtrans
Translations vectors belonging to cell-sub-translations.

symbol
Hermann-Mauguin (or international) symbol for the space group.

symmetry_normalised_reflections(*hkl*)
Returns an array of same size as *hkl*, containing the corresponding symmetry-equivalent reflections of lowest indices.

Example:

```
>>> from muesr.core.spg import Spacegroup
>>> sg = Spacegroup(225)  # fcc
>>> sg.symmetry_normalised_reflections([[2, 0, 0], [0, 2, 0]])
array([[ 0,  0, -2],
       [ 0,  0, -2]])
```

symmetry_normalised_sites(*scaled_positions*, *map_to_unitcell=True*)
Returns an array of same size as *scaled_positions*, containing the corresponding symmetry-equivalent sites of lowest indices.

If *map_to_unitcell* is true, the returned positions are all mapped into the unit cell, i.e. lattice translations are included as symmetry operator.

Example:

```
>>> from muesr.core.spg import Spacegroup
>>> sg = Spacegroup(225)  # fcc
>>> sg.symmetry_normalised_sites([[0.0, 0.5, 0.5], [1.0, 1.0, 0.0]])
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

tag_sites(*scaled_positions*, *symprec=0.001*)
Returns an integer array of the same length as *scaled_positions*, tagging all equivalent atoms with the same index.

Example:

```
>>> from muesr.core.spg import Spacegroup
>>> sg = Spacegroup(225)  # fcc
>>> sg.tag_sites([[0.0, 0.0, 0.0],
...               [0.5, 0.5, 0.0],
...               [1.0, 0.0, 0.0],
...               [0.5, 0.0, 0.0]])
array([0, 0, 0, 1])
```

todict()

translations
Symmetry translations. The invertions are not included for centrosymmetrical crystals.

unique_reflections(*hkl*)
Returns a subset *hkl* containing only the symmetry-unique reflections.

Example:

```
>>> from muesr.core.spg import Spacegroup
>>> sg = Spacegroup(225)  # fcc
>>> sg.unique_reflections([[ 2,  0,  0],
...                        [ 0, -2,  0],
```

```
...                          [ 2,  2,  0],
...                          [ 0, -2, -2]])
array([[2, 0, 0],
       [2, 2, 0]])
```

unique_sites (*scaled_positions*, *symprec=0.001*, *output_mask=False*, *map_to_unitcell=True*)

Returns a subset of *scaled_positions* containing only the symmetry-unique positions. If *output_mask* is True, a boolean array masking the subset is also returned.

If *map_to_unitcell* is true, all sites are first mapped into the unit cell making e.g. [0, 0, 0] and [1, 0, 0] equivalent.

Example:

```
>>> from muesr.core.spg import Spacegroup
>>> sg = Spacegroup(225)  # fcc
>>> sg.unique_sites([[0.0, 0.0, 0.0],
...                  [0.5, 0.5, 0.0],
...                  [1.0, 0.0, 0.0],
...                  [0.5, 0.0, 0.0]])
array([[ 0. ,  0. ,  0. ],
       [ 0.5,  0. ,  0. ]])
```

8.1.2 muesr.i_o – Input Output functions

muesr.i_o.sampleIO – Input Output functions for the Sample object

muesr.i_o.sampleIO.**load_sample**(*filename=''*, *fileobj=None*)

This function load a sample from a file in YAML format.

Parameters

- **filename** (*str*) – the filename used to store data.
- **fileobj** (*file*) – an optional file object. If specified, this supersede the filename input.

Returns a sample object

Return type Sample object or None

Raises ValueError, FileNotFoundError, IsADirectoryError

muesr.i_o.sampleIO.**save_sample**(*sample*, *filename=''*, *fileobj=None*, *overwrite=False*)

This function saves the sample provided in YAML format.

Parameters

- **sample** – the sample object
- **filename** (*str*) – the filename used to store data.
- **fileobj** (*file*) – a file object used in place of filename.
- **bool** (*overwrite*) – if selected file should be overwritten.

Returns None

Return type None

Raises TypeError, ValueError, IsADirectoryError

muesr.i_o.exportFPS – Input Output for FullProf Studio

muesr.i_o.exportFPS.**export_fpstudio**(*sample*, *filename*)
Exports magnetic structure in FullProfStudio format.

Parameters

- **sample** – the sample object
- **filename** (*str*) – the filename for the FP Studio file

Returns None.

Return type None

muesr.i_o.xsf.xsf – Input Output for Xcrysden

muesr.i_o.xsf.xsf.**load_xsf**(*sample*, *filename*)
Loads structural data from Xcrysden Files in sample object.

WARNING: this will reset all current muon positions, magnetic definitions, and the symmetry definition.

Parameters

- **sample** – a sample object.
- **filename** (*str*) – the filename

Returns True if succeful, False otherwise

Return type bool

Raises TypeError

muesr.i_o.xsf.xsf.**save_xsf**(*sample, filename, supercell=[1, 1, 1], addMuon=True*)
Export structure to XCrysDen.

Parameters

- **sample** – a sample object.
- **filename** (*str*) – path of the destination file.
- **supercell** (*list*) – a list containig the number of replica along the three lattice parameters
- **addMuon** (*bool*) – if true, adds the muon positions (if any) in the central unit cell

Returns True if succeful, False otherwise

Return type bool

Raises CellError, TypeError

muesr.i_o.cif.cif – Input in CIF and mCIF format

This file was adapted from ASE. Module to read and write atoms in cif file format.

See http://www.iucr.org/resources/cif/spec/version1.1/cifsyntax for a description of the file format. STAR extensions as save frames, global blocks, nested loops and multi-data values are not supported.

muesr.i_o.cif.cif.**convert_to_float**(*frac_str*)
This function converts fractions to float, ex. -1/3 = -0.33333...

muesr.i_o.cif.cif.**convert_value**(*value*)
Convert CIF value string to corresponding python type.

muesr.i_o.cif.cif.**load_cif**(*sample*, *filename*, *reset_muon=True*, *reset_sym=True*)
Loads the structural information from a Cif file.

Parameters

- **sample** (muesr.core.sample.Sample) – the sample object.
- **filename** (*str*) – the cif file path (path + filename).
- **reset_muon** (*bool*) – if true the muon positions is reinitialized.
- **reset_sym** (*bool*) – if true the symmetry is reinitialized. ALWAYS TRUE

Returns True if succesfull, false otherwise

Return type bool

muesr.i_o.cif.cif.**load_mcif**(*sample*, *filename*, *reset_muon=True*, *reset_sym=True*)
Loads both the crystalline structure and the magnetic order from a mcif file. N.B.: This function is EXPERIMENTAL.

Note: Only the first lattice and magnetic structure is loaded. mCif files hosting multiple structures are not supported.

Parameters

- **sample** (muesr.core.sample.Sample) – the sample object.
- **filename** (*str*) – the mcif file path (path + filename).
- **reset_muon** (*bool*) – if true the muon positions is reinitialized.
- **reset_sym** (*bool*) – if true the symmetry is reinitialized.

Returns True if succesfull, false otherwise

Return type bool

muesr.i_o.cif.cif.**parse_block**(*lines*, *line*)
Parse a CIF data block and return a tuple with the block name and a dict with all tags.

muesr.i_o.cif.cif.**parse_cif**(*fileobj*)
Parse a CIF file. Returns a list of blockname and tag pairs. All tag names are converted to lower case.

muesr.i_o.cif.cif.**parse_items**(*lines*, *line*)
Parse a CIF data items and return a dict with all tags.

muesr.i_o.cif.cif.**parse_loop**(*lines*)
Parse a CIF loop. Returns a dict with column tag names as keys and a lists of the column content as values.

muesr.i_o.cif.cif.**parse_magn_operation_xyz_string**(*tag*)
Parse the symmetry operations of the magnetic part of mcif

muesr.i_o.cif.cif.**parse_multiline_string**(*lines*, *line*)
Parse semicolon-enclosed multiline string and return it.

muesr.i_o.cif.cif.**parse_singletag**(*lines*, *line*)
Parse a CIF tag (entries starting with underscore). Returns a key-value pair.

muesr.i_o.cif.cif.**read_cif**(*fileobj*, *index*, *store_tags=False*, *primitive_cell=False*, *subtrans_included=True*)

Read Atoms object from CIF file. *index* specifies the data block number or name (if string) to return.

If *index* is None or a slice object, a list of atoms objects will be returned. In the case of *index* is *None* or *slice(None)*, only blocks with valid crystal data will be included.

If *store_tags* is true, the *info* attribute of the returned Atoms object will be populated with all tags in the corresponding cif data block.

If *primitive_cell* is true, the primitive cell will be built instead of the conventional cell.

If *subtrans_included* is true, sublattice translations are assumed to be included among the symmetry operations listed in the CIF file (seems to be the common behaviour of CIF files). Otherwise the sublattice translations are determined from setting 1 of the extracted space group. A result of setting this flag to true, is that it will not be possible to determine the primitive cell.

muesr.i_o.cif.cif.**split_chem_form**(*comp_name*)

Returns e.g. AB2 as ['A', '1', 'B', '2']

muesr.i_o.cif.cif.**tags2atoms**(*tags*, *store_tags=False*, *primitive_cell=False*, *subtrans_included=True*)

Returns an Atoms object from a cif tags dictionary. See read_cif() for a description of the arguments.

muesr.i_o.cif.cif.**write_cif**(*fileobj*, *images*, *format='default'*)

Write *images* to CIF file.

8.1.3 muesr.engines – Muesr calculators

8.1.4 muesr.engines.clfc – The C powered Local Fields Calculator

8.1.5 muesr.utilities – Various useful functions

muesr.utilities.dft_grid – Generate grid for DFT simulations

muesr.utilities.dft_grid.**build_uniform_grid**(*sample*, *size*, *min_distance_from_atoms=1.0*)

Generates a grid of symmetry inequivalent interstitial positions with a specified minimum distance from the atoms of the sample. Especially intended for DFT simulations.

Parameters

- **sample** – A sample object.
- **size** (*int*) – The number of steps in the three lattice directions. Only equispaced grids are supported at the moment.
- **min_distance_from_atoms** (*float*) – Minimum distance between a interstitial position and the atoms of the lattice. Units are Angstrom.

Returns A list of symmetry inequivalent positions.

Return type list

muesr.utilities.symsearch – Identify symmetry (based on spglib)

muesr.utilities.symsearch.**symsearch**(*sample*, *precision=0.0001*)

Identifies symmetry operations of the unit cell using spglib and update the sample definition.

Parameters

- **sample** – A sample object.
- **precision** (*float*) – atoms are assumed equivalent if distances are smaller than precision. In Angstrom.

Returns True if successful, False otherwise.

Return type bool

muesr.utilities.ms – Functions facilitating magnetic description

muesr.utilities.ms.**mago_add**(*sample*, *coordinates='b-c'*, *fcs=None*, *kvalue=None*)
Adds a magnetic model (fourier components and K vector). The order is automatically selected if succesfully added.

Parameters

- **sample** – A sample object.
- **coordinates** (*string*) – coordinates system and units of the Fourier components. Options are: 'b-c', 'b/a-l', 'b-l'.

  b-c : Fourier components in Bohr magnetons and Cartesian coordinates.

  b/a-l: Fourier components in Bohr magnetons/Angstrom and lattice coordinates.

  b-l : Fourier components in Bohr magnetons and in lattice coordinates.
- **fcs** (*np.complex*) – Fourier components in coordinate system (default: Bohr magnetoc/ Cartesian coordinates)
- **kvalue** (*np.ndarray*) – Propagation vector in r.l.u.

Returns True if successful, False otherwise.

Return type bool

muesr.utilities.ms.**mago_set_FC**(*sample*, *fcs=None*, *atoms_types=None*, *mm=None*, *inputConvention='b-c'*)
Defines fourier components for the unit cell.

muesr.utilities.ms.**mago_set_k**(*sample*, *kvalue=None*, *mm=None*)
Set propagation with respect to the conventional reciprocal cell kavalue : propagation vector (either string or tuple or list), if None, input is prompetd mm : Magnetic Model to be used. If None the current magnetic model is used returns: None

muesr.utilities.muon – Functions facilitating muon position definition

muesr.utilities.muon.**muon_find_equiv**(*sample*, *eps=0.001*)
Given the unit cell symmetry, finds the equivalent muon sites. Magnetic order is NOT considered :params: eps, number of decimal figures for comparison

muesr.utilities.muon.**muon_reset**(*sample*)
Removes all previously set muon positions. returns: True

muesr.utilities.muon.**muon_set_frac**(*sample*, *arg=None*)
Set muon position in lattice coordinates.

returns: None

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Bibliography

[Sugiyama2011] Jun Sugiyama *et al.*, Phys. Rev. B 84, 054430 (2011)

[Amato2014] Phys. Rev. B 89, 184425

[DalmasdeReotier2016] Phys. Rev. B 93, 144419

Python Module Index

m

Index

K

L

M

N

P

R

S

T

U

W