
SW
SpinW

SpinW

A crash course on SpinW - Part 1

SpinW (spin-double-u) is a MATLAB library that can optimize magnetic structures using mean field theory and calculate spin wave dispersion and spin-spin correlation function for complex crystal and magnetic structures.

PRESENTED BY

Simon Ward

Scientific Software Developer - ESS

Duc Le

Instrument Scientist - ISIS Facility

Linear Spin Wave Theory Review

5 minute recap on LSWT

Heisenberg Hamiltonian of a chain on Bravais lattice:

$$\mathcal{H} = \sum_{i,d} J(d)\mathbf{S}_i \cdot \mathbf{S}_{i+d}$$

After substitution, changing to a rotating co-ordinate system and a bit of trigonometry we get:

```
\begin{align}
        \mathcal{H} = & \sum_{i,j} J(\mathbf{d}) \left( S_i^\eta S_j^\eta + \sin (\mathbf{Q} \cdot \mathbf{d}_{i,j})(S_i^\mu S_j^\xi - S_i^\xi S_j^\eta \right)
        & \cos(\mathbf{Q}\cdot \mathbf{d}_{i,j} \right) \left( S_i^\mu S_j^\mu + S_i^\xi S_j^\xi\right)
        \end{align}
```

Spin state with a small field along the z axis:

$$|n\rangle \equiv S, m = S - n$$

Ladder operators for a harmonic oscillator:

$$\begin{aligned} a^+|n\rangle &= \sqrt{n+1}|n+1\rangle \\ a|n\rangle &= \sqrt{n}|n-1\rangle \\ \left[a, a^+\right] &= 1 \end{aligned}$$

Ladder operators for spin:

$$S^-|n\rangle = \sqrt{2S}\sqrt{1 - \frac{n}{2S}}\sqrt{n+1}|n+1\rangle$$

$$S^+|n\rangle = \sqrt{2S}\sqrt{1 - \frac{n-1}{2S}}\sqrt{n}|n-1\rangle$$

SW

Assuming $\langle n \rangle \ll S$ the coupled harmonic oscillator is a good model. The excited states of the harmonic oscillator are magnons.

Spin wave approximation

Holstein–Primakoff transformation

Ladder operators for spin:

$$S^- = \sqrt{2S}a^+\hat{f}$$
$$S^+ = \sqrt{2S}\hat{f}a$$

Where:

$$\hat{f} = \sqrt{1 - \frac{n}{2S}}$$

Applying the transformation, S in boson creation and annihilation operators:

$$S_i^\eta = \frac{1}{2}(S^+ + S^-) = \sqrt{S/2}\hat{f}(a_i^+ + a_i)$$
$$S_i^\mu = -\frac{i}{2}(S^+ - S^-) = \sqrt{S/2}\hat{f}(a_i^+ - a_i)$$
$$S_i^\xi = S - n_i = S - a_i^+ a_i$$

So:

$$S_i^\eta S_j^\eta = S/2 \hat{f}_i \hat{f}_j (a_i^+ + a_i)(a_j^+ + a_j)$$

etc...

We have a hamiltonian of boson operators which can be expanded in powers of $1/S$ to obtain:

$$\mathcal{H} = E_0 + \mathcal{H}_1 + \mathcal{H}_2 + \mathcal{H}_3 + \mathcal{H}_4 \ldots$$

Where E_0 is the classical result which is derived in many textbooks:

$$E_0 = S^2 \sum_{i,\mathbf{d}} J(\mathbf{d}) \cos(\mathbf{Q} \cdot \mathbf{d})$$

1 operator term:

$$\mathcal{H}_1 = S^{3/2} \sum_{i,j} \frac{i}{2} J(\mathbf{d}_{i,j}) \sin(\mathbf{Q} \cdot \mathbf{d}_{i,j})(a_i^+ - a_i + a_j - a_j^+)$$

2 operator term:

$$\begin{aligned}\mathcal{H}_2 = S \sum_{i,j} \frac{1}{2} J(\mathbf{d}_{i,j})((1 - \cos(\mathbf{Q} \cdot \mathbf{d}_{i,j}))(a_i a_j + a_i a_j) \\ + (1 + \cos(\mathbf{Q} \cdot \mathbf{d}_{i,j}))(a_i a_j^+ + a_i^+ a_j) \\ - 2\cos(\mathbf{Q} \cdot \mathbf{d}_{i,j})(a_i^+ a_i + a_j^+ a_j))\end{aligned}$$

3 operator term:

- Non-zero in non-collinear structures
- Can change the ground state

4 operator term:

- Renormalizes the magnon dispersion
- Gives finite magnon lifetime

Luckily, $\mathcal{H}_2$ can be written in matrix form:

$$\mathcal{H}_2 = \sum_{\mathbf{k}} \mathbf{x}^\dagger H(\mathbf{k}) \mathbf{x}$$

Where $\mathbf{x}$ is a vector of Boson operators:

$$\mathbf{x} = \begin{bmatrix} a_{\mathbf{k}} \\ a^+_{-\mathbf{k}} \end{bmatrix}$$

And the matrix of the Hamiltonian has the form:

$$H = \begin{bmatrix} A & B \\ B & A \end{bmatrix}$$

$$A = J(\mathbf{k}) + J(\mathbf{k}+\mathbf{Q})/2 + J(\mathbf{k}-\mathbf{Q})/2 - 2J(\mathbf{Q})$$
$$B = J(\mathbf{k}) - J(\mathbf{k}+\mathbf{Q})/2 - J(\mathbf{k}-\mathbf{Q})/2$$

In Bogoliubov method, we define new operator b with the following transformation:

$$b = ua + va^{+}$$
$$b^{+} = ua^{+} + va$$

The new operator has to fulfill the commutation relations:

$$[b, b^{+}] = 1$$
$$u^2 + v^2 = 1$$

With the right parameter choice:

$$\mathcal{H}_2 = \sum_{\mathbf{k}} \omega_{\mathbf{k}} \left(b^{+} b + \frac{1}{2} \right)$$

And the spin wave dispersion:

$$\omega_{\mathbf{k}} = \sqrt{A^2 - B^2}$$

The above calculation is equivalent to solving the eigenvalue problem of gH, where $g = [x, x^{\dagger}]$ commutator matrix.

Now we have $\omega_{\mathbf{k}}$ and half the story. Remember for neutron scattering:

$$\frac{d^2\sigma}{d\Omega dE} = C \cdot F^2(\mathbf{K}) \sum_{\alpha,\beta} (\delta_{\alpha,\beta} - \hat{k}_\alpha \hat{k}_\beta)(gSg^T)^{\alpha,\beta}(\mathbf{k}, \omega)$$

With the correlation function:

$$S^{\alpha\beta}(\mathbf{k}, \omega) = \frac{1}{2\pi\hbar} \int dt e^{-i\omega t} \left\langle S^\alpha(\mathbf{k}, 0) S^\beta(-\mathbf{k}, t) \right\rangle$$

General spin Hamiltonian - SpinW

How does SpinW work?

General spin Hamiltonian:

$$H = \sum_{mi,nj} \mathbf{S}_{mi}^{T} \cdot J_{mi} \cdot \mathbf{S}_{nj} + \sum_{mi} \mathbf{S}_{mi}^{T} \cdot A_i \cdot \mathbf{S}_{mi} + \mu_B \mathbf{H}^{T} \sum_{mi} g_i \mathbf{S}_{mi}$$

With the anisotropic and antisymmetric (Dzyaloshinskii-Moriya) exchange interactions:

$$J_S = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix}; J_A = \begin{bmatrix} 0 & D_z & -D_y \\ -D_z & 0 & D_x \\ D_y & -D_x & 0 \end{bmatrix}$$

Key Points:

Non-Bravais lattice
- Additional rotation on every site within unit cell

General interactions
- Multi-q magnetic ground states are possible

SpinW

- Solves the general spin Hamiltonian
- Calculates spin-spin correlation function
- Numerical and symbolical
- Can apply crystal symmetry operators on the Hamiltonian - Solving single-q magnetic structures
- Solves multi-q magnetic structures on a magnetic supercell
- Open source, runs on MATLAB and now python
- More information: http://www.spinw.org
- Download from: https://www.github.com/spinw/spinw

Operating System

- All modern OS's are supported

MATLAB

- All versions after R2014b are fully supported.
- OPTIONAL: Symbolic Toolbox
- OPTIONAL: Parallel Computing Toolbox

Notes

- More memory is needed if you have more magnetic atoms
- More q-points takes longer
- Python is now technically supported but will not be covered here. See https://www.github.com/spinw/PySpinW

Now lets install

Get the code - The manual way:

SpinW is available at:
https://www.github.com/spinw/spinw/releases/latest

Steps:

- Unzip the archive into your preferred directory
- Open MATLAB and run `install_spinw` from inside this directory
- Verify with `s = spinw;`

Updating:

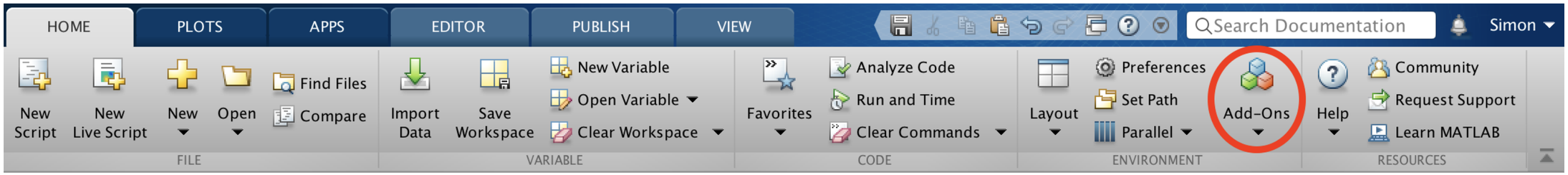
I am trying to make releases 2-3 times a year. It's always nice to be on the latest code!
There is a self update function `sw_update`
This retrieves and installs the package from the link above.

Developing:

If you want to help develop a feature and contribute, please git clone https://www.github.com/spinw/spinw.git and create a pull request (to development branch)

Using the MATLAB package - The easy way

SpinW is now a MATALB add-on and can be downloaded directly from Mathworks.



Then search for SpinW and hit install.

Verify with `s = spinw;`

Updating:

Add-Ons → Check for updates → Update

NOTE

This version may not correspond to the `sw_update` version!

Function help

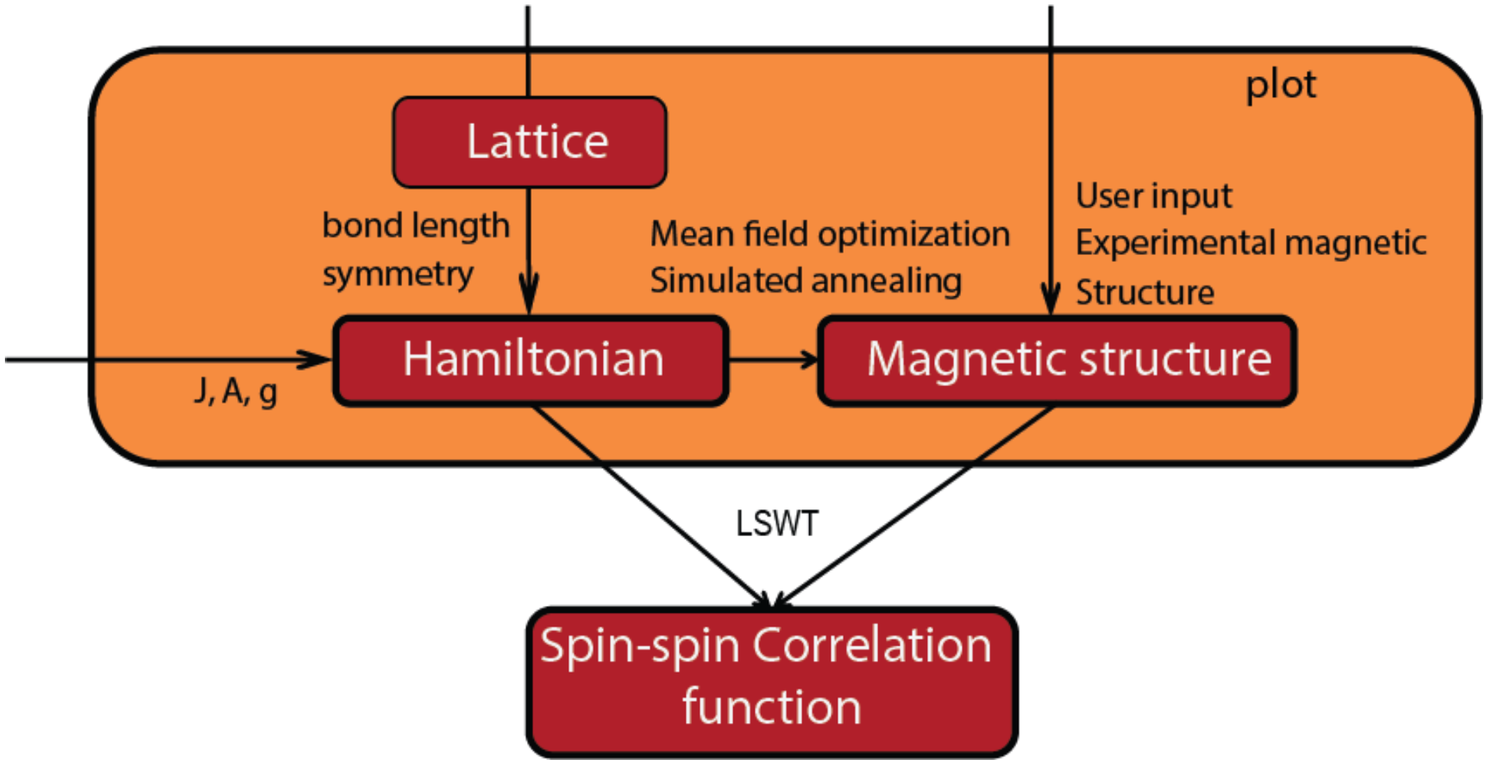
For any function that starts with `sw_*` use:
`help sw_*`

SpinW class methods

for spinw class methods use:
`help spinw.function_name`.
For help on plotting commands, use:
`help swplot`.

Online Documentation

All help can be found on http://www.spinw.org or https://spinw.github.io/spinwdoc



Data transfer diagram in SpinW

Tutorials 1

Getting started in SpinW

Excitations on a triangular lattice

Download the script here: `sw_tutorial_01.m`

$k = [1\,1\,0]/3$ magnetic structure

Let's try this with a $k = [1\,1\,0]/3$ magnetic structure

Creating the lattice

```
tri = spinw;
tri.genlattice('lat_const',[3 3 4],'angled',[90 90 120])
plot(tri)
```

We have:

- Created a SpinW object
- Generated a lattice of $a = 3\text{Å}$, $b = 3\text{Å}$, $c = 4\text{Å}$ and $\alpha = \beta = 90^\circ$, $\gamma = 120^\circ$

In the plot window, you can zoom with the mouse wheel, pan by pressing the Ctrl button while dragging. Change the plot range and view direction by pressing the corresponding button on the top.

Questions:

What is the default symmetry and what does it mean?

Adding atoms

```
tri.addatom('r',[0 0 0],'S',3/2,'label','MCr3')
plot(tri)
```

We have added an magnetic Cr^{3+} at position $[0, 0, 0]$ with spin $S = 3/2$

Creating the Spin-Hamiltonian

We create an antiferromagnetic first neighbor Hamiltonian plus easy plane single ion anisotropy

```
A0 = -0.1;
tri.addmatrix('label','J1','value',1)
tri.addmatrix('label','A','value',[0 0 0;0 0 0;0 0 A0])

tri.gencoupling

tri.addcoupling('mat','J1','bond',1)
tri.addaniso('A')

plot(tri,'range',[3 3 1/2],'cellMode','inside')
```

Red ellipsoids represent the single ion anisotropy on the plot (equienergetic surface)

Questions:

What have we done in each code part?
Examine the plot and test different values of `A0` with different signs

Creating the magnetic structure:

We have seen the ground state magnetic structure of the above Hamltonian is a spiral, with propagation vector of $(1/3, 1/3, 0)$.
We define the plane of the spiral as the ab plane

```
tri.genmagstr('mode', 'helical', 'S', [1;0;0], 'k',[1/3 1/3 0], 'n', [0 0 1], 'nExt', [1 1 1])
plot(tri, 'range', [3 3 1/2], 'cellMode', 'inside', 'magColor', 'red')
```

Careful: the given spin vector is column vector!

Questions:

What are the angles between nearest neighbor moments?

Calculating the spin wave dispersion

We calculate the spin wave dispersion along the $(H, H, 0)$ high symmetry direction

```
spec = tri.spinwave({[0 0 0] [1 1 0] 500}, 'hermit', false);
figure
sw_plotspec(spec, 'mode', 'disp', 'imag', true, 'colormap', [0 0 0], 'colorbar', false)
axis([0 1 0 5])
```

Questions:

How many modes are there and why?
What does the red line mean?
Did you get any warning?

Calculating the spin-spin correlations

The spin-spin correlations are already calculated, however it contains 9 numbers per Q-point per mode. It is not possible to show this on a single plot. But:

1. we can calculate the neutron scattering cross section
2. we can select one of the components $S^{\alpha\beta}(\mathbf{Q}, \omega)$
3. we can sum up the diagonal $S^{\alpha\alpha}(\mathbf{Q}, \omega)$

```
spec = sw_egrid(spec, 'component', {'Sxx+Syy' 'Szz'}, 'Evect', 0:0.01:5);
% Try other components!
figure
sw_plotspec(spec,'mode','color','dE',0.2,'imag',false)
axis([0 1 0 5.5])
caxis([0 3])
```

Questions:

How is it related to the magnetic propagation vector?
Why are some modes gapped? Which correlations are gapped?
Why do we have `Szz`?

$k = 0$ magnetic structure

Let's try this with a $k = 0$ magnetic structure

$k = 0$ magnetic structure

Duplicate the original object using the `.copy()` command,
Why are we using the `.copy()` command?

```
triNew = copy(tri);
triNew.genmagstr('mode','rotate','n',[0 0 1])
phi1 = atan2(triNew.magstr.S(2,1),triNew.magstr.S(1,1));
triNew.genmagstr('mode','rotate','n',[0 0 1],'phi',-phi1)
plot(triNew,'range',[3 3 1])
```

Compare the energy per spin of the old magnetic structure and the new magnetic structure using the `spinw.energy()` function.

Questions:

How does the magnetic structures compare?
Are they the same?
Why?

Calculating the spin wave dispersion

We calculate the spin wave dispersion along the $(H, H, 0)$ high symmetry direction

```
spec = triNew.spinwave({[0 0 0] [1 1 0] 500}, 'hermit', false);
figure
subplot(2, 1, 1)
sw_plotspec(spec, 'mode', 'disp', 'imag', true, 'colormap', [0 0 0], 'colorbar', false)
axis([0 1 0 5])
spec = sw_egrid(spec, 'component', 'Sperp', 'Evect', 0:0.01:5.5);
subplot(2, 1, 2)
sw_plotspec(spec, 'mode', 'color', 'dE', 0.2, 'imag', false)
axis([0 1 0 5.5])
caxis([0 3])
```

Questions:

How many number of modes are there and why?
Is there more than before?
Why are there vertical lines in the dispersion?
Which structure is the correct one?

The FM kagome lattice

Download the script here: `sw_tutorial_02.m`

This tutorial will be up to you, using what you have learned in tutorial 1.

Help is available by the MATLAB command, SpinW website and for a limited time.... Me.

SpinW

A crash course on SpinW - Part 2

SpinW (spin-double-u) is a MATLAB library that can optimize magnetic structures using mean field theory and calculate spin wave dispersion and spin-spin correlation function for complex crystal and magnetic structures.

PRESENTED BY

Simon Ward

Scientific Software Developer - ESS

Duc Le

Instrument Scientist - ISIS Facility

Magnetic Structures in SpinW

Defining and refining a magnetic structure in SpinW

$$\hat{S}^{\pm}|m\rangle = \sqrt{(S \mp m)(S + 1 \pm m)}\,|m \pm 1\rangle$$

$$m = S, S-1, S-2, \ldots, -S$$

$$a|n\rangle = \sqrt{n}\,|n-1\rangle$$

$$a^{\dagger}|n\rangle = \sqrt{n+1}\,|n+1\rangle$$

$$n = 0, 1, 2, \ldots, \infty$$

$n = 0$ corresponds to $m = S$

Holstein-Primakoff Transformation

In linear spin wave theory, the vacuum state $n = 0$ corresponds to the fully ordered state $m = S$

Introduction

Linear spin wave theory is about small deviations of the spins away from their (ordered) ground state

Therefore, before calculating the spin precessions, we need to define the ordered state

There are two main ways to define the magnetic structure in SpinW:

- Directly, by specifying the spin directions in the (super)lattice
- For single-k structures, the propagation vector and initial spin direction can be given instead
- Single-k structures can also be defined by an initial direction and an angular offset

$$\hat{S}^{\pm}|m\rangle = \sqrt{(S \mp m)(S+1 \pm m)}\,|m \pm 1\rangle$$

$$m = S, S-1, S-2, \ldots, -S$$

$$a|n\rangle = \sqrt{n}\,|n-1\rangle$$

$$a^{\dagger}|n\rangle = \sqrt{n+1}\,|n+1\rangle$$

$$n = 0, 1, 2, \ldots, \infty$$

$n = 0$ corresponds to $m = S$

Holstein-Primakoff Transformation

In linear spin wave theory, the vacuum state $n = 0$ corresponds to the fully ordered state $m = S$

Because of the Hamiltonian in SpinW is formulated in a rotating coordinate system, defining a single-k magnetic structure using a propagation vector is computationally more efficient than defining a supercell

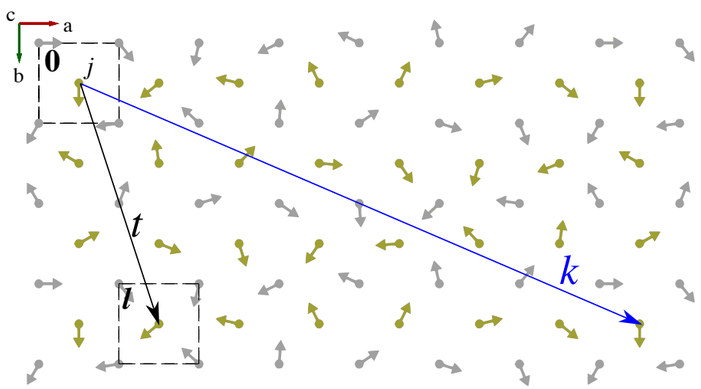
This method will also allow the definition of a true incommensurate structure

This is in contrast to similar programs such as SpinWaveGenie and McPhase which only allow the supercell definition

However, multi-k and more complex structures cannot be defined in this way and will need a supercell

Magnetic Structure Theory

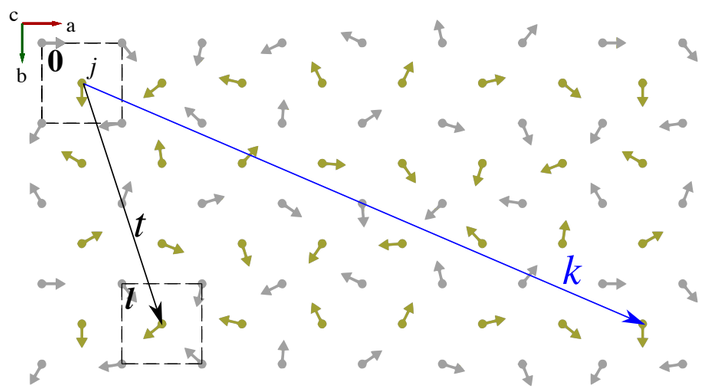
Refresher on basis and propagation vectors



Magnetic Structure Theory Refresher

The j^{th} magnetic moment in unit cell l which is separated from the first unit cell 0 by the vector t can be expressed as a Fourier series,

$$\mathbf{m}_j = \sum_n \mathbf{\Psi}_j^{\mathbf{k}_n} \exp^{-2\pi i \mathbf{k}_n \cdot \mathbf{t}}$$



$$\mathbf{m}_j = \sum_n \mathbf{\Psi}_j^{\mathbf{k}_n} \exp^{-2\pi i \mathbf{k}_n \cdot \mathbf{t}}$$

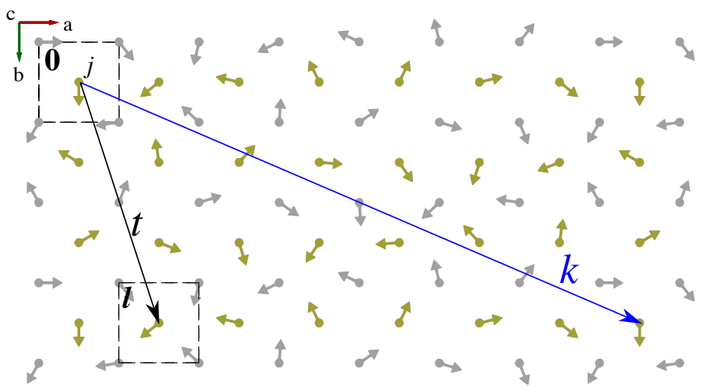
For many materials, there is only a single propagation vector k:

$$\mathbf{m}_j = \mathbf{\Psi}_j^{\mathbf{k}} \exp^{-2\pi i \mathbf{k} \cdot \mathbf{t}}$$

The allowed propagation vectors $\mathbf{k}_n$ are related to each to each other by the rotation symmetry of the crystal structure (they are the *star* of k).

In the case of a single-k magnetic structure, the spin wave Hamiltonian is invariant under all rotations

This allows the Hamiltonian to be expressed in a rotating coordinate system which allows SpinW to calculate more efficiently than codes which define the magnetic structure in terms of a supercell.



$$\mathbf{m}_j = \sum_n \mathbf{\Psi}_j^{\mathbf{k}_n} \exp^{-2\pi i \mathbf{k}_n \cdot \mathbf{t}}$$

The *basis vector* $\mathbf{\Psi}_j^{\mathbf{k}}$ is in general complex.

A complex basis vector requires both $\mathbf{k}$ and $-\mathbf{k}$ components to produce a real moment

$$\mathbf{m}_j = \mathbf{\Psi}_j^{\mathbf{k}} [\cos(-2\pi\mathbf{k}\cdot\mathbf{t}) + i\sin(-2\pi\mathbf{k}\cdot\mathbf{t})] + \mathbf{\Psi}_j^{-\mathbf{k}} [\cos(2\pi\mathbf{k}\cdot\mathbf{t}) + i\sin(2\pi\mathbf{k}\cdot\mathbf{t})]$$

$$= 2\,\mathrm{Re}(\mathbf{\Psi}_j^{\mathbf{k}})\cos(-2\pi\mathbf{k}\cdot\mathbf{t}) + 2\,\mathrm{Im}(\mathbf{\Psi}_j^{\mathbf{k}})\sin(-2\pi\mathbf{k}\cdot\mathbf{t})$$

because $\mathbf{\Psi}_j^{-\mathbf{k}} = (\mathbf{\Psi}_j^{\mathbf{k}})^{\dagger} = \mathrm{Re}(\mathbf{\Psi}_j^{\mathbf{k}}) - i\,\mathrm{Im}(\mathbf{\Psi}_j^{\mathbf{k}})$

A real basis vector will only give a collinear magnetic structure, but possibly with a varying moment magnitude

A complex basis vector with imaginary part perpendicular to the real part can give helical magnetic structures.

Reference: A.S. Wills, *J. Phys. IV France*, **11** (Pr9) 133-158 (2001).

https://doi.org/10.1051/jp4:2001906

spinw.mag_str

How SpinW stores the magnetic structure

SpinW stores the magnetic structure in the **`spinw.mag_str`** field.

It can store arbitrary magnetic structures using Fourier components.

Subfields:

- The propagation vectors $\mathbf{k}$ are stored in **`k`**
- The basis vectors $\mathbf{\Psi}_j$ are stored in **`F`**
- The magnetic supercell in lattice units are stored in **`nExt`**

The experimental magnetization can be obtained by multiplying **`F`** with the g-tensor!

Propagation vector: **`mag_str.k`**

Basis vector: **`mag_str.F`**

Magnetic supercell: **`mag_str.nExt`**

A helical or modulated single-k structure can be stored by using **`k`** and **`F`** and setting **`nExt`** to a single unit cell **`[1 1 1]`**

If there are `n` atoms in the structural unit cell, **`F`** should be an `n`-column matrix.

A true *incommensurate* magnetic structure can be generated by giving an irrational wavevector k

Multi-k incommensurate structures may only be approximated in SpinW (as in other codes) using a supercell

Propagation vector: **`mag_str.k`**

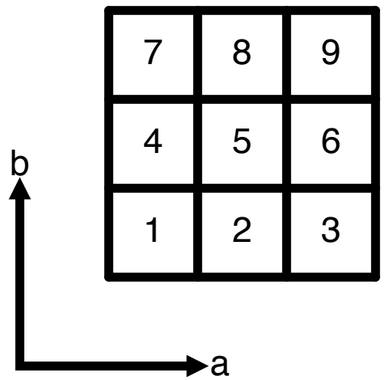
Basis vector: **`mag_str.F`**

Magnetic supercell: **`mag_str.nExt`**

A supercell magnetic structure can be stored by using a real **`F`** and **`nExt`** and setting **`k`** to zero **`[0 0 0]`**.

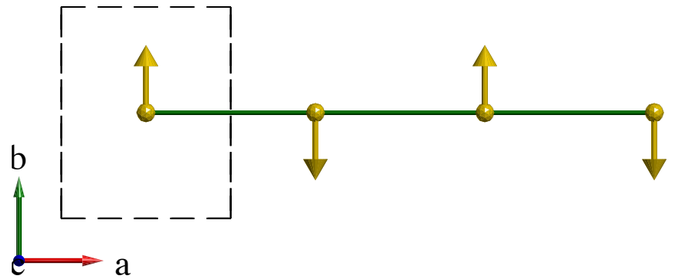
The number of magnetic moments stored in `F` are: `nMagExt = prod(nExt)*nMagAtom`

So, **`F`** should be an `nMagExt`-column matrix with moments in the following order:



Ordering of the unit cell in the magnetic super-cell.

1D AFM spin chain



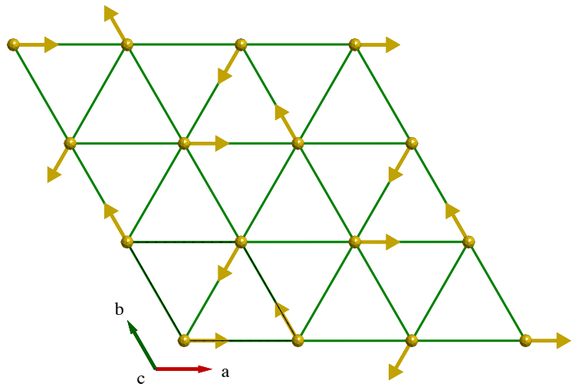
Single-k mode

```
mag_str.N_ext = [1 1 1];
mag_str.k     = [1/2 0 0];
mag_str.F     = [0;
                 1;
                 0];
```

Supercell mode

```
mag_str.N_ext = [2 1 1];
mag_str.k     = [0 0 0];
mag_str.F     = [0 0;
                 1 -1;
                 0 0];
```

120° structure on a triangular lattice



Single-k mode

```
mag_str.N_ext = [1 1 1];
mag_str.k     = [1/3 1/3 0];
mag_str.F     = [1;
                 i;
                 0];
```

Supercell mode

```
mag_str.N_ext = [3 3 1];
mag_str.k     = [0 0 0];
mag_str.F     = [1 -0.5  -0.5  -0.5  -0.5     1  -0.5     1
-0.5;
                 0 0.86 -0.86  0.86 -0.86     0 -0.86     0
0.86;
                 0    0     0     0     0     0     0     0
0];
```

spinw.genmagstr

How to define a magnetic structure in SpinW

Generating a magnetic structure

Rather than changing the `mag_str` field directly, it is recommended to use the `genmagstr()` function to generate the magnetic structure

This function checks your input for errors and also provides short-cuts for common use-cases, using various `modes`:

`helical`	single-k helix
`fourier`	single-k helix or modulated structure
`rotate`	uniform rotation of all moments
`direct`	direct input of structure using all fields `k`, `F`, `nExt`
`tile`	tile a magnetic supercell
`func`	using a function to generate `k`, `F`, `nExt`
`random`	random moments

`genmagstr()` always respects `nExt`, so a combination of an input `nExt` and `k` will generate a magnetic supercell which is extended first by `nExt` and then by `k`

spinw.genmagstr('mode',...)

HELICAL

- Extend the given structure by applying rotations on the moments
- Moments are either given in rotating frame formalism (S,n) or as complex vectors (S)

```
chain.genmagstr('mode', 'helical', 'S', [1; 0; 0], 'n', [0 0 1], 'k', [1/8 0 0])
chain.genmagstr('mode', 'helical', 'S', [1; 1i; 0], 'k', [1/8 0 0])
```

FOURIER

- Generate a single-k structure using Fourier components in S

```
chain.genmagstr('mode', 'fourier', 'nExt', [8 1 1], 'S', {[1; 1i; 0] [1/8 0 0]})
```

ROTATE

- Uniform rotation of all existing moments

```
chain.genmagstr('mode', 'rotate', 'n', [1 0 0])
```

spinw.genmagstr('mode',...)

DIRECT

- Direct input of every field

```
chain.genmagstr('mode', 'direct', 'nExt', [4 1 1], 'S', [1 0 -1 0; 0 1 0 -1; 0 0 0 0]);
```

TILE

- Tile a magnetic supercell using the given data

```
chain.genmagstr('mode', 'tile', 'nExt', [2 1 1], 'S', [1 0; 0 -1; 0 0]);
```

spinw.genmagstr('mode',...)

FUNC

- Give parameters to a constraint function to generate magnetic structure

```
chain.genmagstr('mode', 'func', 'func', @gm_spherical3d, 'x', [pi/2 0.2 pi/2 0.4 pi/2 0.6 pi/2 0.8 0 0 0 0 0])
```

SpinW provides two built-in functions, `gm_planar` and `gm_spherical3d`

- `gm_planar` produces a coplanar structure with fittable relative angles between spins and propagation vectors
- `gm_spherical3d` is a generalisation of this for non-coplanar structure

RANDOM

- Random magnetization vectors

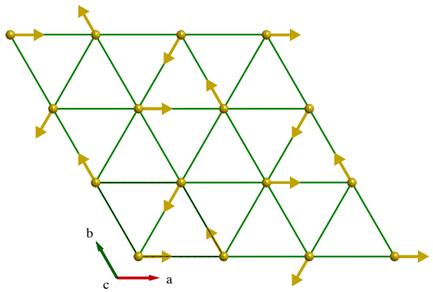
```
chain.genmagstr('mode', 'random', 'nExt', [4 1 1])
```

Examples

Examples of magnetic structures in SpinW

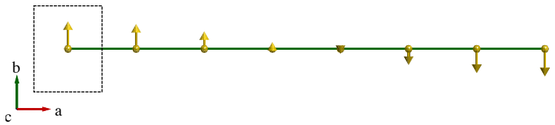
Helical

```
tri = spinw;
tri.genlattice('lat_const', [4 4 6], 'angled', [90 90 120]);
tri.addatom('r', [0 0 0], 'S', 2, 'label', 'MCr3', 'color', 'gold');
tri.genmagstr('mode', 'helical', 'S', [1; 0; 0], 'n', [0 0 1], 'k', [1/3 1/3 0])
```



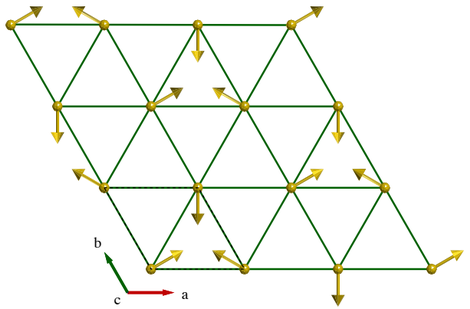
Fourier

```
mmod = spinw;
mmod.genlattice('lat_const', [4 4 6], 'angled', [90 90 90]);
mmod.addatom('r', [0.5 0.5 0.5], 'S', 2, 'label', 'MCr3', 'color', 'gold');
mmod.genmagstr('mode', 'fourier', 'S', [0; 1; 0], 'k', [0.07 0 0])
```



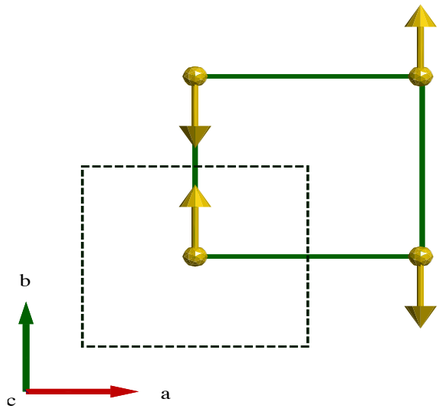
Rotate

```
tri = spinw;
tri.genlattice('lat_const', [4 4 6], 'angled', [90 90 120]);
tri.addatom('r', [0 0 0], 'S', 2, 'label', 'MCr3', 'color', 'gold');
tri.genmagstr('mode', 'helical', 'S', [1; 0; 0], 'n', [0 0 1], 'k', [1/3 1/3 0])
tri.genmagstr('mode', 'rotate', 'n', [0 0 1], 'phid', 30)
```



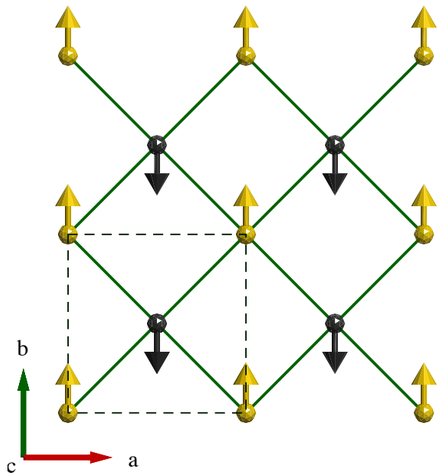
Direct

```
sq = spinw;
sq.genlattice('lat_const', [4 4 6], 'angled', [90 90 90]);
sq.addatom('r', [0.5 0.5 0.5], 'S', 2, 'label', 'MCr3', 'color', 'gold');
sq.genmagstr('mode', 'direct', 'S', [0 0 0 0; 1 -1 -1 1; 0 0 0 0], 'nExt', [2 2 1])
```



Tile

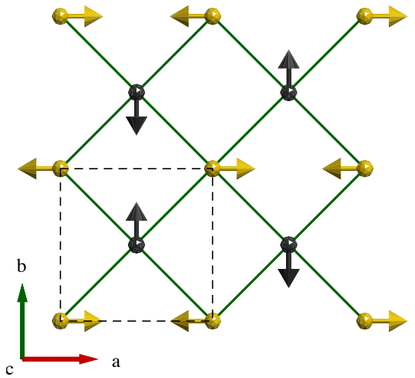
```
fct = spinw;
fct.genlattice('lat_const', [4 4 6], 'angled', [90 90 90]);
fct.addatom('r', [0 0 0], 'S', 2, 'label', 'MCr3', 'color', 'gold');
fct.addatom('r', [0.5 0.5 0], 'S', 2, 'label', 'MCr3', 'color', 'black');
fct.genmagstr('mode', 'tile', 'S', [0 0; 1 -1; 0 0], 'nExt', [2 2 1])
```



Function

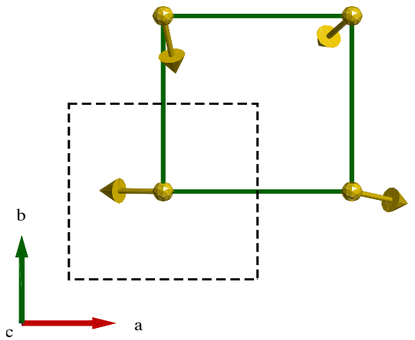
```
fct = spinw;
fct.genlattice('lat_const', [4 4 6], 'angled', [90 90 90]);
fct.addatom('r', [0 0 0], 'S', 2, 'label', 'MCr3', 'color', 'gold');
fct.addatom('r', [0.5 0.5 0], 'S', 2, 'label', 'MCr3', 'color', 'black');
fct.genmagstr('mode', 'func', 'func', @gm_planar, 'x0', [0 pi/2 1/2 1/2 1 0 0])
```

Parameters for gm_planar is: [phi1 phi2 ... kx ky kz n_theta n_phi], n_theta and n_phi define the normal of the plane. phiN are relative phases within the plane



Random

```
fct = spinw;
fct.genlattice('lat_const', [4 4 6], 'angled', [90 90 90]);
fct.addatom('r', [0.5 0.5 0], 'S', 2, 'label', 'MCr3', 'color', 'gold');
fct.genmagstr('mode', 'random', 'nExt', [2 2 1])
```



Optimising magnetic structures

How to refine structures in SpinW

The magnetic structure can be optimised as a classical ground state of the spin Hamiltonian, using:

`sw.optmagk()`

- determines the magnetic propagation vector + n-vector

`sw.optmagtsteep()`

- optimise magnetic structure for a given k-vector by succesively rotating each moment to the Weiss field direction
- fastest
- recommended if k-vector is known

`sw.optmagstr()`

- optimise magnetic structure for minimum energy using non-linear optimization (fminsearch)
- can include a constraint function (`@gm_planar()`, etc.)

`sw.anneal()`

- performs simulated annealing, using the Metropolis algorithm
- can calculate thermodynamic properties

To optimize magnetic structure with constraints, use the `spinw.optmagstr()` method:

- lots of parameters to fit: $3N_{mag} + 6$
- ground state magnetic structure with **constrained optimization** (simplex method)
- constraints through external function:

$$[M, k, n] = @(x)\mathtt{func}(M0, x)$$

- **`@gm_planar()`** planar magnetic structure:

$$[M, k, n] = @\mathtt{gm\_planar}(M0, x)$$
$$x = (\varphi_1, \varphi_2, \ldots, k_x, k_y, k_z, n_\Theta, n_\varphi)$$

  φ_n are phase angles for each of the n spins in the extended unit cell

  n_Θ and n_φ are the angles defining the plane normal
- **`@gm_spherical3d()`** general magnetic structure:

$$[M, k, n] = @\mathtt{gm\_spherical3d}(M0, x)$$
$$x = (\varphi_1, \Theta_1, \varphi_2, \Theta_2, \ldots, k_x, k_y, k_z, n_\Theta, n_\varphi)$$

  φ_n and Θ_n are azimuth and polar phase angles for each of the n spins in the extended unit cell
- limits: **`xmin`** and **`xmax`**, starting value **`x0`**

Spin Wave Calculations

Notes on how magnetic structure is used in spin wave calculations in SpinW

For spin wave calculation, the complex magnetic structure is converted to the rotating frame representation using the `spinw.magstr()` function.

Output of `spinw.magstr()` is a struct with fields:

- size of magnetic supercell in l.u. stored in **`nExt`**
- single k-vector stored in **`k`**
- vector normal to the spiral plane is stored in **`n`**
- real magnetic moment directions (spin quantization axis) are stored in **`S`**

The representation allows an additional `k=0` component parallel to `n`, however this is rarely used.

The conversion from Fourier components to rotating frame representation is not always possible, in this case `magstr()` gives the best approximation and gives a warning. The conversion is approximate:

- multi-k structures
- non-Bravais lattice with counter-rotating incommensurate spirals
- non-Bravais lattice with non-coplanar spirals
- non-Bravais antiferromagnet with non-coplanar moments

The spin wave Hamiltonian is expressed in a basis of creation and annihilation operators, one each for each spin in the magnetic unit cell

For a supercell only description of the magnetic structure $(\text{with a } k = 0 \text{ propagation vector})$, SpinW will use only this basis, so the size of the Hamiltonian will be $2N \times 2N$ where N is the number of spins in the extended unit cell.

For a single-k structure, SpinW also has to consider the $+k$ and $-k$ branches $(\text{as well as } k = 0)$ so there are $6N$ operators in the basis.

In the 1D chain case, we have only 2 spins in the unit cell, so the supercell description gives 4-operator basis but the rotating frame description needs 6 operators.

In the 2D triangular lattice, we have $k = [\frac{1}{3}\,\frac{1}{3}\,1]$ so there are 9 spins in the unit cell giving an 18-operator basis in the supercell method. In contrast in the rotating frame description, we still only have 6 operators

Larger matrices take longer to diagonalise so it's important to try to reduce the size of the Hamiltonian by using the rotating frame description if possible

The Horace-SpinW interface

How to use a SpinW model in Horace for fitting

Defining models in Horace: a recap

Horace accepts a variety of functions to model data:

`y = fn(x1, x2, ..., xn, pars)`	Functions operating directly on data coordinates (e.g. gaussian peaks)
`s = fn(qh, qk, ql, en, pars)`	Model $S(\mathbf{q}, \omega)$ functions evaluated for each ω
`[w, s] = fn(qh, qk, ql, pars)`	General model $S(\mathbf{q}, \omega)$ functions

In all cases, immediately following the coordinates, Horace expects a vector of parameter values to be fitted

After this parameter values, Horace also accepts any other input variables as model constants which will be passed to the model

The fit functions generally only accept the `s = fn(qh, qk, ql, en, pars)` form for $S(\mathbf{q}, \omega)$ models, so energy convolution needs to be done by the modelling code

The SpinW `spinwave` method: a recap

In order to calculate the spin wave spectrum in SpinW, something like the following needs to be used:

```
spec = sw_obj.spinwave(hkl, 'hermit', false, 'formfact', true);
spec = sw_egrid(spec, 'component', 'Sperp', 'Evect', 0:0.05:10);
```

Comparing with what Horace needs, we notice that:

- The model (fittable) parameters are not set here, but much earlier in the definition of the model
- We need the combination of both `spinwave` and `sw_egrid` to get a function of the form `s = fn(qh, qk, ql, en, pars)` which Horace needs

Fortunately the wrapped model function is provided in SpinW: the method `spinw.horace_sqw`

The `spinw.horace_sqw` method

`horace_sqw` has the same signature as a standard Horace $S(\mathbf{q},\omega)$ function, `horace_sqw(qh, qk, ql, en, pars, varargin)`

So, it can be used directly in a Horace `multifit_sqw` call.

In order to define which model parameter is to be varied in the fit, you have to give `horace_sqw` a `mat` parameter which is a cell array of the matrix names to be varied in the order they appear in the `pars` vector

Since the parameters of `pars` are scalars, if the matrix you refer to is not isotropic (e.g. it's not representing a Heisenberg interaction), a special syntax to refer to which matrix element(s) needs to vary has to be used.

A simple example:

```
J = 1.2;
K = 0.1;
tri = sw_model('triAF', J);
tri.addmatrix('label', 'K', 'value', diag([0 0 K]));
tri.addaniso('K');

fwhm = 0.75;
scalefactor = 1;
ws = cut_sqw(sqw_file, [0.05], [-0.1, 0.1], [-0.1, 0.1],
[0.5]);
fitobj = multifit_sqw(ws);
fitobj.set_fun(@tri.horace_sqw);
fitobj.set_pin({[J K fwhm scalefactor], 'mat', {'J_1',
'K(3,3)'}, ...
    'hermit', false, 'useFast', true, 'formfact', true});
ws_sim = fitobj.simulate();
[ws_fit, fit_dat] = fitobj.fit()
```

The vector `[J K fwhm scalefactor]` is the parameters vector. We need to tell SpinW that it corresponds to the Heisenberg nearest neighbour interaction `J_1` and the easy-place anisotropy `K`

Because `J` is isotropic, we can just give the matrix name in `mat`

But, `K` only applies to the *zz* element, so we need to tell SpinW that in `mat`

`fwhm` and `scalefactor` are parameters which are added by `horace_sqw` to denote the energy FWHM and intensity scale factor (may be omitted, in which case it is taken to be unity and fixed)

The other (non-varying) parameters we pass to `multifit` are just standard SpinW keyword arguments

There are a few keyword arguments unique to `horace_sqw`

- `'useFast'` - This tells `horace_sqw` to use a faster but slightly less accurate code than `spinwave.` In particular, this code achieves a speed gain by:
  - Only calculating `Sperp` rather than full $S^{\alpha\beta}$ tensor
  - Only calculating magnon creation (positive energy / neutron energy loss) modes.
  - Ignoring twins
- `'partrans'` - A function handle to transform the input parameters received from Horace before passing to SpinW
- `'coordtrans'` - A 4×4 matrix to transform the input $(Q_h, Q_k, Q_l, \hbar\omega)$ coordinates received from Horace before passing to SpinW
- `'resfun'` - This is tells `horace_sqw` what function to use for the energy convolution. Options are:
  - `'gauss'` - a gaussian (one parameter: fwhm)
  - `'lor'` - a lorentzian (one parameter: fwhm)
  - `'voigt'` - a pseudovoigt (two parameters: fwhm and lorentzian fraction)
  - `'sho'` - a damped harmonic oscilator (parameters: Gamma Temperature Amplitude)
  - A function handle to a function which will be accepted by Horace's `disp2sqw` method

`horace_sqw` appends the parameters needed by `resfun` to the end of the parameter vector and then adds a scale factor between the data and calculation after that

The `'mat'` argument

Horace expects a parameter vector, so we have to tell SpinW which parameter is which

In simple cases, just the name of the corresponding SpinW matrix, or a string denoting which single matrix element suffice

For more complicated cases, an additional parameter `'selector'`, a 3×3 logical matrix needs to be used

This tells the `matparser` function which SpinW uses to decode the `'mat'` argument which matrix elements the parameter corresponds to

```
Dvec = [0.1 0.2 0.3];
swobj.addmatrix('label', 'DM', 'value', Dvec);
swobj.addcoupling('mat', 'DM', 'bond', 1);

sel(:,:,1) = [0 0 0; 0 0 1; 0 -1 0];    % Dx
sel(:,:,2) = [0 0 1; 0 0 0; -1 0 0];    % Dy
sel(:,:,3) = [0 1 0; -1 0 0; 0 0 0];    % Dz

fitobj.set_fun(@swobj.horace_sqw);
fitobj.set_pin({Dvec, 'mat', {'DM', 'DM', 'DM'}, ...
    'selector', sel, 'hermit', false})
fitobj.fit()
```

`'selector'` is a $3 \times 3 \times N$ array where N is the number of parameters

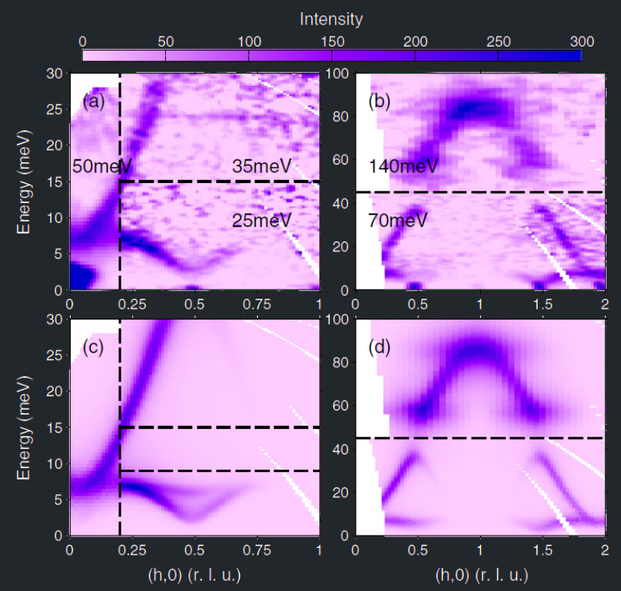
Each 3×3 matrix denotes which elements of the corresponding matrix in `'mat'` goes with that parameter

Example of Horace-SpinW integration

Modelling spin waves in $Pr(Ca_{0.9}Sr_{0.1})_2Mn_2O_7$

Download the scripts here: `pcsmo_eval.m`

This can be done by yourself if you are interested. Speak to one of us



Published data and simulation of $Pr(Ca_{0.9}Sr_{0.1})_2Mn_2O_7$

$Pr(Ca_{0.9}Sr_{0.1})_2Mn_2O_7$ Horace data

The `.sqw` files should be in the folders previously used for Horace

If you're not using a desktop, you can get the files from: Dropbox

Make 2D slices along $(h00)$ and energy transfer and compare to figure 2 of Johnstone et al.

```
ei = [25 35 50 70 140];
proj = projaxes([1 0 0], [0 1 0]);
for ii = 1:numel(ei);
    sqw_file = sprintf('../aaa_my_work/pcsmo_ei%d_base.sqw', ei(ii));
    ws_cut(ii) = cut_sqw(sqw_file, proj, [-5,0.025,5], [-0.2,0.2], ...
        [-inf,inf], [ei(ii)/100]);
    % Symmetrise about h=0
    ws_cut(ii) = symmetrise_sqw(ws_cut(ii), [0 0 1], [0 1 0], [0 0 0]);
    plot(ws_cut(ii))
    lz 0 10
    keep_figure;
end
```

Background subtraction

As in the Horace tutorials, make a cut at high q and use replicate to generate a 2D background slice

Subtract this from the data and compare it to the paper

```
for ii = 1:numel(ei)
    idx = find(sum(ws_cut(ii).data.s,2)>0);
    qmax = ws_cut(ii).data.p{1}(idx(end)) * 0.5;
    ws_bkg(ii) = cut_sqw(ws_cut(ii), [qmax-0.1,qmax], []);
    ws_sub(ii) = ws_cut(ii) - replicate(ws_bkg(ii), ws_cut(ii));
    plot(ws_sub(ii))
    lz 0 10
    lx -2 2
    keep_figure;
end
```

Evaluate the SpinW model

Use the code from the previous ("real world") tutorial to set up a SpinW model of the Goodenough model

Evaluate this model on the cuts you've made:

- Because the model had to use a larger $(2 \times 2 \times 1)$ "structural" unit cell, we need to convert the Q_h and Q_k coordinates given by Horace (multiply them by 2) to get the SpinW equivalent:

```
cpars = {'coordtrans', diag([2 2 1 1])}
```

- Add the usual SpinW options:

```
cpars = {cpars{:}, 'mat', {'JF1', 'JA', 'JF2', 'JF3', 'Jperp', 'D(3,3)'}, ...
    'hermit', false, 'optmem', 0, 'useFast', true, 'formfact', true, ...
    'resfun', 'gauss'};
```

EVALUATE THE SPINW MODEL

Select the 170 meV dataset to see the overal dispersion (including gap)

Then set up a multifit object on this dataset - use a `dnd` object to save time (the `sqw` object will have ~100x the number of pixels)

Finally tell SpinW to use mex files to speed up the calculation and evaluate the SpinW model

```
idx = 5;             % EIs: [25 35 50 70 140], index 5 == 140meV
fwhm = ei(idx)/30;   % Typical resolution ~ 3% of Ei

kk = multifit_sqw(d2d(ws_sub(idx)));
kk = kk.set_fun (@pcsmo.horace_sqw, {[JF1 JA JF2 JF3 Jperp D fwhm] cpars{:}});
kk = kk.set_free ([1, 1, 1, 1, 1, 1, 1]);
kk = kk.set_options ('list',2);

swpref.setpref('usemex',true);
% Time a single iteration
tic
wsim = kk.simulate;
t_spinw_single = toc;
```

Example of Horace-SpinW fitting

Fitting spin waves in bcc-Iron with SpinW and Horace
Download the scripts here: `fe_fit.m`
Scripts and data are on your USB sticks.

MAKE CUTS OF FE DATA

Make the set of standard Q_h 1D cuts at different energies as before

```
proj = projaxes([1 1 0], [-1 1 0]);
energy_range = [80:20:160];
for i = 1:numel(energy_range)
    my_cuts(i) = cut_sqw(sqw_file, proj, [-3,0.05,3], [-1.05,-0.95], [-0.05,0.05], ...
        [-10 10]+energy_range(i));
end
```

Run the same fits with the analytical $S(\mathbf{q}, \omega)$ function as before and note the fitted parameters and how long it takes

SW

SPINW MODEL

Define a SpinW model for bcc-Fe with a single nearest neighbour exchange and a small easy axis anisotropy

```
a = 2.87;

fe = spinw;
fe.genlattice('lat_const', [a a a], 'angled', [90 90 90], 'spgr', 'I m -3 m')  % bcc Fe
fe.addatom('label', 'MFe3', 'r', [0 0 0], 'S', 5/2, 'color', 'gold')
fe.gencoupling()
fe.addmatrix('label', 'J1', 'value', 1, 'color', 'gray')
fe.addmatrix('label', 'D', 'value', diag([0 0 -1]), 'color', 'green')
fe.addcoupling('mat', 'J1', 'bond', 1)
fe.addaniso('D')
fe.genmagstr('mode', 'direct', 'S', [0 0 1; 0 0 1]');  % Ferromagnetic

plot(fe, 'range', [2 2 2])
```

Set the parameters for the fits, and also SpinW options

Note that the analytical expression used previous for $S(\mathbf{q},\omega)$ used JS, whereas SpinW uses J, and we have defined $S=\frac{5}{2}$

```
% Initial parameters:
J = -16;
D = -0.1;
gam = 66;
temp = 10;
amp = 131;

cpars = {'mat', {'J1', 'D(3,3)'}, 'hermit', false, 'optmem', 1, ...
    'useFast', true, 'resfun', 'sho', 'formfact', true};
swpref.setpref('usemex',true);
```

Notice that we use the `sho` resolution function, in common with the analytical expressions

Because the cuts are small, we also force SpinW to calculate all q-points in one chunk with the `{'optmem', 1}` option.

Set up the multifit object on the array of 1D cuts

Use the `horace_sqw` method as the fit function and parameters as defined previously, and run the fit

```
kk = multifit_sqw (my_cuts);
kk = kk.set_fun (@fe.horace_sqw, {[J D gam temp amp] cpars{:}});
kk = kk.set_free ([1, 0, 1, 0, 1]);
kk = kk.set_bfun (@linear_bg, [0.1,0]);
kk = kk.set_bfree ([1,0]);
kk = kk.set_options ('list',2);

[wfit, fitdata] = kk.fit('comp');
```

Compare the fits - are the parameters or correlation matrix consistent?

SpinW

Thank you!

Well done if you're still awake!


SW
SpinW