Info about the Mufit program

B. Roessli

June 23, 2009

0.1 Introduction

The Mufit program has been written to calculate and fit neutron nuclear, magnetic intensities and polarimetry data. It runs under Octave (>= 3.1.55) that can be downloaded from www.octave.org. It is is not compatible with Matlab. Mufit is free and the sources are available and can be modified. To start the program, just type Mufit in an Octave session.

An executable, that does not need to install Octave, can be found in the subdirectory LINUX.

Mufit needs three input files:

- nuc.inp;
- crys.inp;
- mag.inp'

0.2 Input files

An example of a parameter file used to fit a chemical structure. The text between {} should be removed before using this file. Generally, the first line of an input is used to perform simulation, while the second line contains the label of the parameter that is used to fit the data.

```
MN  -0.373     0.41125  0.35137  0.50000  0.363   0.50000
                   N6       N7  0.50000  N17     0.50000
```

The starting values of the parameters are given in the lines

```
!N1     N2    N3        N4         N5       N6       N7        N8       N9        N10
906.   10000  0.13758   0.17149   0.25152  0.41531   0.34961  0.2840   0.16581   0.44452
```

while the steps in the lines

```
!errors dN1 to dN10
1        1      1.       0       1.      1.       1.        1.      1.         1.
```

A 0 value indicates that the parameter is fixed. Note that it is possible to correlate parameters e.g.

```
MN  -0.373     0.41125  0.35137  0.50000  0.363   0.50000
                  N6     -N6     0.50000   N17     0.50000


!wavelength
1.526
!extinction
0.00
abs(N2)  0.000
{Extinction parameter}
!scale
58.24
{scale factor used for simulation}
N1 0.000
{scale factor parameter used to fit the data}
!lattice parameters and angles
7.220708   8.461032   5.660867  90.000000  90.000000  90.000000
! number of atoms
7
{/it list of atoms}
ER   0.779     0.13658  0.17096  0.00000  0.367   0.50000
{Atomic label scattering length atomic positions B-factor occupation factor}
           N3       N4  0.00000  N16     0.50000
{labels of parameters used to fit the data}
MN  -0.373     0.00000  0.50000  0.26053  0.363   0.50000
               0.00000  0.50000       N5  N17     0.50000
MN  -0.373     0.41125  0.35137  0.50000  0.363   0.50000
                   N6       N7  0.50000  N17     0.50000
O    0.5803    0.00000  0.00000  0.27511  0.380   0.50000
               0.00000  0.00000       N8  N18     0.50000
O    0.5803    0.16883  0.44155  0.00000  0.380   0.50000
                   N9      N10  0.00000  N18     0.50000
O    0.5803    0.15360  0.43033  0.50000  0.380   0.50000
                  N11      N12  0.50000  N18     0.50000
O    0.5803    0.39549  0.20745  0.24711  0.380   1.00000
                  N13      N14      N15  N18     1.00000
```

```
!# symmetry operations in space group and symmetry elements
8
{List of the operation of the space group. Note that
Mufit will recognise 1/2-X but not -X+1/2}
      X             Y             Z
    1/2-X         1/2+Y           Z
    1/2+X         1/2-Y           Z
      X             Y            -Z
     -X            -Y            -Z
    1/2+X         1/2-Y          -Z
    1/2-X         1/2+Y          -Z
     -X            -Y             Z
!# number of translations and translations in space group
1
{The translation can be be given as above here}
0 0 0
!N1     N2    N3        N4         N5       N6       N7        N8       N9        N10
906.   10000  0.13758   0.17149   0.25152  0.41531   0.34961  0.2840   0.16581   0.44452
{There must be 10 parameters per line. At moment the maximum number of parameters is
limited to 30.}
!errors dN1 to dN10
1        1      1.       1.       1.      1.       1.        1.      1.         1.
!N11     N12      N13     N14       N15        N16    N17    N18    N19    N20
0.15156  0.43074  0.39471 0.20672   0.22928    0.302     0.3    0.3      0.2     0.
!errors dN1 to dN10
1       1          1       1         1          1         1.     1.      0.    0.
!N21  N22    N23     N24     N25    N26      N27   N28     N29    N30
0       0      0       0       0      0       0.     0.      0.    0.
!errors dN1 to DN10
0       0.       0      0       0     0       0      0.      0.    0.
! fit algorithm: levenberg-marquard (0), simplex (1), simulated annealing (2)
{Three fit algorithms are implemented at moment. The following line is a flag}
1
{The followinf lines contains the fit options: exit if chisq does not change,
maximal number of itereations, etc}
!levenberg-marquard options: stol niter verbose (0/1)
0.001 100 1
!simplex options: stol niter chisq_min simplex verbose (0/1)
1e-12 50 5  0 1
!simulated anneling options
* nt - integer: # of iterations between temperature reductions
* ns - integer: # of iterations between bounds adjustments
* rt - (0 < rt <1): temperature reduction factor
* maxevals - integer: limit on function evaluations
* neps - integer:  number of values final result is compared to
* functol -   (> 0): the required tolerance level for function value comparisons
* paramtol -  (> 0): the required tolerance level for parameters
* verbosity - scalar: 0, 1, or 2.
5  5  0.8  100  5  0.1  0.1 2
!flag intensities/structure factors^2
1
!   h        k          l     I(obs)      sqrt(I)
{List of hkl with intensities/structure factors and standard deviations}
N   4.000000   0.000000   0.000000   6751.580000   95.110000
N   2.000000   0.000000   0.000000   385.880000   11.450000
```

0.3 Examples of input files

- *nuc.inp*

```
!wavelength
1.526
!extinction
0.00
abs(N2)  0.000
!scale
58.24
N1 0.000
!lattice parameters and angles
7.220708   8.461032   5.660867  90.000000  90.000000  90.000000
! # atoms
7
ER   0.779     0.13658  0.17096  0.00000  0.367   0.50000
            N3       N4  0.00000  N16     0.50000
MN  -0.373     0.00000  0.50000  0.26053  0.363   0.50000
         0.00000  0.50000       N5  N17     0.50000
MN  -0.373     0.41125  0.35137  0.50000  0.363   0.50000
              N6       N7  0.50000  N17     0.50000
O    0.5803    0.00000  0.00000  0.27511  0.380   0.50000
        0.00000  0.00000       N8  N18     0.50000
O    0.5803    0.16883  0.44155  0.00000  0.380   0.50000
                    N9      N10  0.00000  N18     0.50000
O    0.5803    0.15360  0.43033  0.50000  0.380   0.50000
                   N11      N12  0.50000  N18     0.50000
O    0.5803    0.39549  0.20745  0.24711  0.380   1.00000
                   N13      N14      N15  N18     1.00000
!# symmetry operations in space group and symmetry elements
8
      X             Y             Z
    1/2-X         1/2+Y           Z
    1/2+X         1/2-Y           Z
      X             Y            -Z
     -X            -Y            -Z
```

```
    1/2+X         1/2-Y          -Z
    1/2-X         1/2+Y          -Z
     -X            -Y            Z
!# number of translations and translations in space group
1
0 0 0
!N1     N2    N3        N4         N5       N6       N7        N8       N9        N10
906.   10000  0.13758   0.17149   0.25152  0.41531   0.34961  0.2840   0.16581   0.44452
!errors dN1 to dN10
1        1      1.       1.        1.       1.       1.        1.       1.        1.
!N11     N12      N13     N14       N15        N16    N17    N18    N19    N20
0.15156  0.43074  0.39471 0.20672   0.22928    0.302     0.3    0.3      0.2     0.
!errors dN1 to dN10
1       1          1       1         1          1         1.     1.      0.    0.
!N21  N22    N23     N24     N25    N26      N27   N28     N29    N30
0       0      0       0       0      0       0.     0.      0.    0.
!errors dN1 to DN10
0       0.       0      0       0      0       0      0.      0.    0.
! fit algorithm: levenberg-marquard (0), simplex (1), simulated annealing (2)
1
!levenberg-marquard options: stol niter verbose (0/1)
0.001 100 1
!simplex options: stol niter chisq_min simplex verbose (0/1)
1e-12 50 5  0 1
!simulated anneling options
* nt - integer: # of iterations between temperature reductions
* ns - integer: # of iterations between bounds adjustments
* rt - (0 < rt <1): temperature reduction factor
* maxevals - integer: limit on function evaluations
* neps - integer:  number of values final result is compared to
* functol -   (> 0): the required tolerance level for function value comparisons
* paramtol -  (> 0): the required tolerance level for parameters
* verbosity - scalar: 0, 1, or 2.
5  5  0.8  100  5  0.1  0.1 2
!flag intensities/structure factors^2
1
!   h        k          l     I(obs)      sqrt(I)
N   4.000000   0.000000   0.000000   6751.580000   95.110000
N   2.000000   0.000000   0.000000   385.880000   11.450000
```

- *mag.inp* that contains the description of the magnetic cell, parameters to fit the data, propagation vectors, magnetic domains and polarimetry data.

```
!lattice
7.2208     8.4610       5.6609       90      90      90
!# atoms
12
!AT   x         y        z        B         occ
Mn41   0.00000   0.50000   0.74300   0.370 1
Mn42   0.00000   0.50000   0.25700   0.370 1
Mn45   0.50000   0.00000   0.74300   0.370 1
Mn46   0.50000   0.00000   0.25700   0.370 1
Mn31   0.91200   0.14900   0.50000   0.370 1
Mn32   0.58700   0.64900   0.50000   0.370 1
Mn33   0.41300   0.35100   0.50000   0.370 1
Mn34   0.08780   0.85100   0.50000   0.370 1
Er01   0.86200   0.82900   0.00000   0.367 1
Er02   0.63800   0.32900   0.00000   0.367 1
Er03   0.36200   0.67100   0.00000   0.367 1
Er04   0.13800   0.17100   0.00000   0.367 1
! input switch 0: components along axis RMx,RMy,RMz,Imx,Imy,Imz,phase; 1: polar coordinates Rm,Rphi,Rtheta,Im,Iphi,Itheta,phase
1
!       F(Q)      Rm        Rphi    Rtheta      Im       Iphi      Itheta     phase (units of 2*pi)
12
Mn41  MF_Mn4     3.270        -1.8     90.000       0.44   180.000    +0.000     0.1537
                   C6          C1      90.000         C8   180.000    +0.000     0.125+C12
Mn42  MF_Mn4     3.270        -1.8     90.000       0.44   180.000    +0.000     0.0963
                   C6          C1      90.000         C9   180.000    +0.000     0.125-C12
Mn43  MF_Mn4     3.270      -178.2     90.000       0.44   180.000    +0.000     0.1537
                   C6    -C1-180      90.000        C10   180.000    +0.000     0.125+C12
Mn44  MF_Mn4     3.270      -178.2     90.000       0.44   180.000    +0.000     0.0963
                   C6    -C1-180      90.000        C11   180.000    +0.000     0.125-C12
Mn31  MF_Mn3     4.151         13      90.000     -1.05     0.000    +0.000    0.125
                   C7          C2       90.000       -C13     0.000    +0.000    0.125
Mn32  MF_Mn3     4.151        -13      90.000      1.05     0.000    +0.000    0.125
                   C7         -C2       90.000        C14     0.000    +0.000    0.125
Mn33  MF_Mn3     4.151        167      90.000     -1.05     0.000    +0.000    0.125
                   C7      180-C2       90.000       -C15     0.000    +0.000    0.125
Mn34  MF_Mn3     4.151         13      90.000      1.05     0.000    +0.000    0.125
                   C7          C2       90.000        C16     0.000    +0.000    0.125
Er01  MF_Er3      0.00        133      90.000      1.62         0    +0.000    -0.0
                    0          C3      90.000        C17         0    +0.000     0
Er02  MF_Er3      0.00        -47      90.000     -2.34       180    +0.000    -0.0
                    0    -180+C3      90.000       -C18       180    +0.000     0
Er03  MF_Er3      0.00        -21      90.000     -0.42         0    +0.000    -0.0
                    0          C4      90.000        C19         0    +0.000     0
Er04  MF_Er3      0.00        -21      90.000     -0.42         0    +0.000    -0.0
                    0          C4      90.000        C20         0    +0.000     0
!# symmetry operations in space group and symmetry elements
1
        X             Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
        X             Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
        X             Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
        X             Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
        X             Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
        X             Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
```

```
       X            Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
       X            Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
       X            Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
       X            Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
       X            Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
       X            Y           Z     RMx     RMy     RMz     IMx     IMy     IMz     0.0
!# of translations and translations
0
! propagation vectors
1
k01    0.5 0       0.25
!#spin domains, populations and parameters
2
!population and rotation matrices for domains
0.10    1 0 0    0  1 0    0  0  1  T
0.10   -1 0 0    0 -1 0    0  0 -1  T
!c1     c2     c3     c4     c5     c6     c7     c8     c9     c10
 +5.70 +14.41 +157.16 +37.41  +1.22  +2.71  +4.13  +0.61  +1.68  +1.76
!steps dC1 to DC30
 +0.50  +0.50  +0.00  +0.00  +0.00  +0.00  +0.50  +0.50  +0.50  +0.50
!c11    c12    c13    c14    c15    c16    c17    c18    c19    c20
 +0.67  +0.05  +2.41  +2.22  -0.02  -0.39  +3.21  +3.31  +0.49  +0.72
!steps dC11 to DC20
 +0.50  +0.50  +0.50  +0.50  +0.50  +0.50  +0.50  +0.50  +0.50  +0.50
!c21    c22    c23    c24    c25    c26    c27    c28     c29    c30
 +0.12  +0.12  +0.12  +0.12  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00
!steps dC1 to DC30
 +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00
!D1     D2     D3     D4     D5     D6     D7     D8      D9     D10
 +0.50  +0.50  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00
!steps dD1 to DD30
 +0.5   +0.5   +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00
! fit algorithm: levenberg-marquard (0), simplex (1), simulated annealing (2)
0
!levenberg-marquard options: stol niter verbose (0/1)
0.1 10 1
!simplex options: stol niter chisq_min simplex verbose (0/1)
0.001 3000 5  0 1
!simulated anneling options
* nt - integer: # of iterations between temperature reductions
* ns - integer: # of iterations between bounds adjustments
* rt - (0 < rt <1): temperature reduction factor
* maxevals - integer: limit on function evaluations
* neps - integer:  number of values final result is compared to
* functol -   (> 0): the required tolerance level for function value comparisons
* paramtol -  (> 0): the required tolerance level for parameters
* verbosity - scalar: 0, 1, or 2.
20  5  0.95  100000  5  0.1  0.1 2
! flag for calculation k0 + -k0 (1); flag for pure magnetic reflection (0)
0     0
! polarizer/analyzer efficiencies 0.5 <= P_1, P_2 <= 1
1 1
! hkl, prop. vector, vector in scattering plane, incident pol., scattered pol., error on scatt. pol.
+0.500000 +3.000000 +0.250000  0  -1  0     1  0  0  -0.99 +0.06 -0.10   +0.01 +0.02 +0.02
+0.500000 +3.000000 +0.250000  0  -1  0     0  1  0  -0.12 +0.38 -0.91   +0.02 +0.02 +0.01
+0.500000 +3.000000 +0.250000  0  -1  0     0  0  1  -0.19 -0.88 -0.40   +0.02 +0.02 +0.02
+0.500000 +3.000000 +0.250000  0  -1  0    -1  0  0  +0.96 -0.04 +0.05   +0.01 +0.02 +0.02
+0.500000 +3.000000 +0.250000  0  -1  0     0 -1  0  -0.20 -0.36 +0.89   +0.02 +0.02 +0.02
```

The translations are only used to calculate the direction of the magnetic moments in translated magnetic cells (option 1):

```
!# of translations and translations
2
1 0 0
0 0 1
```

At moment only one propagation vector can be defined (in rlu).

```
! propagation vectors
1
k01    0.5 0       0.25
```

The operations to create the magnetic domains are given in

```
!population and rotation matrices for domains
0.10    1 0 0    0  1 0    0  0  1  T
0.10   -1 0 0    0 -1 0    0  0 -1  T
```

Each line contains

- the domain population (used only to calculate polarisation matrices. To fit the domain populations see below);
- a 3×3 matrix that acts on the hkl-list. The operations have to be defind in a cartesian system

– a label (not used in this version)

The parameters used to fit the domain populations are given in the lines

```
!D1     D2     D3     D4     D5     D6     D7     D8      D9     D10
 +0.50  +0.50  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00
!steps dD1 to DD30
 +0.5   +0.5   +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00
```

At moment a maximum 10 domains can be fitted simultaneously.

- *crys.inp* that is used to fit magnetic intensities.

```
! lambda, extinction, scale factor
1.18  0  51
! Int (0) / fsqr (1)
1
!   h        k          l             I(obs)        sqrt(I)
M +0.500000  +0.000000  +0.250000      9.620000      0.237000
M +0.500000  -1.000000  +0.250000      64.730000      0.742000
M +0.500000  +0.000000  +0.750000      0.910000      0.300000
M +0.500000  -1.000000  +0.750000      40.100000      0.824000
M +1.500000  +0.000000  +0.250000      10.980000      0.328000
M +0.500000  +0.000000  +1.250000      5.110000      0.246000
M +1.500000  -1.000000  +0.250000      49.130000      0.522000
M +1.500000  +0.000000  +0.750000      48.930000      1.044000
M +0.500000  -2.000000  +0.250000      81.590000      0.638000
```

0.4 The magnetic formfactor

The magnetic form factor is calculated in formfac.m. It uses the computation described in the International Tables. Most probably the list in function *formfac.m* does not contain the coefficients for all magnetic ions yet and has to be completed. Consequently you should edit this file and check that the formafactor for your magnetic moment is

```
function [j_0]=formfac(Q,at_n);

%function calculates the neutron form factor
%input: Q vector and ion lablel
%output: form factor according to Jane Brown in International Tables
%broessli@psi.ch, 16.02.2007

fm_at=[];
%coefficient of the form factor
MF_FE3= [  0.3972  13.2442   0.6295   4.9034  -0.0314   0.3496   0.0044 ];
MF_ND3= [  0.0733   4.4124   0.3715   4.0196   0.5395   1.5580   0.0173 ];
MF_Er3= [  0.0389   5.3118   0.2598   8.1732   0.6784   2.0828   0.0222 ];
MF_Mn3= [  0.3760  12.5661   0.6602   5.1329  -0.0372   0.5630   0.0011 ];
MF_Mn4= [  0.4198  14.2829   0.6054   5.4689   0.9241  -0.0088  -0.9498 ];

fm_at=eval(eval("at_n"));

Q=Q/4/pi;
q2=Q*Q;
j_0=fm_at(1)*exp(-fm_at(2)*q2)+fm_at(3)*exp(-fm_at(4)*q2)+fm_at(5)*exp(-fm_at(6)*q2)+fm_at(7);

endfunction
```

0.5 Magfac.m

The magnetic structure factor $F_M(\vec{Q})$ is the Fourier transform of the magnetisation density which is written as

$$\vec{M}(\vec{r}+\vec{l}) = \hat{\vec{p}}M_p(\vec{r})\cos(\vec{\tau}\cdot\vec{l}+\phi_r) + \hat{\vec{q}}M_q(\vec{r})\sin(\vec{\tau}\cdot\vec{l}+\phi_r) \quad (1)$$

$$= \Phi(\vec{r})\exp^{i(\vec{l}\cdot\vec{\tau}+\phi_r)} + c.c., \quad (2)$$

with $\Phi(\vec{r}) = \frac{1}{2}[\hat{\vec{p}}M_p(\vec{r}) - i\hat{\vec{q}}M_p(\vec{r})]\exp(i\phi_r)$. In Eq. 2, $\vec{M}$ is the magnetic vector at site $\vec{r}$ in the cell $\vec{l}$; $\vec{\tau}$ is the propagation vector and ϕ_r the phase. The computed

Fourier transform of eq. 2 is

$$\begin{aligned}
\vec{M}(-\vec{Q}) &= \sum_{\vec{r},\vec{l}} \vec{M}(\vec{r}+\vec{l}) \exp(-i\vec{Q}\cdot(\vec{r}+\vec{l})) = \\
&= \sum_{\vec{r},\vec{l}} \phi(\vec{r}) \exp(-i\vec{Q}\cdot(\vec{r}+\vec{l})) \exp(i\vec{l}\cdot\vec{\tau}) + \sum_{\vec{r},\vec{l}} \phi^*(\vec{r}) \exp(-i\vec{Q}\cdot(\vec{r}+\vec{l})) \exp(-i\vec{l}\cdot\vec{\tau}) \\
&= \sum_{\vec{r},\vec{l}} \phi(\vec{r}) \exp(-i\vec{Q}\cdot\vec{r}) \exp(i\vec{l}\cdot(\vec{\tau}-\vec{Q})) + \sum_{\vec{r},\vec{l}} \phi^*(\vec{r}) \exp(-i\vec{Q}\cdot\vec{r}) \exp(-i\vec{l}\cdot(\vec{\tau}+\vec{Q})) \\
&= \sum_{\vec{g}} \phi(\vec{Q})\delta(\vec{\tau}-\vec{Q}-\vec{g}) + \sum_{\vec{g}} \phi^*(\vec{Q})\delta(-\vec{\tau}-\vec{Q}-\vec{g}), \qquad (3)
\end{aligned}$$

with

$$\Phi(\vec{Q}) = \sum_{\vec{r}} \Phi(\vec{r}) \exp(-i\vec{Q}\cdot\vec{r}) \qquad (4)$$

$$\Phi^*(\vec{Q}) = \sum_{\vec{r}} \Phi^*(\vec{r}) \exp(-i\vec{Q}\cdot\vec{r}). \qquad (5)$$

- For the case where $\vec{\tau}+\vec{g} = -\vec{\tau}+\vec{g}' = \vec{Q}$ the magnetic structure factor is equal to

$$\vec{M}(-\vec{Q}) = [\hat{\vec{p}} M_p(\vec{r}) \cos(\phi_r) + \hat{\vec{q}} M_q(\vec{r}) \sin(\phi_r)] \exp(-i\vec{Q}\cdot\vec{r}). \qquad (6)$$

- Taking the Fourier transform for $+\vec{Q}$

$$\vec{M}(\vec{Q}) = \sum_{\vec{r},\vec{l}} \vec{M}(\vec{r}+\vec{l}) \exp(+i\vec{Q}\cdot(\vec{r}+\vec{l}))$$

  gives

$$\sum_{\vec{g}} \phi(\vec{Q})\delta(\vec{\tau}+\vec{Q}-\vec{g}) + \sum_{\vec{g}} \phi^*(\vec{Q})\delta(-\vec{\tau}+\vec{Q}-\vec{g}), \qquad (7)$$

  with $\phi^{(*)}(\vec{Q}) = \sum_{\vec{r}} \phi^{(*)}(\vec{r}) \exp(+i\vec{Q}\cdot\vec{r})$.

Example: $\vec{Q} = [1.5, 0, 0.75]$ with the propagation vector $\vec{\tau} = [0.5, 0, 0.25]$ yields $\vec{g} = \vec{\tau} + \vec{Q} = [2, 0, 2] \rightarrow F(\vec{Q}) = \sum_{\vec{r}} \phi(\vec{r}) \exp(+i\vec{Q}\cdot\vec{r})$ or $\vec{g} = -\vec{\tau} - \vec{Q} = [-2, 0, -2] \rightarrow F(-\vec{Q}) = \sum_{\vec{r}} \phi^*(\vec{r}) \exp(-i\vec{Q}\cdot\vec{r}) = F^*(\vec{Q})$. Hence, the chirality term in the neutron cross-section changes sign for $\vec{Q} \rightarrow -\vec{Q}$ or equivalently if the inversion operation $\bar{1}$ is applied to the [H,K,L] list (chiral domain).
Finally the magnetic structure factor is calculated from

$$F_M(\vec{Q}) = 0.2695 \cdot \hat{\vec{Q}} \cdot (\vec{M}(\vec{Q}) \times \hat{\vec{Q}}). \qquad (8)$$

```
## PURPOSE.  See the GNU General Public License for more
## details.
##
## You should have received a copy of the GNU General Public
## License along with Octave; see the file COPYING.  If not,
## write to the Free Software Foundation, Inc., 51 Franklin Street,
## Fifth Floor, Boston, MA 02110-1301, USA.
##
## usage: mufit
##

function [mag_inter_vec,fsqr,mag_vec_cart]=magfac2(spins_cell,fstar,Q,flag_k0,ffq);

%calculate the magnetic structure factor M(Q)=(Mp(q)-I*Mq(q))*exp(-I*Q*r) and
%the magnetic interaction vector kx(M(Q)xk) in cartesian coordinates in crystal system
%input:
% 1.- list with spins coordinates Mp(r)*exp(i*phi) and Mq(r)*exp(i*phi)
% 2.- the scattering vector Q in reciprocal Angstroem
% 3.- the propagation vector
%output:
% 1.- the magnetic interaction vector mag_inter_vec
% 2.- the square of the magnetic structure factor
% 3.- M(Q)
%16.02.07
%added switch for domain types: 15.06.07
%broessli@psi.ch
%check if HKL+tau_0 is reciprocal vector
%vectorized 07.03.2009

%---- compute the fourier transform
switch flag_k0
case 0 %---- usual case
nr_spins=spins_cell{1};
A=[2:nr_spins+1];
phase=fstar'*[[spins_cell{1,A}].phase];
p=[[spins_cell{1,A}].spins_p];
spinsp=(reshape(p,3,columns(p)/3))';
q=[[spins_cell{1,A}].spins_q];
spinsq=(reshape(q,3,columns(p)/3))';
atpos=[[spins_cell{1,A}].ionpar];
ionpar=(reshape(atpos,5,columns(atpos)/5))';
B=ionpar(:,4);
occ=ionpar(:,5);

%---- fourier transform; occupation number; form factor; Debye-Waller factor
QQ=dot(Q',Q').^0.5;
QQ=QQ';

%---- occupation number: size nr_atoms x hkl
occt=repmat(occ,1,rows(Q));
%---- Fourier transform: size hkl x nr_atoms
ftr=[exp(-I.*ionpar(:,1:3)*Q')'.*exp(-1/4.*QQ*B').*occt'.*ffq];

%---- Real and Imaginary parts: size hkl x nr_atoms
phase_partA=ftr.*exp(-I.*2.*pi*phase);
ffstar=repmat(fstar',1,nr_spins);
phase_partB=phase_partA.*ffstar;

%---- x component
P_part=phase_partA*spinsp(:,1);
Q_part=phase_partB*spinsq(:,1);
%---- the sum is done automatically
Smagstrufac_x=P_part.+I*Q_part;

%---- y component
P_part=phase_partA*spinsp(:,2);
Q_part=phase_partB*spinsq(:,2);
Smagstrufac_y=P_part.+I*Q_part;

%---- z component
P_part=phase_partA*spinsp(:,3);
Q_part=phase_partB*spinsq(:,3);
Smagstrufac_z=P_part.+I*Q_part;


case 1 %---- if k0 and -k0 produce Bragg peaks at same position

nr_spins=spins_cell{1};
A=[2:nr_spins+1];
phase=fstar'*[[spins_cell{1,A}].phase];
p=[[spins_cell{1,A}].spins_p];
spinsp=(reshape(p,3,columns(p)/3))';
q=[[spins_cell{1,A}].spins_q];
spinsq=(reshape(q,3,columns(p)/3))';
atpos=[[spins_cell{1,A}].ionpar];
ionpar=(reshape(atpos,5,columns(atpos)/5))';
B=ionpar(:,4);
occ=ionpar(:,5);

%---- fourier transform; occupation number; form factor; Debye-Waller factor
QQ=dot(Q',Q').^0.5;
QQ=QQ';
%---- occupation number: size nr_atoms x hkl
occt=repmat(occ,1,rows(Q));

%---- Fourier transform: size hkl x nr_atoms
ftr=[exp(-I.*ionpar(:,1:3)*Q')'.*exp(-1/4.*QQ*B').*occt'.*ffq];

%---- the sum is done automatically
```

```
%---- x component
Smagstrufac_x=2.*sum((cos(2.*pi.*phase).*repmat(spinsp(:,1),1,rows(QQ))').*ftr.+(sin(2.*pi.*phase).*repmat(spinsq(:,1),1,rows(QQ))').*ftr,2);

Smagstrufac_y=2.*sum((cos(2.*pi.*phase).*repmat(spinsp(:,2),1,rows(QQ))').*ftr.+(sin(2.*pi.*phase).*repmat(spinsq(:,2),1,rows(QQ))').*ftr,2);

%---- z component
Smagstrufac_z=2.*sum((cos(2.*pi.*phase).*repmat(spinsp(:,3),1,rows(QQ))').*ftr.+(sin(2.*pi.*phase).*repmat(spinsq(:,3),1,rows(QQ))').*ftr,2);

endswitch

%-------------------------------------------------------------------------------
%---- calculate magnetic interaction vector k x (M x k)
mag_vec_cart=0.2695.*[Smagstrufac_x Smagstrufac_y Smagstrufac_z];
Qnorm=dot(Q,Q,2).^0.5;
Qvec_norm=Q./repmat(Qnorm,1,columns(Q));

mag_inter_vec=cross(Qvec_norm,cross(mag_vec_cart,Qvec_norm,2),2);

%---- calculate the square of the magnetic structure factor
fsqr=mag_inter_vec(:,1).*conj(mag_inter_vec(:,1)).+...
mag_inter_vec(:,2).*conj(mag_inter_vec(:,2)).+...
mag_inter_vec(:,3).*conj(mag_inter_vec(:,3));


endfunction
```

0.6 Moment.m

The magnetic moments are described according to Eq. 2 and hence the input of the two vectors $\hat{\vec{p}}M_p$ and $\hat{\vec{q}}M_q$ is required. Note that $\vec{p}$ and $\vec{q}$ are orthogonal to each other and defined in a *cartesian* coordinate system. Polar coordinates can be used and are defined as

$$M_x = M \cdot \sin(\theta) \cdot \cos(\phi); M_y = M \cdot \sin(\theta) \cdot \sin(\phi); M_z = M \cdot \cos(\theta);$$

with ϕ in the (a,b)-plane and θ is the angle from the c-axis. Alternatively the magnetic moments can be given directly in the crystal space. The phase of the magnetic moment is given in units of 2π. Symmetry elements can be applied in the form $RMx\ RMy\ RMz$ for the real part and $IMx\ IMy\ IMz$ for the imaginary part of the magnetic moment and a phase added. In case the atomic part of the symetry operation contains a translation the magnetic moment is accordingly multiplied by $\exp(i\vec{\tau} \cdot (\vec{T}|\vec{r}))$. The components of the spins, i.e. $\vec{M}(\vec{r}) = (M_x, M_y, M_z)(\vec{r})$, are calculated according to Eq. 2 for the atoms in the primitive cell. If translations are given the spin components in the translated cells are also calculated and printed out.

```
## Copyright (C) 2007,2008,2009 Bertrand Roessli <bertrand.roessli@psi.ch>
##
##
## MuFit is free software; you can redistribute it and/or
## modify it under the terms of the GNU General Public
## License as published by the Free Software Foundation;
## either version 2, or (at your option) any later version.
##
## Mufit is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied
## warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
## PURPOSE.  See the GNU General Public License for more
## details.
##
## You should have received a copy of the GNU General Public
## License along with Octave; see the file COPYING.  If not,
## write to the Free Software Foundation, Inc., 51 Franklin Street,
## Fifth Floor, Boston, MA 02110-1301, USA.
##

function [M_cell,S_cart_sys_cell]=moment(lat_cell,ion_cell,moment_cell,propvec_cell_rlu,propvec_cell_AA,spgsym_cell,spgtrans_cell);

% input:4 lists containing:
% 1.- the atomic positions
% 2.- the space group symmetries
% 3.- the translations
% 4.- the parameters (Mx,My,Mz IMx,IMy,IMz phase) or polar coordinates
% output: 2 lists containg
% 1.- the spins coordinates in the unit cell used to calculate the magnetic structure factor
% 2.- the magnetic moment directions/size in different cells according to the translation list
```

```
as=lat_cell.ll(1);
bs=lat_cell.ll(2);
cs=lat_cell.ll(3);
aa=lat_cell.angl(1);
bb=lat_cell.angl(2);
cc=lat_cell.angl(3);
[Mcryst2cart,Mrec2cart]=lat2cart(as,bs,cs,aa,bb,cc);

%---- 1) transform from polar coordinates to xyz coordinates if necessary
%---- note that polar coordinates should be defined in a cartesian coordinate system in the input file
moment_cart_cell=cell(moment_cell{1}+1,1);
moment_cart_cell{1}=moment_cell{1};

if (moment_cell{columns(moment_cell)}.type  == 1)
invMcryst2cart=Mcryst2cart^-1;
for i=1:moment_cell{1}
mom.fq=moment_cell{i+1}.fq;
RM=moment_cell{i+1}.rm;
IM=moment_cell{i+1}.im;
rtheta=moment_cell{i+1}.rtheta/180*pi;
rphi=moment_cell{i+1}.rphi/180*pi;
itheta=moment_cell{i+1}.itheta/180*pi;
iphi=moment_cell{i+1}.iphi/180*pi;
rmz=RM.*cos(rtheta);
         rmx=RM.*sin(rtheta)*cos(rphi);
         rmy=RM.*sin(rtheta)*sin(rphi);
         imz=IM.*cos(itheta);
         imx=IM.*sin(itheta).*cos(iphi);
         imy=IM.*sin(itheta).*sin(iphi);

%
%transform to crystal system to apply later symmetry operations
%
         rm_cart=invMcryst2cart*[rmx; rmy; rmz];
         moment_cell{i+1}.mx=rm_cart(1);moment_cell{i+1}.my=rm_cart(2);moment_cell{i+1}.mz=rm_cart(3);
         im_cart=invMcryst2cart*[imx; imy; imz];
         moment_cell{i+1}.imx=im_cart(1);moment_cell{i+1}.imy=im_cart(2);moment_cell{i+1}.imz=im_cart(3);
endfor
endif

mag_vec_cart=[0;0;0];
k0=[propvec_cell_AA{2}.kx propvec_cell_AA{2}.ky propvec_cell_AA{2}.kz];

%---- 1) calculate magnetic moments in the primitive cell to
%---- be used for the structure factors
atoms_pos=zeros(1:3,1);
l=1;
%---- create list of ion/magnetic moment according to symmetry elements
spins_cell=cell(spgsym_cell{1}*ion_cell{1}+1,1);
spins_cell{1}=ion_cell{1}*spgsym_cell{1};

while (l<=spgsym_cell{1}*ion_cell{1})
for k=1:ion_cell{1}
         X=ion_cell{k+1}.x;
         Y=ion_cell{k+1}.y;
         Z=ion_cell{k+1}.z;
         for j=1:spgsym_cell{1}
%---- apply symmetry operations to atomic positions and transform to cartesian coordinates
atoms_pos(1:3)=Mcryst2cart*([eval(spgsym_cell{l+1}.x).*as eval(spgsym_cell{l+1}.y).*bs eval(spgsym_cell{l+1}.z).*cs])';

%---- apply symmetry operations to magnetic moments
RMx=moment_cell{k+1}.mx;
RMy=moment_cell{k+1}.my;
RMz=moment_cell{k+1}.mz;
IMx=moment_cell{k+1}.imx;
IMy=moment_cell{k+1}.imy;
IMz=moment_cell{k+1}.imz;

%%propagate the magnetic moment with the propagation vector and translation part of symmetry operations
a1=spgsym_cell{l+1}.x;a2=spgsym_cell{l+1}.y;a3=spgsym_cell{l+1}.z;
b1=a1(max([rindex(a1,"X"),rindex(a1,"Y"),rindex(a1,"Z")])+1:length(a1));
b2=a2(max([rindex(a2,"X"),rindex(a2,"Y"),rindex(a2,"Z")])+1:length(a2));
b3=a3(max([rindex(a3,"X"),rindex(a3,"Y"),rindex(a3,"Z")])+1:length(a3));
if (length(b1)==0)
b1='0';
endif
if (length(b2)==0)
b2='0';
endif
if (length(b3)==0)
b3='0';
endif
trans=exp(I*k0*Mcryst2cart*[eval(b1)*as;eval(b2)*bs;eval(b3)*cs]);
atoms(l,1:11)=[atoms_pos ion_cell{k+1}.B ion_cell{k+1}.occ...
eval(spgsym_cell{l+1}.rmx).*trans eval(spgsym_cell{l+1}.rmy).*trans eval(spgsym_cell{l+1}.rmz).*trans...
eval(spgsym_cell{l+1}.imx).*trans eval(spgsym_cell{l+1}.imy).*trans eval(spgsym_cell{l+1}.imz).*trans];
MM.ion=ion_cell{k+1}.ion;
                 MM.fq=moment_cell{k+1}.fq;
                 MM.ionpar=atoms(l,1:5);
                 MM.spins_p=real(atoms(l,6:8))./2;
                 MM.spins_q=real(atoms(l,9:11))./2;
%the phase is the sum of two components, nth(spgsym,l+1).phase is given by symmetry
                 MM.phase=moment_cell{k+1}.phase.+spgsym_cell{l+1}.phase;
                 spins_cell{l+1}=MM;
                 l++;
endfor
endfor
endwhile
```

```
%---- 2) calculate the magnetic moments not only in the primitive cell but
%---- also according to the translations given in the input file
M_cell=cell(2.*rows(atoms)+spgtrans_cell{1}+1,1);
M_cell{1}=(spgtrans_cell{1}+1)*rows(atoms);
for l=1:rows(atoms)
        M.label=spins_cell{l+1}.ion;
        M.fq=spins_cell{l+1}.fq;
        M.mat(1:3)=atoms(l,1:3);
        M.T=[0;0;0];

%---- it is cpu-cheaper to put content of cell first in a variable
sphase=2.*pi.*spins_cell{l+1}.phase;
        M.mpx=atoms(l,6).*cos(sphase);
        M.mpy=atoms(l,7).*cos(sphase);
        M.mpz=atoms(l,8).*cos(sphase);
        M.mqx=atoms(l,9).*sin(sphase);
        M.mqy=atoms(l,10).*sin(sphase);
        M.mqz=atoms(l,11).*sin(sphase);
        M_cell{l+1}=M;
endfor

kk=rows(atoms)+2;
for i=1:spgtrans_cell{1}
Tx=spgtrans_cell{i+1}.x;Ty= spgtrans_cell{i+1}.y;Tz=spgtrans_cell{i+1}.z;
T=(Mcryst2cart*[Tx;Ty;Tz])';
        for l=1:rows(atoms)
                M.label=spins_cell{l+1}.ion;
                M.fq=spins_cell{l+1}.fq;
                M.mat(1:3)=atoms(l,1:3).+T;
                M.T=T;
sphase=T*k0'.+2.*pi.*spins_cell{l+1}.phase;
                M.mpx=atoms(l,6).*cos(sphase);
                M.mpy=atoms(l,7).*cos(sphase);
                M.mpz=atoms(l,8).*cos(sphase);
                M.mqx=atoms(l,9).*sin(sphase);
                M.mqy=atoms(l,10).*sin(sphase);
                M.mqz=atoms(l,11).*sin(sphase);
                M_cell{kk}=M;
kk=kk+1;
        endfor
endfor

%---- 3)transform spins to cartesian coordinates for calculation of structure factor
S_cart_cell=cell(1,spins_cell{1}+1);
S_cart_sys_cell{1}=spins_cell{1};
atoms=zeros(spgsym_cell{1}*ion_cell{1},11);
for k=1:spins_cell{1}
p.fq=spins_cell{k+1}.fq;
p.ion=spins_cell{k+1}.ion;
p.ionpar=spins_cell{k+1}.ionpar;

p1=spins_cell{k+1}.spins_p(1);p2=spins_cell{k+1}.spins_p(2);p3=spins_cell{k+1}.spins_p(3);
q1=spins_cell{k+1}.spins_q(1);q2=spins_cell{k+1}.spins_q(2);q3=spins_cell{k+1}.spins_q(3);
if (moment_cell{columns(moment_cell)}.type  == 1)
%magnetic moments in polar coordinates are already defined in a cartesian system
p.spins_p=[p1;p2;p3]';
p.spins_q=[q1;q2;q3]';
else
%otherwise transform to cartesian system
p.spins_p=(Mcryst2cart*[p1;p2;p3])';
p.spins_q=(Mcryst2cart*[q1;q2;q3])';
endif
p.phase=spins_cell{k+1}.phase;
  S_cart_sys_cell{k+1}=p;
endfor

endfunction
```

0.7 Ncs.m

The function *ncs.m* calculates the polarisation matrices according to the Blume's equations. The polarisation is defined in the following coordinate system $\hat{\vec{x}}$ is parallel to $\vec{Q} = \vec{k}_i - \vec{k}_f$, $\hat{\vec{z}}$ is perpendicular to the scattering plane ('up') and $\hat{\vec{y}}$ is in the scattering plane and defines a right-handed coordinate system. Although very useful to calculate to polarisation matrices, the Blume's equations have the following restrictions:

- to be used the polarisation data have to be corrected for the finite polarisation of the beam and efficiency of the benders;
- the scattered intensities cannot be calculated for all incident and scattered polarisations.

- the Blume's equations cannot be used when domains are present.

```
## Copyright (C) 2007,2008,2009 Bertrand Roessli <bertrand.roessli@psi.ch>
##
##
## MuFit is free software; you can redistribute it and/or
## modify it under the terms of the GNU General Public
## License as published by the Free Software Foundation;
## either version 2, or (at your option) any later version.
##
## Mufit is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied
## warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
## PURPOSE.  See the GNU General Public License for more
## details.
##
## You should have received a copy of the GNU General Public
## License along with Octave; see the file COPYING.  If not,
## write to the Free Software Foundation, Inc., 51 Franklin Street,
## Fifth Floor, Boston, MA 02110-1301, USA.
##
## usage: mufit
##

function [P_f,cs_n]=ncs(H,K,L,A1,P0_x,P0_y,P0_z,mag_inter_vec,flag_nmi,det_r);
%---- calculates neutron cross-section and polarisation with the Blume's equations

%input:
% Q in cartesian coordinates (A^-1) [H;K;L]
% A1 second vector in scattering plane (A^-1, cartesian coordinates)
% incident polarisation [P0_x;P0_y;P0_z]
% mag_inter_vec: magnetic interaction vector
% flag_nmi:flag for mixed magnetic/nuclear reflections
%output: final polarisation and total cross-section

P_f=zeros(1,3);

%---- polarisation of incident neutron beam
P_0=[P0_x;P0_y;P0_z]; %in MuPAD frame

V1=[H;K;L];
%---- component of vector in scattering plane in cartesian coordinates (1/A)
V2=A1;

if (dot(V1/norm(V1),V2/norm(V2)) == 1 )
error("\nredefine scattering plane\n");
return;
endif

%----- Form unit vectors V1, V2, V3 in scattering plane
V3=cross(V1,V2);
V2=cross(V3,V1);
V3=V3/sqrt(sum(V3.*V3));
V2=V2/sqrt(sum(V2.*V2));
V1=V1/sqrt(sum(V1.*V1));
V3=V3.*det_r;

%---- coordinates of HKL in V1, V2, V3 frame
%---- transformation matrix
U=[V1';V2';V3'];

%---- rotate magnetic interaction vector into MuPAD system
magstruc=U*mag_inter_vec;

%---- calculate cross-section terms
%---- 1) sigma nuclear
if (flag_nmi == 0)
nucstruc=0;
else
[nucstruc]=feval("nucfac",H,K,L);
endif
sigma_n=nucstruc.*conj(nucstruc);

%---- 2)sigma magnetic
sigma_m=magstruc'*magstruc;

%---- 3)nuclear magnetic interference
sigma_imn=P_0'*(nucstruc.*conj(magstruc).+conj(nucstruc).*magstruc);

%---- 4)chiral term
sigma_ch=P_0'*I*cross(conj(magstruc),magstruc);

%---- cross-section
cs_n=sigma_n.+sigma_m.+sigma_imn.+sigma_ch;
if (cs_n == 0)
cs_n = 10^-8;
endif

%---- calculates final polarisation vector according to Blume
%---- 1) nuclear
P_i_n=P_0.*sigma_n;

%---- 2)magnetic
P_i_m=-P_0.*sigma_m+conj(magstruc).*(P_0'*magstruc).+(magstruc'*P_0).*magstruc;

%---- 3)interference term
nmi=conj(nucstruc).*magstruc.-conj(magstruc).*nucstruc;
P_i_imn=nucstruc.*conj(magstruc)+conj(nucstruc).*magstruc+I.*cross(nmi,P_0);
```

```
%---- 4)chiral
P_i_ch=-I.*cross(conj(magstruc),magstruc);

%---- complete neutron cross-section
P_i=P_i_n.+P_i_m.+P_i_imn.+P_i_ch;
P_i(find(abs(P_i)<0.000001))=0;

%---- polarisation vector after scattering
P_f=real(P_i./cs_n);

endfunction;
```

0.8 Nucfac.m

The function *nucfac.m* calculates the nuclear structure factor. Symmetry elements and translations can be given.

```
f## Copyright (C) 2007,2008,2009 Bertrand Roessli <bertrand.roessli@psi.ch>
##
##
## MuFit is free software; you can redistribute it and/or
## modify it under the terms of the GNU General Public
## License as published by the Free Software Foundation;
## either version 2, or (at your option) any later version.
##
## Mufit is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied
## warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
## PURPOSE.  See the GNU General Public License for more
## details.
##
## You should have received a copy of the GNU General Public
## License along with Octave; see the file COPYING.  If not,
## write to the Free Software Foundation, Inc., 51 Franklin Street,
## Fifth Floor, Boston, MA 02110-1301, USA.
##

function [strufac2]=nucfac(hkl_crys_rlu,hkl_crys_AA)

% NUCFAC(H,K,L) calculates nuclear structure factor
% cell, atomic parameters and symmetry elements are read in readnuc
% calls readnuc
% input: H,K,L in A^-1
% called by ncs
% output: nuclear structure factor
% written by broessli@psi.ch (2006)
% last modified  28.05.2007
% added simple extinction correction 28.01.08


nuc_stru_fac=0;

%---- get cell parameters etc... from readnuc
[genpar,as,bs,cs,aa,bb,cc,ion_cell, spgsym_cell,spgtrans_cell,singlecrys,parameters,dparameters,simplexpar]=readnuc;

%---- transformation matrices to cartesian system
[Mcryst2cart,Mrec2cart]=lat2cart(as,bs,cs,aa,bb,cc);


atoms=zeros(ion_cell{1}*spgsym_cell{1}*spgtrans_cell{1},6);
atoms_pos=zeros(1,3);

l=1;
%---- loop over translations/symmetry elements of the space group to create ions in the unit cell
for k=1:ion_cell{1}
X=ion_cell{k+2}.x;
Y=ion_cell{k+2}.y;
Z=ion_cell{k+2}.z;
for i=1:spgtrans_cell{1}
for j=1:spgsym_cell{1}
%create atomic positions in AA
atoms_pos(1:3)=Mcryst2cart*([eval(spgsym_cell{j+1}.x).*as eval(spgsym_cell{j+1}.y).*bs eval(spgsym_cell{j+1}.z).*cs].+...
                    [spgtrans_cell{i+1}.x.*as spgtrans_cell{i+1}.y.*bs spgtrans_cell{i+1}.z.*cs])';
%create array with scattering length, x,y,z, B-factor and occupation number
atoms(l,1:6)=[ion_cell{k+2}.bd atoms_pos ion_cell{k+2}.B ion_cell{k+2}.occ];
l=l+1;
endfor
endfor
endfor

%structure factor
sl=[atoms(:,1)];    %scattering length
ap=[atoms(:,2:4)];  %atomic positions
B_iso=[atoms(:,5)]; %B-factor
occ=[atoms(:,6)];   %occupation number

%vectorised version of the structure factor (27.03.09)
Q2=dot(hkl_crys_AA,hkl_crys_AA,2);
ft=exp(I.*ap*hkl_crys_AA'); %.*exp(-1/4.*Q2.^0.5*abs(B_iso)')';
strufac=(sl.*occ)'*ft;
```

```
%the sum over the atoms in the unit cell is done automatically

%add extinction correction
if (genpar.zach ~= 0)
        x=real(strufac.^2);
        zachpar=genpar.zach;
        lambda=genpar.lambda;
        fhandle=@(x) extinc(x,lambda,zachpar,hkl_crys_AA);
        [ys]=fhandle(x);
else
        ys=ones(1,rows(hkl_crys_rlu));
endif

%Lorentz factor
if (singlecrys{1}.intfsqr == 0)
        Q=hkl_crys_AA;
        d_sp=2*pi./dot(Q,Q,2).^0.5;
        th_angle=asin(genpar.lambda./2./d_sp);
        lor=1./sin(2.*th_angle);
        strufac2=strufac.*conj(strufac).*ys.*lor';
else
        strufac2=strufac.*conj(strufac).*ys;
endif
strufac2=genpar.scale.*real(strufac2);

endfunction
```

0.9 Onedomain.m

The function calculates the neutron intensities and polarisations for given incident polarisation direction $\vec{P}_i$ and scattered neutron polarisation $\vec{P}_f$ in the MuPAD system. The scattered polarisation is obtained by calculating

$$P_{i,j} = \frac{I^{i,j} - I^{i,-j}}{I^{i,j} + I^{i,-j}}, i, j = x, y, z. \tag{9}$$

The neutron intensities $I^{i,j}$ are calculated in the function *sigma-tot.m*.

```
function [Ip_ij,Im_ij,polar]=onedomain(P_1,P_2,hhkl,Q_rlu,AA1,flag_k0,flag_nmi,spins_cell,dtype,pi_direction,propvec_cell_rlu,det_r)

%calculate scattered neutron polarisation for one magnetic domain
%uses sigma_tot.m
%input: bender efficiencies and directions, list of parameters, HKL, second vector to define scattering plane, number of atoms, flags for
%calculation of +k0 and/or -k0 and pure magnetic or mixed reflection
%output: intensites and polarisation vector
%broessli@psi.ch, 16.02.2007

s_px=1/sqrt(2)*[1; 1];s_mx=1/sqrt(2)*[1; -1];
s_py=1/sqrt(2)*[1; I];s_my=1/sqrt(2)*[1; -I];
s_pz=[1; 0];s_mz=[0; -1];
vec_p={s_px,s_py,s_pz};
vec_m={s_mx,s_my,s_mz};

%first calculate neutron cross-sections for incident polarisation Pi
%i.e. get back from pol_sd 2x2 matrix
[poln_cs]=pol_sd(hhkl,AA1,Q_rlu,flag_k0,flag_nmi,spins_cell,propvec_cell_rlu,dtype,det_r);
%define wave-vectors and calculates cross section
if (length(pi_direction) == 5)
%if (pi_direction == 'pi_up')

%1) sigma x -> x, sigma x -> -x and pol_xx, and teh same for y,z
for i=1:3
for j=1:3
[I_pp]=sigma_tot(vec_p{i},vec_p{j},vec_m{i},vec_m{j},poln_cs,P_1,P_2);
[I_pm]=sigma_tot(vec_p{i},vec_m{j},vec_m{i},vec_p{j},poln_cs,P_1,P_2);
Ip_ij(i,j)=I_pp;
Im_ij(i,j)=I_pm;
if ((I_pp + I_pm) == 0);
polar(i,j)=0;
else
polar(i,j)=(I_pp-I_pm)/(I_pp+I_pm);
endif
endfor
endfor
else
%incident polarisation along -x,-y,-z
for i=1:3
for j=1:3
[I_mp]=sigma_tot(vec_m{i},vec_p{j},vec_p{i},vec_m{j},poln_cs,P_1,P_2);
[I_mm]=sigma_tot(vec_m{i},vec_m{j},vec_p{i},vec_p{j},poln_cs,P_1,P_2);
Ip_ij(i,j)=I_mp;
Im_ij(i,j)=I_mm;
if ((I_mp + I_mm) == 0)
polar(i,j)=0;
else
polar(i,j)=(I_mp-I_mm)/(I_mp+I_mm);
endif
```

```
endfor
endfor
endif
%calculate different cross-section, taking into acount bender efficiencies
endfunction
```

0.10 Pol-sd.m

The function *Pol-sd* calculates the neutron interaction potential $V(\vec{Q}) = N(\vec{Q}) + \vec{M}_{\perp}(\vec{Q}) \cdot \vec{S}$ where S_x, S_y and S_z are the Pauli matrices; $N(\vec{Q})$ the nuclear potential and $\vec{M}_{\perp}(\vec{Q})$ the magnetic interaction vector.

```
function [poln_cs]=pol_sd(hhkl,AA1,Q_rlu,flag_k0,flag_nmi,spins_cell,propvec_cell_rlu,dtype,det_r)

% calculates neutron intensities as a 2x2 matrix
% using Pauli matrices;
% for one magnetic domain;
% calls magfac
% input: list of parameters
%  hhkl: [H;K;L];
%  AA1: second vector to define scattering plane
% flag_k0: flag for calculations of +k0 and -k0
% flag_nmi: flag for pure magnetic/mixed nuclear-magnetic reflections
% spins_list: list containing spin positions and directions
% output: neutron cross-section


%----- Q vector in cartesian coordinates (1/A)
hkl=hhkl';
H=hkl(1);
K=hkl(2);
L=hkl(3);
Q=[H;K;L]';

%calculate magnetic interaction vector
[mag_inter_vec,fsqr,mag_vec_cart]=magfac(spins_cell,Q,Q_rlu,propvec_cell_rlu,flag_k0,dtype,det_r);

P_f=zeros(1,3);
%Pauli Matrices
sigx=[0 1; 1 0];
sigy=[0 -I; I 0];
sigz=[1 0; 0 -1];

%define coordinate system in polarimetry coordinates
%component of (HKL) in cartesian coordinates (1/A)
%x along Q
%H,K,L already in cartesian coordinates
%V1=Q2c*[H;K;L];
%V2=Q2c*A1;
V1=[H;K;L];
%second vector in scattering plane
%first vector is given by Q
A1=AA1';
%component of vector in scattering plane in cartesian coordinates (1/A)
V2=A1;

if (dot(V1/norm(V1),V2/norm(V2)) == 1 )
error("\nredefine scattering plane\n");
return;
endif
%----- Form unit vectors V1, V2, V3 in scattering plane
V3=cross(V1,V2);
V2=cross(V3,V1);
V3=V3/sqrt(sum(V3.*V3));
V2=V2/sqrt(sum(V2.*V2));
V1=V1/sqrt(sum(V1.*V1));
V3=V3.*det_r;
%coordinates of HKL in V1, V2, V3 frame
%transformation matrix
U=[V1';V2';V3'];

%rotate magnetic interaction vector into MuPAD system
magstruc=U*mag_inter_vec;

%fprintf(stdout,"magnetic interaction vector is: %f +i%f, %f +i%f, %f +i%f\n", real(magstruc(1)),imag(magstruc(1)),...
%real(magstruc(2)),imag(magstruc(2)),real(magstruc(3)),imag(magstruc(3)));
%calculate cross-section terms
%1) sigma nuclear

if (flag_nmi == 0)
nucstruc=0;
else
[nucstruc]=feval("nucfac",H,K,L);
endif

poln_cs=nucstruc.+(magstruc(1).*sigx.+magstruc(2).*sigy.+magstruc(3).*sigz);


endfunction
```

0.11 Sigma-tot.m

The neutron intensity $I^{i,j}$ is calculated from the interaction potential $V(\vec{Q}) = N(\vec{Q}) + \vec{M}_{\perp}(\vec{Q}) \cdot \vec{S}$ where i is the direction of the incident polarisation and j the scattered polarisation and corrected for the final efficiency of the benders P_1 and P_2:

$$I^{i,j}(\vec{Q}) = | < j|V(\vec{Q})|i > |^2, i, j = x, y, z \tag{10}$$

with $|x >= 1/\sqrt{2}[1, 1]$,$|y >= 1/\sqrt{2}[1, I]$,$|z >= [1, 0]$. It is useful to recal that the polatisation of the neutron beam is defined as $P_0 = 2f - 1$ where f is the proportion of neutrons in the 'up'-channel, so that $-1 \leq P_0 \leq 1$ for $0 \leq f \leq 1$. Mufit uses f and not P_0.

```
## Copyright (C) 2007,2008,2009 Bertrand Roessli <bertrand.roessli@psi.ch>
##
##
## MuFit is free software; you can redistribute it and/or
## modify it under the terms of the GNU General Public
## License as published by the Free Software Foundation;
## either version 2, or (at your option) any later version.
##
## Mufit is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied
## warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
## PURPOSE.  See the GNU General Public License for more
## details.
##
## You should have received a copy of the GNU General Public
## License along with Octave; see the file COPYING.  If not,
## write to the Free Software Foundation, Inc., 51 Franklin Street,
## Fifth Floor, Boston, MA 02110-1301, USA.
##

function [Intensity]=sigma_tot(f1,f2,f3,f4,poln_cs,P_1,P_2)

%---- calculate neutron intensities corrected for bender efficiencies
%---- input : spin matrix, neutron cross-section and incoming/outgoing polarisations e.g. [0.9 0 0],[0 0.9 0],[0 0 0.9]
%---- output neutron intensities along x,y,z

Intensity=(f2'*poln_cs*f1)'*(f2'*poln_cs*f1).*P_1.*P_2+...
          (f4'*poln_cs*f1)'*(f4'*poln_cs*f1).*P_1.*(1.-P_2).+...
          (f2'*poln_cs*f3)'*(f2'*poln_cs*f3).*(1-P_1).*P_2.+...
          (f4'*poln_cs*f3)'*(f4'*poln_cs*f3)*(1-P_1).*(1.-P_2);

endfunction
```

An alternative method would be to correct the measured polarisation for the monochromator/analyser efficiencies. If the polarisation of the scattered neutrons beam is P_s then the number of neutrons in the 'up'-channel is $\propto \frac{P_s+1}{2}$ and in the down channel $\propto \frac{1-P_s}{2}$. Hence if the analyser efficiency is P_a then the neutrons oriented 'up' by the analyser is $(1+P_s)/2(1+P_a)/2+(1-P_s)/2(1-P_a)/2 = 1+P_sP_a$ and for the down analyser $(1-P_s)/2(1+P_a)/2+(1+P_s)/2(1-Pa)/2 = 1-P_aP_s$. The measured polarisation is then P_sP_a and the matrix elements can be corrected by dividing by that factor.

0.12 The fit

3 fit algorithms have been implemented in Mufit

- Levenberg-Marquard (leasrq.m)
- Nelder-Mead (nmsmax.m)
- Simulated Annealing (samin.cc)

These functions are free software and belong to the optim package that can be downloaded from www.octaveforge.com. They have been partly modified to fit into the Mufit program.

- The Levenberg-Marquard function uses dfdp.m to calculate the Jacobian.
- The Nelder-Mead algorithm is a direct search algorithm that does not use derivatives. As the size of the initial simplex depends upon the number of free parameters it is recommended that the maximum number of iterations is large (Nmax $\sim$ 1000 for 20 parameters).
- Siman.cc is a simulated annealing function that determines the starting temperature automatically.

More information about the fit options can be obtained either by editing these functions or using the help in Octave (e.g. help leasqr). The fit options can be changed in the input files directly.

```
!levenberg-marquard options: stol niter verbose (0/1)
0.1 10 1
!simplex options: stol niter chisq_min simplex verbose (0/1)
0.001 3000 5  0 1
!simulated anneling options
* nt - integer: # of iterations between temperature reductions
* ns - integer: # of iterations between bounds adjustments
* rt - (0 < rt <1): temperature reduction factor
* maxevals - integer: limit on function evaluations
* neps - integer:  number of values final result is compared to
* functol -   (> 0): the required tolerance level for function value comparisons
* paramtol -  (> 0): the required tolerance level for parameters
* verbosity - scalar: 0, 1, or 2.
20  5  0.95  100000  5  0.1  0.1 2
```