

Implementation of Magnetic Fields and Precession in NISP

P. A. Seeger

October 31, 2009

Caveat: NISP is written completely in Fortran, so when I use words such as “class” and “method,” they come from my less-than-complete understanding of those concepts. I have endeavored to “structure” to code, but always within the limitations of Fortran90. For instance, rather than placing all of the methods associated with an object in a single code module, I am more likely to have a module with a particular method for all of the members of a class. I have also used Fortran indexed arrays to contain multiple structures.

Magnetic regions co-exist with material regions, and they may or may not be coterminous with material regions. They do not have to fill all space and do not have connectivity. All coordinates must be given in the “world” system.

The parameters used by all NISP algorithms are stored sequentially in a single block. Each region has a NAME, and an INDEX pointer to the PARAMETER block. In general, the entry at the index is the TYPE number of the region, and all parameters are offset from that location using integer offsets that have the assigned name of the parameter. The variable names listed below are pointers, whose values are defined in file `mc_element.inc`.

Magnetic Field Class: region types 60 through 69.

Magnetic field regions represented by a magnetic induction vector \mathbf{B} .

The only parameters inherited by all magnetic field types are those associated with time dependence for a pulsed source. The idea is that the field may decrease during a pulse so that $|\mathbf{B}|$ is proportional to $|v|$ and the precession rate is the same over a range of neutron energies. The four parameters are

B_PERIOD, repetition time (μs); 0 for a steady field

B_T0, pulse reference time (μs); the induction vector \mathbf{B} decays as $1/(t - T_0)$

B_T1, time after T_0 to begin field variation (μs); \mathbf{B} is calculated at T_0+T_1 , and $\mathbf{B}(t)=\mathbf{B}\times T_1/(t - T_0)$

B_T2, latest field time measured from T_0 (μs).

Type 60: Uniform magnetic field vector.

B1x, B1y, B1z, ABSB_1, components and magnitude of magnetic induction vector \mathbf{B} (T)

Type 61.n: Field Interpolated between constant values \mathbf{B}_1 and \mathbf{B}_2 on two arbitrary surfaces.

61.0: Linear interpolation of each component of \mathbf{B}

61.1: “Helical” interpolation, linear in magnitude and angle

B1x, B1y, B1z, ABSB_1, components and magnitude of vector \mathbf{B}_1 on surface 1 (T)

B2x, B2y, B2z, ABSB_2, components and magnitude of vector \mathbf{B}_2 on surface 2 (T)

SURF1: structure with 10 elements defining surface 1, normalized to give distance in (m)

SURF2: structure with 10 elements defining surface 2, normalized to give distance in (m)

Bhelix: six matrix elements for helical interpolation (/T)

B1thetaB2: angle between \mathbf{B}_1 and \mathbf{B}_2 (rad)

Type 62: Superposition of any number of Current Loops, including point dipoles. This type requires an external procedure from MCLIB, BLOOP.

N_LOOPS: number of current loops contributing to the region

LOOPN: number of parameters to define each loop, always 8, used to index multiple loops.

LOOPX, LOOPY, LOOPZ: coordinates of center of loop (m)

LOOPALF, LOOPBET, LOOPGAM: direction cosines of loop axis

LOOPRAD: radius of loop, 0 for point dipole (m)

LOOPCRNT: current (A), or (if radius=0) dipole moment/ π (A-m²)
repeat LOOPN parameters for additional loops

Type 63: Solenoid, using external procedure SOLENOID.

SOL_XCENT, SOL_YCENT, SOL_ZCENT: coordinates of center of solenoid (m)

SOL_ALF, SOL_BET, SOL_GAM: direction cosines of solenoid axis

SOL_R, SOL_HL: radius and half-length of the solenoid (m)

SOL_J: surface current density (A-turns/m); the thickness of the current layer is infinitesimal .

Type 64: Finite coaxial alternating Current Sheets. NOTE: this type not yet implemented, because I lack an algorithm for *finite* current sheets. Parameter names have been defined.

N_SHEETS: number of current sheets

SHT_SPACE: spacing of the sheets (m)

SHT_XCENT, SHT_YCENT, SHT_ZCENT: coordinates of center of 1st sheet (m)

SHT_ALF, SHT_BET, SHT_GAM: directions cosines of sheet normal (axis of array)

SHT_DX, SHT_DY: transverse dimensions of sheets (m)

SHT_I: current in sheet (A)

SHT_PHI: azimuthal direction of current in odd sheets (rad); even sheets opposite direction

Type 65: Field defined in File. NOTE: this type has not yet been implemented, because I have not found a suitable file format.

The NAME given to the region will be the File Name.

No parameters have yet been defined.

Type 66: Multiple connected Wire Segments. Uses external procedure BWIRES.

W_NODES: number of nodes, 1 more than number of wire segments

W_CURRENT: common current through all wires (A)

W_X1, W_Y1, W_Z1: coordinates of 1st node (m)

repeat additional triplets for all nodes.

Type 67: not assigned

Type 68: not assigned

Type 69: Superposition of any other types. When a single region contains fields generated by various means, they must be encapsulated in a region of this type. The parameter blocks will be referenced by the offset of the block from this one.

N_FIELDS: number of superposed fields

B_OFFSET: offset to parameter block with field definition
additional pointers to all field definitions.

The **Methods** for calculating **B** for all field types are contained as separate functions attached to the Fortran source file BFIELD.f. All such procedures have the form

```
REAL*8 FUNCTION TYPEnn_Bfunction(t, B, PART, PARAMS, NAME)
```

All of these methods can be called from procedure BFIELD in order to find the **B** vector at the current location of a neutron specified in PART. BFIELD uses a case structure based on the type of the magnetic region to call the proper method. More important, in most cases BFIELD will also call RKPRECES and pass to it the link to the appropriate procedure. The field integration routines have an external function name Bfunction in their calling sequences, which is eventually passed to routine BINTEGRAL. Since the function is called many thousands (or even millions) of times, it is faster not to have to use a case structure for each call. Also, code maintenance is easier with all of the methods defined within the single module, BFIELD.

Polarization is represented by a 3-vector **P**, with magnitude $P \leq 1$. The probability of the spin of the neutron being in direction **P** is $(1 + P)/2$, and the probability of the opposite spin is $(1 - P)/2$. Probabilities in an arbitrary direction given by a unit vector **n** depend on the dot product **n•P**: $(1 + \mathbf{n}\cdot\mathbf{P})/2$ and $(1 - \mathbf{n}\cdot\mathbf{P})/2$. Thus $P = 0$ is an unpolarized beam, with probability in *any* direction being 50%. This notation is completely equivalent to tracking separate spin-up and spin-down neutrons with appropriate statistical weights. The hierarchy of subroutine calls for moving a polarized neutron through a field is

```
MOVEB
  BREGION
  BFIELD
    TYPEnn_Bfunction
    RKPRECES
      TYPEnn_Bfunction
      RK4BLOCH
        TYPEnn_Bfunction
        BINTEGRAL
          TYPEnn_Bfunction
```

The routine BREGION is called to determine whether a neutron is inside any magnetic region, or if its trajectory will intersect a magnetic region. Then if BFIELD determines that the direction of **B** is constant, the complete solution of Bloch's equation is taken as a simple line integral along the neutron trajectory. Otherwise, BFIELD will make an estimate of a step size over which **B** is *approximately* parallel, and then call RKPRECES, which is a quality control manager for the 4th-order Runge-Kutta stepper routine RK4BLOCH. For efficiency in the program, it is essential to decouple the Runge-Kutta step size from the integration step size. The component of **B** orthogonal to **P** produces a *rapid* precession rate, with *slowly* varying phase angles resulting from the change of direction of **B**. The "clever" part of the program is to rotate coordinates in order to separate out the fast part as a one-dimensional line integral, and to use Bloch's equation for the slower variation. Full details of the procedure are given in Seeger&Daemen, NIM A 457 (2001) 338. The actual integrations are done in routine BINTEGRAL, using the Romberg algorithm to minimize the number of points at which the field must be evaluated for each step.