ADDENDUM C:

DNP via Thermal Mixing: Beyond the High Temperature Approximation

This addendum concerns an extension to low spin temperature of the treatment of Provotorov's theory and its application to dynamic nuclear polarization (DNP) via fast thermal mixing—Sections 4.3.1 to 4.3.3, 5.3.2, 5.3.3 and 8.3.3 of Essentials of Dynamic Nuclear Polarization, Spindrift Publications, 2016—henceforward shortly denoted as *EofDNP*. The extension itself is presented in [1]. Here we consider numerical methods to solve two sets of equations derived in that paper. The first part of this addendum concerns Equations (73) and (74) yielding the stationary state of the electron spin system. The second part is devoted to Equations (66) and (67) providing the evolution of the nuclear spin polarization in the case of fast thermal mixing. FORTRAN codes implementing these numerical methods are given in two appendices.

C.1 Stationary State of the Electron Spin System

C.1.1 Stationary Generalized Provotorov Equations

The extension of Provotorov's theory presented in [1] applies for ESR spectra $g(\omega)$ that are inhomogeneously broadened by g-tensor anisotropy. As in *EofDNP* we normalize these spectra, so

$$\int_{\infty}^{\infty} \mathrm{d}\omega\, g(\omega) \;=\; 1, \tag{1}$$

define their centre of gravity ω_0, such that

$$\int_{-\infty}^{\infty} \mathrm{d}\omega\, (\omega-\omega_0) g(\omega) \;=\; 0 \tag{2}$$

and their second moment

$$D^2 \;=\; \int_{-\infty}^{\infty} \mathrm{d}\omega\, (\omega-\omega_0)^2 g(\omega). \tag{3}$$

As discussed in [1], fast spectral diffusion assures that the electron spin polarization as a function of frequency can always be written in the shape

$$P(\omega) \;=\; \tanh \tfrac{1}{2}\big(\omega_0\alpha + (\omega-\omega_0)\beta_{\mathrm{NZ}}\big) \tag{4}$$

proposed by Provotorov. Thus the electron spin system is described by two inverse spin temperatures α and β_{NZ}. These are traditionally called the inverse electron Zeeman temperature and the inverse electron non-Zeeman temperature, though at low spin temperature neither the electron Zeeman energy

$$U_{\mathrm{Z}} \;=\; \tfrac{1}{2}\hbar\omega_0 \int_{-\infty}^{\infty} \mathrm{d}\omega\, g(\omega) P(\omega) \tag{5}$$

is solely determined by α, nor the electron non-Zeeman energy

$$U_{\rm NZ} \;=\; \tfrac{1}{2}\hbar \int_{-\infty}^{\infty} {\rm d}\omega\; (\omega-\omega_0) g(\omega) P(\omega) \tag{6}$$

is solely a function of $\beta_{\rm NZ}$.

We consider the stationary state that is reached after applying a microwave field with a frequency $\omega_{\rm m}$ and an amplitude B_1 for a long time. According to [1], in this stationary state

$$\begin{aligned} F_1 \;&=\; \int_{-\infty}^{\infty} {\rm d}\omega\; g(\omega) P_S(\omega) \;-\; P_{\rm L} \;+\; 2W(\omega_{\rm m}) T_{1S} P_S(\omega_{\rm m}) \qquad\qquad &=\; 0\,, \\ F_2 \;&=\; \int_{-\infty}^{\infty} {\rm d}\omega\; (\omega-\omega_0) g(\omega) P_S(\omega) + 2W(\omega_{\rm m}) T_{1S} (\omega_{\rm m}-\omega_0) P_S(\omega_{\rm m}) \;&=\; 0\,. \end{aligned} \tag{7}$$

In these equations

$$P_{\rm L} \;=\; \tanh \tfrac{1}{2}\omega_0 \beta_{\rm L}, \qquad\qquad \beta_{\rm L} \;=\; \frac{\hbar}{k_{\rm B} T_{\rm L}} \tag{8}$$

is the average electron spin polarization when the electron spin system is in thermal equilibrium with the lattice. Here $T_{\rm L}$ and $\beta_{\rm L}$ are the lattice temperature and the inverse lattice temperature, while $\hbar$ and $k_{\rm B}$ are Planck's and Boltzmann's constants. Furthermore

$$W(\omega_{\rm m}) \;=\; \tfrac{1}{2}\pi \omega_{1S}^2 g(\omega_{\rm m}), \qquad\qquad \omega_{1S} \;=\; \gamma_S B_1 \tag{9}$$

is the rate at which the microwave field induces electron spin flips. Here we ignore the anisotropy of the g-tensor and set γ_S to be the average gyromagnetic ratio of the electron spins.

C.1.2 Numerical Methods

Numerical solutions of the set of equations (7) are found using a Newton-Raphson method for nonlinear systems of equations described in [2]. To solve a set of n nonlinear equations

$$F_i(x_1, \cdots, x_j, \cdots, x_n) \;=\; 0, \tag{10}$$

we insert trial solutions x_j^0 and expand up to first order:

$$\begin{aligned} &F_i(x_1^0 + \delta x_1, \cdots, x_j + \delta x_j, \cdots, x_n + \delta x_n) \\ &\qquad =\; F_i(x_1^0, \cdots, x_j^0, \cdots, x_n^0) \;+\; \sum_{j=1}^{n} \left(\frac{\partial F_i}{\partial x_j} \right)_{x_j = x_j^0} \delta x_j \;+\; \cdots \end{aligned} \tag{11}$$

Next we solve the set of equations

$$\left(\frac{\partial F_i}{\partial x_j} \right)_{x_j = x_j^0} \delta x_j \;=\; 0, \tag{12}$$

and set new trial solutions equal to $x_j^0 + \delta x_j$. This process is continued until

$$\sum_{i=1}^{n} \left| F_i(x_1, \cdots, x_j, \cdots, x_n) \right| \tag{13}$$

is less than a pre-determined threshold.

To implement this method we need the derivatives of the functions F_1 and F_2 defined in (7). We determine them analytically. We recall (4) showing that $P(\omega)$ is a tangent hyperbolic function, and note that

$$\frac{\partial}{\partial x}\tanh x \;=\; 1-\tanh^2 x. \tag{14}$$

We furthermore recall (2) and (1). As a result we find:

$$\frac{\partial F_1}{\partial \alpha} \;=\; \tfrac{1}{2}\omega_0\left[1\;-\int_{-\infty}^{\infty}\mathrm{d}\omega\; g(\omega)P_S^2(\omega)\;+\;2W(\omega_\mathrm{m})T_{1S}\big(1-P_S^2(\omega_\mathrm{m})\big)\right], \tag{15}$$

$$\frac{\partial F_1}{\partial \beta_\mathrm{NZ}} \;=\; \tfrac{1}{2}\left[-\int_{-\infty}^{\infty}\mathrm{d}\omega\;(\omega-\omega_0)g(\omega)P_S^2(\omega)\right.$$
$$\left.+\;2W(\omega_\mathrm{m})T_{1S}(\omega_\mathrm{m}-\omega_0)\big(1-P_S^2(\omega_\mathrm{m})\big)\right], \tag{16}$$

$$\frac{\partial F_2}{\partial \alpha} \;=\; \tfrac{1}{2}\omega_0\left[-\int_{-\infty}^{\infty}\mathrm{d}\omega\;(\omega-\omega_0)g(\omega)P_S^2(\omega)\right.$$
$$\left.+\;2W(\omega_\mathrm{m})T_{1\mathrm{NZ}}(\omega_\mathrm{m}-\omega_0)\big(1-P_S^2(\omega_\mathrm{m})\big)\right], \tag{17}$$

$$\frac{\partial F_2}{\partial \beta_\mathrm{NZ}} \;=\; \tfrac{1}{2}\left[D^2-\int_{-\infty}^{\infty}\mathrm{d}\omega\;(\omega-\omega_0)^2g(\omega)P_S^2(\omega)\right.$$
$$\left.+\;2W(\omega_\mathrm{m})T_{1\mathrm{NZ}}(\omega_\mathrm{m}-\omega_0)^2\big(1-P_S^2(\omega_\mathrm{m})\big)\right], \tag{18}$$

in which D^2 is given by (3).

C.1.3 Example Results

Figure 1 presents solutions of (7) for $\beta_\mathrm{NZ}/\beta_\mathrm{L}$ as a function of the microwave frequency ω_m obtained using the FORTRAN codes given in the Appendix. The calculation is performed for TEMPO in an externally applied magnetic field $B_0 = 3.4$ T, at a temperature $T_\mathrm{L} = 1.5$ K, and for five different strength of the microwave power, such that $s_0 = W(\omega_0)T_{1S} = 0.01$, 0.1, 1, 10 and 100 at the centre of gravity ω_0 of the ESR spectrum. This spectrum was determined by first calculating the exact spectrum of just the electron spin and the ^{14}N spin using the components of the g-tensor and the hyperfine tensor provided in [3]. Next we included the hyperfine and super-hyperfine interaction with the remaining nuclear spins by convoluting this spectrum with a Gaussian with a width of 7.1 MHz—see Addendum B to *EofDNP*.

For comparison Figure 2 presents results that would have been obtained using the high temperature approximation. In this figure expression (5.81) from *EofDNP*,

$$\frac{\beta_\mathrm{NZ}}{\beta_\mathrm{L}} \;=\; -\,\frac{2s_\mathrm{m}\omega_0(\omega_\mathrm{m}-\omega_0)}{2s_\mathrm{m}\big[(\omega_\mathrm{m}-\omega_0)^2+D^2\big]+D^2}, \tag{19}$$

is used to calculate the curves. Here $s_\mathrm{m} = W(\omega_\mathrm{m})T_{1S}$. Especially at the edges of the ESR spectrum and at weaker microwave powers the general equations (7) predict a

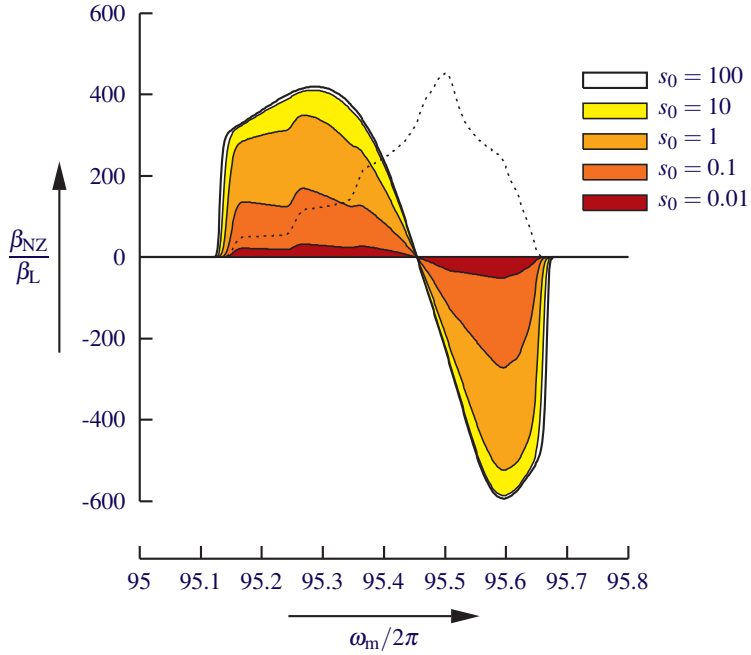

Figure 1: Example solutions of (7): β_{NZ}/β_L as a function of the microwave frequency ω_m for TEMPO in an externally applied magnetic field $B_0 = 3.4$ T, at a temperature $T_L = 1.5$ K and for five different strengths of the microwave power, such that $s_0 = W(\omega_0)T_{1S} = 0.01$, 0.1, 1, 10 and 100 at the centre of gravity of the ESR spectrum.

larger enhancement of the ratio β_{NZ}/β_L than the high temperature approximation. The reason is that the high temperature approximation does not take into account that the energy that can be stored in the electron non-Zeeman reservoir is considerably reduced when the electron spin polarization is high—see Section 4.3.1 of *EofDNP*. As a result the change of β_{NZ} is larger than predicted by the high temperature approximation.

C.2 Evolution of the Nuclear Spin Polarization

C.2.1 Equations for Fast Thermal Mixing

Figure 3 reproduces Figure 8.10 from *EofDNP* and shows the flows of energy in DNP via fast thermal mixing. Then the rate τ_{SSI}^{-1} of triple spin flips coupling the nuclear Zeeman reservoir to the electron non-Zeeman reservoir is much faster than all other processes. As a result the inverse nuclear Zeeman temperature β_I is always equal to the inverse electron non-Zeeman temperature β_{NZ}, and the polarization of nuclear spins $I = \frac{1}{2}$ is simple given by

$$P_I = \tanh \tfrac{1}{2}\omega_{0I}\beta_{NZ}, \tag{20}$$

in which ω_{0I} is the NMR frequency. Then the growth of the nuclear spin polarization

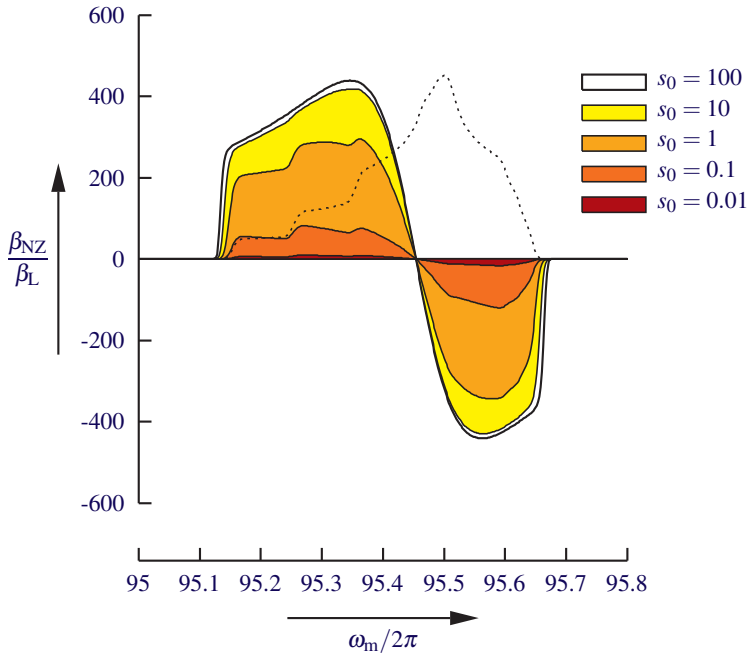

Figure 2: The high temperature approximation: expression (19) used to calculate β_{NZ}/β_L as a function of the microwave frequency ω_m for TEMPO in an externally applied magnetic field $B_0 = 3.4$ T and for five different strengths of the microwave power, such that $s_0 = W(\omega_0)T_{1S} = 0.01$, 0.1, 1, 10 and 100 at the centre of gravity of the ESR spectrum.

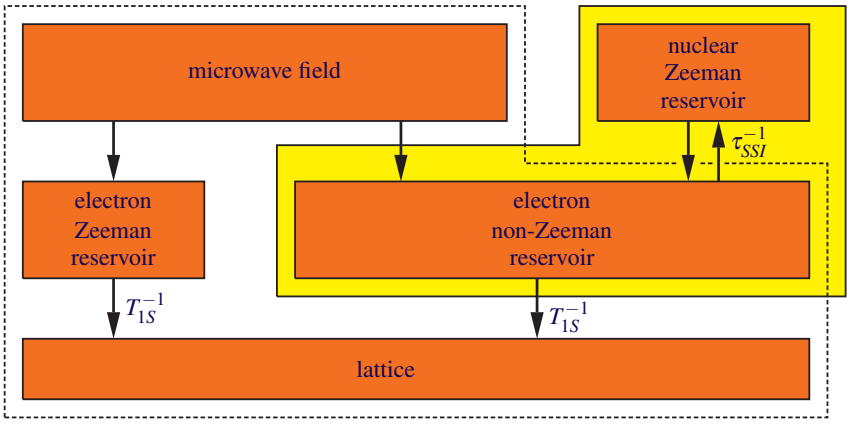

Figure 3: The flows of energy in DNP via fast thermal mixing.

as a function of time t can be solved from equations (66) and (67) in [1]:

$$\begin{aligned}
\omega_{0I}\frac{\partial P_I}{\partial t} &= -2W(\omega_\mathrm{m})\,\frac{N_S}{N_I}\,(\omega_\mathrm{m}-\omega_0)\tanh\tfrac{1}{2}\big(\omega_0\alpha+(\omega_\mathrm{m}-\omega_0)\beta_\mathrm{NZ}\big) \\
&\quad -\frac{1}{T_{1S}}\,\frac{N_S}{N_I}\int_{-\infty}^{\infty}\mathrm{d}\omega\; g(\omega)(\omega-\omega_0)\tanh\tfrac{1}{2}\big(\omega_0\alpha+(\omega-\omega_0)\beta_\mathrm{NZ}\big), \qquad (21) \\
0 &= -2W(\omega_\mathrm{m})\tanh\tfrac{1}{2}\big(\omega_0\alpha+(\omega_\mathrm{m}-\omega_0)\beta_\mathrm{NZ}\big) \\
&\quad -\frac{1}{T_{1S}}\left[\int_{-\infty}^{\infty}\mathrm{d}\omega\; g(\omega)\tanh\tfrac{1}{2}\big(\omega_0\alpha+(\omega-\omega_0)\beta_\mathrm{NZ}\big)\;-\;P_\mathrm{L}\right], \qquad (22)
\end{aligned}$$

in which N_S is the total number of electron spins, N_I is the total number of nuclear spins, while all other parameters are defined in Section C.1.1.

C.2.2 Numerical Methods and Example Solutions

The code presented in Appendix C.B solves time t as a function of nuclear spin polarization P_I. First the inverse electron non-Zeeman temperature β_NZ as a function of P_I is solved from (20). Next, the inverse electron Zeeman temperature α as a function of β_NZ is determined using (22). Inserting the two inverse temperatures α and β_NZ in (21) then yields $\partial P_I/\partial t$ as a function of P_I. Finally, the integral

$$t \;=\; \int_0^{P_I}\mathrm{d}P_I'\left(\frac{\partial P_I'}{\partial t}\right)^{-1} \qquad (23)$$

yields time t as a function of the nuclear spin polarization P_I.

As an example Figure 4 shows the evolution of the proton spin polarization in a sample doped with TEMPO. The calculations were performed for $B_0 = 3.4$ T, $T_L = 1.5$ K, $\omega_\mathrm{m}/2\pi = 95.5828$ GHz and $s_0 = W_0T_{1S} = 100$, 10, 1, 0.1, where $W_0 = \pi\omega_{1S}^2 g(\omega_0)$. As in Figures 1 and 2 were determined using the method described in Addendum B of *EofDNP*. Notice that results for larger values of s_0 coincide with those for $s_0 = 100$.

References

[1] W.Th. Wenckebach: *J. Magn. Res.* (2017) DOI:10.1016/j.jmr.2017.01.020.

[2] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery: *Numerical Recipes in Fortran*, 2nd Ed., Cambridge University Press, Cambridge, 1992.

[3] S.T. Goertz, J. Harmsen, J. Heckmann, Ch. Hess, W. Meyer, E. Radtke, G. Reicherz: *Nucl. Instrum. Methods Phys. Res. A* 526 (2004) 43-52 DOI:10.1016/j.nima.2004.03.148.

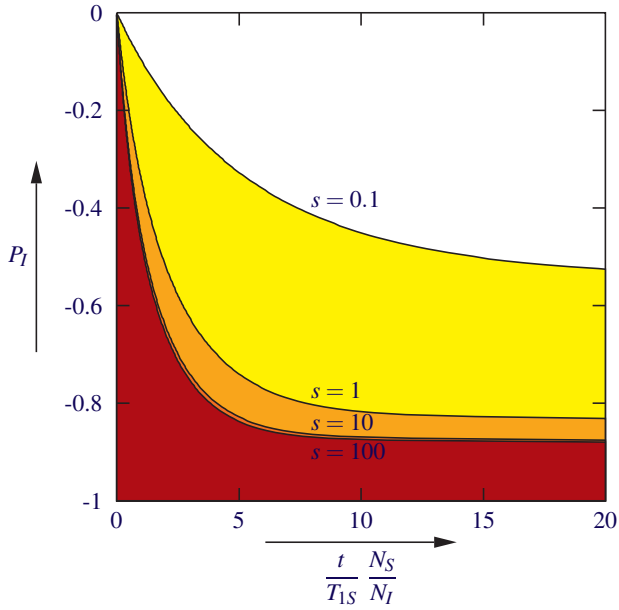

Figure 4: Evolution of the proton spin polarization for $B_0 = 3.4$ T, $T_L = 1.5$ K, $\omega_m/2\pi = 95.5828$ GHz and $s_0 = W_0 T_{1S} = 100$, 10, 1, 0.1, where $W_0 = \pi\omega_{1S}^2 g(\omega_0)$. Results for larger values of s_0 coincide with those for $s_0 = 100$.

Appendix: FORTRAN Codes

This appendix lists the FORTRAN codes. They were tested with `gfortran` on a Dell Mobile Precision ® 5510 CTO running Xubuntu 16.04 for 64 bits processors and a Dell Latitude ® E62230 running Debian 7.0.0 for 64 bits processors. The compiled code requires a sub-directory `inout/` containing a data file `inf.dat` with input parameters. The format of this data file with example parameters is also given. The output is written to the file `outf.txt` in the sub-directory `inout/`. The output file of the codes solving (7) consists of three columns: the microwave frequency $\omega_m/2\pi$ in units GHz and the inverse spin temperatures $2\pi\alpha$ and $2\pi\beta$ in units GHz^{-1}. The output file of the codes solving (7) consists of two columns: time and nuclear spin polarization P_l.

C.A Fortran Codes to Solve (7)

Main program:

```
C*************************************************************************
C*                              statsol.f                                *
C*************************************************************************
C* Calculates stationary solutions of generalized Provotorov equations   *
C*************************************************************************
C* Output:                                                               *
C* esr(1,im) = microwave frequency                    (double real) *
C* x(1)      = inverse spin temperature alpha         (double real) *
C* x(2)      = inverse spin temperature beta          (double real) *
C*************************************************************************
C* Uses:                                                                 *
C* blocks.f                  for other declarations                      *
C*    s0     = WT_1S = saturation factor at om0       (double real) *
C*    s      = WT_1S = saturation factor at esr(1,im) (double real) *
C*             W = (1/2) omega_1S^2 g(omega)                             *
C*    eps    = minimum for s                          (double real) *
C*    spec0  = amplitude ESR spectrum at its centre of gravity (double real) *
C*    betal  = inverse lattice temperature            (double real) *
C*************************************************************************
C* Calls:                                                                *
C* init(ntrial,tolx,tolf)      for input                                 *
C* mnewt(ntrial,tolx,tolf,x)   to solve the equations                    *
C*    ntrial = number of iteration in mnewt           (integer)     *
C*    tolx   = precision parameter in mnewt           (double real) *
C*    tolf   = precision parameter in mnewt           (double real) *
C*************************************************************************

      program main

      implicit none
      include "blocks.f"

      integer ntrial
      double precision tolx, tolf
      double precision x(1:2)
      integer i, n
      character outf*60

      outf = "./inout/outf.txt"
      open(30,file=outf)
```

```
C*************************************************************************
C* initiate parameters                                                   *
C*************************************************************************
      call init(ntrial,tolx,tolf)

C*************************************************************************
C* perform calculations                                                  *
C*************************************************************************
      n = 1200
      do im=1,n,2
        s = s0*esr(2,im)/spec0
        if (s.lt.eps) then
          x(1) = beta1
          x(2) = 0.0d0
          goto 90
        else
          call mnewt(ntrial,tolx,tolf,x)
        endif

C*************************************************************************
C* print x(1) = alpha and x(2) = beta on standard output and file        *
C*************************************************************************
 90     write(*,100)  esr(1,im), x(1), x(2)
        write(30,200) esr(1,im), x(1), x(2)
      end do

      close(30)

 100  format("omega_m =",f12.7,
     1"   alpha =",f12.7,"  (GHz^-1)   beta =",f12.7,"  (GHz^-1)")
 200  format(f12.7,"   ",f12.7,"   ",f12.7)

      end
```

Subroutine init:

```
C*************************************************************************
C*                              init.f                                   *
C*************************************************************************
C* reads input parameters from rdat = inf.dat                            *
C* reads ESR spectrum from resr = esr-34000.txt                          *
C* determines norm, centre of gravity and second moment of ESR spectrum  *
C* determines inverse lattice temperature, thermal equilibrium polarization *
C*************************************************************************
C* Input read from rdat:                                                 *
C* e        = e                                          (double real) *
C* pi       = pi                                         (double real) *
C* hbar     = Planck's constant                          (double real) *
C* kb       = Boltzmann's constant                       (double real) *
C* b0       = B_0 = static magnetic field (T)            (double real) *
C* tl       = lattice temperature                        (double real) *
C* s0       = WT_lS = saturation factor at om0           (double real) *
C*            W = (1/2) omega_lS^2 g(omega)                            *
C* eps      = minimum for s                              (double real) *
C* ntrial   = number of iteration in mnewt               (integer)     *
C* tolx     = precision parameter in mnewt               (double real) *
C* tolf     = precision parameter in mnewt               (double real) *
C*************************************************************************
C* Output:                                                               *
C* ntrial   = number of iteration in mnewt               (integer)     *
C* tolx     = precision parameter in mnewt               (double real) *
C* tolf     = precision parameter in mnewt               (double real) *
C*************************************************************************
```

```
C*************************************************************************
C* Loaded to common block:                                               *
C* esr(1,i) = ESR spectrum, frequency                    (double real) *
C* esr(2,i) = ESR spectrum, intensity                    (double real) *
C* norm     = normalization ESR spectrum                 (double real) *
C* om0      = centre of gravity omega_0 of the ESR spectrum (double real) *
C* spec0    = amplitude spectrum at centre of gravity    (double real) *
C* d2       = second moment of spectrum                  (double real) *
C* betal    = inverse lattice temperature                (double real) *
C* pl       = thermal equilibrium polarization           (double real) *
C*************************************************************************
C* Internal parameters:                                                  *
C* ib0      = 10^4*B_0                                   (integer)     *
C* c        = conversion factor freq. vs. temp.          (double real) *
C* arg      = exponent Boltzmann factor                  (double real) *
C*************************************************************************
C* Calls:                                                                *
C* rnul(rdat,linenr)            to read empty line from rdat             *
C* rint(rdat,linenr,intin)      to read integer from rdat                *
C* rreal(rdat,linenr,intin)     to read double real from rdat            *
C*************************************************************************

      subroutine init(ntrial,tolx,tolf)

      implicit none

      include "blocks.f"

      integer ntrial
      double precision tolx, tolf
      integer ib0
      integer i, n
      double precision error
      double precision e, pi, hbar, kb, c, arg

      character rdat*60
      character resr*60

C*************************************************************************
C* read data file                                                        *
C*************************************************************************
      rdat = "./inout/inf.dat"
      open(10,file=rdat)
      call rnul(rdat,1)
      call rnul(rdat,2)
      call rnul(rdat,3)
      call rnul(rdat,4)
      call rnul(rdat,5)
      call rreal(rdat,6,e)
      call rreal(rdat,7,pi)
      call rreal(rdat,8,hbar)
      call rreal(rdat,9,kb)
      call rnul(rdat,10)
      call rnul(rdat,11)
      call rreal(rdat,12,b0)
      call rreal(rdat,13,t1)
      call rreal(rdat,14,s0)
      call rnul(rdat,15)
      call rnul(rdat,16)
      call rreal(rdat,17,eps)
      call rint(rdat,18,ntrial)
      call rreal(rdat,19,tolx)
      call rreal(rdat,20,tolf)
```

```
C*************************************************************************
C* load lineshape of TEMPO                                               *
C*************************************************************************
      ib0 = idnint(1.0d4*b0)
      if(ib0.eq.34000) then
        resr  = "./inout/esr-34000.txt"
      else
        write(*,*) "no input file for spectrum"
        stop
      endif
      open(20,file=resr)

      n = 1200
      do i=0,n,1
        read (20,*) esr(1,i), esr(2,i)
      end do
      close(20)

C*************************************************************************
C* Normalize spectrum                                                    *
C* Determine centre of gravity om0 of spectrum                           *
C* Determine amplitude spec0 spectrum at om0                             *
C* Determine second moment d2 of spectrum                                *
C*************************************************************************
      norm = 0.0d0
      do i=1,n-1,1
        norm = norm+esr(2,i)
      end do
      write(*,*) "norm     =", norm

      om0 = 0.0d0
      do i=1,n,1
        om0 = om0+esr(1,i)*esr(2,i)
      end do
      om0 = om0/norm
      write(*,*) "omega_0 =", om0, "GHz"

      do i=1,n-1,1
        error = dabs(om0-esr(1,i))
        if (error.lt.esr(1,i+1)-esr(1,i)) spec0 = esr(2,i)
      end do
      write(*,*) "spec0    =", spec0

      d2  = 0.0d0
      do i=1,n-1,1
        d2  = d2+(esr(1,i)-om0)*(esr(1,i)-om0)*esr(2,i)
      end do
      d2  = d2/norm
      write(*,*) "D        =", sqrt(d2), "GHz"

C*************************************************************************
C* calculate thermal equilibrium polarization                            *
C*************************************************************************

      write(*,*) "T_L      =", t1, "K"
      write(*,*) "B_0      =", b0, "T"
      c     = kb/(2.0d0*pi*hbar)/1.0d9
      beta1 = 1.0d0/(c*t1)
      write(*,*) "beta_L   =", beta1, "GHz^-1"
      arg   = 0.5d0*om0/(c*t1)
      p1    = (e**arg-e**(-arg))/(e**arg+e**(-arg))
      write(*,*) "P_L      =", p1

      end
```

Subroutines reading lines of data file:

```
C*************************************************************************
C*                              rnul.f                                   *
C*************************************************************************
C* reads double empty line from iofile                                   *
C*************************************************************************
C* Internal variables:                                                   *
C* dummy    = character string for intermediate use                      *
C* linenr   = number of line being read                    (integer)     *
C* iofile   = character string denoting input file                       *
C*************************************************************************

      subroutine rnul(iofile,linenr)

      implicit none

      integer   linenr
      character iofile*60
      character dummy*80

      read (10,"(a80)",err=70) dummy
      go to 1000



   70 continue
      write (*,*) 'ERROR reading line ', linenr, ' of ', iofile

 1000 continue
      end

C*************************************************************************
C*                              rreal.f                                  *
C*************************************************************************
C* reads double real number from iofile                                  *
C*************************************************************************
C* Internal variables:                                                   *
C* dummy    = character string for intermediate use                      *
C* dummy1   = character string for intermediate use                      *
C* linenr   = number of line being read                    (integer)     *
C* pos      = number of column being read                  (integer)     *
C* iofile   = character string denoting input file                       *
C* realin   = real to be read from file                    (double real) *
C*************************************************************************

      subroutine rreal(iofile,linenr,realin)

      implicit none

      integer   linenr, pos
      character iofile*60
      character dummy*80
      character dummy1*80
      double precision realin

      read (10,"(a80)",err=70) dummy
      pos = index(dummy,':') + 2
      dummy1 = dummy(pos:)
      dummy = dummy1
      read (dummy,*,err=70) realin
      go to 1000

   70 continue
      write (*,*) 'ERROR reading line ', linenr, ' of ', iofile

 1000 continue
      end
```

```
C*****************************************************************************
C*                              rint.f                                       *
C*****************************************************************************
C* reads integer from iofile                                                 *
C*****************************************************************************
C* Internal variables:                                                       *
C* dummy    = character string for internal use                              *
C* dummy1   = character string for internal use                              *
C* linenr   = number of line being read                    (integer)        *
C* pos      = number of column being read                  (integer)        *
C* iofile   = chacter string denoting input file                             *
C  intin    = integer to be read from file                 (integer)        *
C*****************************************************************************

      subroutine rint(iofile,linenr,intin)

      implicit none

      integer linenr, pos
      character iofile*60
      character dummy*80
      character dummy1*80
      integer intin


      read (10,'(a80)',err=70) dummy
      pos = index(dummy,':') + 2
      dummy1 = dummy(pos:)
      dummy = dummy1
      read (dummy,*,err=70) intin
      go to 1000

   70 continue
      write (*,*) 'ERROR reading line ', linenr, ' of ', iofile

 1000 continue
      end
```

Example data file read by subroutine init:

```
01..5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0....5
02. Input file for the calculation of the static solutions of
03. the generalized Provotorov equations
04.
05. Basic constants:
06. e                          : 2.7182818d0      (double real)
07. pi                         : 3.1415926d0      (double real)
08. hbar                       : 1.0545716d-34    (Js double real)
09. kb                         : 1.3806504d-23    (J/K double real)
10.
11. Data for calculation:
12. B_0                        : 3.4              (T, double real)
13. T_L                        : 1.5              (K, double real)
14. W(omega_0)T_1S             : 1.0              (double real)
15.
16. Parameters for mnewt:
17. eps                        : 1.0d-6           (double real)
18. ntrial                     : 10000            (integer)
19. tolx                       : 1.0d-5           (double real)
20. tolf                       : 1.0d-5           (double real)
```

Common blocks storing data read from data file:

```
C*************************************************************************
C*                                blocks.f                               *
C*************************************************************************
C* Contains common declarations                                          *
C*************************************************************************
C* b0        = magnetic field                                (double real) *
C* t1        = lattice temperature                           (double real) *
C* s0        = WT_1S = saturation factor at om0              (double real) *
C* s         = WT_1S = saturation factor at im               (double real) *
C*             W = (1/2) omega_1S^2 g(omega)                               *
C* eps       = minimum for s                                 (double real) *
C* esr(1,i)  = ESR spectrum, frequency                       (double real) *
C* esr(2,i)  = ESR spectrum, intensity                       (double real) *
C* norm      = normalization ESR spectrum                    (double real) *
C* om0       = centre of gravity omega_0 of the ESR spectrum (double real) *
C* spec0     = amplitude spectrum at centre of gravity       (double real) *
C* d2        = second moment of spectrum                     (double real) *
C* betal     = inverse lattice temperature                   (double real) *
C* pl        = thermal equilibrium polarization              (double real) *
C* im        = microwave frequency                           (integer)     *
C*************************************************************************

      double precision b0, t1, s0
      double precision eps
      double precision esr(1:2,1:1200)
      double precision norm, om0, spec0, d2
      integer im
      double precision s
      double precision betal, pl

      common /block1/b0, t1, s0
      common /block2/s, eps
      common /block3/esr
      common /block4/norm, om0, spec0, d2
      common /block5/betal, pl
      common /block6/im
      save /block1/,/block2/,/block3/,/block4/,/block5/,/block6/
```

Subroutine mnewt:

This subroutine is modified from [2]. Copyrights are owned by Cambridge University Press, and only the headings showing the modifications are provided here.

```
C*************************************************************************
C*                                mnewt.f                                *
C*************************************************************************
C* Modified from Numerical Recipies in Fortran, 2nd Edition, 1994        *
C*************************************************************************
C* Newton-Raphson method to solve 2 variables                            *
C* from a set of 2 non-linear equations F_i = 0                          *
C*************************************************************************
C* Input:                                                                *
C* ntrial     = maximum number of iterations                             *
C* tolf       = iteration stops if tolf > sum_i abs(F_i)                 *
C* tolx       = iteration stops if tolx > sum_i abs(delta x_i)           *
C*              delta x_i difference between two subsequent steps        *
C*************************************************************************
C* Output:                                                               *
C* x          = solution vector                                          *
C*************************************************************************
```

```
C*************************************************************************
C* Internal parameters                                                   *
C* fjac       = Jacobian 2x2 matrix with elements dF_i/dx_j              *
C* fvec       = 2-dim vector with functions F_i (i=1,2)                  *
C* p          = 2-dim vector = solution of fjac*p = -fvec                *
C* errf       = sum_i abs(F_i)                                           *
C* errx       = sum_i abs(delta x_i)                                     *
C*************************************************************************
C* Calls:                                                                *
C* calf(x,fvec,fjac)       determines fvec and fjac for given x(1) and x(2) *
C* lineqs(fjac,fvec,p)     solves p from fjac*p = -fvec                  *
C*************************************************************************

      subroutine mnewt(ntrial,tolx,tolf,x)

      implicit none

      include "blocks.f"

      integer ntrial
      double precision tolx, tolf
      double precision x(1:2)

      integer i, j, k
      double precision fjac(1:2,1:2)
      double precision fvec(1:2), p(1:2)
      double precision errf, errx
```

Subroutine calf:

```
C*************************************************************************
C*                              calf.f                                   *
C*************************************************************************
C* calculates components of the vector and jacobian for mnewt            *
C*************************************************************************
C* input:                                                                *
C* x(1)     = alpha = inverse electron Zeeman temperature     (double real) *
C* x(2)     = beta  = inverse electron non-Zeeman temperature (double real) *
C*************************************************************************
C* loaded from common block:                                             *
C* esr(1,i) = ESR spectrum, frequency                         (double real) *
C* esr(2,i) = ESR spectrum, intensity                         (double real) *
C* norm     = normalization ESR spectrum                      (double real) *
C* om0      = centre of gravity omega_0 of the ESR spectrum   (double real) *
C* s        = WT_1S = saturation factor                       (double real) *
C*            W = (1/2) omega_1S^2 g(omega)                               *
C* spec0    = amplitude spectrum at centre of gravity         (double real) *
C* d2       = second moment of spectrum                       (double real) *
C* im       = microwave frequency                             (integer)     *
C* pl       = thermal equilibrium polarization                (double real) *
C*************************************************************************
C* output:                                                               *
C* fv(1)   = F_1 = P_S - P_L + 2WT_1S*P_S(omega_m)                       *
C* fv(2)   = F_2 = U_NZ + 2WT_1S*(omega_m - omega_0)*P_S(omega_m)        *
C* fj(1,1) = dF_1/dx_1                                                   *
C* fj(1,2) = dF_1/dx_2                                                   *
C* fj(2,1) = dF_2/dx_1                                                   *
C* fj(2,2) = dF_2/dx_2                                                   *
C*************************************************************************
C* internal parameters:                                                  *
C* pm       = P_S(omega_m) = polarization at microwave freq.  (double real) *
C* s(1)     = saturation parameter = WT_{1S}                             *
C* s(2)     = saturation parameter = WT_{1NZ}                            *
C*            W = (1/2) omega_1S^2 g(omega)                              *
C* intgr(1,1) = intgral d omega g(omega) P^2(omega)                      *
C* intgr(1,2) = intgral d omega (omega-omega_0) g(omega) P^2(omega)      *
C* intgr(2,2) = intgral d omega (omega-omega_0)^2 g(omega) P^2(omega)    *
C*************************************************************************
```

```
      subroutine calf(x,fv,fj)
      implicit none

      include "blocks.f"

      double precision x(1:2), fv(1:2)
      double precision fj(1:2,1:2)

      integer i, n
      double precision e, arg, pm, tnh
      double precision intgr(1:2,1:2)

      e          = 2.7182818d0
      n          = 1000
      fv(1)    = 0.0d0
      fv(2)    = 0.0d0
      intgr(1,1) = 0.0d0
      intgr(1,2) = 0.0d0
      intgr(2,1) = 0.0d0
      intgr(2,2) = 0.0d0

      arg = 0.5d0*(om0*x(1)+(esr(1,im)-om0)*x(2))
      pm  = (e**arg-e**(-arg))/(e**arg+e**(-arg))
      do i=1,n,1
        arg      = 0.5d0*(om0*x(1)+(esr(1,i)-om0)*x(2))
        tnh      = (e**arg-e**(-arg))/(e**arg+e**(-arg))
        fv(1)    = fv(1)+esr(2,i)*tnh
        fv(2)    = fv(2)+esr(2,i)*(esr(1,i)-om0)*tnh
        intgr(1,1) = intgr(1,1)+0.5d0*esr(2,i)*tnh*tnh
        intgr(1,2) = intgr(1,2)+0.5d0*esr(2,i)*(esr(1,i)-om0)*tnh*tnh
        intgr(2,2) = intgr(2,2)
     1               +0.5d0*esr(2,i)*((esr(1,i)-om0)**2)*tnh*tnh
      end do

      fv(1)    = fv(1)/norm
      fv(2)    = fv(2)/norm
      intgr(1,1) = intgr(1,1)/norm
      intgr(1,2) = intgr(1,2)/norm
      intgr(2,2) = intgr(2,2)/norm

      fv(1)    = fv(1)-p1+2.0d0*s*pm
      fv(2)    = fv(2)+2.0d0*s*(esr(1,im)-om0)*pm
      fj(1,1) = 0.5d0*om0*(1.0d0-intgr(1,1)+2.0d0*s*(1.0d0-pm*pm))
      fj(1,2) = 0.5d0*(-intgr(1,2)
     1           +2.0d0*s*(esr(1,im)-om0)*(1.0d0-pm*pm))
      fj(2,1) = 0.5d0*om0*(-intgr(1,2)
     1           +2.0d0*s*(esr(1,im)-om0)*(1.0d0-pm*pm))
      fj(2,2) = 0.5d0*(d2-intgr(2,2)
     1           +2.0d0*s*((esr(1,im)-om0)**2)*(1.0d0-pm*pm))

      end
```

Subroutine lineqs:

```
C*************************************************************************
C*                                lineqs.f                               *
C*************************************************************************
C* solves a set of 2 coupled linear equations m*x = -b                   *
C*************************************************************************
C* Input:                                                                *
C* m = 2x2 matrix                                                        *
C* b = 2-dim vector                                                      *
C*************************************************************************
C* Output:                                                               *
C* p = 2-dim solution vector                                             *
C*************************************************************************
```

```
C*************************************************************************
C* Internal parameters                                                   *
C* d = determinant of m                                                  *
C*************************************************************************

      subroutine lineqs(m,b,p)

      implicit none

      double precision m(1:2,1:2)
      double precision b(1:2)
      double precision p(1:2)

      double precision d

      d    = m(1,1)*m(2,2)-m(1,2)*m(2,1)
      p(1) = (m(2,2)*b(1)-m(1,2)*b(2))/d
      p(2) = (m(1,1)*b(2)-m(2,1)*b(1))/d

      end
```

C.B Fortran Codes to Solve (21) and (22)

Main program:

```
C*************************************************************************
C*                                dynsol.f                               *
C*************************************************************************
C* Calculates evolution of the proton spin polarization following from   *
C* the generalized Provotorov equations                                  *
C*************************************************************************
C* Parameters from common blocks:                                        *
C* omm      = microwave frequency (GHz)                  (double real) *
C* om0      = centre of gravity omega_0 of the ESR spectrum (double real) *
C* omp      = NMR frequency proton spins                 (double real) *
C* nstep    = number of steps in nuclear polarization    (integer)     *
C* x1       = lower limit variable = tanh(0.5*om0*alpha)  (double real) *
C* x2       = upper limit variable = tanh(0.5*om0*alpha)  (double real) *
C* xacc     = accuracy rtbis                             (double real) *
C*************************************************************************
C* Internal parameters:                                                  *
C* ppmax    = maximum nuclear spin polarization          (double real) *
C* dpp      = step size nuclear spin polarization        (double real) *
C* p(i)     = nuclear spin polarization                  (double real) *
C* x0       = variable = tanh(0.5*om0*alpha)             (double real) *
C* args     = om0*alpha                                  (double real) *
C* a        = alpha = inverse Zeeman temperature         (double real) *
C* argp     = omp*beta                                   (double real) *
C* b        = alpha = inverse non-Zeeman temperature     (double real) *
C* y0       = omp*dpdt                                   (double real) *
C* dpdt     = (dP_I/dt)                                  (double real) *
C* dt       = time step                                  (double real) *
C* t(i)     = time                                       (double real) *
C*************************************************************************

      program main
      implicit none

      include "blocks.f"

      integer i
      double precision ppmax, dpp
      double precision rtbis
      double precision x0, args, a, argp, b
      double precision func2, y0, dpdt
      double precision t(0:500), p(0:500)
      double precision dt

      character outf*60

      outf = "./inout/outf.txt"
      open(30,file=outf)

      call init

C*************************************************************************
C* step nuclear spin polarization p(i) from 0 to 1 in nstep steps        *
C* determine the time t(i) needed to reach p(i)                          *
C*************************************************************************
      if (omm.lt.om0) then
        ppmax  = 1.0d0
      else if (omm.gt.om0) then
        ppmax  = -1.0d0
      endif
      dpp    = ppmax/nstep
      p(0)   = 0.0d0
      do i=1,nstep-1,1
        p(i) = p(i-1)+dpp
      end do
```

```
      t(0)  = 0.0d0
      do i=1,nstep-1,1
C*************************************************************************
C* rtbis solves x0 (=alpha*omega_0) for given p(i)                       *
C*************************************************************************
        x0   = rtbis(x1,x2,xacc,p(i-1))
        args = dlog((1.0d0+x0)/(1.0d0-x0))
        a    = args/om0
        argp = dlog((1.0d0+p(i-1))/(1.0d0-p(i-1)))
        b    = argp/omp
        y0   = func2(a,b)
        dpdt = y0/omp
        dt = -1.0d0/dpdt*dpp
        if (dt.lt.0.0d0) then
          goto 10
        endif
        t(i) = t(i-1)+dt

C*************************************************************************
C* write t(i) and p(i) to standard output and to file                    *
C*************************************************************************
        write (*,*) i, "  t =", t(i),"  P_I = ", p(i)
        write (30,100) t(i), p(i)
      end do
 10   continue

      close(30)

 100  format(f12.7,'    ',f12.7)

      end
```

Subroutine init:

```
C*************************************************************************
C*                              init.f                                   *
C*************************************************************************
C* read spectrum of TEMPO                                                *
C* determines centre of gravity and norm of the spectrum                 *
C* initiate some constants                                               *
C*************************************************************************
C* Input read from rdat:                                                 *
C* pi       = pi                                        (double real) *
C* hbar     = Planck's constant                         (double real) *
C* kb       = Boltzmann's constant                      (double real) *
C* gp       = gamma_I                                   (double real) *
C*************************************************************************
C* Input read from rdat and loaded to common blocks:                     *
C* e        = e                                         (double real) *
C* b0       = B_0 = static magnetic field (T)           (double real) *
C* tl       = lattice temperature                       (double real) *
C* omm      = microwave frequency (GHz)                 (double real) *
C* s0       = WT_1S = saturation factor at om0          (double real) *
C*            W = (1/2) omega_1S^2 g(omega)                            *
C* nstep    = number of steps in nuclear polarization   (integer)     *
C* x1       = lower limit Boltzmann factor electron spins (double real) *
C* x2       = upper limit Boltzmann factor electron spins (double real) *
C* xacc     = accuracy rtbis                            (double real) *
C*************************************************************************
C* Read from resr and loaded to common blocks:                           *
C* esr(1,i) = ESR spectrum, frequency                   (double real) *
C* esr(2,i) = ESR spectrum, intensity                   (double real) *
C*************************************************************************
```

```
C*************************************************************************
C* Loaded to common blocks:                                              *
C* norm      = normalization ESR spectrum                (double real) *
C* om0       = centre of gravity omega_0 of the ESR spectrum (double real) *
C* spec0     = amplitude spectrum at centre of gravity   (double real) *
C* s         = WT_1S = saturation factor at omm          (double real) *
C*             W = (1/2) omega_1S^2 g(omega)                           *
C* pl        = thermal equilibrium polarization          (double real) *
C* omp       = NMR frequency (GHz)                       (double real) *
C*************************************************************************
C* Internal parameters:                                                  *
C* ib0       = 10*B_0                                    (integer)     *
C* error     = dabs(omm-esr(1,i))                        (double real) *
C*             (needed to determine im)                                *
C* c         = conversion factor freq. vs. temp.         (double real) *
C* arg       = Boltzmann factor                          (double real) *
C*************************************************************************

      subroutine init

      implicit none

      include "blocks.f"

      integer ib0
      integer i, n
      double precision error
      double precision pi, hbar, kb, s0, gp, c, arg

      character rdat*60
      character resr*60

C*************************************************************************
C* read data file                                                        *
C*************************************************************************
      rdat = "./inout/inf.dat"
      open(10,file=rdat)
      call rnul(rdat,1)
      call rnul(rdat,2)
      call rnul(rdat,3)
      call rnul(rdat,4)
      call rnul(rdat,5)
      call rreal(rdat,6,e)
      call rreal(rdat,7,pi)
      call rreal(rdat,8,hbar)
      call rreal(rdat,9,kb)
      call rnul(rdat,10)
      call rnul(rdat,11)
      call rreal(rdat,12,b0)
      call rreal(rdat,13,t1)
      call rreal(rdat,14,omm)
      call rreal(rdat,15,s0)
      call rreal(rdat,16,gp)
      call rnul(rdat,17)
      call rnul(rdat,18)
      call rint(rdat,19,nstep)
      call rreal(rdat,20,x1)
      call rreal(rdat,21,x2)
      call rreal(rdat,22,xacc)
      close(10)
```

```
C*************************************************************************
C* load lineshape of TEMPO                                               *
C*************************************************************************

      ib0 = idnint(1.0d4*b0)
      if(ib0.eq.34000) then
        resr  = "./inout/esr-34000.txt"
      else
        write(*,*) "no input file for spectrum"
        stop
      endif
      open(20,file=resr)

      n = 1000
      do i=1,n,1
        read (20,*) esr(1,i), esr(2,i)
      end do
      close(20)

C*************************************************************************
C* Normalize spectrum                                                    *
C* Determine centre of gravity om0 of spectrum                           *
C* Determine amplitude spec0 spectrum at om0                             *
C* Determine microwave frequency                                         *
C* Determine saturation parameter at microwave frequency                 *
C*************************************************************************

      norm = 0.0d0
      do i=1,n,1
        norm = norm+esr(2,i)
      end do
      write(*,*) "norm     =", norm

      om0 = 0.0d0
      do i=1,n,1
        om0 = om0+esr(1,i)*esr(2,i)
      end do
      om0 = om0/norm
      write(*,*) "omega_0 =", om0, "GHz"

      do i=1,n,1
        error = dabs(om0-esr(1,i))
        if (error.lt.esr(1,i+1)-esr(1,i)) spec0 = esr(2,i)
      end do
      write(*,*) "spec0    =", spec0

      do i=1,n,1
        error = dabs(omm-esr(1,i))
        if (error.lt.0.5001d0*(esr(1,i+1)-esr(1,i))) im = i
      end do
      write(*,*) "im       =", im
      write(*,*) "omm      =", omm
      s = s0*esr(2,im)/spec0
      write(*,*) "WT_1S    =", s

C*************************************************************************
C* calculate thermal equilibrium polarization                            *
C*************************************************************************
      write(*,*) "T_L      =", t1, "K"
      write(*,*) "B_0      =", b0, "T"
      c    = kb/(2.0d0*pi*hbar)/1.0d9
      beta1 = 1.0d0/(c*t1)
      write(*,*) "beta_L   =", beta1, "GHz^-1"
      arg  = 0.5d0*om0/(c*t1)
      p1   = (e**arg-e**(-arg))/(e**arg+e**(-arg))
      write(*,*) "P_L      =", p1
```

```
C*************************************************************************
C* calculate NMR frequency                                               *
C*************************************************************************
      omp     = gp*b0
      write(*,*) "omega_I =", omp*1.0d3, "MHz"
      write(*,*)

      end
```

Subroutines reading lines of data file:

```
C*************************************************************************
C*                              rnul.f                                   *
C*************************************************************************
C* reads double empty line from iofile                                   *
C*************************************************************************
C* Internal variables:                                                   *
C* dummy    = character string for intermediate use                      *
C* linenr   = number of line being read                    (integer)    *
C* iofile   = character string denoting input file                       *
C*************************************************************************

      subroutine rnul(iofile,linenr)

      implicit none

      integer   linenr
      character iofile*60
      character dummy*80

      read (10,"(a80)",err=70) dummy

      go to 1000

   70 continue
      write (*,*) 'ERROR reading line ', linenr, ' of ', iofile

 1000 continue
      end



C*************************************************************************
C*                              rreal.f                                  *
C*************************************************************************
C* reads double real number from iofile                                  *
C*************************************************************************
C* Internal variables:                                                   *
C* dummy    = character string for intermediate use                      *
C* dummy1   = character string for intermediate use                      *
C* linenr   = number of line being read                    (integer)    *
C* pos      = number of column being read                  (integer)    *
C* iofile   = character string denoting input file                       *
C* realin   = real to be read from file                    (double real)*
C*************************************************************************

      subroutine rreal(iofile,linenr,realin)

      implicit none

      integer   linenr, pos
      character iofile*60
      character dummy*80
      character dummy1*80
      double precision realin
```

```
      read (10,"(a80)",err=70) dummy
      pos = index(dummy,':') + 2
      dummy1 = dummy(pos:)
      dummy = dummy1
      read (dummy,*,err=70) realin
      go to 1000

   70 continue
      write (*,*) 'ERROR reading line ', linenr, ' of ', iofile

 1000 continue
      end



C*************************************************************************
C*                          rint.f                                       *
C*************************************************************************
C* reads integer from iofile                                             *
C*************************************************************************
C* Internal variables:                                                   *
C* dummy    = character string for internal use                          *
C* dummy1   = character string for internal use                          *
C* linenr   = number of line being read                    (integer)     *
C* pos      = number of column being read                  (integer)     *
C* iofile   = chacter string denoting input file                         *
C  intin    = integer to be read from file                 (integer)     *
C*************************************************************************

      subroutine rint(iofile,linenr,intin)

      implicit none

      integer linenr, pos
      character iofile*60
      character dummy*80
      character dummy1*80
      integer intin

      read (10,'(a80)',err=70) dummy
      pos = index(dummy,':') + 2
      dummy1 = dummy(pos:)
      dummy = dummy1
      read (dummy,*,err=70) intin

      go to 1000

   70 continue
      write (*,*) 'ERROR reading line ', linenr, ' of ', iofile

 1000 continue
      end
```

Example data file read by subroutine init:

```
01..5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0....5
02. Input file for the calculation of the static solutions of
03. the generalized Provotorov equations
04.
05. Basic constants:
06. e                       : 2.7182818d0      (double real)
07. pi                      : 3.1415926d0      (double real)
08. hbar                    : 1.0545716d-34    (Js, double real)
09. kb                      : 1.3806504d-23    (J/K, double real)
10.
```

```
11. Data for calculation:
12. B_0                          : 3.4              (T, double real)
13. T_L                          : 1.50             (K, double real)
14. omega_m                      : 95.5828d0        (GHz, double real)
15. W(omega_0)T_1S               : 1.0d2            (double real)
16. gamma_I                      : 42.5775d-3       (GHz/T, double real)
17.
18. Parameters for calculation:
19. nstep                        : 250              (integer)
20. x1 = min. Boltzmann factor   : -0.9999d0        (double real)
21. x2 = max. Boltzmann factor   : 0.9999d0         (double real)
22. xacc = accuracy rtbis        : 1.0d-8           (double real)
```

Common blocks storing data read from data file:

```
C*************************************************************************
C*                                blocks.f                               *
C*************************************************************************
C* Contains common declarations                                          *
C*************************************************************************
C* Input read from rdat and loaded to common blocks:                     *
C* e         = e                                        (double real) *
C* b0        = B_0 = static magnetic field (T)          (double real) *
C* tl        = lattice temperature                      (double real) *
C* omm       = microwave frequency (GHz)                (double real) *
C* s0        = WT_1S = saturation factor at om0         (double real) *
C*             W = (1/2) omega_1S^2 g(omega)                           *
C* nstep     = number of steps in nuclear polarization  (integer)     *
C* x1        = lower limit Boltzmann factor electron spins (double real) *
C* x2        = upper limit Boltzmann factor electron spins (double real) *
C* xacc      = accuracy rtbis                           (double real) *
C*************************************************************************
C* Read from resr and loaded to common blocks:                           *
C* esr(1,i) = ESR spectrum, frequency                   (double real) *
C* esr(2,i) = ESR spectrum, intensity                   (double real) *
C*************************************************************************
C* Calculated in init and loaded to common blocks:                       *
C* norm      = normalization ESR spectrum               (double real) *
C* om0       = centre of gravity omega_0 of the ESR spectrum (double real) *
C* spec0     = amplitude spectrum at centre of gravity  (double real) *
C* s         = WT_1S = saturation factor at omm         (double real) *
C*             W = (1/2) omega_1S^2 g(omega)                           *
C* pl        = thermal equilibrium polarization         (double real) *
C* omp       = NMR frequency (GHz)                      (double real) *
C*************************************************************************

      integer im
      double precision e, b0, tl, omm, x1, x2, xacc
      integer nstep
      double precision esr(1:2,1:1000)
      double precision om0, norm, s, spec0, specm
      double precision betal, pl, omp



      common /block1/im
      common /block2/e, b0, tl, omm, x1, x2, xacc
      common /block3/nstep
      common /block4/esr
      common /block5/om0, norm, s, spec0, betal, pl, omp

      save /block1/,/block2/,/block3/,/block4/,/block5/
```

Subroutine rtbis:

This subroutine is modified from [2]. Copyrights are owned by Cambridge University Press, and only the headings showing the modifications are provided here.

```
C**********************************************************************
C*                              rtbis.f                               *
C**********************************************************************
C* Modified from Numerical Recipies in Fortran, 2nd Edition, 1994     *
C**********************************************************************
C* calculates root of function by the bisection method                *
C**********************************************************************
C* Input:                                                             *
C* funct      = function of a single variable              (double real) *
C* x1         = lower limit of the root                    (double real) *
C* x2         = upper limit of the root                    (double real) *
C* xacc       = required precision of the root             (double real) *
C**********************************************************************
C* Output:                                                            *
C* rtbis      = root                                       (double real) *
C**********************************************************************
C* Internal parameters:                                               *
C* j          = iteration number                           (integer)     *
C* jmax       = maximum number of iterations               (integer)     *
C**********************************************************************

      function rtbis(x1,x2,xacc,pp)

      implicit none

      double precision rtbis, x1, x2, xacc, func1, pp
      integer j, jmax
      double precision dx, f, fmid, xmid

      jmax = 40
```

Subroutine func1:

```
C**********************************************************************
C*                              func1.f                               *
C**********************************************************************
C* calculates the function of which rtbis determines the root         *
C**********************************************************************
C* Input:                                                             *
C* x          = variable = tanh(0.5*om0*alpha)             (double real) *
C* pp         = proton spin polarization                   (double real) *
C**********************************************************************
C* Output:                                                            *
C* func1      = value of the function                      (double real) *
C**********************************************************************
C* Parameters from common blocks:                                     *
C* e          = e                                          (double real) *
C* esr(1,i)   = ESR spectrum, frequency                    (double real) *
C* esr(2,i)   = ESR spectrum, intensity                    (double real) *
C* om0        = centre of gravity of ESR spectrum          (double real) *
C* s          = WT_1S = saturation factor at omm           (double real) *
C*              W = (1/2) omega_1S^2 g(omega)                          *
C* pl         = thermal equilibrium polarization           (double real) *
C* omp        = NMR frequency proton spins                 (double real) *
C**********************************************************************
```

```
C*************************************************************************
C* Internal parameters:                                                  *
C* args        = om0*alpha                                (double real) *
C* argp        = omp*beta                                 (double real) *
C* a           = alpha = inverse Zeeman temperature       (double real) *
C* b           = alpha = inverse non-Zeeman temperature   (double real) *
C* argm        = 0.5*(om0*alpha+(omm-om0)*beta)           (double real) *
C* pm          = tanh(argm)                               (double real) *
C* arg         = 0.5*(om*alpha+(omm-om)*beta)             (double real) *
C* tnh         = tanh(arg)                                (double real) *
C*************************************************************************

      function func1(x,pp)

      implicit none

      include "blocks.f"

      integer i, n
      double precision func1, x, pp
      double precision args, a, argp, b, argm, pm, arg, tnh, y

      n    = 1000
      args = dlog((1.0d0+x)/(1.0d0-x))
      a    = args/om0
      argp = dlog((1.0d0+pp)/(1.0d0-pp))
      b    = argp/omp

      argm = 0.5d0*(om0*a+(esr(1,im)-om0)*b)
      pm   = (e**argm-e**(-argm))/(e**argm+e**(-argm))
      y    = 0.0d0
      do i=1,n,1
        arg = 0.5d0*(om0*a+(esr(1,i)-om0)*b)
        tnh = (e**arg-e**(-arg))/(e**arg+e**(-arg))
        y   = y+esr(2,i)*tnh
      end do
      y     = y/norm
      y     = y-p1+2.0d0*s*pm
      func1 = y

      end
```

Subroutine func2:

```
C*************************************************************************
C*                                 func2.f                               *
C*************************************************************************
C* calculates the function of which rtbis determines the root            *
C*************************************************************************
C* Input:                                                                *
C* a           = alpha = inverse Zeeman temperature       (double real) *
C* b           = beta  = inverse non-Zeeman temperature   (double real) *
C*************************************************************************
C* Output:                                                               *
C* func2       = value of the function                    (double real) *
C*************************************************************************
C* Parameters from common blocks:                                        *
C* e           = e                                        (double real) *
C* esr(1,i)    = ESR spectrum, frequency                  (double real) *
C* esr(2,i)    = ESR spectrum, intensity                  (double real) *
C* om0         = centre of gravity of ESR spectrum        (double real) *
C* s           = WT_1S = saturation factor at omm         (double real) *
C*               W = (1/2) omega_1S^2 g(omega)                           *
C*************************************************************************
```

```
C*************************************************************************
C* Internal parameters:                                                  *
C* argm        = 0.5*(om0*alpha+(omm-om0)*beta)          (double real) *
C* pm          = tanh(argm)                              (double real) *
C* arg         = 0.5*(om*alpha+(omm-om)*beta)            (double real) *
C* tnh         = tanh(arg)                               (double real) *
C*************************************************************************

      function func2(a,b)

      implicit none

      include "blocks.f"

      integer i, n
      double precision func2, a, b
      double precision argm, arg, pm, tnh, y

      n    = 1000

      argm = 0.5d0*(om0*a+(esr(1,im)-om0)*b)
      pm   = (e**argm-e**(-argm))/(e**argm+e**(-argm))
      y    = 0.0d0
      do i=1,n,1
        arg = 0.5d0*(om0*a+(esr(1,i)-om0)*b)
        tnh = (e**arg-e**(-arg))/(e**arg+e**(-arg))
        y   = y+esr(2,i)*(esr(1,i)-om0)*tnh
      end do
      y    = y/norm
      y    = y+2.0d0*s*(esr(1,im)-om0)*pm
      func2 = y

      end
```