Computer Simulations in Solid-State NMR. II. Implementations for Static and Rotating Samples

MATTIAS EDÉN

*Physical Chemistry Division, Arrhenius Laboratory, Stockholm University, SE-*106 91 *Stockholm, Sweden*

ABSTRACT: In the first article in this series (Concepts Magn Reson Part A 17A:117–154, 2003), we outlined a theoretical framework for calculating solid-state NMR time-domain signals and frequency-domain spectra. This article explains how the theory may be implemented on a computer for simulating NMR signals from a single crystal in a static and rotating sample. The general building blocks of a computer program for numerically calculating NMR signals are discussed. Computer algorithms for carrying out the simulations are presented as flowcharts and implemented in C/C++ code. © 2003 Wiley Periodicals, Inc. Concepts Magn Reson Part A 18A: 1–23, 2003

KEY WORDS: solid-state NMR; numerical simulation; computer algorithm; static solids; magic-angle-spinning; dynamically inhomogeneous Hamiltonian; spherical tensor

INTRODUCTION

In Ref. (*1*), henceforth referred to as "Part I," we gave a theoretical formalism for how to calculate NMR time-domain signals and frequency-domain spectra in solid-state NMR. This article, which builds directly on Part I, discusses in more detail the various building blocks of a numerical simulation program and outlines how such a program may be written for simulation of NMR spectra of a single molecular orientation, i.e., a single crystal. The NMR problems treated here conform to so-called *dynamically inhomogeneous* cases as defined by Maricq and Waugh (*2*). As explained in Part I, these correspond to evolution of the nuclear spins under either a time-independent Hamiltonian or a time-dependent Hamiltonian that commutes with itself at all times. This applies, for instance, to isolated spins and heteronuclear spin systems under MAS conditions.

The algorithms are summarized as flowcharts that are suitable for implementation in most programming languages, such as FORTRAN (*3*), C (*4, 5*), C++ (*6*), and MATLAB (*7*). Some explicit examples of computer code in C/C++ are provided. The aim of this article is not to teach computer programming but merely to guide those already acquainted with elementary programming toward converting the theoretical NMR formalism introduced in Part I into a work-

Received 5 October 2001; revised 22 October 2002; accepted 20 November 2002

Correspondence to: M. Edén; E-mail: mattias@physc.su.se

Concepts in Magnetic Resonance Part A, Vol. 18A(1) 1–23 (2003)

Published online in Wiley InterScience (www.interscience.wiley.com). DOI 10.1002/cmr.a.10064

ing computer code, using well-established NMR simulation techniques. These are presented to aid generalization to more complex problems, as one of the purposes of this article is to help guide the reader toward articles that discuss implementations of more advanced and efficient simulations strategies.

This article is organized as follows: The next section summarizes the most important results and equations of Part I. The third section describes general aspects of the computer code given in the remainder of the article. As outlined in Part I, the three main parts of a numerical simulation program are (i) initialization of tensors and construction of matrices for the spin operators, (ii) the spin dynamics calculation, leading to the NMR time-domain signal, and (iii) postprocessing of the calculated data. Parts (i–iii) are discussed in detail. Finally, the last section briefly discusses extensions of the routines given here to more complicated spin dynamics calculations. We stress that this article deals only with calculation of the NMR response from a *single* molecular orientation. Simulations of time-domain signals and frequency-domain spectra of powders require powder averaging (*8–13*). This stage of the simulation will be included in an upcoming article, where the fragments of the computer code presented here will be merged together to form the final simulation programs.

SUMMARY OF PART I

Here we summarize the most important aspects of the theory that is to be implemented numerically.

The spin density operator $\hat{\rho}$ represents the state of an ensemble of equivalent nuclear spin systems. At thermal equilibrium in a strong magnetic field, the density operator is proportional to $\hat{\mathbf{I}}_z$, written for brevity (*14–19*)

$$\hat{\rho}_{\mathrm{eq}} = \hat{\mathbf{I}}_z \qquad [1]$$

To obtain a detectable NMR signal, it is necessary to create single-quantum coherences (1QC) by applying a radio-frequency pulse to the spin ensemble. Next, the signal acquisition starts at time point $t = 0$, with the density operator $\hat{\rho}(0)$ taken to be proportional to the spin operator $\hat{\mathbf{I}}_x$.

The NMR time signal corresponds to the expectation value of an observable operator $\hat{\mathbf{Q}}$, and may be calculated by forming the following trace (*14–19*):

$$s(t) = \langle \hat{\mathbf{Q}} \rangle(t) = \mathrm{Tr}\{\hat{\rho}(t)\hat{\mathbf{Q}}\} \qquad [2]$$

In the case of quadrature detection (which applies to all modern spectrometers), $\hat{\mathbf{Q}}$ is proportional to $\hat{\mathbf{I}}^+$ (*14–19*), which corresponds to detection of -1QC. The density operator $\hat{\rho}(t)$ evolves in time according to

$$\hat{\rho}(t) = \hat{\mathbf{U}}(t, t_0)\hat{\rho}(t_0)\hat{\mathbf{U}}(t, t_0)^\dagger \qquad [3]$$

The operator $\hat{\mathbf{U}}(t, t_0)$ is called the *propagator*, and transforms the density operator from the time point t_0 to t. It is related to the Hamiltonian through the *Schrödinger equation:*

$$\frac{d}{dt}\hat{\mathbf{U}}(t, t_0) = -i\hat{\mathbf{H}}(t)\hat{\mathbf{U}}(t, t_0) \qquad [4]$$

Equations [2], [3], and [4] are the fundamental expressions for calculating the time-domain signal and frequency-domain spectrum from any NMR experiment. Solving the Schrödinger equation provides the major challenge as it lacks an analytic solution in general and is the main reason why numerical simulation programs are needed. However, in Part I we confined the discussion to so-called dynamically inhomogeneous problems, where the spin Hamiltonian is either time independent or time periodic and self-commuting at all time points. In these cases, Eq. [4] may be solved analytically, and we showed that the NMR time-domain signal then has the following general form:

$$s(t) = \sum_{u,v=1}^{\mathcal{N}} \langle u|\hat{\rho}(0)|v\rangle\langle v|\hat{\mathbf{Q}}|u\rangle \exp\{i\Phi_{uv}(t, 0)\} \qquad [5]$$

The expression for the *dynamic phase* $\Phi_{uv}(t, 0)$ dictates the form of $s(t)$ and hence also the appearance of the NMR spectrum.

In the case of a *time-independent Hamiltonian*, $\Phi_{uv}(t, 0)$ was demonstrated to be given by the product $\omega_{uv}t$, with the frequency ω_{uv} obtained as the difference between the Hamiltonian eigenvalues ω_v and ω_u:

$$\omega_{uv} = \omega_v - \omega_u \qquad [6]$$

At the start of signal acquisition, defined as $t_0 = 0$, this gives the following expression for the dynamic phase:

$$\Phi_{uv}(t, 0) = \omega_{uv}t \qquad [7]$$

and Eq. [5] casts as

$$s(t) = \sum_{u,v=1}^{\mathcal{N}} a_{uv} \exp\{i\omega_{uv}t\} \quad [8]$$

The amplitude a_{uv} is a product of matrix elements of the initial density operator and observable, both expressed in the *eigenbasis* of the Hamiltonian:

$$a_{uv} = \langle u|\hat{\rho}(0)|v\rangle\langle v|\hat{\mathbf{Q}}|u\rangle \quad [9]$$

Upon Fourier transformation of Eq. [8], the frequency-domain spectrum from a spin system evolving under a time-independent Hamiltonian is obtained in the following form:

$$S(\omega) = \sum_{u,v=1}^{\mathcal{N}} a_{uv}\delta(\omega, \omega_{uv}) \quad [10]$$

It is represented by a set of $\mathcal{N}^2$ delta functions, each representing a spectral peak with amplitude a_{uv} positioned at the frequency coordinate $\omega = \omega_{uv}$.

For the second case with a Hamiltonian being a sum of *time-periodic* but mutually *commuting Hamiltonians*, we demonstrated that the dynamic phase is given by a sum of two terms

$$\Phi_{uv}(t, 0) = \omega_{uv}^{(0)} t + \Phi'_{uv}(t, 0) \quad [11]$$

where $\omega_{uv}^{(0)}$ is defined analogously to Eq. [6], with the distinction that it involves the difference between the $m = 0$ *Fourier components* of the Hamiltonian *eigenvalues:*

$$\omega_{uv}^{(0)} = \omega_v^{(0)} - \omega_u^{(0)} \quad [12]$$

with the Fourier components defined from the series

$$\omega_u(t) = \sum_{m=-2}^{2} \omega_u^{(m)} \exp\{im\omega_r t\} \quad [13]$$

We showed that under magic-angle-spinning conditions (MAS), each Hamiltonian eigenvalue Fourier component $\omega_u^{(0)}$ only depends on the isotropic (i.e., orientational-independent) parts of the spin interactions. The $m \neq 0$ Fourier components, on the other hand, contain the anisotropic (orientationally dependent) parts of the spin interactions. These, in turn, are contained in the dynamic phase $\Phi'_{uv}(t, 0)$ of Eq. [11]. $\Phi'_{uv}(t, 0)$ is periodically time dependent and given by

$$\Phi'_{uv}(t, 0) = (i\omega_r)^{-1} \times \sum_{m\neq 0} m^{-1}\{\omega_v^{(m)} - \omega_u^{(m)}\}(\exp\{im\omega_r t\} - 1) \quad [14]$$

The expression for the time-domain signal is finally obtained after Fourier expanding the function $\exp\{i\Phi_{uv}(t, 0)\}$ and inserting the result into the generic expression for the signal, Eq. [5]:

$$s(t) = \sum_{u,v=1}^{\mathcal{N}} \sum_{k=-\infty}^{\infty} a_{uv}^{(k)} \exp\{i\omega_{uv}^{(k)} t\} \quad [15]$$

The spectral amplitudes and frequencies are given by

$$a_{uv}^{(k)} = a_{uv} c_{uv}^{(k)} \quad [16]$$

and

$$\omega_{uv}^{(k)} = \omega_{uv}^{(0)} + k\omega_r \quad [17]$$

respectively, with $c_{uv}^{(k)}$ being a Fourier component defined in Eq. [I-208]. After Fourier transformation of the time-domain signal, the NMR spectrum takes the following form:

$$S(\omega) = \sum_{u,v=1}^{\mathcal{N}} \sum_{k=-\infty}^{\infty} a_{uv}^{(k)} \delta(\omega, \omega_{uv}^{(k)}) \quad [18]$$

As discussed in Part I, each pair of Hamiltonian eigenstates $\{|u\rangle, |v\rangle\}$ produces an NMR spectrum corresponding to a sideband manifold. A peak within a manifold is separated from its neighbor by the spinning frequency ω_r, according to the expression for the frequencies given by Eq. [17]. The total spectrum is the sum over all such manifolds.

In the following sections, we present computer routines for numerically evaluating these expressions.

GENERAL COMMENTS ON THE COMPUTER CODE

The mathematical formalism underlying the NMR theory must be converted into computer code in the numerical simulation program. There exist a large number of programming languages that are suitable; the most commonly used are FORTRAN (*3*), C (*4, 5*), C++ (*6*), and MATLAB (*7*). The fragments of C code presented in this article are relatively compact to illustrate the implementation of the theory presented above, using the same definitions of the NMR interactions, tensors, and trans-

formations as presented in Part I. These routines (typed in monotype font) are programmed mainly for the sake of clarity and generality rather than for efficiency of execution or implementational elegance. They should be regarded as *examples* of implementations rather than "ultimate solutions."

Each entity introduced in Part I will be represented in the computer program by suitable data type. For example, all operators [e.g., $\hat{\rho}(t)$, $\hat{\mathbf{Q}}$, and $\hat{\mathbf{H}}$] are represented by matrices. Second-rank irreducible spherical tensors correspond to (1×5) matrices, i.e., *row* vectors. The elements of matrices, as well as the components of tensors, are represented by complex numbers. Especially when using object-oriented languages such as C++ (*6*), the computer code may be formulated to closely resemble equations as they are formulated on paper.

The code was programmed using the GNU C++ (*20*) compiler, but should be compatible with many other compilers. The code is based on the C++ classes for handling complex numbers and matrices provided by GAMMA (*21, 22*), which is a general simulation environment comprising C++ code for a large variety of NMR simulations. The code in this article (and the following article in this series) does not, however, use the GAMMA library of routines for NMR simulations, but represents an extract of an alternative code. We refer to the GAMMA online manual (*22*) for details on the use of the complex/matrix routines used here. For brevity, only the code for the parts of the programs that are directly relevant for the NMR implementations are reproduced. Auxiliary routines, such as those for memory allocations of arrays and Fourier transformations are excluded: The complete source code may be found in Ref. (*23*).

Array Indexing

The source code makes use of the standard data types provided in C but employs additional routines for dynamic memory allocation (*3, 4, 6*), meaning that the memory needed for storing the elements of an array are reserved and released when needed during runtime of the program. Note, however, that older versions of FORTRAN (e.g., FORTRAN 77) do not support dynamic memory allocation. Array indexing follows the standard convention in C/C++ (*4, 6*). For example, a one dimensional array of double-precision floating point numbers may be declared as

```
double *double_list
```

and allocated to comprise 100 elements by the instruction

```
double_list=d_array1(100)
```

where "`d`" specifies "double precision" and "`1`" implies it is a one dimensional array. The first and last elements of this array correspond to `double_list[0]` and `double_list[99]`, respectively. Likewise, "`c_array1(5)`" allocates memory for five complex numbers. A similar indexing also applies to the matrices provided by GAMMA. For example, for a variable `M` (declared as `matrix M`), `M(3,1)` represents the matrix element M_{42}, i.e., the element of the fourth row and second column of the matrix **M**.

Second-rank irreducible spherical tensor elements are ordered as in Eq. [I-50], i.e., $(A_{22}, A_{21}, \ldots, A_{2-2})$. For example, for the variable `tensor` [declared as a (1×5) matrix] the element `tensor(0,3)` corresponds to the tensor component A_{2-1}.

Representing Euler Angles

For handling the Euler angles, a data type `euler` is defined through

```
struct euler {
//euler angle: {alpha,beta,gamma}
double alpha,beta,gamma;
};
```

A variable `angle` declared as type `euler` contains a triplet of Euler angles $\{\alpha, \beta, \gamma\}$. The numerical values of α, β, and γ are set by calling the routine `setEuler`:

```
void setEuler(euler &angle,double alpha,double beta,double gamma)
//constructs the euler angle triplet
//INPUT: (alpha,beta,gamma) in degrees
//OUTPUT: angle with the components in rads
{
 angle.alpha=(alpha*Pi/180.);
 angle.beta=(beta*Pi/180.);
 angle.gamma=(gamma*Pi/180.);
}
```

The constant `Pi` is defined to hold the value of $\pi = 3.141592$. Note that the routine inputs the Euler angle components in units of degrees and constructs the Euler angle variable with the components in units of radians. For example, `setEuler(angle, 120.,45.,90.)` assigns the value $\{\alpha, \beta, \gamma\} = \{2\pi/3, \pi/4, \pi/2\}$ to `angle`. The values of the angles α, β, and γ are accessed through `angle.alpha`, `angle.beta`, and `angle.gamma`, respectively.

The following three sections discuss the main parts of a numerical simulation program, as outlined earlier in Part I: *initialization*, *spin dynamics calculation*, and data *postprocessing*.

INITIAL STEPS OF NUMERICAL SIMULATIONS

Input Data

There are various ways to provide the input data. For example, it may be entered explicitly by the user when the program is executed. A more flexible approach, however, is to construct a separate "parameter file" for each simulation program. Such a parameter file should contain all necessary information for the calculations, and is subsequently read from disk at the start of the program.

One has to choose suitable units for the input data. In the examples here, the magnitudes of the spin interactions are input in units of Hz and the PAS tensor orientations in degrees. However, when the program is executed these values are directly converted into angular frequency units (rad s^{-1}) and radians for the magnitudes and orientations, respectively, as these are the relevant units when constructing the Hamiltonian.

Constructing Spatial Tensors

Next, it is necessary to construct the spatial tensors from the interaction parameters and represent them in a convenient form in the computer memory. Initialization of a second-rank tensor in its PAS may be programmed according to

```
matrix L2TensorSetup( double w_aniso, double asym )
// This routine may be used for constructing the second-rank spatial tensor of the anisotropic
// chemical shift, J, dipolar and first order quadrupolar interactions, provided that the
// following input is used for w_aniso:
// CS: w_aniso==delta_aniso*w0 (w0 is the Larmor frequency)
// Q: w_aniso==Qcc/( 2I(2I-1) ), where Qcc is the quadrupolar coupling constant
// D: w_aniso=2*.b_jk (Note factor of 2!)
{
 matrix tensor(1,5,0.);
 tensor(0,0)=-(asym*w_aniso)/2.;
 tensor(0,2)=sqrt(1.5)*w_aniso;
 tensor(0,4)=tensor(0,0);
 return tensor;
}
```

This routine may be used for constructing any second-rank spatial tensor (according to the expressions given in Table I-2 and I-3), where the anisotropy of the tensor, labeled `w_aniso`, corresponds to $\delta_{\text{aniso}}\omega_0$, J_{aniso}, $2b_{jk}$, and $\chi_Q/(2I(2I-1))$ for the CSA, J, dipolar and quadrupolar interactions respectively.

Because several interactions are normally involved in a simulation, it is beneficial to initially transform each tensor to a molecular frame (as discussed in Part I) and use this as input to the subsequent spin dynamics routine. This transformation (Eq. [I-89]) may easily be accomplished by a matrix multiplication as

```
tens_M=tens_P*WignerD(angle_PM)
```

where `tens_P` and `tens_M` is the tensor in the PAS and molecular frame, respectively, and the routine `WignerD2(angle_PM)` returns the matrix representation of the operator $\hat{\mathbf{D}}^{(2)}(\Omega_{PM})$. However, since such transformations are repeatedly used during the course of the simulation, the computational effort is reduced by first analytically evaluating the matrix–vector multiplication for a general case and using that in the computer implementation. The code `L2tensor_transform` below is an example of such a routine for transforming an irreducible second-rank tensor, represented as a 1×5 matrix, from a general reference frame F into another frame G, and may be used, for example, for the transformations $P \rightarrow M$ and $M \rightarrow R$.

```
double sqr(double x) {return x*x}; //calculate the square of the number x
matrix L2tensor_transform( matrix tensor, euler angle )
// transforms a tensor between two frames (G is the final one) using analytical equations (angles in rad)
// tensor elements are ordered as tensor(0,0)=A_22, tensor(0,2)=A_20
{
 double a,b,g,cosb,cos2b,sinb,sin2b,cosb_Pone,cosb_Mone;
 complex X22,X21,A20,c_X21,c_X22,exp_Ma;
 matrix A_G(1,5,0.);

 a=angle.alpha;b=angle.beta;g=angle.gamma; //get each of the Euler angle components

 cosb=cos(b);cos2b=cos(2.*b);cosb_Mone=cosb-1.;cosb_Pone=cosb+1.;
 sinb=sin(b);sin2b=sin(2.*b);exp_Ma=exp(-I*a);
 X22=( tensor(0,0)*sqr(exp_Ma) );c_X22=conj(X22);
 X21=( tensor(0,1)*exp_Ma );c_X21=conj(X21);
 A20=tensor(0,2);

 //construct m=2 component
 A_G(0,0)=exp(-2.*I*g)*( X22*sqr(cosb_Pone) + c_X22*sqr(cosb_Mone)
        + sqrt(3./2.)*A20*(1.-cos2b) + 2.*sinb*(X21*cosb_Pone + c_X21*cosb_Mone) )/4.;
 //m=1
 A_G(0,1)=exp(-I*g)*( -sinb*(X22*cosb_Pone + c_X22*cosb_Mone)
        + sqrt(3./2.)*A20*sin2b + X21*(cosb+cos2b) - c_X21*(cosb-cos2b) )/2.;
 //m=0
 A_G(0,2)=sqrt(3./8.)*( sqr(sinb)*( 2.*Re(X22) )
        + A20*(1.+3.*cos2b)/sqrt(6.) - sin2b*( 2.*Re(X21) ) );
 //use the symmetries of the components
 A_G(0,3)=-conj(A_G(0,1));A_G(0,4)=conj(A_G(0,0));

 return A_G;
}
```

With our indexing conventions for the tensor components, the elements $[A_{22}]^G$, $[A_{21}]^G$, and $[A_{20}]^G$ are represented *in the computer code* by `A_G(0,0)`, `A_G(0,1)`, and `A_G(0,2)`, respectively. Note that only these three components are calculated *explicitly* by the analytic transformations. The elements $[A_{2-1}]^G$ and $[A_{2-2}]^G$ are constructed directly from the others using the symmetries of the spherical tensor components expressed by Eq. [I-51], as discussed in Part I.

Matrix representations for the spin operators are also needed. As outlined in Part I, this is done preferably by taking successive direct products of smaller matrices. Because the implementations of such calculations are heavily dependent on the available library of matrix routines, we will not discuss them here.

SPIN DYNAMICS CALCULATION

Constructing the Hamiltonian

We continue by outlining a computer implementation for constructing spin Hamiltonians. This step is normally required repeatedly during the course of execution of the program.

According to Eq. [I-60], the Hamiltonian for an interaction Λ is calculated by multiplying the component $[A_{20}^{\Lambda}]^L$ (a real number) with its corresponding spin tensor operator $\hat{\mathbf{T}}_{20}^{\Lambda}$ (a matrix). It is now required to convert the vector representing the spatial tensor in either the PAS or the molecular frame into the $m = 0$ laboratory frame component, as expressed by Eqs. [I-91] and [I-95], respectively. These transformations may be implemented in various ways. Since only *one* component, $[A_{20}^{\Lambda}]^L$, is needed, it is convenient to use an analytic expression for this element, instead of first constructing the whole tensor and then extracting the $m = 0$ component. The computer implementation is

```
double L2tensorToLab_m0( matrix tensor, euler angle )
// transforms a 2nd rank tensor from the molecular (or PAS) frame to the lab-frame,
// using analytical equations (angles in rad), and returns the m=0 component.
{
 double a,b;
 complex X22,X21,exp_Ma;
 a=angle.alpha;b=angle.beta;
 exp_Ma=exp(-I*a);X22=( tensor(0,0)*sqr(exp_Ma) );X21=( tensor(0,1)*exp_Ma );
 return sqrt(3./2.)*( sqr(sin(b))*Re(X22)- sin(2.*b)*Re(X21)+
     (Re(tensor(0,2))*(1.+3.*cos(2.*b))/(2.*sqrt(6.)) );
}
```

The routine `L2TensorToLab_m0` is suitable for calculations involving static solids. On the other hand, for a rotating solid, requiring two transformations $M \rightarrow R \rightarrow L$, it is necessary to calculate the Fourier components $\omega_\Lambda^{(m)}$ of Eq. [I-105]. The routine `getSpatialFourierComponents` given below may then be employed (with an arbitrary angle β_{RL} as input):

```
matrix getSpatialFourierComponents( matrix tensor, euler MR,double betaRL )
//INPUT: a 2nd rank tensor in the molecular (or PAS) frame
//OUTPUT: the Fourier components of the tensor in the laboratory frame
}
 tensor=L2tensor_transform( tensor,MR );          //do M->R transformation
 //multiply with the d_m0(betaRL) elements
 tensor(0,0)=sqrt(3./8.)*sqr(sin(betaRL))*tensor(0,0);
 tensor(0,1)=-sqrt(3./8.)*sin(2*betaRL)*tensor(0,1);
 tensor(0,2)=( tensor(0,2)*(3.*sqr(cos(betaRL))-1.) )/2.;
 //use the symmetries of the components
 tensor(0,3)=conj(tensor(0,1));tensor(0,4)=conj(tensor(0,0));
 return tensor;
}
```

Note that first the molecular frame is transformed into the rotor frame using the routine `L2Tensor_transform` introduced earlier. Next, Eq. [I-105] is implemented by multiplying each rotor frame component $[A_{2m}^{\Lambda}]^R$ with the corresponding reduced Wigner element $d_{m0}^2(\beta_{RL})$. The routine returns a matrix containing the desired components, ordered as $\{\omega_\Lambda^{(2)}, \omega_\Lambda^{(1)}, \omega_\Lambda^{(0)}, \omega_\Lambda^{(-1)}, \omega_\Lambda^{(-2)}\}$. Analogously to the routine `L2Tensor_transform`, only the components $\omega_\Lambda^{(m)}$ with $m \geq 0$ are calculated explicitly; the remaining two components are obtained directly from the symmetry Eq. [I-106].

Propagators from Time-Independent Hamiltonians

In this section we describe how to numerically calculate the propagator $\hat{\mathbf{U}}(t, t_0)$ generated from a *time-independent* Hamiltonian over the interval $\tau = t - t_0$. As discussed in Part I, this represents the simplest possible spin dynamics scenario, and applies to a static solid in the absence of RF fields or in the presence of an RF field of constant amplitude, phase, and frequency. However, as we shall see soon, all more complicated time-dependent cases are solved by reducing the problem to a sequence of "*locally time-independent* Hamiltonians" by considering smaller time segments in the integration of the Schrödinger equation.

The gist of this calculation is to first apply Eq. [I-5], i.e., diagonalizing the Hamiltonian to obtain $\hat{\mathbf{H}}_{\text{diag}}$. Formally, this is expressed

$$\hat{\mathbf{H}}_{\text{diag}} = \hat{\mathbf{X}}^\dagger \hat{\mathbf{H}} \hat{\mathbf{X}} \qquad [19]$$

and is carried out in practice by using a "matrix diagonalization" routine from some library of matrix routines, for example, those provided in Refs. (*5, 21, 22*). Next, the propagator is obtained as the exponential of the operator $(-i\tau\hat{\mathbf{H}}_{\text{diag}})$ according to Eq. [I-21]:

$$\hat{\mathbf{U}}(t, t_0) \equiv \exp\{-i\tau\hat{\mathbf{H}}\} = \hat{\mathbf{X}} \exp\{-i\tau\hat{\mathbf{H}}_{\text{diag}}\}\hat{\mathbf{X}}^\dagger \qquad [20]$$

The procedure is carried out by the following steps:

1. Diagonalize the Hamiltonian $\hat{\mathbf{H}}$, i.e., find its eigenvalues and eigenvectors numerically.
2. Construct the diagonal eigenvalue matrix and the transformation matrix $\hat{\mathbf{X}}$ having the eigenvectors of $\hat{\mathbf{H}}$ as columns.
3. Calculate the matrix having the exponentiated Hamiltonian eigenvalues (ω_u) on the diagonal using Eq. [I-20]:

$$\exp\{-i\tau\hat{\mathbf{H}}_{\text{diag}}\} = \begin{pmatrix} \exp\{-i\tau\omega_1\} & 0 & 0 & \cdots & 0 \\ 0 & \exp\{-i\tau\omega_2\} & 0 & \cdots & 0 \\ 0 & 0 & \exp\{-i\tau\omega_3\} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \exp\{-i\tau\omega_N\} \end{pmatrix} \quad [21]$$

4. Form the product $\hat{\mathbf{X}}\exp\{-i\tau\hat{\mathbf{H}}_{\text{diag}}\}\hat{\mathbf{X}}^{\dagger}$ according to Eq. [20].

These steps are implemented in the C routine below, which uses a a time-independent Hamiltonian (`Ham`) and time segment τ (`tau`) as input parameters:

```
matrix propagator(const matrix &Ham, double tau)
// INPUT: a Hamiltonian (Ham)
// OUTPUT: the propagator exp(-I*Ham*tau)
{
 int n_states=Ham.rows();                    //get number of eigenstates
 matrix Ham_diag,X,expHam_diag(n_states,n_states,0.); //declare the necessary matrices

 diag(Ham,Ham_diag,X);                       //diagonalize the Hamiltonian
 for(int u=0;u<n_states;u++)                 //calculate exponential of Ham_diag
  expHam_diag(u,u)=exp( -I*Ham_diag.get(u,u)*tau );

 return X*expHam_diag*adjoint(X);            //return the propagator
}
```

Steps 1 and 2 above are carried out by the matrix routine `diag(Ham,Ham_diag,X)` of the GAMMA package. It takes the Hamiltonian matrix as input and returns the diagonal eigenvalue matrix (`Ham_diag`), as well as the transformation matrix (`X`). Next, step 3 is implemented as a `for` loop over all values u = 0, 1, . . . , `n_states`-1, where `n_states` is the number of Hamiltonian eigenstates. Finally, the matrix product in Eq. [20] is returned.

Propagators from Time-Periodic Hamiltonians

Here we discuss useful properties of propagators from a *time-periodic* Hamiltonian that fulfill the condition

$$\hat{\mathbf{H}}(t + NT) = \hat{\mathbf{H}}(t) \quad [22]$$

for integral N. For samples undergoing MAS, T is equal to the rotational period τ_r. Although the results discussed here *hold in general for any periodic Hamiltonian*, they are proven under the assumption of a dynamically inhomogeneous Hamiltonian, obeying $[\hat{\mathbf{H}}(t), \hat{\mathbf{H}}(t')] = 0$ for all time points t and t'. The consequence is that time ordering need not be taken into account (see page 139 of Part I). The corresponding proof in the general case of a dynamically *homogeneous* Hamiltonian are similar in spirit but require a more technical formalism; more complete discussions on the topic of periodic Hamiltonians may be found in Refs. (*18, 24, 25*) and applications to numerical problems are discussed in Refs. (*26–32*).

Consider a time segment $\tau_{ba} = t_b - t_a$ between two time points *within* the *first period* T, i.e., $0 \leq t_a \leq T$ and $0 \leq t_b \leq T$. For a dynamically inhomogeneous Hamiltonian, the general form of the propagator over the time interval τ_{ba} is given by Eq. [I-141]:

$$\hat{\mathbf{U}}(t_b, t_a) = \exp\left\{-i\int_{t_a}^{t_b} dt'\hat{\mathbf{H}}(t')\right\} \quad [23]$$

Then, consider two additional time points, $t_b + NT$ and $t_a + NT$, each obtained by time shifting t_b and t_a, respectively, by exactly N periods T. By definition, the propagator between the two time points $t = t_a + NT$ and $t = t_b + NT$ is given by

$$\hat{\mathbf{U}}(t_b + NT, t_a + NT) = \exp\left\{-i \int_{t_a+NT}^{t_b+NT} dt' \hat{\mathbf{H}}(t')\right\} \quad [24]$$

By making the following substitution of the integration variable

$$t = t' - NT \quad [25]$$

we obtain

$$\hat{\mathbf{U}}(t_b + NT, t_a + NT) = \exp\left\{-i \int_{t_a}^{t_b} dt \hat{\mathbf{H}}(t + NT)\right\} \quad [26]$$

From the periodicity of the Hamiltonian, $\mathbf{H}(t + NT) = \mathbf{H}(t)$, it then follows that (*25*)

$$\hat{\mathbf{U}}(t_b + NT, t_a + NT) = \hat{\mathbf{U}}(t_b, t_a) \quad [27]$$

Equation [27] states that the propagator $\hat{\mathbf{U}}(t_b, t_a)$ over the interval from $t = t_a$ to $t = t_b$ is the *same* as that over the interval from $t = t_a + NT$ and $t = t_b + NT$. In other words, the propagator over a given time segment has the *same periodicity* as the Hamiltonian (cf. Eqs. [27] and [22]).

Because the propagator is the exponential function of the *integral* of the Hamiltonian, it is instructive to relate the meaning of Eq. [27] to the following simple case: Assume a function $f(x)$ of the real variable x, fulfilling the periodicity $f(x + C) = f(x)$ for a given constant C. Now we select two values of the variable, x_a and x_b, both within the interval $0 \le x \le C$, and define the function $F(x_b, x_a)$ as the integral of $f(x)$ between x_a and x_b:

$$F(x_b, x_a) = \int_{x_a}^{x_b} dx f(x) \quad [28]$$

$F(x_b, x_a)$ may be interpreted as the *area* enclosed by the curve $y = f(x)$ and the line $y = 0$ between the values x_a and x_b. Since $f(x)$ is periodic, it follows that the area enclosed between the values $x = x_a$ and $x = x_b$ is the same as that between the two points $x_a + C$ and $x_b + C$ (obtained from a shift by C); hence, $F(x_b, x_a) = F(x_b + C, x_a + C)$. This is an analogous statement to the periodicity property of the propagator, expressed by Eq. [27].

Equation [27] may be used to demonstrate another useful property of propagators obtained from time-periodic Hamiltonians (*25*):

$$\hat{\mathbf{U}}(\tau + T, 0) = \hat{\mathbf{U}}(\tau, 0)\hat{\mathbf{U}}(T, 0) \quad [29]$$

Note that the right side of Eq. [29] only comprises time points within the first period of the Hamiltonian (i.e., $0 \le t \le T$). Consequently, the propagator obtained as the solution of the Schrödinger equation over *any* time interval may be expressed as a product of propagators, each involving *only time points within the first period T*. For instance, assume that we want to determine the propagator acting from $t = 0$ to $t = \tau + T$, i.e., over the time segment $\tau_{\text{tot}} = [0 \le t \le \tau + T]$. This corresponds to the operator on the left side of Eq. [29]. We assume that $\tau < T$. Following the procedure discussed on page 138 of Part I, we divide this interval into two smaller segments: $\tau_A = [0 \le t < T]$ and $\tau_B = [T \le t < \tau + T]$. The former interval corresponds to the *first* completed period T, while the latter interval starts at the *next* period. According to Eq. [I-121], we may express the propagator over τ_{tot} as the *time-ordered product* (*14–18*) of the propagators over the smaller time segments τ_A and τ_B:

$$\hat{\mathbf{U}}(\tau + T, 0) = \hat{\mathbf{U}}(\tau_B)\hat{\mathbf{U}}(\tau_A) \quad [30]$$

$$= \hat{\mathbf{U}}(\tau + T, T)\hat{\mathbf{U}}(T, 0) \quad [31]$$

However, from Eq. [27] it follows that $\hat{\mathbf{U}}(\tau + T, T) = \hat{\mathbf{U}}(\tau, 0)$, giving $\hat{\mathbf{U}}(\tau + T, 0) = \hat{\mathbf{U}}(\tau, 0)\hat{\mathbf{U}}(T, 0)$, which is the desired result, Eq. [29].

In this case, we considered time points not extending further than $2T$. However, by a straightforward extension of these arguments one can show that the accumulated propagator from $t = 0$ out to a time point $t = \tau + NT$ within the $N + 1$ period may be calculated according to

$$\hat{\mathbf{U}}(\tau + NT, 0) = \hat{\mathbf{U}}(\tau, 0)\{\hat{\mathbf{U}}(T, 0)\}^N \quad [32]$$

which involves raising the period propagator $\hat{\mathbf{U}}(T, 0)$ to the Nth power (and may be done using the results of Part I).

The consequences of Eqs. [27] and [32] for numerical simulations of rotating solids are that the Schrödinger equation needs only to be integrated over the

first rotational period, as it contains all information required for calculating the spin dynamics (i.e., propagating the density operator) out to arbitrarily large time points. Because the numerical diagonalization required to construct the propagators (see the previous section) are in general the most time-consuming step in the simulation, exploiting Eqs. [27] and [32] may give large savings in computational time. Such issues are discussed in more detail in Ref. (*32*).

Time-Domain Versus Frequency-Domain Simulations

There are two main approaches for carrying out spin dynamics calculations: One involves calculating the time signal as a set of discrete time points, thereby emulating the recording of the experimental NMR signal. The simulated signal is subsequently converted into a frequency-domain spectrum by a discrete Fourier transformation. This corresponds to a numerical implementation of Eq. [2] and is referred to as "simulation in the time domain." The other approach is to generate the spectrum directly by calculating a set of frequencies and amplitudes through Eq. [10] or [18]. This is referred to as a "frequency-domain calculation." The choice of a particular simulation technique depends on the given problem and preferences of the programmer. The frequency-domain calculations are often more efficient and also have the advantage of providing unlimited frequency resolution, whereas the time-domain approach is physically more intuitive and more straightforward to implement numerically. Below, we describe both the time- and frequency-domain approaches.

Time-Domain Calculation: "Direct Method"

The time-domain calculation outlined here is referred to as the "direct method" (*33*) and is the most widespread simulation technique. We first outline its most general form, applicable to any Hamiltonian, not necessarily self-commuting. Then, we discuss how it may be more efficiently evaluated in the dynamically inhomogeneous cases considered in this article.

Assume that the time span of the calculation is τ_{acq}, analogous to the experimental acquisition interval introduced on page 122 of Part I. Further, assume that the experiment starts at $t_0 = 0$ and that q samples of the NMR signal are calculated at the evenly spaced time points

$$t_j = j\tau = j\tau_{acq}/q, \quad j = 0, 1, 2, \ldots, (q-1) \quad [33]$$

Now, label the interval τ_j so that it includes all time points $t_{j-1} \leq t < t_j$. For q time points t_j there are consequently $(q-1)$ intervals $\tau_1, \tau_2, \ldots, \tau_{q-1}$, all of equal duration τ. Note that the interval τ has exactly the same role for the simulation as τ_{dwell} (Eq. [I-31]) has for the acquisition of the experimental time-domain signal.

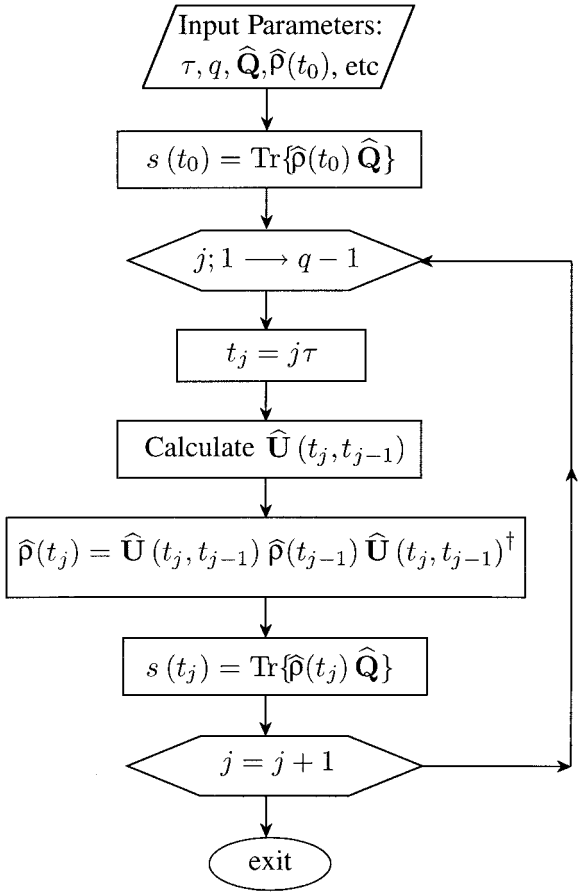

Figure 1 Flowchart for the direct method in the time domain. Relevant parameters, such as observable and initial density operators, number of sampled points (q), and the duration of the time segments (τ), are used to calculate the NMR time-domain signal. After the parameters are input, the program proceeds as follows: The signal at $t = t_0$ is calculated. Then follows a loop over the index j that extends over all sampled points $j = 0, 1, 2, \ldots, (q-1)$. For each value of j, the following consecutive steps are executed: (1) Each time point t_j is calculated from Eq. [33]; (2) each propagator $\hat{\mathbf{U}}(\tau_j)$ is estimated numerically as discussed in the text; (3) the density operator $\hat{\rho}(t_j)$ is obtained from Eq. [44]; (4) the time signal $s(t_j)$ is calculated (Eq. [45]). An explicit C code implementation of this algorithm is given in Example 1.

The "direct method" for obtaining the NMR time signal from a single molecular orientation proceeds as explained below and outlined in the flowchart of Fig. 1. C code implementations of the direct method are given in Examples 1 and 2.

```
void getSignalStatic( const matrix &Ham, const matrix &rho0, const matrix &Q,
                      complex* FID, int q, double dwell )
// time-domain calculation: "direct method" for static Hamiltonian
// INPUT: (time-independent) Hamiltonian (Ham), initial density operator
//        (rho0), observable (Q) and the time-resolution (dwell)
// OUTPUT: q points of the time-signal (FID)
{
 matrix U=propagator(Ham,dwell);      //get the propagator
 matrix U_adjoint=adjoint(U);         //calculate the inverse (adjoint) operator

 matrix rhoTemp=rho0;
 for(int j=0;j<q;j++) {
  FID[j]=trace( rhoTemp,Q );          //one point of the FID
  rhoTemp=( U*rhoTemp*U_adjoint );    //update the density operator
 }
}
```

Example 1 Routine `getSignalStatic` for generating the NMR time-domain signal by the direct method, assuming arbitrary initial density (`rho0`) and observable (`Q`) operators, and a time-*independent* Hamiltonian (`Ham`), according to the flowchart of Fig. 1. The routine outputs the signal (`FID`) as an array of `q` complex numbers, corresponding to the signals $s(t_j)$, with $j = 0, 1, \ldots, (q - 1)$.

Constructing the Propagator in the General Case. The first step toward obtaining the NMR signal involves integration of the Schrödinger equation (Eq. [4]) over each of the time intervals τ_j. This results in $q - 1$ propagators, $\hat{\mathbf{U}}(\tau_j) \equiv \hat{\mathbf{U}}(t_j, t_{j-1})$, with $1 \leq j \leq (q - 1)$. Each of these are in general obtained as described below.

Because the Hamiltonian is often time dependent during τ_j, one has to find an approximate form of the propagator $\hat{\mathbf{U}}(t_j, t_{j-1})$. This is done by dividing τ_j into $(q' - 1)$ smaller time segments τ', over each of which the Hamiltonian is constant (to a good approximation). Assume the propagator over the j'th time segment within the segment τ_j is to be estimated. This interval is denoted $\tau_{j,j'}$ and corresponds to the time points $t_{j-1} + (j' - 1)\tau' \leq t < t_{j-1} + j'\tau'$. Figure 2 depicts the relationship between the various time points, time intervals, and propagators in use.

The instantaneous Hamiltonian at the midpoint of the interval $\tau_{j,j'}$ is then diagonalized, i.e., its eigenvectors and eigenvalues are calculated numerically:

$$\hat{\mathbf{H}}_{\text{diag}}(t_{\text{mid}}) = \hat{\mathbf{X}}(t_{\text{mid}})^{\dagger} \cdot \hat{\mathbf{H}}(t_{\text{mid}}) \cdot \hat{\mathbf{X}}(t_{\text{mid}}),$$

$$t_{\text{mid}} = t_{j-1} + \left(j' - \frac{1}{2}\right)\tau' \quad [34]$$

Here, $\hat{\mathbf{H}}_{\text{diag}}$ is the diagonal matrix of the eigenvalues of the Hamiltonian and the columns of the matrix $\hat{\mathbf{X}}$ correspond to the normalized eigenvectors (implying that $\hat{\mathbf{X}} \cdot \hat{\mathbf{X}}^{\dagger} = \hat{1}$). Finally, the propagator acting over $\tau_{j,j'}$ is estimated by calculating the exponential of the diagonalized Hamiltonian

$$\hat{\mathbf{U}}(\tau_{j,j'}) = \hat{\mathbf{X}}(t_{\text{mid}}) \cdot \exp\{-i\hat{\mathbf{H}}_{\text{diag}}(t_{\text{mid}})\} \cdot \hat{\mathbf{X}}^{\dagger}(t_{\text{mid}}) \quad [35]$$

Note that Eqs. [34] and [35] correspond exactly to Eqs. [19] and [20], respectively: Provided that small enough time segments are considered (so that the Hamiltonian is nearly constant over each segment), the calculation of a propagator from a time-dependent Hamiltonian reduces to that of a series of time-independent ones.

Once each of the $(q' - 1)$ propagators $\hat{\mathbf{U}}(\tau_{j,j'})$ are calculated, the construction of the "segment propagator" $\hat{\mathbf{U}}(\tau_j)$ follows by forming a product of the "small-segment" propagators $\hat{\mathbf{U}}(\tau_{j,j'})$

$$\hat{\mathbf{U}}(\tau_j) \equiv \hat{\mathbf{U}}(t_j, t_{j-1}) = \hat{\mathbf{U}}(\tau_{j,q'-1}) \cdot \cdots \cdot \hat{\mathbf{U}}(\tau_{j,2}) \cdot \hat{\mathbf{U}}(\tau_{j,1}) \quad [36]$$

Note that the product is time ordered such that propagators involving *later* time points appear to the *left*, i.e., the index j' increases from right to left (see Part I). This is necessary when the Hamiltonian does not commute with itself at two different time points, i.e., for dynamically homogeneous problems. For dynamically inhomogeneous cases, however, time ordering need not be considered.

In the general case, this procedure involves applying Eq. [35] $(q - 1)(q' - 1)$ times to obtain all $(q - 1)$ propagators $\hat{\mathbf{U}}(t_j, t_{j-1})$. From this set of operators, it is also possible to calculate the "accumulated propagators" (shown at the top of Fig. 2), acting

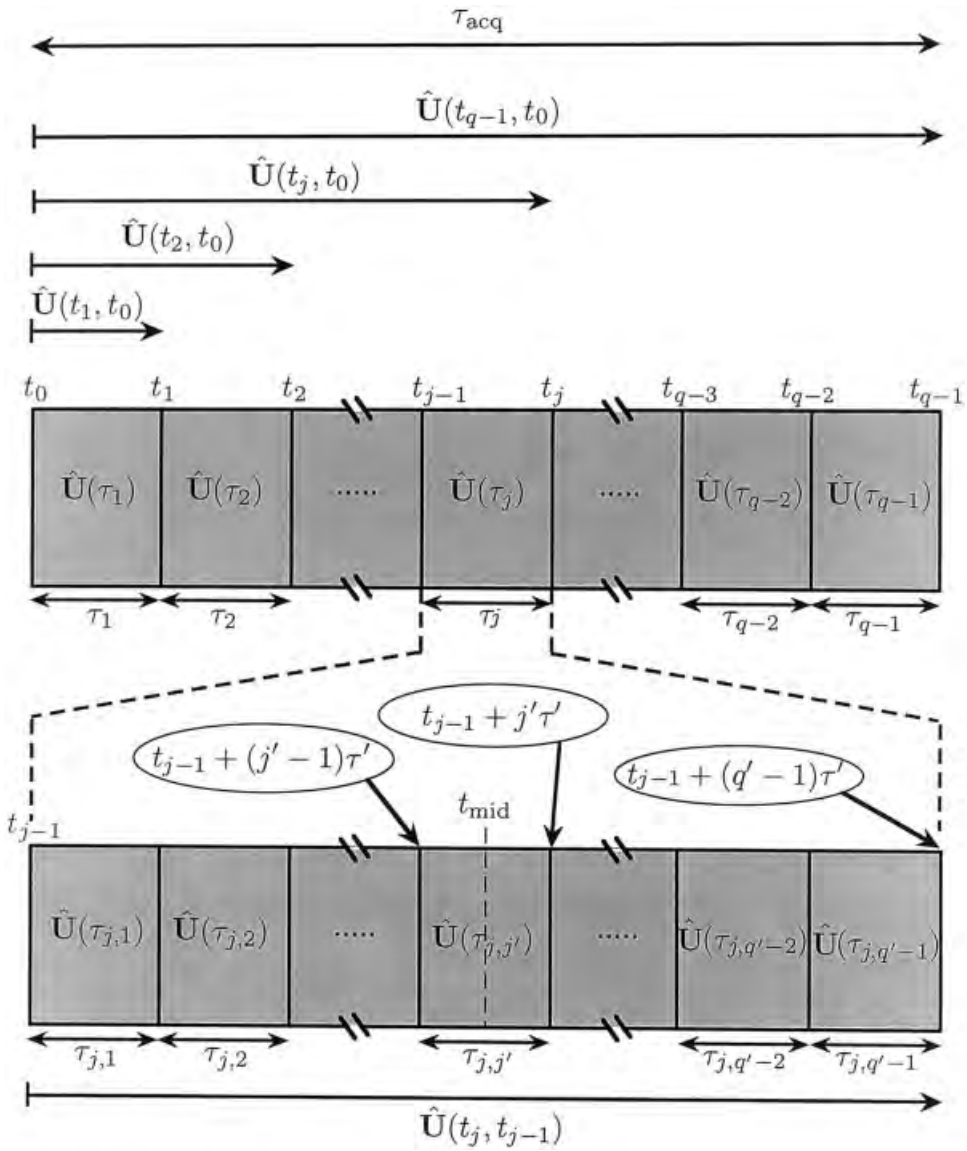

Figure 2 Relationship between time points and propagators over time segments used in the "direct time domain" calculation. The total time-span τ_{acq}, over which the density operator is to be propagated, is divided into $(q - 1)$ segments τ_j (Eq. [33]). These time segments are indicated beneath the top block of shaded rectangles. Each block represents a propagator. For example, $\hat{\mathbf{U}}(\tau_j)$ is calculated as follows: First, each segment τ_j is divided up further into $(q' - 1)$ segments $\tau_{j,j'}$, with q' chosen such that the Hamiltonian is to a good approximation time independent over $\tau_{j,j'}$. The bottom block of shaded rectangles illustrates this division for *one* segment τ_j. Next, the "small-segment" propagator $\hat{\mathbf{U}}(\tau_{j,j'})$ is estimated from the Hamiltonian (evaluated at the midpoint of the interval $\tau_{j,j'}$) according to Eq. [35]. This calculation is repeated for all intervals $\tau_{j,j'}$, and the propagator $\hat{\mathbf{U}}(\tau_j) \equiv \hat{\mathbf{U}}(t_j, t_{j-1})$ over the segment τ_j is calculated from a time-ordered product (Eq. [36]). These calculations are repeated for all segments τ_j. Once all propagators $\hat{\mathbf{U}}(\tau_j)$ $(j = 0, 1, \ldots, (q - 1))$ are calculated, they may be used to form a set of "accumulated propagators" (indicated by *arrows* in the top of the figure) according to Eq. [37].

```
void getSignalPeriodic( matrix *Ham_list, const matrix &rho0, const matrix &Q,
                        double T,int n,complex* FID, int q)
// time-domain calculation: "direct method" for a time-periodic Hamiltonian with
// the period T divided into n time-segments. The parameters should fulfill
// T/n*m = tau_acq, where tau_acq is the acquisition time, and m is any positive integer
// INPUT: initial density operator (rho0), and observable (Q), and a list of
//        Hamiltonians (Ham_list), where Ham_list[p] is the value of the Hamiltonian at
//        the midpoint of the pth segment: (p+0.5)*T/n,
// OUTPUT: q samplings of the time-signal (FID) using a dwell-time = tau_acq/q
{
 int n_states=Ham_list[0].rows();                    //get number of eigenstates
 matrix *U_list;
 U_list=m_array1(n,n_states,n_states);               //allocate memory for n propagators
 for(int p=0;p<n;p++)
  U_list[p]=propagator( Ham_list[p],T/double(n) );   //construct list of propagators

 matrix rhoTemp=rho0;
 for(int j=0;j<q;j++) {
  FID[j]=trace( rhoTemp,Q );                         //one point of the FID
  //update rho using the current propagator of U_list (a%b corresponds to a MOD b)
  rhoTemp=( U_list[j%n]*rhoTemp*adjoint(U_list[j%n]) );
 }
 del_m_array1(U_list);                               //free allocated memory
}
```

Example 2 Routine `getSignalPeriodic` for generating the NMR time-domain signal by the direct method, assuming a time-periodic Hamiltonian with period T. For a rotating sample, $T \equiv \tau_r$. The code is implemented according to the flowchart in Fig. 1. $\hat{\mathbf{H}}(t)$ is assumed to be approximated as a list (`Ham_list`) of n piece-wise time-independent Hamiltonians, sampled at equal time steps over the interval $[t_0, t_0 + T]$. t_0 is the initial time point and `Ham_list[p]` is the Hamiltonian at the midpoint of the pth segment: $t = t_0 + (p + 0.5)T/n$.

from $t = 0$ out to $t = t_j$, by again forming a time-ordered product:

$$\hat{\mathbf{U}}(t_j, 0) = \hat{\mathbf{U}}(\tau_j) \cdot \cdots \cdot \hat{\mathbf{U}}(\tau_2) \cdot \hat{\mathbf{U}}(\tau_1) \quad [37]$$

$$= \hat{\mathbf{U}}(t_j, t_{j-1}) \cdot \cdots \cdot \hat{\mathbf{U}}(t_2, t_1) \cdot \hat{\mathbf{U}}(t_1, 0) \quad [38]$$

Special Case: Time-Periodic Hamiltonians. As discussed earlier, it is possible to reduce the computational effort significantly if the Hamiltonian is either *time independent* or *periodic* in time. If the Hamiltonian is *periodic*, $\hat{\mathbf{H}}(t + \tau_r) = \hat{\mathbf{H}}(t)$ (applying to experiments with rotating solids), it is sufficient to construct the propagators over *only* the *first* rotational period τ_r, as they carry all the relevant information about the spin dynamics at *all* times.

Assume that $n + 1$ time points are sampled such that

$$t_p = p\tau_r/n, \quad p = 0, 1, 2, \ldots, n \quad [39]$$

Note the range of p, which includes both $p = 0$ and $p = n$. The sampling of the $n + 1$ time points results in a division of the interval $0 \leq t \leq \tau_r$ into n time segments of equal duration τ_r/n. This should be small enough that the Hamiltonian may be approximated as time independent over each segment, as discussed above. Note that the interval τ_r/n is *not necessarily* related to the "dwell time" τ defined in Eq. [33] and used for sampling the NMR time-domain signal. However, to fully exploit the periodicity of the Hamiltonian, it is desirable to synchronize the intervals τ_r/n and τ as follows:

$$\tau = n_1\tau_r/n \quad [40]$$

where n_1 is any positive integer. This means that τ should be an integral multiple of τ_r/n. An example of this "synchronization procedure" with $q = 6$, $n = 4$, and $n_1 = 2$ is given in Fig. 3.

Next, all n propagators $\hat{\mathbf{U}}(t_p, t_{p-1})$ for $1 \leq p \leq n$ are calculated from Eq. [35]. As follows from Eq. [27], the propagators over the time segments τ_r/n are periodic with τ_r, i.e., periodic with the parameter n according to

$$\hat{\mathbf{U}}(t_{p+n}, t_{p+n-1}) = \hat{\mathbf{U}}(t_p, t_{p-1}) \quad [41]$$

By applying Eq. [29] it follows that the propagator acting from $t = 0$ to $t = t_{p+n}$ may be calculated as a product involving the propagator from $t = 0$ to $t = t_p$ and the propagator from $t = 0$ to $t = t_n$:

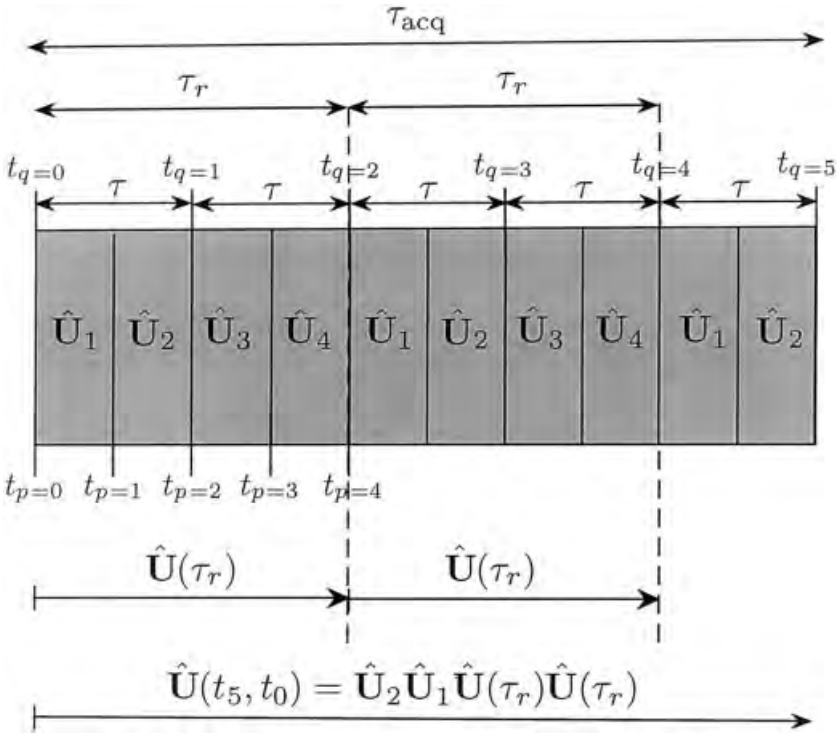

Figure 3 Relationship between propagators over various time segments for a periodic Hamiltonian in a rotating solid, i.e., a Hamiltonian obeying $\hat{\mathbf{H}}(t + \tau_r) = \hat{\mathbf{H}}(t)$. In this example, the acquisition interval τ_{acq} is divided into $q = 6$ smaller intervals τ and the rotational period τ_r is divided into $n = 4$ smaller segments $\tau_r/4$. The index q indicates points where the time-domain signal $s(t_q)$ is to be sampled, whereas the index p indicates the time points confining each segment of τ_r. Note that in the present case τ_{acq} spans exactly 2.5 rotational periods. The given divisions of smaller intervals of τ_{acq} and τ_r allows exploiting the periodicity of the Hamiltonian in the numerical simulation to avoid unnecessary calculations. This means that the Schrödinger equation need only be integrated from t_0 to $t_0 + \tau_r$, resulting in the four propagators $\hat{\mathbf{U}}_p$. The propagator over each interval τ is obtained from a time-ordered product of $\hat{\mathbf{U}}_p$ using Eq. [38]. This is exemplified by the calculation of the propagator $\hat{\mathbf{U}}(t_5, t_0)$, as indicated in the bottom of the figure.

$$\hat{\mathbf{U}}(t_{p+n}, 0) = \hat{\mathbf{U}}(t_p, 0)\hat{\mathbf{U}}(t_n, 0) \qquad [42]$$

Provided that the "synchronization condition" Eq. [40] is met, the propagator involving *any* time point being an integer multiple of the time segment τ_r/n may be constructed by using propagators involving time points *only* within the *first* rotational period $0 \leq t \leq \tau_r$. Consequently, the Schrödinger equation need only be integrated between $t = 0$ and $t = \tau_r$ to allow the calculation of $s(j\tau)$ for arbitrarily large j.

Propagation of the Density Operator. The density operator at time point t_j may be calculated either from $\hat{\rho}(0)$ by using the propagator acting from $t = 0$ to $t = t_j$

$$\hat{\rho}(t_j) = \hat{\mathbf{U}}(t_j, 0)\hat{\rho}(0)\hat{\mathbf{U}}(t_j, 0)^{\dagger} \qquad [43]$$

or recursively from the density operator at the previous time point $\hat{\rho}(t_{j-1})$

$$\hat{\rho}(t_j) = \hat{\mathbf{U}}(t_j, t_{j-1})\hat{\rho}(t_{j-1})\hat{\mathbf{U}}(t_j, t_{j-1})^{\dagger} \qquad [44]$$

by using the segment propagator acting from $t = t_{j-1}$ to $t = t_j$. In the flowchart of Fig. 1 and the C code implementations of Examples 1 and 2, the latter approach is employed. The density operator propagation is repeated for all q time points, which is in practice implemented as a loop over the variable j (see Fig. 1 as well as Examples 1 and 2).

Time-Domain Signal and Spectrum. The NMR time-domain signal at $t = t_j$ is obtained from Eq. [2]

$$s(t) = \langle \hat{\mathbf{Q}} \rangle(t) = \mathrm{Tr}\{\hat{\rho}(t)\hat{\mathbf{Q}}\} \qquad [45]$$

and is preferably implemented as a pair-wise multiplication of the elements of the two operators (Eq. [I-134]):

$$s(t_j) = \sum_{a,b=1}^{\mathcal{N}} (\hat{\rho}(t_j))_{ab}(\hat{\mathbf{Q}})_{ba} \qquad [46]$$

This is computationally more efficient than directly using Eq. [45] (where the trace is taken *after* forming the product) because the redundant off-diagonal elements of the matrix product $\hat{\rho}(t_j)\hat{\mathbf{Q}}$ are then not evaluated (*32*).

This calculation is repeated for all q time points. Finally, Fourier transformation of the set $\{s(t_j)\}$ produces a set of spectral amplitudes $\{a_j(\omega_j)\}$. The corresponding frequency coordinates are converted into units of Hz: They are distributed evenly over a frequency range $\{-1/(2\tau) + \upsilon_{\text{res}}, 1/(2\tau)\}$ with the resolution $\upsilon_{\text{res}} = \omega_{\text{res}}/2\pi = \tau_{\text{acq}}^{-1} = (q\tau)^{-1}$, as discussed in Part I. Note that the larger the acquisition interval the finer the frequency resolution.

Frequency-Domain Simulation

As explained in Part I, Hamiltonians being either time independent or time periodic and self-commuting gives rise to spin dynamics that may be solved analytically. In the time-independent case, the expression for the frequency-domain spectrum (Eq. [10]) was derived. Likewise, for the periodic Hamiltonian, the corresponding expression was shown to correspond to Eq. [18]. This section outlines how the mathematical procedure may be converted into computer code to numerically calculate the NMR frequency-domain spectrum of a single molecular orientation. Additional information about such calculations are given in Ref. (*32*).

Time-Independent Hamiltonian. Frequency-domain calculations of the NMR spectrum for the case of a time-independent Hamiltonian comprise the following steps:

1. Obtain the sets of $\mathcal{N}$ eigenvalues ω_u and the transformation matrix $\hat{\mathbf{X}}$ that diagonalizes the Hamiltonian (Eq. [19]).
2. Transform the observable and initial density operator to the eigenbasis of the Hamiltonian:

$$\hat{\mathbf{O}} \rightarrow \hat{\mathbf{X}}^{\dagger}\hat{\mathbf{O}}\hat{\mathbf{X}} \qquad [47]$$

   The matrix elements in the eigenbasis are given by $\langle u|\hat{\mathbf{O}}|v\rangle \equiv (\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{O}}\hat{\mathbf{X}})_{uv}$.
3. Construct a set of $\mathcal{N}^2$ amplitudes $a_{uv} = \langle u|\hat{\rho}(0)|v\rangle\langle v|\hat{\mathbf{Q}}|u\rangle$ (Eq. [9]).
4. Calculate the eigenvalue differences ω_{uv} for each pair of eigenstates (Eq. [6]). These correspond to the frequency positions of the peaks in the spectrum.
5. The spectrum is specified by the set of coordinates $\{\omega_{uv}, a_{uv}\}$, $u, v = 1, 2, \ldots, \mathcal{N}$.

This algorithm is represented as a detailed flowchart in Fig. 4 and implemented in C in Example 3. The flowchart corresponds to the following events: First, the components of each spin and spatial tensor are constructed from the input data, the Hamiltonian is diagonalized (step 1 above), and the observable and initial density operators are transformed (step 2). Then, the program enters two nested loops over each of the eigenvalue indices u and v. For each pair (u, v), the following steps are carried out: The amplitude a_{uv} is constructed (step 3) and the eigenvalue differences ω_{uv} are formed (step 4). Because there are $\mathcal{N}$ eigenvalues (in Example 3, $\mathcal{N}$ is represented by the constant `n_states`), and two nested loops, there are in principle $\mathcal{N}^2$ distinct amplitudes and frequencies. However, in practice, many of these amplitudes are zero due to the sparseness of the matrix representations of the initial density operator and observable operator. This results in zero values of the products $\langle u|\hat{\rho}(0)|v\rangle\langle v|\hat{\mathbf{Q}}|u\rangle$ for many transitions. To avoid using these insignificant spectral peaks, the magnitude of each amplitude a_{uv} is compared with a certain threshold value (taken as 10^{-8} in the code in Example 3). If a_{uv} is smaller, the frequency–amplitude pair is rejected; otherwise, each amplitude and frequency is stored separately in arrays (`amp_list` and `freq_list`, respectively). A counter, `n_transitions`, is employed to keep track of the number of "significant amplitudes."

Repeating the steps above for all $\mathcal{N}^2$ pairs of eigenstates results in two lists, each of which comprise `n_transitions` frequencies ω_{uv} and amplitudes a_{uv}, respectively. The resulting spectrum is consequently represented by a "stick spectrum" (histogram) of delta functions.

Time-Periodic Self-Commuting Hamiltonian. This section discusses the numerical implementation of the formal calculations in Part I for generating the frequency-domain spectrum from a spin system evolving under a time-periodic Hamiltonian, that involves several interactions with commuting spin parts.

The calculation of a spectrum containing a manifold of n sidebands (for each pair of Hamiltonian eigenstates) requires that the rotational period τ_r is divided into an even number (n) of time segments. Then, $n + 1$ time points are sampled according to Eq. [39]. The procedure outlined below is a special case of the COMPUTE method (*27*) for time-periodic Hamiltonians, here applied to the situation of several commuting interactions. The detailed algorithm is il-

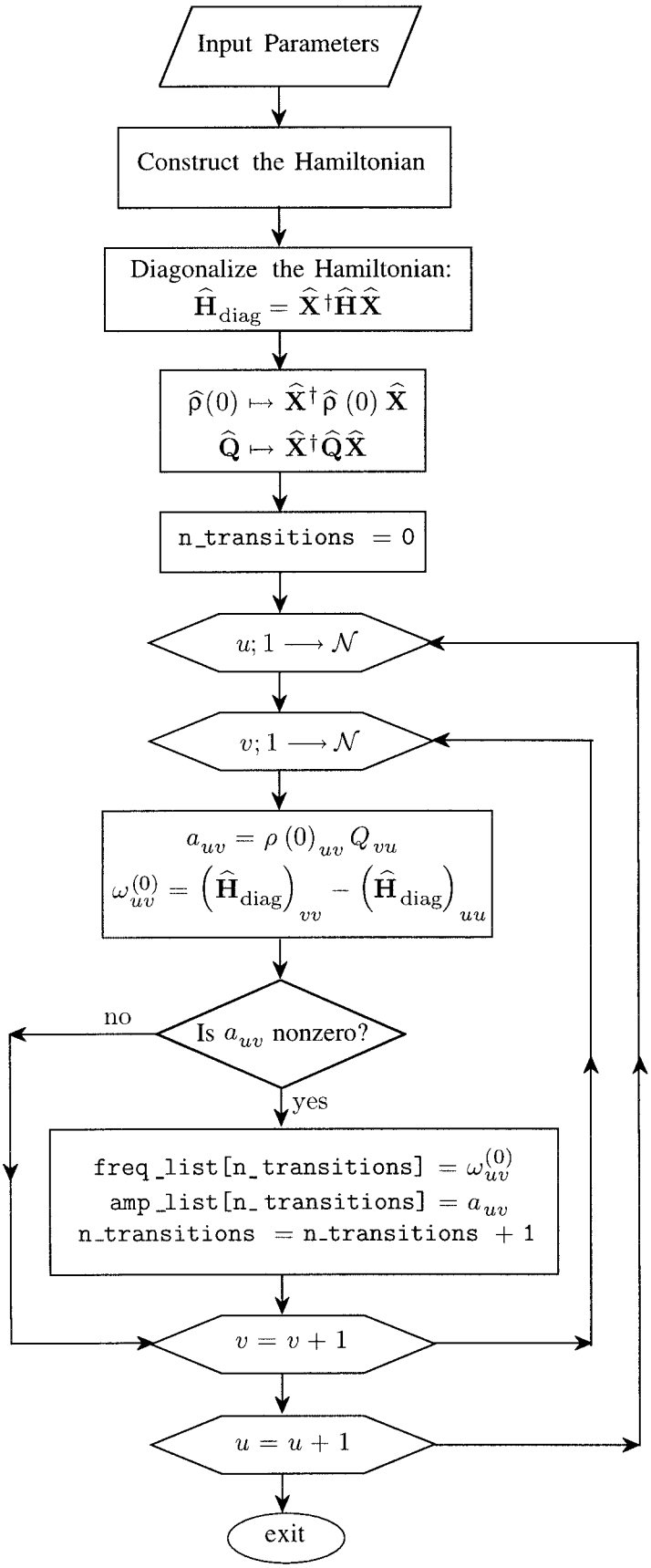

Figure 4 Flowchart for a frequency-domain calculation (explained in the *Time Dependent Hamiltonian* section) of the NMR spectrum using arbitrary initial density and observable operators and a *time-independent* Hamiltonian.

```
void getSpectrumStatic( const matrix &Ham, const matrix &rho0, const matrix &Q,
                        double* freq_list, complex* amp_list,
                        int &n_transitions )
// frequency-domain calculation for static Hamiltonian
// INPUT: time-indpendent Hamiltonian (Ham), initial density operator (rho0) and observable (Q)
// OUTPUT: lists of frequencies and amplitudes for those transitions having nonzero
// amplitude: in total n_transitions data-points in the stick-spectrum
{
 int n_states=Ham.rows();
 complex a_uv;
 matrix rho0_eigenBasis,Q_eigenBasis,Ham_diag,X,X_adjoint;

 diag(Ham,Ham_diag,X);              //diagonalize the Hamiltonian
 Ham_diag /= (2.*Pi);               //convert to units of Hz
 X_adjoint=adjoint(X);
 rho0_eigenBasis=X_adjoint*rho0*X; //transform density operator to the Hamiltonian eigenbasis
 Q_eigenBasis=X_adjoint*Q*X;        //transform observable operator to the Hamiltonian eigenbasis

 n_transitions=0;
 for(int u=0;u<n_states;u++) {     //loop over u index
  for(int v=0;v<n_states;v++) {    //loop over v index
   a_uv=( rho0_eigenBasis.get(u,v)*Q_eigenBasis.get(v, u) ); //get the amplitude a_uv
   if ( norm(a_uv)>1e-8 ) {        //place the frequencies and amplitudes
    freq_list[n_transitions]=( (Ham_diag.get(v,v)).re) - ((Ham_diag.get(u,u)).re );
    amp_list[n_transitions++]=a_uv;
   }
  } //closes v loop over states
 } //closes u loop over states
}
```

Example 3 Routine `getSpectrumStatic` for calculating the frequency-domain NMR spectrum generated from arbitrary initial density operator (`rho0`) and observable (`Q`) and assuming a time-independent Hamiltonian. The code is implemented according to the flowchart in Fig. 4 and returns two arrays containing the NMR spectral frequencies and amplitudes, respectively. The array of amplitudes (`amp_list`) should be allocated to contain `n_states`×`n_states` elements. Note that only those amplitudes that are larger than a given threshold value (in this case, 10^{-8}) are retained. The total number of nonzero amplitudes is contained in the parameter `n_transitions`, the value of which is returned when exiting `getSpectrumStatic`.

lustrated as a flowchart in Fig. 5: Its basic structure is similar to that of the time-independent calculation in Fig. 4, but the time-dependent calculations are somewhat more elaborate. The upper left panel of Fig. 5 comprise the following steps:

1. Obtain the transformation matrix $\hat{\mathbf{X}}$.
2. Transform the observable and initial density operator to the eigenbasis of the Hamiltonian (Eq. [47]). In most dynamically inhomogeneous cases, the high-field Hamiltonian commutes with $\hat{\mathbf{I}}_z$, which means that it is already diagonal in the Zeeman basis, and no diagonalization is necessary. Steps 1 and 2 above may then be omitted.

   Next follows a loop over all interactions, involving steps 3 and 4 below.

3. Calculate the Fourier components $\omega_\Lambda^{(m)}$ of each interaction Λ (Eq. [I-105]), and transform the corresponding spin operator $\hat{\mathbf{T}}_\Lambda$ to the eigenbasis of the Hamiltonian.

   Then follows the main calculation loop over the indices u and v, exactly as in the time-independent case: The magnitude of each amplitude a_{uv} is checked and, only if it is nonzero, the subsequent "inner loops" are executed. These are shown in the right panel of Fig. 5 and comprise the following steps.

4. Construct the five Fourier components $\omega_u^{(m)}$ of each eigenvalue from Eq. [I-194].
5. Compute the dynamic phases $\Phi'_{uv}(t_p, 0)$ for the n time segments by Eq. [14]. This corresponds to the loop over p, with each time point t_p obtained from Eq. [39].
6. Obtain the Fourier coefficients $c_{uv}^{(k)}$ by applying a discrete Fourier transform (*5*) (as explained in Part I) to the set of phases. This is carried out according to

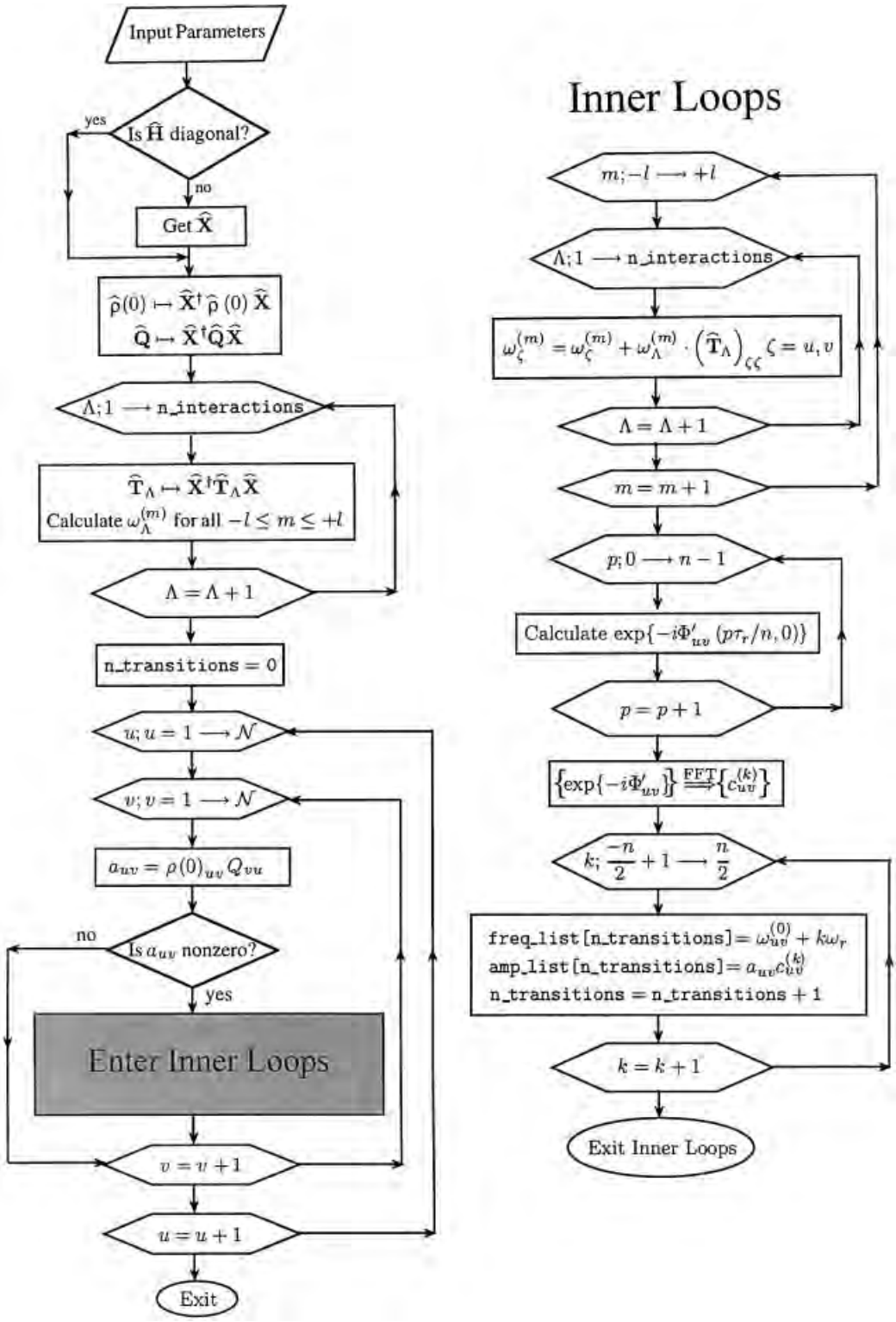

Figure 5 Flowchart for a frequency-domain calculation (explained in the *Time Periodic Self-Commuting Hamiltonian* section) of the NMR spectrum using arbitrary initial density and observable operators and a *time-periodic* self-commuting Hamiltonian.

$$c_{uv}^{(k)} = n^{-1} \sum_{p=1}^{n} \exp\{i(\Phi'_{uv}(t_p, 0) - 2\pi pk/n)\};$$

$$k = -\frac{n}{2} + 1, -\frac{n}{2} + 2, \cdots \frac{n}{2} \qquad [48]$$

If n is a power of two, a fast Fourier transform (FFT) (*5*) may be used, which improves the computational speed considerably for large n.

7. For all $k = -n/2 + 1, -n/2 + 2, \cdots n/2$, calculate the frequency of the kth sideband from Eq. [17] and the corresponding amplitude $a_{uv}^{(k)}$ from Eq. [16].

Steps 4–7 are repeated for each pair of Hamiltonian eigenstates $\{|u\rangle, |v\rangle\}$, giving rise to a *nonzero spectral amplitude* a_{uv}; the check for this leads to fairly large savings in the computational time as, overall, the inner loops comprise the most computer-intensive calculations.

This algorithm leads to a "stick spectrum" parameterized by the following set of frequencies and amplitudes:

$$\{\omega_{uv}^{(k)}, a_{uv}^{(k)}\} = \{\omega_{uv}^{(0)} + k\omega_r, c_{uv}^{(k)} a_{uv}\},$$
$$u, v = 1, 2, \ldots, \mathcal{N};$$
$$k = -\frac{n}{2} + 1, -\frac{n}{2} + 2, \cdots \frac{n}{2} \qquad [49]$$

We do not provide the explicit C code for this calculation here; an implementation for the direct generation of the *powder* averaged spectral amplitudes will be given in the following article.

Because the sideband index k extends over the range $\{-n/2 + 1, n/2\}$, the calculated spectral window is $\{[-n/2 + 1]\omega_r, [n/2]\omega_r\}$ or, equivalently in units of Hz, $\{[-n/2 + 1](\omega_r/2\pi), (n/2)(\omega_r/2\pi)\}$. It is important to choose n large enough that the calculated spectral width covers all sidebands of significant intensity. If n is too small, the coefficients $c_{uv}^{(k)}$ obtained in the Fourier transformation procedure will be erroneous. Physically, this translates into the following: Sidebands having larger (absolute) frequencies than those covered by the calculated window will appear as "folded" on top of the sidebands within the window. The simulated spectrum, therefore, comes out "missing" the high-frequency sidebands, as well as having incorrect amplitudes of the peaks *within* the window. In practice, convergence is checked by performing a series of calculations with increasing n, until the calculated spectra do not change significantly. A conservative estimate for convergence is that n should fulfill $n \geq 2|k_{\max}|$, where $k_{\max}$ is the highest order of the spectral sideband of significant intensity (*27*).

Note that "folding" may also plague spectra calculated from the direct method (as well as *experimental* spectra) if the time interval τ is not chosen small enough. In this case it is required that τ should fulfill $\tau \leq \pi(\omega_r|k_{\max}|)^{-1}$.

POSTPROCESSING OF CALCULATED DATA

The NMR time-domain signal or frequency-domain spectrum, output of the spin dynamics calculation, is given as a set of coordinates $\{t_j, s(t_j)\}$ or $\{\omega_j, a_j(\omega_j)\}$, respectively. These coordinates are subsequently stored in a file that can be viewed and plotted. This section mainly discusses how additional broadening of the spectral peaks may be applied.

The time-domain calculation produces a spectrum with a frequency resolution given by the inverse of the time span of the calculation. As for any experimentally acquired spectrum, the frequency resolution may be increased by "zero filling" the calculated time signal, i.e., appending signals of zero amplitude prior to the Fourier transformation. To avoid introducing distortions of the spectral peaks (showing up as "sinc wiggles" around each peak center) due to truncation of the time signal, it is recommended to apodize the time signal with a decaying function (such that its last data points are zero) prior to the transformation (*15, 17*).

The frequency-domain simulation, on the other hand, outputs a set of "exact" frequencies and amplitudes (within numerical errors and the accuracy of the spin Hamiltonian model of the physical problem). For further processing, this set is converted into a discrete "stick spectrum," i.e., a histogram of amplitudes. The frequencies may be given in units of Hz by first dividing each angular frequency ω by 2π: $\nu = \omega/2\pi$. The procedure for constructing a stick spectrum with a given frequency resolution is as follows.

Over a desired "*spectral window*" $\mathcal{S}_{\text{sw}} = \{-|\nu_{\max}| + \nu_{\text{res}}, |\nu_{\max}|\}$, arrays of frequencies $\{\nu_j\}$ and amplitudes $\{a_j\}$ are generated with a suitable frequency resolution

$$\nu_{\text{res}} = \nu_j - \nu_{j-1} = 2|\nu_{\max}|/m \qquad [50]$$

where m should be an even integer and represents the number of discrete frequency coordinates. As discussed on page 124 of Part I, the convention used in these *Concepts* articles for the spectral representation is that the frequency $+ |\nu_{\max}|$ is *included* in the window, but *not* the frequency $- |\nu_{\max}|$. Note that the

spectral window is the frequency span of the NMR spectrum and plays the same role as the parameter ω_{samp} of Part I. However, the value of $\mathcal{S}_{sw}$ is chosen *arbitrarily* (and may be decided after the frequency-domain simulation is finished), while ω_{samp} is linked to the parameters τ_{acq} and τ_{dwell}, which are decided *prior to* the experimental signal acquisition (or, alternatively, at the start of the direct time-domain simulation). The value of the jth frequency coordinate within $\mathcal{S}_{sw}$ is given by

$$v_j = -|v_{max}| + jv_{res}, \quad j = 1, 2, \cdots, m \qquad [51]$$

Now, we need need to construct the histogram of amplitudes. Initially, all amplitudes a_j are set to zero, where a_j is at the jth frequency coordinate. The discrete set $\{\omega_{uv}/2\pi,\ a_{uv}\}$ is then mapped onto $\mathcal{S}_{sw}$ as follows: The value a_{uv} is added to a_j at the frequency coordinate v_j closest to $\omega_{uv}/2\pi$. The index j may be calculated from

$$j = \left[\frac{\omega_{uv}}{2\pi v_{res}}\right] + m/2 \qquad [52]$$

where the function $[x]$ returns the nearest integer to x. This procedure is repeated for all members of the set $\{\omega_{uv}/2\pi,\ a_{uv}\}$. In C, this may be programmed as follows:

```
#define sign(x) ((x<0) ? -1:1 ) //return the sign of x
void CreateStickSpectrum( double* freq_in, complex* amp_in,int n_in,
                          double* freq_out, complex* amp_out,int n_out,double sw)
//INPUT: lists of frequencies and amplitudes (assumed to be complex numbers); both of dimension n_in
//OUTPUT: the amplitudes sorted into "bins", i.e. a stick-spectrum {n_out frequencies and
//        amplitudes} extending over the spectral window 'sw' with a frequency resolution sw/n_out
{
 int i,m;
 double freq,res=sw/double(n_out);               //define the frequency resolution

 for(i=0; i<n_out; i++) {
  freq_out[i]= ( (-sw/2)+((i+1)*res) );          //generate the output frequencies
  amp_out[i]=complex(0.,0.);                     //start with an empty array of amplitudes
 }

 for(i=0; i<n_in; i++) {
  //get the value of m, using that int(x) returns the integer part of x.
  //next, place the i:th amplitude from the input list at th m:th position of amp_out
  m=int( freq_in[i]/res+(sign(freq_in[i])*0.5) )+n_out/2-1;
  amp_out[m] += amp_in[i];                       //update the array of amplitudes
 }
}
```

Note that the computer implementation employs array indexing starting from zero (see above); hence, the value of `m` is one less than that given in Eq. [52].

Broadening of the spectral peaks may now conveniently be introduced by first converting this stick spectrum into a time-domain signal by an *inverse* Fourier transform. Next, the time-domain signal is apodized with an appropriate decaying function, for example, an exponential decay. Finally, another Fourier transform produces the line broadened frequency-domain spectrum. The shape of the spectral peaks depends on the form of the decaying function being used. The procedure is illustrated in Fig. 6 and implemented numerically as follows:

```
void lineBroaden( complex *list,double width,int n_points, double dwell,int function_flag)
// INPUT: list of spectral amplitudes
// OUTPUT: real part of spectrum; the FWHH of the line is is (width) Hz, with
//        (function_flag=0/1) giving Lorentzian and Gaussian shape, respectively
{
 IFFT( list,n_points);                    //inverse FT; transform to time-domain
 for(int i=0;i<n_points;i++)              //apply decaying function
  if (function_flag)
   list[i] = exp(-sqr(Pi*width*i*dwell)/4.*log(2.)) )*list[i];   //Gaussian decay
  else
   list[i] = exp(-Pi*width*i*dwell)*list[i];    //exponential decay
 FFT(list,n_points,1);                          //transform back to the f-domain
}
```

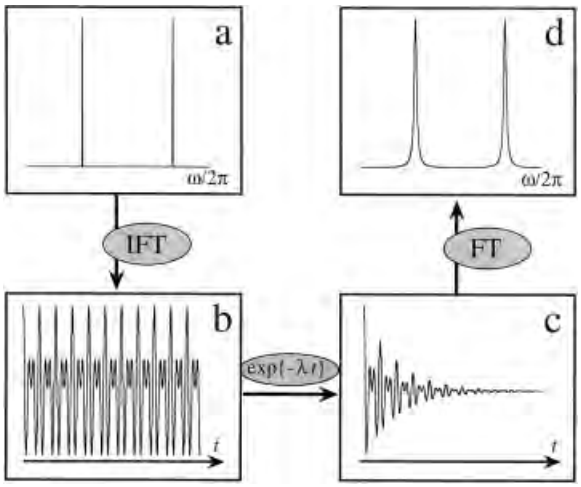

Figure 6 Procedure for introducing line broadening to a stick spectrum, shown in (a). The time signal (b) is obtained by applying an inverse Fourier transform. It is subsequently multiplied with an exponentially decaying function (c). The line-broadened spectrum with Lorentzian peak shapes is obtained after Fourier transformation, shown in (d).

The routines FFT and IFFT carries out the direct and inverse FFTs, respectively. They are based on the routines in Ref. (*5*) and their explicit computer codes are given in Ref. (*23*).

SUMMARY AND DISCUSSION

Computational Efficiency

We implemented the algorithms for spin dynamics calculations outlined in Part I. The code here was not optimized for speed but designed so as to be straightforward and transparent. A general discussion on issues such as implementation of more efficient simulation algorithms may be found in Ref. (*32*), and we also refer to the literature for fast algorithms for the special case of periodic Hamiltonians (*26–32*). Here, we limit ourselves to briefly commenting on the relative computational efficiencies of the frequency-domain simulation and the direct method. In their current versions, the frequency-domain approach is much more efficient and in some cases many orders of magnitude faster than the direct method. However, as discussed in Ref. (*32*), it is feasible to carry out simulations in the time domain employing the eigenbasis of the Hamiltonian as in the frequency-domain calculations, thereby giving the two methods roughly equal efficiencies.

Extensions to More Complicated Problems

The dynamically inhomogeneous cases (*2*) primarily considered in this article are important from an NMR application standpoint because they comprise many applications where NMR interaction parameters may be determined from spectral sideband analysis combined with numerical fitting techniques. However, they comprise only a minor class of all potential scenarios. For example, simulations of strongly coupled homonuclear spins, or experiments involving RF fields during the signal acquisition, require modifications of the computer algorithms presented here. This applies especially to cases where *both* the spatial parts and the spin parts of the Hamiltonian are time dependent. Sample rotation is used to modulate the spatial tensors. In addition, time-dependent RF fields render the spin operators time dependent. This is exploited for suppressing certain spin interactions while retaining others and used, for example, in spin decoupling problems (*14–18*). Such spin systems evolve under dynamically homogeneous Hamiltonians. Below, we briefly outline some of the additional considerations required when simulating such cases. The various scenarios may roughly be classified as follows:

1. *RF fields applied to a static sample.* Assume that a sequence of RF pulses is applied to the spins but the sample remains static. In this case, only the spin part of the Hamiltonian is time dependent. The expression for the RF Hamiltonian for a pulse of constant RF phase and amplitude is given by Eq. [I-138]: If the amplitude and phase is constant during each pulse (but may be varied between the pulses), the rotating frame RF Hamiltonian is *piece-wise time independent*, i.e., takes the form of a step function. The propagators needed when numerically calculating the spin dynamics of such experiments may be obtained by applying Eq. [34] to the Hamiltonian of *each* pulse. Next, the accumulated propagator over the entire pulse sequence is constructed from a time-ordered product (Eq. [38]) of the propagators for the pulses. The time signal is finally calculated by propagating the density operator according to the recipe of the direct method (section 5.5).
2. *Rotating solid in the absence of RF fields.* In this case, the Hamiltonian is periodically modulated exactly as for the dynamically inhomogeneous cases, with the additional complication that it is no longer self-commuting. This re-

quires dividing the rotational period into small segments and applying Eq. [34] to the Hamiltonian at $t = t_{\mathrm{mid}}$ of each segment, as discussed above.

3. *Rotating solid in the presence of RF fields.* This is a superposition of cases 1 and 2. The major difficulty here is to find a suitable way to divide the time interval of the experiment into small time segments during each of which the Hamiltonian may be approximated as piece-wise time independent. Once this is done, the calculation follows the standard recipe of the direct method.

Of course, there also exist a large number of experiments requiring more extensive computational algorithms, such as simulations of multidimensional experiments or problems involving chemical exchange or relaxation. These are outside the scope of this article, and we refer to Ref. (*32*) for more information about writing code for such simulations.

This article only involved calculations of NMR responses from a single crystal under static and MAS conditions. The simulations of spectra of powders, i.e., a collection of many single crystals, will be treated in the following paper, where explicit computer code for complete programs based on the routines provided in this article will be given.

ACKNOWLEDGMENTS

I would like to thank L. Frydman, C.V. Grant, A. Sebald, and S. Vega for helpful comments on the manuscript and M.H. Levitt for many discussions and for providing *Mathematica* routines used for generating some of the figures. A postdoctoral fellowship from The Swedish Foundation for International Cooperation in Research and Higher Education (STINT) is appreciated.

REFERENCES

1. Edén M. 2003 Computer simulations in solid state NMR: Part I. Spin dynamics theory. Concepts Magn Reson Part A 17A:117–154.
2. Maricq MM, Waugh JS. NMR in rotating solids. J Chem Phys 1979; 70:3300–3316.
3. Ellis TMR, Philips IR, Lahey TM. Fortran 90 Programming. New York: Addison-Wesley; 1994.
4. Schildt H. C: The Complete Reference. Berkeley, CA: Osborne McGraw-Hill; 1995.
5. Press WH, Flannery BP, Teukolsky SA, Vetterling VT. Numerical Recipies in C. The Art of Scientific Computing. Cambridge, UK: Cambridge University Press; 1986.
6. Stroustrup B. The C++ Programming Language. New York: Addison-Wesley; 1991.
7. Matlab. Natick, MA: The Mathworks, Inc. (2003).
8. Alderman DW, Solum MS, Grant DM. Methods for analyzing spectroscopic line shapes. NMR solid powder patterns. J Chem Phys 1986; 84:3717–3725.
9. Wang D, Hanson GR. A new method for simulting randomly oriented powder spectra in magnetic resonance: The Sydney Opera House (SOPHE) method. J Magn Reson A 1995; 117:1–8.
10. Mombourquette MJ, Weil JA. Simulation of magnetic resonance powder spectra. J Magn Reson 1992; 99:37–44.
11. Bak M, Nielsen NC. REPULSION—A novel approach to efficient powder averaging in solid-state NMR. J Magn Reson 1997; 125:132–139.
12. Edén M, Levitt MH. Computation of orientational averages in solid-state NMR by Gaussian spherical quadrature. J Magn Reson 1998; 132:220–239.
13. Ponti A. Simulation of magnetic resonance static powder lineshapes: A quantitative assessment of spherical codes. J Magn Reson 1999; 138:288–297.
14. Haeberlen U. High Resolution NMR in Solids. Selective Averaging. New York: Academic Press; 1976.
15. Ernst RR, Bodenhausen G, Wokaun A. Principles of Nuclear Magnetic Resonance in One and Two Dimensions. Oxford, UK: Clarendon Press; 1987.
16. Munowitz M. Coherence and NMR. New York: Wiley; 1988.
17. Schmidt-Rohr K, Spiess HW. Multidimensional Solid-State NMR and Polymers. New York: Academic; 1994.
18. Mehring M, Weberuß VA. Object-Oriented Magnetic Resonance. Classes and Objects, Calculations and Computations. London: Academic Press; 2001.
19. Levitt MH. Spin Dynamics. Basics of Nuclear Magnetic Resonance. Chichester, UK: Wiley; 2001.
20. http://www.gnu.org.
21. Smith SA, Levante TO, Meier BH, Ernst RR. Computer simulations in magnetic resonance. An object-oriented programming approach. J Magn Reson A 1994; 106: 75–105.
22. GAMMA program. Florida State University, Tallahassee, FL. http://gamma.magnet.fsu.edu.
23. http://www.fos.su.se/physical/mattias.
24. Slichter CP. Principles of Magnetic Resonance. New York: Springer-Verlag; 1990.
25. Salzman WR. Quantum mechanics of systems periodic in time. Phys Rev A 1974; 10:461–465.
26. Kubo A, Imashiro F, Terao T. Fine structures of ^{1}H-coupled ^{13}C MAS NMR spectra for uniaxially rotating molecules in deuterated surroundings: Conformations of *n*-alkane molecules enclathrated in urea channels. J Phys Chem 1996; 100:10854–10860.
27. Edén M, Lee YK, Levitt MH. Efficient simulation of

periodic problems in NMR. Application to decoupling and rotational resonance. J Magn Reson A 1996; 120: 56–71.
28. Charpentier T, Fermon C, Virlet J. Efficient time propagation technique for MAS NMR simulation: Application to quadrupolar nuclei. J Magn Reson 1998; 132: 181–190.
29. Charpentier T, Fermon C, Virlet J. Numerical and theoretical analysis of multiquantum magic-angle-spinning experiments. J Chem Phys 1998; 109:3116–3130.
30. Levitt MH, Edén M. Numerical simulation of periodic NMR problems: Fast calculation of carousel averages. Mol Phys 1998; 95:879–890.
31. Hohwy M, Bildsøe H, Jakobsen HJ, Nielsen NC. Efficient spectral simulations in NMR of rotating solids. The γ-COMPUTE algorithm. J Magn Reson 1999; 136: 6–14.
32. Hodgkinson P, Emsley L. Numerical simulation of solid-state NMR experiments. Progr NMR Spectrosc 2000; 36:201–239.
33. Banwell CN, Primas H. On the analysis of high-resolution nuclear magnetic resonance spectra. I. Methods of calculating NMR spectra. Mol Phys 1962; 6:225–256.

BIOGRAPHY

Mattias Edén was born in 1971 in Stockholm, Sweden. He received his B.S. in chemistry from Stockholm University in 1994 and continued there with doctoral studies in the group of Malcolm H. Levitt. Their work primarily involved pulse sequence design for determining molecular structures by solid-state NMR, as well as the development of methods for numerically simulating NMR experiments. He did postdoctoral work in the field of solid-state NMR on quadrupolar nuclei with Lucio Frydman at the University of Illinois at Chicago (2000) and at the Weizmann Institute of Science in Israel (2001). Currently, Dr. Edén is an assistant professor at Stockholm University, focusing on solid-state NMR methodology development for structural investigations of inorganic materials.