Computing and Fitting SSNMR powder patterns with the Arithmetic-Geometric Mean and Edge Detection[1]

January 8, 2007

J. K. Denny*
Department of Mathematics
Mercer University
Macon, GA 31207
denny_jk@mercer.edu

M. B. Daniel
Department of Mathematics
Mercer University
Macon, GA 31207
daniel_mb@acadmn.mercer.edu

and

F. A. Kovacs
Department of Chemistry
University of Nebraska–Kearney
Kearney, NE 68849
kovacsfa@unk.edu

Abstract

This paper presents a very efficient technique for approximating the ideal SSNMR powder pattern using the arithmetic-geometric mean and demonstrates finding an initial fitting of the ideal powder pattern to an experimental spectrum via Marr-Hildreth edge detection. In particular, the edge detection approach is used to identify possible values for the principal values of the chemical shielding tensor. These possibilities are then evaluated using a heuristic approach for choosing the best estimates of the principal values based on a measure of edge strength and the sign of the third derivative of the broadened experimental spectrum. We present a detailed mathematical development of the ideal SSNMR powder spectrum and of the arithmetic geometric mean and summarize the fundamental ideas of line broadening and edge detection. The algorithms in the paper are demonstrated in a program supplied in the appendix and are applied to experimental data from [$^{13}C_1$]-leucine.

Key Words: Solid-state NMR, powder spectrum, arithmetic-geometric mean, edge detection

[1]This is a preprint of an article published in "Concepts in Magnetic Resonance", January 2007.

1 Introduction

Solid-state nuclear magnetic resonance (SSNMR) is a powerful tool for determining 3-dimensional molecular structures of membrane proteins, which have proven difficult to study using crystallography or solution NMR. From SSNMR experiments in which membrane proteins are inserted into membranes and aligned with the magnetic field direction, orientational constraints on the structure can be obtained that lead to characterizations of a protein's structure [1, 2, 3, 4]. To obtain orientational constraints from 1-dimensional experiments on oriented samples, the principal values of the chemical shielding tensor must be determined from 1-dimensional powder pattern spectra. The method of edge detection provides an efficient approach for finding the principal values of the chemical shielding tensor.

Simulated powder pattern spectra for axially symmetric tensors can be easily computed; however, nonaxially symmetric tensors are represented as elliptic integrals of the first kind, which cannot be computed analytically. Section 2 will present the development of this elliptic integral.

Many previous simulation methods have approximated the required integrals using summation and grid methods, which demand significant amounts of computer memory and processor time. The arithmetic-geometric mean (AGM) developed independently by Gauss and Lagrange, however, gives an algorithm for approximating an elliptic integral of the first kind that is simple, converges very quickly, and requires little computer memory. Section 3 explains the AGM and its properties with emphasis on the key theorem connecting the AGM to the elliptic integral of the first kind.

Sections 4 and 5 present line broadening and the method of edge detection along with a heuristic algorithm for selecting the principal values of the chemical shielding tensor from the results of performing edge detection. Finally, section 6 demonstrates the use of the above methods in finding the principal values of the chemical shielding tensor and fitting the ideal spectrum to the experimental spectrum using experimental ^{13}C data

2 Computing 1D SSNMR Powder Patterns

Based on quantum physics [5, 6], chemical shielding can be represented as the value of a quadratic form associated with a symmetric second–rank tensor at a unit magnetic field vector, $\mathbf{B}$. When written in its principal axis frame (PAF), the chemical shielding tensor is diagonalized with principal values $\delta_{33} \leq \delta_{22} \leq \delta_{11}$ on the diagonal, as represented in the deshielding δ-scale [7]. Thus, the chemical shielding, ω, can be expressed as

$$\begin{aligned}\omega &= \mathbf{B}^{\mathrm{t}} \begin{pmatrix} \delta_{33} & 0 & 0 \\ 0 & \delta_{22} & 0 \\ 0 & 0 & \delta_{11} \end{pmatrix} \mathbf{B} \\ &= \delta_{33} \cos^2\alpha \sin^2\beta + \delta_{22} \sin^2\alpha \sin^2\beta + \delta_{11} \cos^2\beta, \end{aligned} \tag{1}$$

where the unit magnetic field vector $\mathbf{B}$ is written in spherical coordinates in the PAF as

$$\mathbf{B} = (\cos\alpha \sin\beta, \sin\alpha \sin\beta, \cos\beta)^{\mathrm{t}} \tag{2}$$

with ^{t} denoting matrix transpose. Here, α is the polar angle and β is the azimuthal angle (see Figure 1) with $0 \leq \alpha < 360^\circ$ and $0 \leq \beta \leq 180^\circ$.

In a solid-state NMR powder spectrum, the intensity, I, at a particular frequency, δ, is proportional to the probability that the magnetic field vector, $\mathbf{B}$, will have coordinates (α, β) in the PAF of a randomly selected molecule in a given sample [5]. Equivalently, I can be viewed as proportional to the probability that a randomly chosen molecule will resonate at frequency δ [8]. Thus, the goal is to compute the relevant probability density function.

Since δ_{33}, δ_{22}, and δ_{11} are the parameters in computing a powder pattern, denote the intensity as a function of δ by $I(\delta; \delta_{33}, \delta_{22}, \delta_{11})$. Taking $|\mathbf{B}| = 1$, the required probability density function is the derivative of the cumulative probability that $\omega(\mathbf{B}) \leq \delta$. Thus,

$$I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) = \frac{d}{d\delta}\Big(\text{Prob}(\omega(\mathbf{B}) \leq \delta)\Big). \tag{3}$$

To calculate the cumulative probability $\text{Prob}(\omega(\mathbf{B}) \leq \delta)$, integrate over the portion of the unit sphere on which $\omega(\mathbf{B})$ is less than or equal to δ. Call this region Ω. Normalizing by the total area of the sphere gives the density function

$$I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) = \frac{1}{4\pi}\frac{d}{d\delta}\iint_\Omega \sin\beta \, d\beta \, d\alpha, \tag{4}$$

where $\Omega = \{(\alpha, \beta) | \omega(\alpha, \beta) \leq \delta\}$ is the portion of the sphere on which the chemical shielding is less than or equal to the given chemical shielding value of δ. The difficulties that arise in computing this integral depend on the shape of the region Ω.

2.1 An Axially Symmetric Case

In the axially symmetric case, $\delta_{33} = \delta_{22}$, and the integral in Eq. (4) is taken over the region Ω with $\delta_{33} + (\delta_{11} - \delta_{33})\cos^2\beta \leq \delta$. So, $0 \leq \alpha < 2\pi$, and β must fall between

$$b_+ = \cos^{-1}\left(\sqrt{\frac{\delta - \delta_{33}}{\delta_{11} - \delta_{33}}}\right) \text{ and } b_- = \cos^{-1}\left(-\sqrt{\frac{\delta - \delta_{33}}{\delta_{11} - \delta_{33}}}\right). \tag{5}$$

Thus, the intensity function is

$$\begin{aligned} I(\delta; \delta_{33}, \delta_{33}, \delta_{11}) &= \frac{1}{4\pi}\frac{d}{d\delta}\int_0^{2\pi}\int_{b_+}^{b_-} \sin\beta \, d\beta \, d\alpha \\ &= \frac{1}{2\sqrt{(\delta - \delta_{33})(\delta_{11} - \delta_{33})}}. \end{aligned} \tag{6}$$

2.2 The Nonaxially Symmetric Case

The situation for a nonaxially symmetric tensor with $\delta_{33} < \delta_{22} < \delta_{11}$ is more complex and involves an elliptic integral. To begin, the shape of the region Ω over which we need to integrate must be determined. Solving

$$\delta_{33}\cos^2\alpha\sin^2\beta + \delta_{22}\sin^2\alpha\sin^2\beta + \delta_{11}\cos^2\beta \leq \delta \tag{7}$$

for β demonstrates that β is restricted to the values between

$$b_+ = \arccos\left(\left(\frac{\delta - \delta_{22}\sin^2\alpha - \delta_{33}\cos^2\alpha}{\delta_{11} - \delta_{22}\sin^2\alpha - \delta_{33}\cos^2\alpha}\right)^{1/2}\right) \text{ and}$$
$$b_- = \arccos\left(-\left(\frac{\delta - \delta_{22}\sin^2\alpha - \delta_{33}\cos^2\alpha}{\delta_{11} - \delta_{22}\sin^2\alpha - \delta_{33}\cos^2\alpha}\right)^{1/2}\right). \tag{8}$$

To determine the possible values of α, first observe that, by symmetry, we can consider only the orientations of $\mathbf{B}$ with $0 \leq \alpha \leq \pi/2$ and multiply the integral in Eq. (4) by 4. Now, to ensure that b_+ and b_- are defined, the values of α are limited by the requirement that

$$\frac{\delta - \delta_{22}\sin^2\alpha - \delta_{33}\cos^2\alpha}{\delta_{11} - \delta_{22}\sin^2\alpha - \delta_{33}\cos^2\alpha} > 0. \tag{9}$$

Solving for $\cos^2\alpha$ gives

$$\cos^2\alpha > \frac{\delta_{22} - \delta}{\delta_{22} - \delta_{33}}, \tag{10}$$

which is certainly always satisfied when $\delta_{22} < \delta < \delta_{11}$. For $\delta_{33} < \delta < \delta_{22}$, this relationship only holds for $0 \leq \alpha < \arccos\left(\sqrt{\frac{\delta_{22}-\delta}{\delta_{22}-\delta_{33}}}\right)$.

Clearly, the region Ω is more complex than in the axially symmetric case, since there are two cases which are determined by the value of δ. To simplify the remainder of the calculations, we will allow α to take on values between 0 and $\pi/2$. To prevent this from making b_+ and b_- undefined, we will consider inverse cosine as a complex-valued function and only deal with the real part of the result. The two forms of the region Ω will again play a key role in the final calculations of this section.

Now, integrating α from 0 to $\pi/2$, multiplying by 4, and taking the real part of the integral, the intensity function is

$$I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) = \frac{1}{\pi}\frac{d}{d\delta}\ \Re\mathrm{e}\left[\int_0^{\pi/2}\int_{b_+}^{b_-} \sin\beta\, d\beta\, d\alpha\right]. \tag{11}$$

Moving the derivative inside the outer integral, integrating with respect to β, and differentiating gives

$$I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) =$$
$$\frac{1}{\pi}\Re\mathrm{e}\left[\int_0^{\pi/2} \frac{d\alpha}{\sqrt{(\delta - \delta_{22}\sin^2\alpha - \delta_{33}\cos^2\alpha)(\delta_{11} - \delta_{22}\sin^2\alpha - \delta_{33}\cos^2\alpha)}}\right]. \tag{12}$$

Proceeding as in [8], let $s = \delta_{33}\cos^2\alpha + \delta_{22}\sin^2\alpha$ so that

$$d\alpha = \frac{ds}{2\sqrt{(\delta_{22} - s)(s - \delta_{33})}}. \tag{13}$$

Thus,

$$I(\delta;\delta_{33},\delta_{22},\delta_{11}) = \\ \frac{1}{2\pi}\Re\mathrm{e}\left[\int_{\delta_{33}}^{\delta_{22}} \frac{ds}{\sqrt{(\delta_{11}-s)(s-\delta_{33})(\delta-s)(\delta_{22}-s)}}\right], \tag{14}$$

which is an elliptic integral of the first kind.

To write the above integral in another standard form, make the intermediate substitution

$$s = \frac{\delta_{33}(\delta_{22}-\delta_{11}) - t\,\delta_{11}(\delta_{22}-\delta_{33})}{(\delta_{22}-\delta_{11}) - t\,(\delta_{22}-\delta_{33})} \tag{15}$$

to obtain the integral

$$I(\delta;\delta_{33},\delta_{22},\delta_{11}) = \frac{1}{2m\,\pi}\ \Re\mathrm{e}\Big[\int_0^1 \frac{dt}{\sqrt{t(1-t)(1-\kappa^2 t)}}\Big], \tag{16}$$

where

$$\kappa = \sqrt{\frac{(\delta_{11}-\delta)(\delta_{22}-\delta_{33})}{(\delta-\delta_{33})(\delta_{11}-\delta_{22})}} \text{ and } m = \sqrt{(\delta_{11}-\delta_{22})(\delta-\delta_{33})}. \tag{17}$$

Observe that $I(\delta;\delta_{33},\delta_{22},\delta_{11})$ is not real-valued for $\delta_{33} \le \delta < \delta_{22}$, since $\kappa > 1$. But, $I(\delta;\delta_{33},\delta_{22},\delta_{11})$ is real-valued for $\delta_{22} < \delta \le \delta_{11}$, since $\kappa < 1$. Importantly, for $\delta = \delta_{22}$, $\kappa = 1$, and hence, $I(\delta_{22};\delta_{33},\delta_{22},\delta_{11})$ does not exist.

Next, substitute $t = u^2$ when $\delta_{22} < \delta \le \delta_{11}$ and $t = u^2/\kappa^2$ when $\delta_{33} \le \delta < \delta_{22}$ to produce a standard form of the elliptic integral of the first kind. The result agrees with [9, 10] and is

$$I(\delta;\delta_{33},\delta_{22},\delta_{11}) = \begin{cases} \dfrac{1}{2\sqrt{(\delta_{11}-\delta_{33})(\delta_{22}-\delta_{33})}}, & \text{if } \delta = \delta_{33} \\ \dfrac{1}{m\kappa\,\pi}K\left(\dfrac{1}{\kappa}\right), & \text{if } \delta_{33} < \delta < \delta_{22} \\ \dfrac{1}{m\,\pi}K(\kappa), & \text{if } \delta_{22} < \delta < \delta_{11} \\ \dfrac{1}{2\sqrt{(\delta_{11}-\delta_{22})(\delta_{11}-\delta_{33})}}, & \text{if } \delta = \delta_{11} \\ 0, & \text{if } x < \delta_{33} \text{ or } x > \delta_{11}, \end{cases} \tag{18}$$

where

$$K(k) = \int_0^1 \frac{du}{\sqrt{(1-u^2)(1-k^2u^2)}}$$

is the complete elliptic integral of the first kind and κ, m are functions of δ, δ_{33}, δ_{22}, and δ_{11} as above. This intensity function has singularities at the resonances δ_{33}, δ_{22}, and δ_{11}. Unfortunately, neither integral in Eq. (18) can be computed analytically and must be approximated.

3 The AGM and Powder Patterns

For applications to SSNMR powder patterns, the challenge of computing the elliptic integral in Eq. (18) has been approached using several different algorithms. Typically, a sum is used to approximate the elliptic integral [8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]. Such methods require the use of correction factors to account for the errors that occur when approximating an integral by a sum. Another method for specifically approximating elliptic integrals, however, exists that does not require time-consuming sums, is easy to program, and can quickly give great accuracy. This algorithm is called the *arithmetic-geometric mean (AGM)* and is used to compute elliptic integrals in packages such as *Maple* and *Mathematica*.

The AGM and its properties were known in the nineteenth century to Gauss [26] and Lagrange [27] who discovered them independently. More recently, the book *Pi and the AGM* by Borwein and Borwein [28] has renewed interest in the AGM.

Define the arithmetic mean a_{n+1} and the geometric mean b_{n+1} by

$$a_{n+1} = \frac{a_n + b_n}{2}, \quad b_{n+1} = \sqrt{a_n b_n} \tag{19}$$

for nonnegative integers n and positive numbers $a_0 > b_0$. Before seeing the connection between these means and elliptic integrals of the first kind, we observe several important facts.

First, we have the *arithmetic-geometric mean inequality*

$$a_{n+1} = \frac{a_n + b_n}{2} \geq \sqrt{a_n b_n} = b_{n+1} \tag{20}$$

for all n. Moreover, a_n is a decreasing sequence of values, while b_n is an increasing sequence. Thus,

$$a_0 \geq a_1 \geq \cdots \geq a_n \geq \cdots \geq b_n \geq \cdots \geq b_1 \geq b_0. \tag{21}$$

Next, as in a recent review [29] of Gauss's work on the AGM, observe that

$$a_{n+1} - b_{n+1} \leq a_{n+1} - b_n = \frac{a_n - b_n}{2}. \tag{22}$$

Induction using this inequality shows that

$$0 \leq a_n - b_n \leq \frac{1}{2^n}(a_0 - b_0) \to 0 \text{ as } n \to \infty. \tag{23}$$

Thus, a_n and b_n converge to the same value as n grows large, and we can define this value to be a function of the initial values a_0 and b_0,

$$M(a_0, b_0) = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n. \tag{24}$$

Note also that, for all integers n, $a_n \geq M(a_0, b_0) \geq b_n$.

Finally, the connection between the AGM and complete elliptic integrals of the first kind can be given using the following remarkable theorem [28, 30, 31]:

$$K(k) = \frac{\pi}{2\, M(1, \sqrt{1-k^2})}. \tag{25}$$

(An excellent derivation of this theorem is given in [32].) Since Eq. (18) is a complete elliptic integral of the first kind, Eq. (25) implies that $I(\delta;\delta_{33},\delta_{22},\delta_{11})$ can be computed using the AGM as follows.

$$I(\delta;\delta_{33},\delta_{22},\delta_{11}) = \begin{cases} \dfrac{1}{2\sqrt{(\delta_{11}-\delta_{33})(\delta_{22}-\delta_{33})}}, & \text{if } \delta = \delta_{33} \\ \dfrac{1}{2m\kappa\, M(1,\sqrt{1-1/\kappa^2})}, & \text{if } \delta_{33} < \delta < \delta_{22} \\ \dfrac{1}{2m\, M(1,\sqrt{1-\kappa^2})}, & \text{if } \delta_{22} < \delta < \delta_{11} \\ \dfrac{1}{2\sqrt{(\delta_{11}-\delta_{22})(\delta_{11}-\delta_{33})}}, & \text{if } \delta = \delta_{11} \\ 0, & \text{if } x < \delta_{33} \text{ or } x > \delta_{11}, \end{cases} \tag{26}$$

where

$$\kappa = \sqrt{\frac{(\delta_{11}-\delta)(\delta_{22}-\delta_{33})}{(\delta-\delta_{33})(\delta_{11}-\delta_{22})}} \text{ and } m = \sqrt{(\delta_{11}-\delta_{22})(\delta-\delta_{33})}. \tag{27}$$

Using the AGM, the intensity function $I(\delta;\delta_{33},\delta_{22},\delta_{11})$ can be computed numerically by iterating the AGM sequences several times (See Figure 2). As will be seen shortly, only a few iterations are needed to produce very good accuracy.

To understand the rate of convergence, observe that straightforward calculation with the definition of the AGM gives

$$a_n^2 - b_n^2 = \frac{1}{4}(a_{n-1}-b_{n-1})^2. \tag{28}$$

Then, combining this identity with the definition of the AGM and the inequality $a_{n+1} \geq M(a_0,b_0)$ produces

$$(a_n - b_n) = \frac{a_n^2-b_n^2}{2a_{n+1}} \leq \frac{a_n^2-b_n^2}{2M(a_0,b_0)} = \frac{(a_{n-1}-b_{n-1})^2}{8M(a_0,b_0)}. \tag{29}$$

Thus, the sequences a_n and b_n converge quadratically. This implies that the number of digits in a_n and b_n that agree will approximately double with every iteration of the AGM. For most powder patterns, five iterations of the AGM will give an accuracy of at least 10^{-8}.

4 Line Broadening

Having examined the theoretical development of a function for an ideal SSNMR powder pattern and an efficient method for computing such, we now turn to modifying the ideal spectrum

to include the smoothness and broadness seen in typical SSNMR powder experiments. This modification is called *line broadening* or *apodization* and is simulated mathematically using *convolution*.

The *convolution* of two real-valued functions f and g is defined

$$f * g(x) = \int_{-\infty}^{\infty} f(x-t)g(t)\,dt. \tag{30}$$

Simulated spectra, however, are treated as discrete functions. In this case, the integral can be approximated using Riemann sums or Simpson's Rule.

The convolution operation mixes two functions to produce a new function that reflects some of the properties of each (See Figure 3). This mixing action will allow us to produce broadened spectra by combining our intensity function, $I(\delta; \delta_{33}, \delta_{22}, \delta_{11})$, from Eq. (26) with a smooth function. In addition, the convolution operation has convenient mathematical properties that make it particularly useful. For example, the change of variables $u = x - t$ can be used to show that $f * g = g * f$. Some additional properties will be summarized in the next section.

To get the broadening typically seen in SSNMR powder spectra, the intensity function, $I(\delta; \delta_{33}, \delta_{22}, \delta_{11})$, is convolved with one of the following smooth functions.

1. The Gaussian, or normal distribution, function is given by

$$N(x; \mu, s) = \frac{1}{s\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2s^2}\right) \tag{31}$$

   with mean μ and standard deviation s.

2. The Lorentzian function is defined

$$L(x; x_0, \Gamma) = \frac{1}{\pi} \frac{\Gamma}{(x-x_0)^2 + \Gamma^2} \tag{32}$$

   with center x_0 and width parameter Γ.

3. The Voigt function is the convolution of independent Gaussian and Lorentzian functions:

$$V(x; x_0, \Gamma) = N(x; 0, 1) * L(x; x_0, \Gamma). \tag{33}$$

These functions are chosen to produce different forms of line broadening. For example, the Gaussian is relatively wide at half-height, while the Lorentzian is thinner at half-height and the Voigt function produces a shape between the Gaussian and Lorentzian. Figure 4 shows the convolution of the intensity function, $I(\delta; \delta_{33}, \delta_{22}, \delta_{11})$, with a Gaussian to give a broadened ideal powder spectrum.

5 Edge Detection

To obtain orientational constraints from one-dimensional SSNMR data on oriented samples, the principal values, $\delta_{33}, \delta_{22}, \delta_{11}$, of the chemical shielding tensor in Eq. (1) must be known. These values can be found by applying one-dimensional *Marr-Hildreth edge detection* to an experimental SSNMR powder spectrum. This technique seeks to find the jumps in broad, noisy data that correspond to the singularities seen in ideal spectra. This method is commonly used in image and signal analysis and is implemented in electron paramagnetic resonance spectrometers [33]. For many situations edge detection will be quite effective, but several other techniques for extracting parameters from spectra exist to handle more complicated spectra [34, 35, 36].

As a basic example of one-dimensional edge detection, consider the step function

$$\chi_{[a,\infty)}(x) = \begin{cases} 1, & \text{if } x \geq a \\ 0, & \text{otherwise} \end{cases} \tag{34}$$

to be the ideal spectrum, and let the convolution $\chi_{[a,\infty)} * N(x; 0, s)$ be the "experimental" spectrum. Figure 5 indicates that the inflection point in the graph of $\chi_{[a,\infty)} * N(x; 0, s)$ occurs exactly at $x = a$ where the discontinuity in $\chi_{[a,\infty)}$ occurs. To verify this, we need some key properties of the convolution. (See the excellent text [37] for detailed discussion of these properties.)

Properties: If f is integrable with two continuous derivatives and g is integrable on a closed interval, then

1. $f * g$ is continous,
2. $\dfrac{d}{dx}(f * g) = f * \dfrac{dg}{dx}$,
3. $\dfrac{d^2}{dx^2}(f * g) = f * \dfrac{d^2 g}{dx^2}$, and
4. $\dfrac{d}{dx}(f * \chi_{[a,\infty)}) = f(x - a)$.

(Note that Property 4 indicates that, in essence, the derivative of the step function $\chi_{[a,\infty)}$ is the Dirac delta function, δ_a, at a.) Combining Properties 2 and 4,

$$\frac{d^2}{dx^2}\Big(\chi_{[a,\infty)} * N(x; 0, s)\Big) = N'(x - a; 0, s) \tag{35}$$

so that the inflection point in $\chi_{[a,\infty)} * N(x; 0, s)$ occurs exactly at $x = a$, where $N(x - a; 0, s)$ has a maximum, as seen in Figure 5.

When working with an SSNMR powder spectrum, the situation is not so simple, since $I(\delta; \delta_{33}, \delta_{22}, \delta_{11})$ is not differentiable at δ_{22} and is a much more complicated function than $\chi_{[a,\infty)}$. Indeed, the convolution $I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) * N(\delta; 0, s)$ does not have inflection points exactly at the jumps at δ_{33} and δ_{11}. The inflection points, however, do occur *near* δ_{33} and δ_{11}.

In fact, the wider the interval $[\delta_{33}, \delta_{11}]$ is, the closer the inflection points of $I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) * N(\delta; 0, s)$ are to δ_{33} and δ_{11}.

To address the noise found in experimental spectra, we assume that the spectrum is given by

$$F(x) = I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) * N(\delta; 0, s) + \eta(\delta), \tag{36}$$

where η is the noise function. To approximate δ_{33} and δ_{11}, we wish to find F'' and locate the inflection points for F. However, differentiating a noise function will obscure the information contained in the signal and produce a very noisy derivative function. So, first, we perform *Gaussian smoothing* by convolving F with a Gaussian having a relatively large standard deviation, t. Then,

$$F(\delta) * N(\delta; 0, t) = I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) * N(\delta; 0, s) * N(\delta; 0, t) + \eta(\delta) * N(\delta; 0, t). \tag{37}$$

This convolution will dramatically reduce and smooth the noise, allowing us to find the desired derivatives.

Now, the edge detection strategy is to use Property 3 to find the second derivative

$$\frac{d^2}{d\delta^2}(F(\delta) * N(\delta; 0, t)) = F(\delta) * N''(\delta; 0, t), \tag{38}$$

which can be easily programmed by analytically finding $N''(\delta; 0, t)$ and then convolving it with the experimental spectrum F. Then, we can find the inflection points for $F(\delta) * N(\delta; 0, t)$ and take these as initial estimates for δ_{33} and δ_{11}.

In practice, the inflection points are determined by finding the zero-crossings of $F(\delta) * N''(\delta; 0, t)$. Based on the shape of the second derivative of a broadened ideal powder spectrum, $I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) * N''(\delta; 0, t)$, there should be exactly six zero-crossings with δ_{33} being the first and δ_{11} being the sixth. (See Figure 6.) Of course, noise and possible contributions from other anisotropic interactions in the signal F will cause additional zero-crossings to appear in the second derivative, $F(\delta) * N''(\delta; 0, t)$. Unfortunately, no general method exists for selecting candidates for δ_{33} and δ_{11} from these zero-crossings that will choose appropriate values for every possible signal F. Thus, some input from the user will be necessary or a heuristic must be adopted to guide the algorithm to the best selections in many cases. The knowledge that six zero-crossings are expected and that the sign of the slope of the second derivative at each of the six crossings is known can be used to help evaluate the zero-crossings of $F(\delta) * N''(\delta; 0, t)$ as candidates for δ_{33} and δ_{11}. In particular, the third derivative should be negative at the first crossing and then should alternate between positive and negative until reaching the sixth crossing. (See Figure 6.) In addition, the strength of the edge at each zero-crossing can be computed by examining the length of the interval between the two relative extrema in $F(\delta) * N''(\delta; 0, t)$ that most closely border each zero crossing. Larger such intervals correspond to stronger edges and will be more likely to be one of the six anticipated zero-crossings.

Finally, the discontinuity at δ_{22} in $I(\delta; \delta_{33}, \delta_{22}, \delta_{11})$ corresponds closely to the location of the maximum in $F(\delta) * N(\delta; 0, t)$ and thus can be found by using Property 2 or by finding the minimum of $F(\delta) * N''(\delta; 0, t)$. Further fitting of $\delta_{33}, \delta_{22}, \delta_{11}$ can be performed using a simple grid search or other methods.

6 Example Application

To demonstrate the edge detection algorithm of the previous section, we will apply it to the SSNMR powder spectrum for [$^{13}C_1$]-leucine (Cambridge Isotope Laboratories: Cambridge, MA). The spectrum was obtained on a Bruker 300 MHz DSX spectrometer using a 4mm MAS Bruker probe without spinning. Our analysis of the data is limited to the region of the spectrum containing the signal from the labeled carbonyl. The spectrum was recorded at room temperature and is referenced to TMS via the literature [39, 40].

The algorithms from this paper were implemented in a python program which is presented in the appendix. In particular, this program uses the following approach.

1. Input the experimental spectrum from a comma separated values (csv) file.
2. Create the first and second derivatives of $F(\delta) * N(\delta; 0, t)$ using a large value of t to smooth the noise as described in section 5 and using the AGM to compute the intensity $I(\delta; \delta_{33}, \delta_{22}, \delta_{11})$ as in Eq. (26).
3. Locate the zero-crossings of $F(\delta) * N'(\delta; 0, t)$ and $F(\delta) * N''(\delta; 0, t)$ along with the minimum of $F(\delta) * N''(\delta; 0, t)$.
4. Determine the edge strength and value of the third derivative at each zero-crossing of the second derivative.
5. Request that the user then select an estimate for $\delta_{33}, \delta_{22}, \delta_{11}$ and a guess for the standard deviation s of the Gaussian for broadening.
6. Refine the value of s by fitting the simulated spectrum to the given data via a simple grid search.
7. Output the resulting simulated spectrum to a csv file.

Applying our program to the experimental powder spectrum for [$^{13}C_1$]-leucine, we found the minimum of the second derivative, $F(\delta) * N''(\delta; 0, t)$, to occur at 179.8 ppm, providing an estimate of δ_{22}. The program also found the zero-crossings for the second derivative, their edge strengths, and the sign of third derivative as reported in Table 1. Clearly, the zero-crossings at 79.3 ppm and 259.5 ppm have the smallest edge strengths, making them unlikely candidates for δ_{33} and δ_{11}. In addition, we know that the third derivative at δ_{33} should be negative, so 89.7 is not δ_{33}. Hence, we estimate that $\delta_{33} = 114.5$ and $\delta_{11} = 238.8$ ppm. (See Figure 7.) A simple grid search suggests a good fit to the data via Gaussian broadening with standard deviation $s = 6.8$ ppm, as seen in Figure 8. From the literature, we were able to find two sets of chemical shielding data for the tensor principal values of [$^{13}C_1$]-Leucine which were, $\delta_{33} = 109, \delta_{22} = 177, \delta_{11} = 243$ [39] and $\delta_{33} = 108, \delta_{22} = 180, \delta_{11} = 242$ [40]. Both of these sets were determined using MAS but fit our values from a static powder reasonably well.

Zero-crossing of second derivative	Edge strength	Sign of third derivative
79.3	9.3	-
89.7	17.6	+
114.5	36.3	-
148.7	13.5	+
170.5	13.5	-
187.0	13.5	+
218.1	41.4	-
238.8	11.4	+
255.4	12.4	-
259.5	8.3	+

Table 1: Our program computed the zero-crossings of the second derivative along with the edge strength and the sign of the third derivative at each. Values 79.3 ppm and 259.5 ppm have the smallest edge strengths and are not viewed as likely values of δ_{33} and δ_{11}. Since the third derivative at δ_{33} should be negative, we rule out 89.7 ppm as a possibility for δ_{33}, and estimate $\delta_{33} = 114.5$ ppm and $\delta_{11} = 238.8$ ppm.

7 Summary

The methods presented here provide an efficient approach to computing one-dimensional SSNMR powder patterns and determining the principal values of the chemical shielding tensor. The arithmetic-geometric mean provides a particularly fast method for computing the elliptic integral involved in simulating the powder spectrum for a nonaxially symmetric powder. Marr-Hildreth edge detection uses the zeros of the second derivative of a broadened experimental spectrum to find candidate values for δ_{33} and δ_{11}. These values are evaluated by considering the edge strength and the sign of the third derivative at each value. Combined with knowledge of the shape of the third derivative of a broadened ideal spectrum, we can then determine the best estimates for δ_{33} and δ_{11}. Finally, δ_{22} can be found by looking for the minimum value of the second derivative. In our example, the algorithms gave values for δ_{33}, δ_{22}, and δ_{11} for [^{13}C$_1$]-Leucine that were consistent with the literature.

8 Appendix

The program in this appendix implements the algorithm described in this paper for simulating one-dimensional solid-state NMR powder spectra using the arithmetic-geometric mean and for finding the principal values of the chemical shielding tensor using edge detection. This program can be downloaded from `http://faculty.mercer.edu/denny_jk/cmr.html`. The program is written in the python programming language (version 2.4) which is freely available at `http://www.python.org`.

```
#Filename: powder.py
#Import the necessary libraries
```

```
import string
import math
import operator

###################################
#                                 #
#       define functions           #
#                                 #
###################################


#*********************************************************************
# Function   : rdcsv                                                 *
# Purpose    : Read in data from a csv file                          *
# Parameters : name (string)                                         *
# Returns    : list containing a list of data in each line of file   *
#*********************************************************************
def rdcsv(name):
   f = open(name,"r")
   data = []
   tmp = f.readline()
   while tmp !="":
       onerow = tmp.split(",")
       last = len(onerow)-1
       k = len(onerow[last])
       if k!= 1:
          onerow[last] = onerow[last][0:k-1]
       if len(onerow) !=  0:
          data.append(onerow)
       tmp = f.readline()
   f.close()
   return(data)

#*********************************************************************
# Function   : outcsv                                                *
# Purpose    : Write data out to a csv file                          *
# Parameters : name (string)                                         *
#              data (list of data points)                            *
# Returns    : n/a                                                   *
#*********************************************************************
def outcsv(name,data):
   f = open(name,"w")
   for i in range(0,len(data)):
       tmp = ""
```

```
        for j in range(0,len(data[i])):
            tmp = tmp + str(data[i][j]) + ","
        tmp = tmp[0:len(tmp)-1]+"\n"
        f.writelines(tmp)
    f.close()

#**********************************************************************
# Function   : norm                                                   *
# Purpose    : Integrate the data using Simpson's Rule                *
# Parameters : data (list of data points)                             *
# Returns    : integral of the data                                   *
#**********************************************************************
def norm(data):
    sum=0
    dx = abs(data[1][0]-data[0][0])
    for i in range(0,len(data)):
        sum+=data[i][1]*2**((i % 2)+1)
    return(sum*dx/3.0)

#**********************************************************************
# Function   : normalize                                              *
# Purpose    : Normalize data so that the integral is 1               *
# Parameters : data (list of data points)                             *
# Returns    : list of data points                                    *
#**********************************************************************
def normalize(data):
    area = norm(data)
    for i in range(0,len(data)):
        data[i][1] = data[i][1]/area
    return(data)

#**********************************************************************
# Function   : agm                                                    *
# Purpose    : Compute the arithmetic-geometric mean M(a,b)           *
# Parameters : a,b (float)                                            *
# Returns    : float                                                  *
#**********************************************************************
def agm(a,b):
    a1 = a
    b1 = b
    for i in range(0,5):
        a2 = (a1+b1)/2.0
        b2 = math.sqrt(a1*b1)
        a1=a2
        b1=b2
```

```
   return(a1)

#*********************************************************************
# Function   : intensity                                             *
# Purpose    : Compute the intensity function for the ideal powder   *
#              spectrum (nonaxially symmetric case)                  *
# Parameters : delta,d1,d2,d3 (float)                                *
# Returns    : float                                                 *
#*********************************************************************
def intensity(delta,d3,d2,d1):
   if (delta<=d3):
       return(0)
   elif (delta<d2):
       m = math.sqrt((d1-d2)*(delta-d3))
       kappa = math.sqrt(((d1-delta)*(d2-d3))/((delta-d3)*(d1-d2)))
       return(1/(2*m*kappa*agm(1,math.sqrt(1-1/(kappa*kappa)))))
   elif (delta==d2):
       return(intensity(d2+.0001,d3,d2,d1))
   elif (delta<d1):
       m = math.sqrt((d1-d2)*(delta-d3))
       kappa = math.sqrt(((d1-delta)*(d2-d3))/((delta-d3)*(d1-d2)))
       return(1/(2*m*agm(1,math.sqrt(1-kappa*kappa))))
   else:
       return(0)


#*********************************************************************
# Function   : convolution                                           *
# Purpose    : Compute the convolution of two data sets using        *
#              Simpson's Rule                                        *
# Parameters : a,b (list)                                            *
#              dx (float)                                            *
# Returns    : list of floats                                        *
#*********************************************************************
def convolution(a,b,dx):
   c = []
   if (len(a)!=len(b)):
       print("ERROR")
   else:
       for i in range(0,2*len(a)):
           c.append(0)
           for j in range(0,len(a)):
               k = i-j
               if not((k<0) or (k>len(a)-1)):
                   c[i]+=a[k]*b[j]*2**((j % 2)+1)
```

```
                else:
                    c[i]+=0
            c[i] = c[i]*dx/3.0

    return(c)

#*********************************************************************
# Function   : Gaussian                                              *
# Purpose    : Compute the Gaussian at x with mean mu                *
#              and standard deviation sigma                          *
# Parameters : x,mu,sigma (float)                                    *
# Returns    : float                                                 *
#*********************************************************************
def Gaussian(x,mu,sigma):
    pi = 4*math.atan(1)
    return(1/(sigma*math.sqrt(2*pi))*math.exp(-((x-mu)*(x-mu))/
                                                (2*sigma*sigma)))

#*********************************************************************
# Function   : GaussianFirstDeriv                                    *
# Purpose    : Compute the first derivative of the Gaussian at       *
#              x with mean mu and standard deviation sigma           *
# Parameters : x,mu,sigma (float)                                    *
# Returns    : float                                                 *
#*********************************************************************
def GaussianFirstDeriv(x,mu,sigma):
    pi = 4*math.atan(1)
    coeff = -(x-mu)/(sigma*sigma*sigma*math.sqrt(2*pi))
    exponential = math.exp(-((x-mu)*(x-mu))/(2*sigma*sigma))
    return(coeff*exponential)

#*********************************************************************
# Function   : GaussianSecondDeriv                                   *
# Purpose    : Compute the second derivative of the Gaussian at      *
#              x with mean mu and standard deviation sigma           *
# Parameters : x,mu,sigma (float)                                    *
# Returns    : float                                                 *
#*********************************************************************
def GaussianSecondDeriv(x,mu,sigma):
    pi = 4*math.atan(1)
    coeff = 1/(sigma*sigma*sigma*math.sqrt(2*pi))*((x-mu)*(x-mu)/
                                                (sigma*sigma)-1)
    exponential = math.exp(-((x-mu)*(x-mu))/(2*sigma*sigma))
    return(coeff*exponential)
#*********************************************************************
```

```
# Function   : GaussianThirdDeriv                                    *
# Purpose    : Compute the third derivative of the Gaussian at      *
#              x with mean mu and standard deviation sigma          *
# Parameters : x,mu,sigma (float)                                   *
# Returns    : float                                                *
#*******************************************************************
def GaussianThirdDeriv(x,mu,sigma):
   pi = 4*math.atan(1)
   coeff = 1/(sigma**7*math.sqrt(2*pi))*((x-mu)*(3*sigma*sigma-x*x
                                                    +2*x*mu-mu*mu))
   exponential = math.exp(-((x-mu)*(x-mu))/(2*sigma*sigma))
   return(coeff*exponential)

#*******************************************************************
# Function   : findzeros                                            *
# Purpose    : Compute the list of independent variable values at   *
#              which the dependent variable values cross the        *
#              independent axis                                     *
# Parameters : data (list of data points)                           *
# Returns    : list of zero crossings                               *
#*******************************************************************
def findzeros(data):
   zeros = []
   for i in range(0,len(data)-1):
       if data[i][1]*data[i+1][1]<0:
           zeros.append(data[i])
   return(zeros)

#*******************************************************************
# Function   : phi                                                  *
# Purpose    : Compute l^2 norm of the difference between the data  *
#              lists                                                *
# Parameters : data1 (list of data points)                          *
#              data2 (list of data points)                          *
# Returns    : float                                                *
#*******************************************************************
def phi(data1,data2):
    dx = data1[1][0]-data1[0][0]
    k=abs(int(round((data1[0][0]-data2[0][0])/dx)))
    sum = 0
    for i in range(0,len(data1)):
        sum += (data1[i][1] - data2[k+i][1])**2
    norm = math.sqrt(sum*dx)
    return(norm)
```

```
#*********************************************************************
# Function   : edgedetect                                            *
# Purpose    : Compute possible independent variable values at which*
#              the data has an edge and at which the data will have *
#              a maximum                                             *
# Parameters : data (list of data points)                           *
# Returns    : list of first deriv. zero crossings                  *
#              list of second deriv. zero crossings                 *
#              list of minimum values of the second deriv.          *
#*********************************************************************
def edgedetect(data):
    #compute stepsize dx from the data set
    dx = abs(data[1][0]-data[0][0])

    #extract the dependent variable values from the
    #data set for analysis
    n = len(data)
    dep_var_values = []
    for i in range(0,n):
       dep_var_values.append(data[i][1])

    #generate first and second derivatives of the broadening function
    gx0 = -n*dx/2.0
    gaussian_first_deriv = []
    gaussian_second_deriv = []
    indep_var_values = []
    for i in range(0,n):
       indep_var_values.append(gx0+i*dx)
       #make sigma large to smooth noise
       sigma  = 5.0
       gaussian_first_deriv.append(GaussianFirstDeriv(gx0+i*dx,0,sigma))
       gaussian_second_deriv.append(GaussianSecondDeriv(gx0+i*dx,0,sigma))

    #compute the convolutions
    convolved_first_deriv =
                        convolution(dep_var_values,gaussian_first_deriv,dx)
    convolved_second_deriv =
                        convolution(dep_var_values,gaussian_second_deriv,dx)

    #Reconstruct data points
    gaussian_first_deriv_pts=[]
    gaussian_second_deriv_pts=[]
    for i in range(0,n):
       gaussian_first_deriv_pts.append([indep_var_values[i],
                                                gaussian_first_deriv[i]])
```

```
        gaussian_second_deriv_pts.append([indep_var_values[i],
                                                gaussian_second_deriv[i]])

    convol_first_deriv_pts=[]
    convol_second_deriv_pts=[]
    for i in range(0,len(convolved_first_deriv)):
        convol_first_deriv_pts.append([indep_var_values[0]+data[0][0]+i*dx,
                                                convolved_first_deriv[i]])
        convol_second_deriv_pts.append([indep_var_values[0]+data[0][0]+i*dx,
                                                convolved_second_deriv[i]])

    #find the zeros of the first and second derivatives
    first_deriv_zeros= findzeros(convol_first_deriv_pts)
    second_deriv_zeros= findzeros(convol_second_deriv_pts)

    #find the region of the minimum of the second derivative
    convol_second_deriv_pts_sorted = sorted(convol_second_deriv_pts,
                                                key=operator.itemgetter(1))
    min_second_deriv = convol_second_deriv_pts_sorted[0:5]

    return(first_deriv_zeros,second_deriv_zeros,min_second_deriv)

#*****************************************************************
# Function   : gridsearch                                        *
# Purpose    : Use a simple grid search to optimize the fitting.  *
# Parameters : data (list of data points)                         *
#              delta33,delta22,delta11,sigma (float)              *
# Returns    : list of data for fitted, simulated spectrum        *
#*****************************************************************
def gridsearch(data,delta33,delta22,delta11,sigma):
    n = len(data) - (len(data) % 2)
    dx = abs(data[1][0]-data[0][0])
    small = [1.0,0]
    numpts = 20
    step = 2.0/float(numpts)
    for j in range(0,numpts+1):
        stdev = sigma-1+j*step

        #produce the ideal spectrum
        x0 = data[0][0]
        simulated_dep = []
        simulated_indep = []
        for i in range(0,n):
            simulated_indep.append(x0+dx*i)
            simulated_dep.append(intensity(x0+dx*i,delta33,delta22,delta11))
```

```
        #produce the gaussian
        gx0 = -n*dx/2.0
        gaussian_indep = []
        gaussian_dep = []
        for i in range(0,n):
           gaussian_indep.append(gx0+i*dx)
           gaussian_dep.append(Gaussian(gx0+i*dx,0,stdev))

        #do the convolution
        broadened = convolution(simulated_dep,gaussian_dep,dx)

        #add dependent coordinates
        broadened_pts=[]
        gaussian_pts=[]
        for i in range(0,n):
            gaussian_pts.append([gaussian_indep[i],gaussian_dep[i]])
        for i in range(0,len(broadened)):
            broadened_pts.append([gx0+simulated_indep[0]+i*dx,broadened[i]])

        broadened_pts=normalize(broadened_pts)
        r = phi(data,broadened_pts)
        print "||data - broadened|| = "+str(r)
        print [r,stdev]

        #compare and update
        if r < small[0]:
            small = [r,stdev]
            print small
    print small
    return(broadened_pts)

#*********************************************************************
# Function   : gatherinformation                                     *
# Purpose    : Collect strength and third deriv. values for use      *
#              in determining key zero-crossings of second deriv.    *
# Parameters : info (list of the results from edgedetect function    *
#              data (list of data points)                            *
# Returns    : list of data for fitted, simulated spectrum           *
#*********************************************************************
def gatherinformation(info,data):
    second = info[1]

    #generate the third derivative
    dx = abs(data[1][0]-data[0][0])
```

```
    n = len(data)
    dep_var_values = []
    for i in range(0,n):
       dep_var_values.append(data[i][1])
    gx0 = -n*dx/2.0
    gaussian_third_deriv = []
    indep_var_values = []
    for i in range(0,n):
       indep_var_values.append(gx0+i*dx)
       #make sigma large to smooth noise
       sigma  = 5.0
       gaussian_third_deriv.append(GaussianThirdDeriv(gx0+i*dx,0,sigma))
    convolved_third_deriv = convolution(dep_var_values,gaussian_third_deriv,dx)
    convol_third_deriv_pts=[]
    for i in range(0,len(convolved_third_deriv)):
       convol_third_deriv_pts.append([indep_var_values[0]+data[0][0]+i*dx,
                                                    convolved_third_deriv[i]])

    third = findzeros(convol_third_deriv_pts)

    result = []

    for i in range(0,len(second)):
        for j in range(i,len(third)):
            if (second[i][0] > third[j][0]) and (second[i][0]<third[j+1][0]):
                 result.append([second[i][0],third[j+1][0]-third[j][0]])

    #find the value of the third deriv. at each zero-crossing
    for i in range(0,len(result)):
        for j in range(0,len(convol_third_deriv_pts)):
            if (result[i][0] == convol_third_deriv_pts[j][0]):
                result[i].append(convol_third_deriv_pts[j][1])

    return(result)

####################################
#                                  #
#       main section of code       #
#                                  #
####################################

#
#Edge Detection routine
#
```

```
#read in data
filename = "spectrum.csv"
data = rdcsv(filename)
data.reverse()
#convert data from strings to floats
for i in range(0,len(data)):
   for j in range(0,2):
       data[i][j] = float(data[i][j])
#normalize the data set
data = normalize(data)

results = edgedetect(data)

print "The approximate zeros of the first derivative are:"
for i in range(0,len(results[0])):
    print results[0][i]
print
print "The region of the minimum of the second derivative is:"
for i in range(0,len(results[2])):
    print results[2][i]
print
print "Use the above values to approximate delta22."
print
print "Approximate zeros of the second derivative:"
print
information = gatherinformation(results,data)
print "Edge candidate\t edge strength\t sign of third derivative"
for i in range(0,len(information)):
    print str(information[i][0])+"\t "+str(information[i][1])+"\t "
                                                        +str(information[i][2])
print

print "Now, use the above informatio to determine values for"
print "delta33 < delta22 < delta11."
delta33 = float(raw_input("Enter delta33: "))
delta22 = float(raw_input("Enter delta22: "))
delta11 = float(raw_input("Enter delta11: "))
sigma = float(raw_input("Enter a guess for the standard deviation of
                                              the Gaussian for broadening:"))
print

fit_simulation = gridsearch(data,delta33,delta22,delta11,sigma)
fit_simulation.reverse()
data.reverse()
outcsv("fit.csv",fit_simulation)
```

```
outcsv("newdata.csv",data)
```

Acknowledgments

The authors thank Dr. J.R. Quine at Florida State University for suggesting this investigation and Dr. L. Thompson at the University of Massachusetts at Amherst for the use of her spectrometer.

References

[1] Ketchem RR, Hu W, Cross TA. 1993. High-resolution conformation of gramicidin A in a lipid bilayer by solid-state NMR. Science 261:1457-1460.

[2] Cross TA, Quine JR. 2000. Protein structure in anisotropic environments: development of orientational constraints. Concepts Magn Reson 12:55-70.

[3] Quine JR, Cross TA, Chapman MS, Bertram R. 2004. Mathematical aspects of protein structure determination with NMR orientational restraints. Bulletin Math Biol 66:1705-1730.

[4] Smurnyy Y, Opella SJ. 2006. Calculating protein structures directly from anisotropic spin interaction constraints. Magn Res Chem 44:283-293.

[5] Gerstein BC, Dybowski CR. Transient techniques in NMR of solids: an introduction to theory and practice. New York: Academic Press; 1985.

[6] Duer MJ. Introduction to Solid-State NMR Spectroscopy. Oxford: Blackwell Science; 2004.

[7] Sherwood MH. Chemical shift tensors in single crystals. In: Grant DM, Harris RK, editors. Encyclopedia of NMR. New York: Wiley; 1995. p. 1298-1320.

[8] Varner SJ, Vold RL, Hoatson GL. 1996. An efficient method for calculating powder patterns. J Mag Res 123:72-80.

[9] Haeberlen U. High Resolution NMR in Solids. New York: Academic Press; 1976.

[10] Mehring M. Principles of High Resolution NMR in Solids. New York: Springer-Verlag; 1983.

[11] Zaremba SK. 1966. Good lattice points, discrepancy, and numerical integration. Ann Mat Pure Appl 4-73:293.

[12] Conroy H. 1967. Molecular Schrodinger equation. VIII. A new method for the evaluation of multidimensional integration. J Chem Phys 47:5307-5318

[13] Cheng VB, Suzukawa H, Wolfsberg M. 1973. Investigations of a nonrandom numerical method for multidimensional integration. J Chem Phys 59:3992-3999.

[14] Alderman DW, Solum MS, Grant DM. 1986. Methods for analyzing spectroscopic line shapes: NMR solid powder patterns. J Chem Phys 84:3717-3725.

[15] Teng Q, Iqbal M, Cross TA. 1992. Determination of the C-13 chemical shift and N-14 electric field gradient tensor orientations with respect to the molecular frame in a polypeptide. J Am Chem Soc 114:5312-5321.

[16] Wang C, Teng Q, Cross TA. 1992. Solid-state C-13 NMR spectroscopy of a C-13 carbonyl-labeled polypeptide, Biophy J 61:1550-1556.

[17] Mombourquette MJ, Weil JA. 1992. Simulation of Magnetic Resonance Powder Spectra. J Mag Res 99:37-44.

[18] Andreozzi L,Giordano M, Leporini D. 1993. A fast algorithm for magnetic resonance lineshapes of powder samples. J Mag Res A 104:166-171.

[19] Wang D, Hanson GR. 1995. A New Method for Simulating Randomly Oriented Powder Spectra in Magnetic Resonance: The Sydney Opera House (SOPHE) Method. J Mag Res A 117:1-8.

[20] Bak M, Nielson NC. 1997. REPULSION, A novel approach to efficient powder averaging in solid-state NMR. J Mag Res 125:132-139.

[21] Ponti A. 1999. Simulation of magnetic resonance static powder lineshapes: A quantitative assessment of spherical codes. J Mag Res 138:288-297.

[22] Ponti A. 1999. Simulation of one-dimensional magnetic resonance static powder lineshapes reduced to area computation. Chem Phys Lett 302:224-230.

[23] Hodgkinson P, Emsley L. 2000. Numerical simulation of solid-state NMR experiments. Prog NMR Spectrosc 36:201-239.

[24] Eden M. 2003. Computer simulations in solid-state NMR. III. Powder averaging. Concepts Magn Reson 18A:24-55.

[25] Stevensson B, Eden M. 2006. Efficient orientational averaging by the extension of Lebedev grids via regularized octahedral symmetry expansion. J Mag Res 181:162-176.

[26] Gauss CF, Werke Vol. 3. Göttingen; 1866.

[27] Lagrange JL, Œuvres Vol. 2, Gauthier-Villars, Paris; 1868.

[28] Borwein JM, Borwein PB. Pi and the AGM: A study in analytic number theory and computational complexity. New York: John Wiley and Sons; 1987.

[29] Cox DA. 1984. The arithmetic-geometric mean of Gauss. L'Enseign Math 30:275-330.

[30] Hijab O. Introduction to Calculus and Classical Analysis. New York: Springer-Verlag; 1997. p. 182-188.

[31] Nievergelt Y, Coomes J. UMAP Module 774: Elliptic integrals and elliptic functions in calculus and beyond. In: Tools for Teaching 1999, Lexington, MA: COMAP; 1999.

[32] Weisstein EW. CRC concise encyclopedia of mathematics. New York: CRC Press; 1999.

[33] Poole CP. Electron Spin Resonance. New York: John Wiley and Sons; 1983.

[34] Oas TG, Drobny GP, Dahlquist FW. 1988. A new iterative least-squares method for the extraction of NMR parameters from nonideal powder patterns. J Mag Res 78:408-424.

[35] Kim AJ, Butler LG. 1992. Nonlinear least-squares fitting procedure for solid-state NMR powder patterns. Concepts Magn Reson 4:205-226.

[36] Koons JM, Hughes E, Cho HM, Ellis PD. 1995. Extracting multitensor solid-state NMR parameters from lineshapes. J Mag Res 114:12-23.

[37] Zorich VA. Mathematical Analysis II. New York: Springer-Verlag; 2004.

[38] Marr D, Hildreth E. 1980. Theory of Edge Detection. Proc R Soc Lond B 207:187-217.

[39] Ye C, Fu R, Hu J, Hou L, Ding S. 1993. Carbon-13 chemical shift anisotropies of solid amino acids. Mag Res Chem 13:699-704.

[40] Gu Z, McDermott A. 1993. Chemical shielding anisotropy of protonated and deprotonated carboxylates in amino acids. JACS 115:4282-4285.

Figure 1

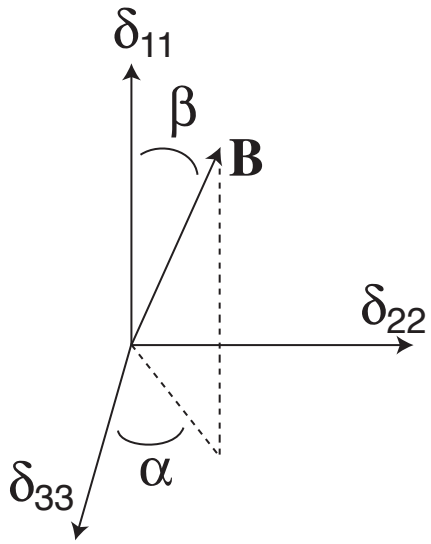


Figure 2

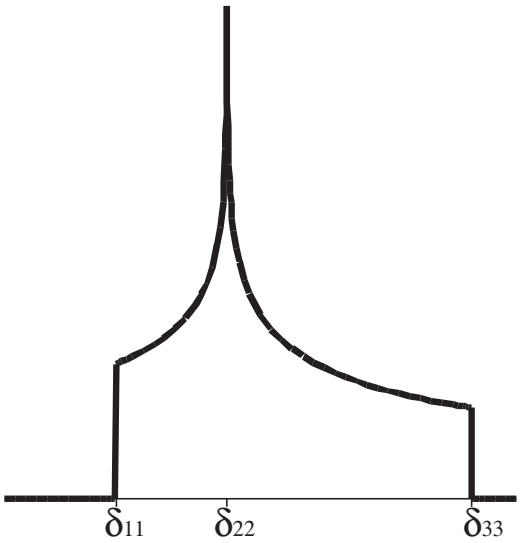
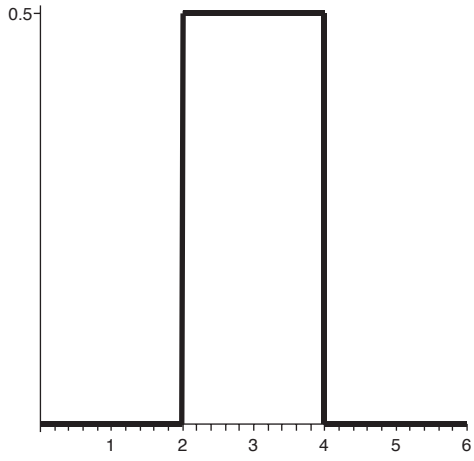
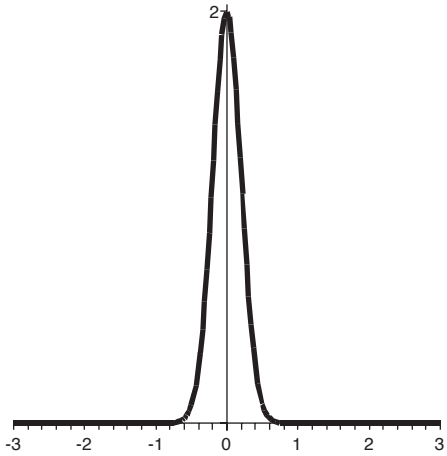


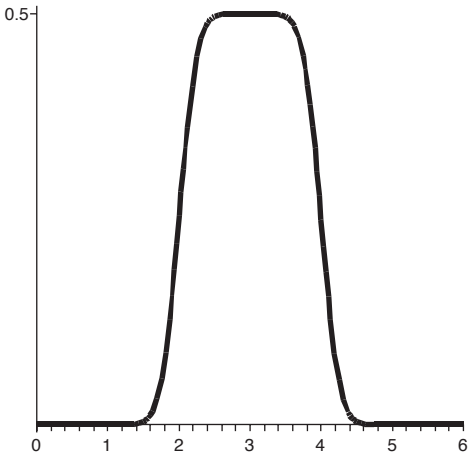
Figure 3



(a)



(b)



(c)

Figure 4

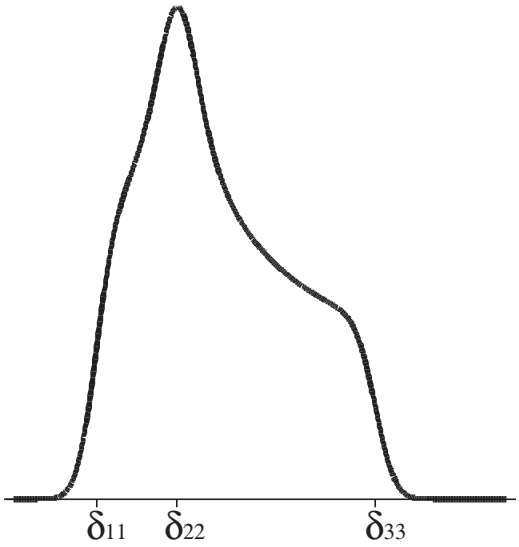


Figure 5

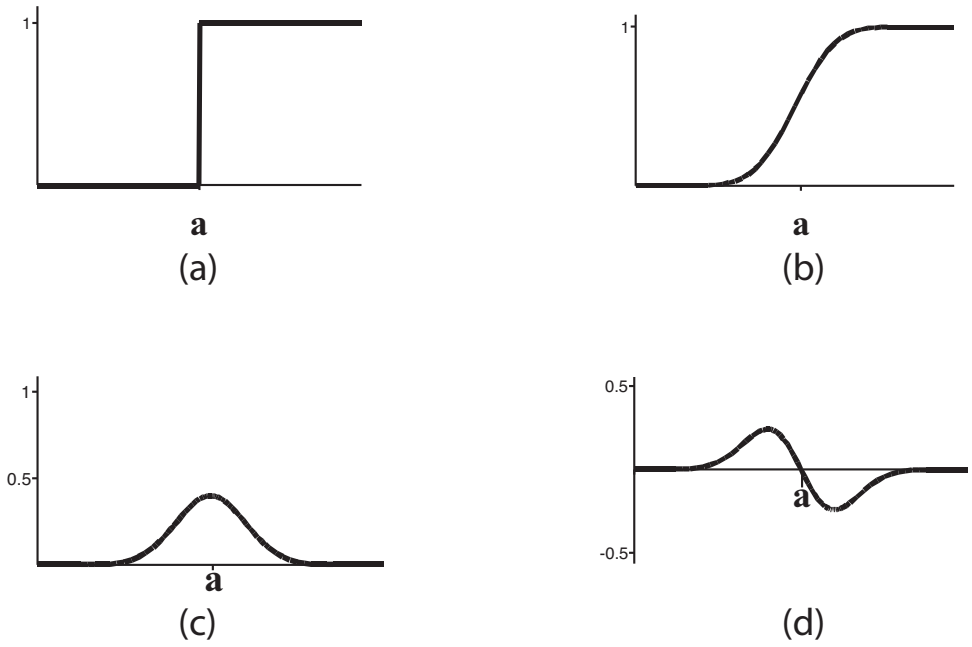


Figure 6

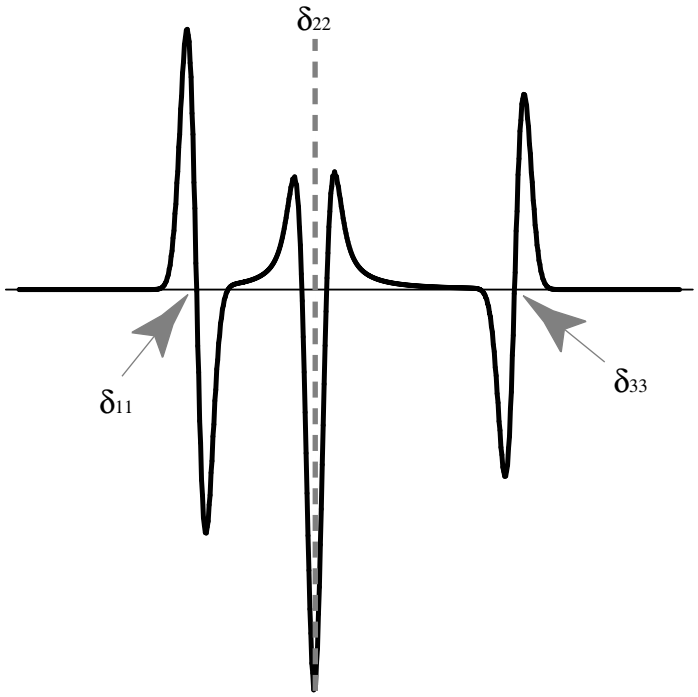


Figure 7

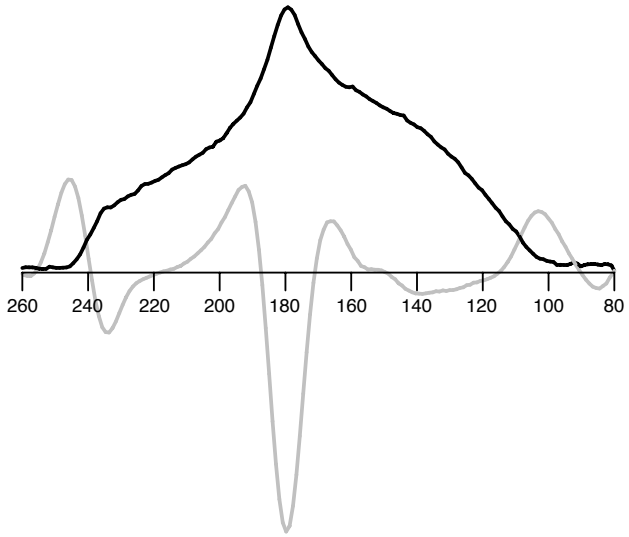
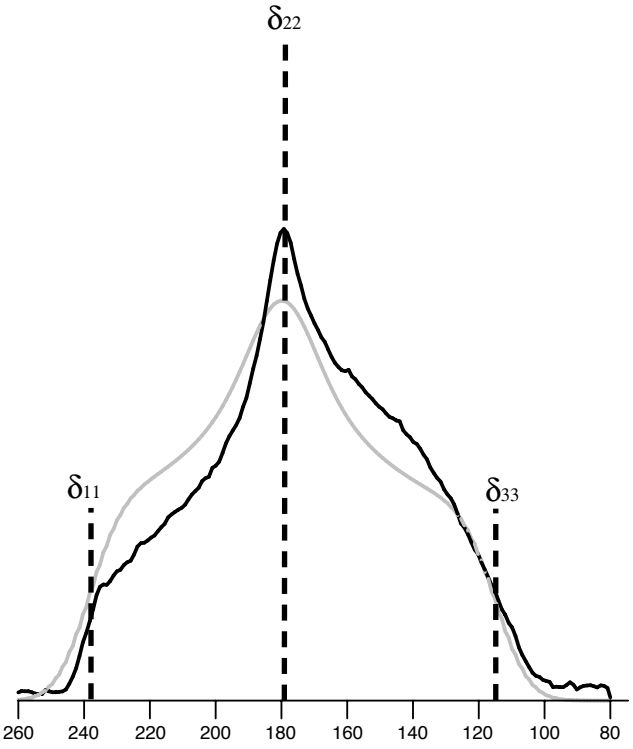


Figure 8



Captions for "Computing and Fitting SSNMR Powder Patterns with the Arithmetic-Geometric Mean and Edge Detection"
by
Denny, Daniel, and Kovacs

Figure 1

The angles α and β are the spherical coordinates of the unit magnetic field direction vector, $\mathbf{B}$, in the principal axis frame (PAF).

Figure 2

Given the principal values $\delta_{33}, \delta_{22}, \delta_{11}$ of the chemical shielding tensor, the arithmetic-geometric mean (AGM) gives an efficient method for computing each point, $I(\delta; \delta_{33}, \delta_{22}, \delta_{11})$, on a nonaxially symmetric ideal powder spectrum.

Figure 3

The square wave (a) is convolved with $N(x; 0, 0.2)$ shown in (b). The convolution operation results in the function plotted in (c) which is continuous and differentiable and is a mixture of the shapes from (a) and (b).

Figure 4

An ideal powder spectrum is convolved with a Gaussian to produce this broadened, smooth ideal spectrum, $I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) * N(\delta; 0, s)$.

Figure 5

As an elementary example of edge detection, consider the broadened spectrum $\chi_{[a,\infty)} * N(x; 0, s)$, shown in (b). The value a at which the ideal spectrum, $\chi_{[a,\infty)}$, in (a) has its step can be found exactly by finding the inflection point in the broadened spectrum (b). The inflection point occurs at the maximum of the first derivative (c) or at the zero-crossing of the second derivative (d). For more complex ideal spectra, the location of the inflection point of the broadened spectrum is a close approximation to the location of a jump in the ideal spectrum.

Figure 6

The second derivative of the broadened ideal powder spectrum, $I(\delta; \delta_{33}, \delta_{22}, \delta_{11}) * N''(\delta; 0, t)$, has six zero-crossings. The first and last zero-crossing estimate the values of δ_{33} and δ_{11}, respectively. Note that the slope of the second derivative (and thus the value of the third derivative) is negative at δ_{33} and alternates until reaching δ_{11}. In addition, the minimum of the second derivative occurs at δ_{22}.

Figure 7

Edge detection is applied to an SSNMR powder spectrum, $F(\delta)$, for [$^{13}C_1$]-leucine. The plot shows the experimental spectrum in black containing the signal from the labeled carbonyl and the second derivative $F(\delta) * N''(\delta; 0, 5)$ in gray. The zero-crossings of the second derivative are summarized in Table 1. Based on the strength of the edges and the values of the third derivatives at these zero-crossings, we estimate that $\delta_{33} = 114.5$ ppm and $\delta_{11} = 238.8$ ppm. In addition, the minimum of the second derivative corresponds to the maximum of $F(\delta)$, giving $\delta_{22} = 179.8$ ppm.

Figure 8

The ideal spectrum $I(\delta; 114.5, 178.9, 238.8)$ broadened by convolution with the Gaussian $N(\delta; 0, 6.8)$ is plotted in gray with the experimental spectrum in black. The asymmetry in the experimental spectrum may represent the contribution of another anisotropic interaction to the spectrum.