2 STRAFI

2.1 Overview

This document deals with simulations of NMR experiments in stray fields: STRAFI. The examples herein make extensive use of the Class ***sys_gradz*** which defines an isotropic spin system in a Bo z-gradient.

2.2 STRAFI Sections

2.3 STRAFI Figures & Tables

2.4 STRAFI Programs

2.5 Description

The basis for most of GAMMA's STRAFI calculations is class **sys_gradz** which is depicted in the following figure.

Spin System in a z-Gradient

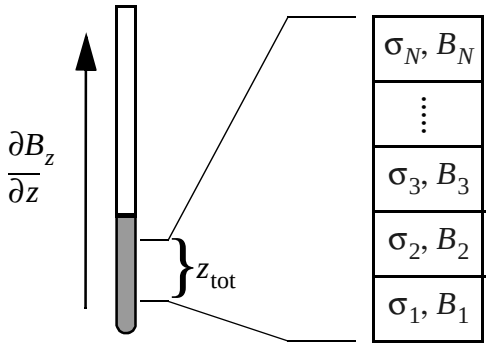

Figure 2-1 Components of ***sys_gradz***, a spin system in a z-gradient. Not shown are the isotropic values shared by all "sub-systems". Only the isotropic shifts change with distance in the sample due to the gradient.

In addition to an isotropic spin system (spins, shifts, Js,...), variables of type **sys_gradz** contain the essential elements to track a spin system in a z-gradient. The user assigns a gradient value (in T/m), a total sample length (m), and the number of sub-systems to treat within the sample. In principle, all calculations performed with such a spin system are the sum of calculations performed with a single sub-system and each sub-system has a particular field value depending upon its distance from the gradient center. Thus there is no diffusion between spins in different sub-systems.

To gain some intuition in dealing with field gradients we can have a look at the applied gradient versus distance in our sample. This will provide an idea of what shift offsets one might expect in a typical field gradient[1].

1. Figure generated from program GzPOffset.cc given at the end of this Chapter, page 65

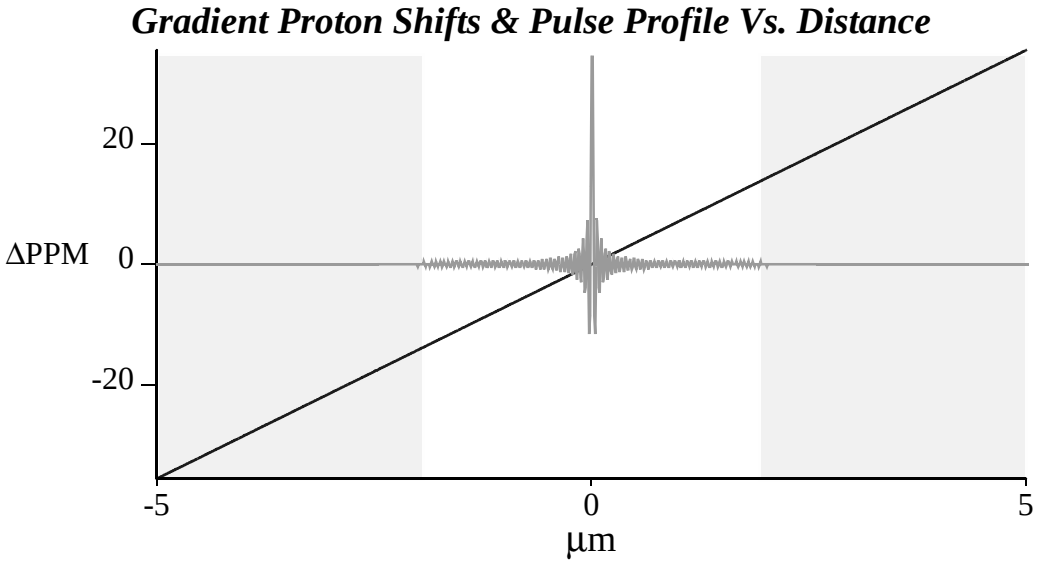

Figure 2-2 **Simulated shift offsets (blue) and excitation profile (green) versus position in a z-Gradient. The gradient was set to 50 T/M and the system was a single proton in a 300 MHz field. The pulse angle was set to 90 on resonance and the length to 10 ms. The yellow shaded region**

The above plot shows that, for proton shift averages, a 10 μm length in a 50 T/m gradient is overkill. Such settings may be fine for spanning other interactions (e.g. dipolar couplings) and/or other isotopes. However: *Blind use of **sys_gradz** can lead to trouble under certain circumstances*. In particular, setting an effective sample length in turn sets the field under which the outer sub-systems evolve (zero gradient is the system center). Much of STRAFI work depends upon pulse power and length. These two factors combine to produce an excitation envelope relative to offset frequency and that implies spins outside this envelope (at large offsets) do not experience **any** pulse effects.

To illustrate this point, the figure also displays a pulse excitation profile following a 90 pulse lasting 10 ms. If such conditions were used in a GAMMA simulation, sample areas in low intensity parts of the excitation profile are unaffected by the pulse. That implies that computations in this region are unfruitful. If such were the case, the effective sample length could be shortened to ~4 μm in order to produce better gradient averaging with no increase in CPU time. Users should insure that their effective sample lengths lie within the applied pulse envelope. A rule of thumb is to take +/- $50/t_p$ as the frequency width (Hz) that is affected by the pulse. This is then matched by the product of the field gradient 6 effective length 6 gamma.

$$\gamma \frac{\Delta B_z}{\Delta z} z \approx \frac{100}{t_p} \qquad z \approx \frac{100}{\gamma t_p} \left\lceil \frac{\Delta z}{\Delta B_z} \right\rceil \tag{2-1}$$

For a 10 ms ^{1}H pulse & a 50 T/m gradient, 50 sinc nodes on each side of zero are spanned in

$$z \approx \frac{100}{(2.7x10^8 radT^{-1}s^{-1})(1cycle/(2\pi rad))(10x10^{-3}\mathrm{s})(50Tm^{-1})} \approx 4.5\mu m \tag{2-2}$$

2.6 Benson & McDonald Simulations

2.6.1 Mag. vs. Distance, Single 1H

As a first test of GAMMA's ability to treat STRAFI problems a simple program has been made which will generate plots of xy-magnetization vs. sample distance in a gradient. In particular, the program will be set to reproduce the 1st figure in the paper by Benson and McDonald[1].

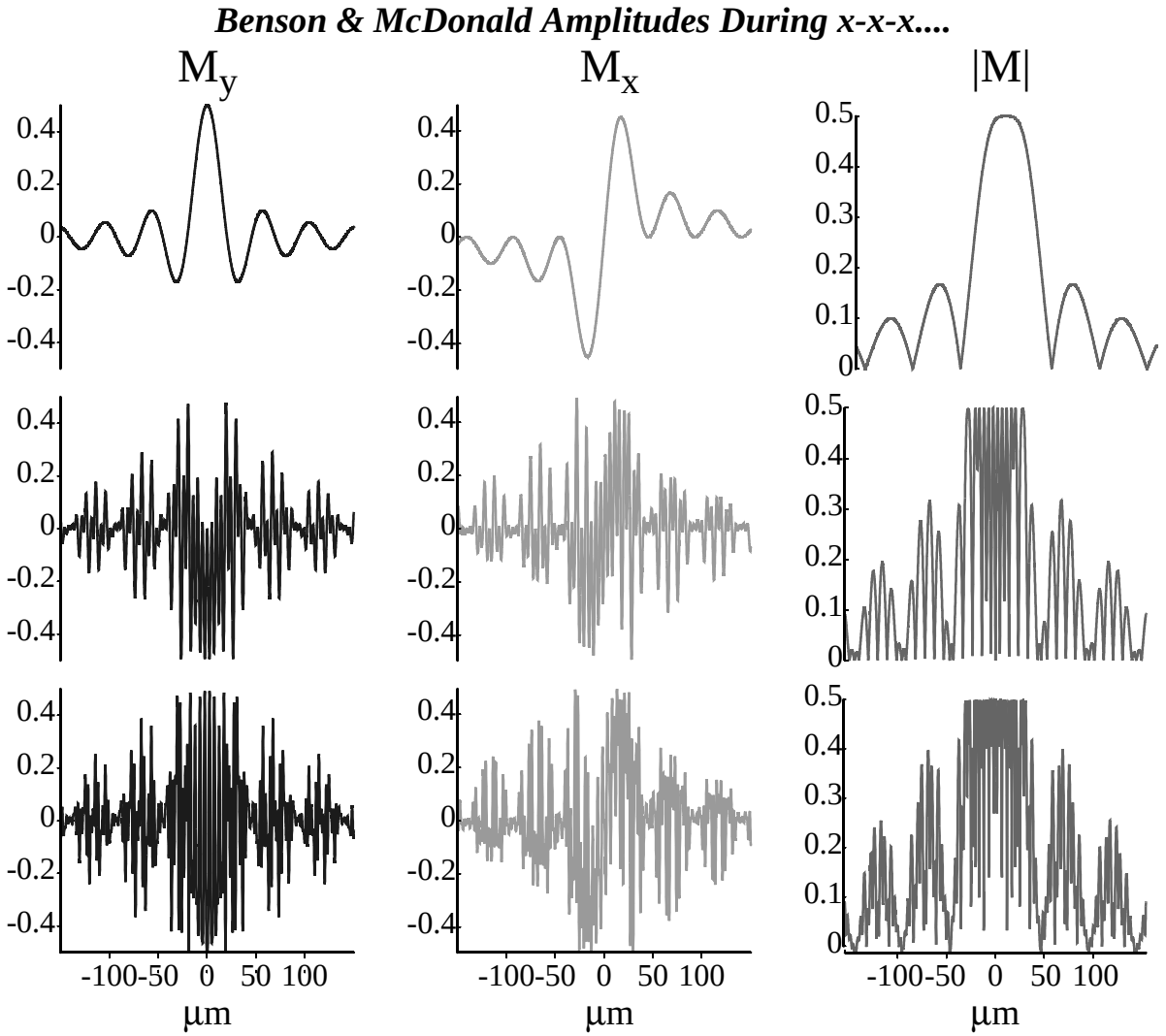

Figure 2-3 **Simulated magnetization during STRAFI sequence 90_x-(τ-90_x-τ–)$_n$. Row 1 are the value immediately after the pulse. Row 2 are those at the 1st STRAFI echo and Row 3 are those at the 2nd STRAFI echo. Here t_p = 10 μs, tau = 50 μs.**

This figure, as well the following one, used parameters similar to those in the citation. However,

1. "Profile Amplitude Modulation in Stray-Field Magnetic-Resonance Imaging", T.B. Benson and P.J. McDonald, *J. Magn. Reson. Ser. A*,**112**, 17-23, (1995).

GAMMA uses a density operator treatment rather than the Bloch treatment used by the authors.

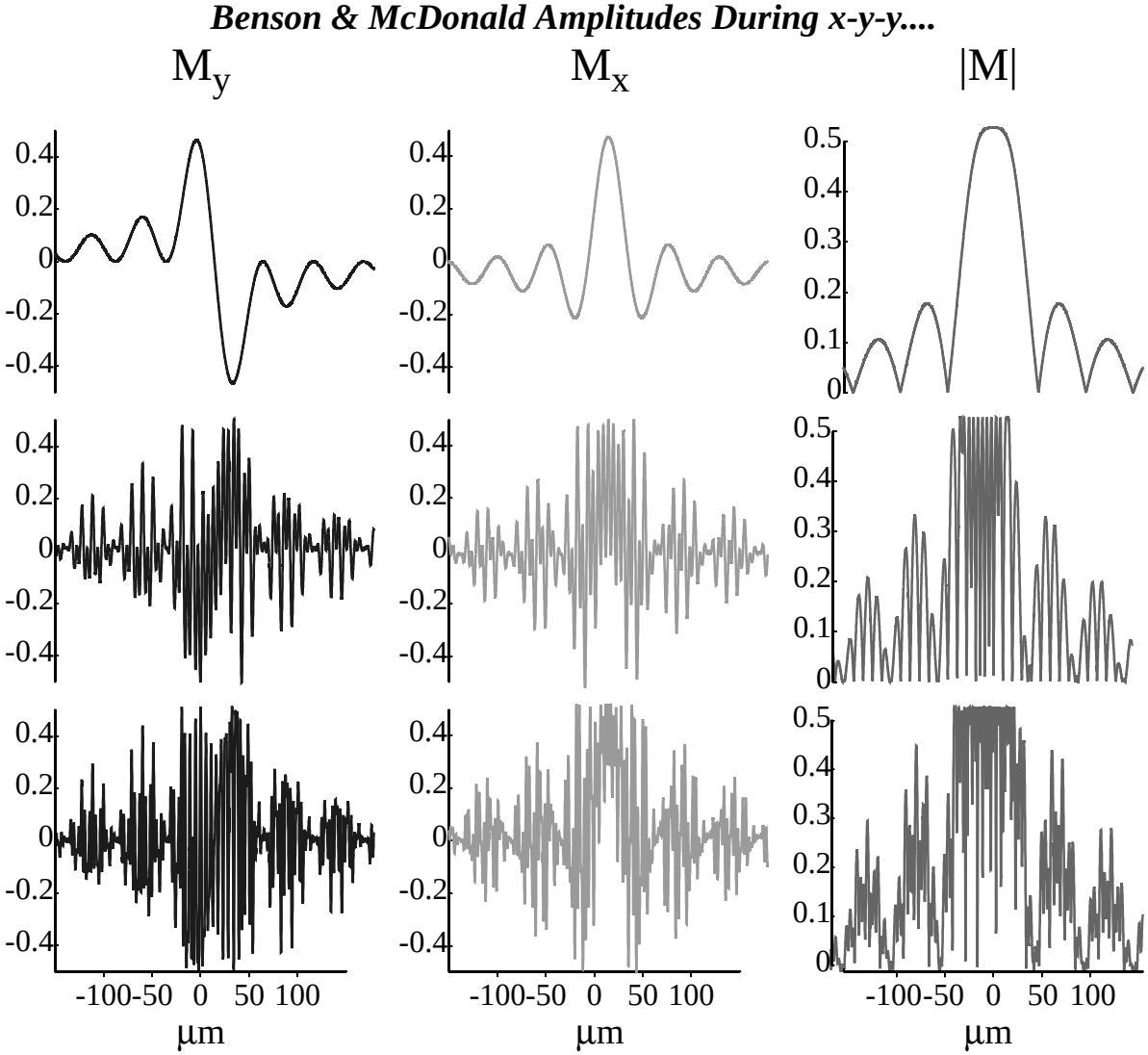

Figure 2-4 **Simulated magnetization during STRAFI sequence 90_x-(τ-90_y-τ–)$_n$. Row 1 are the value immediately after the pulse. Row 2 are those at the 1st STRAFI echo and Row 3 are those at the 2nd STRAFI echo. Here t_p = 10 μs, tau = 50 μs.**

Both figures were generated from the same GAMMA program, SFIBenson.cc, which is found at the end of this Chapter, page 66. These results are virtually identical to the publication.

Before we leave this type of calculation behind we can have a look at some other interesting effects. Will these profiles look the same on higher spins (I=1,3/2,...) and with muliti-spin systems? How will pulse length, strength, and shape affect the magnetization? Can higher order coherences influence the results? What about different interactions in multi-spin systems?

2.6.2 A Single Deuteron, I=1

The following figure is a repeat of the Benson & McDonald computation ala GAMMA, except that now a single deuteron is being used instead of a proton.

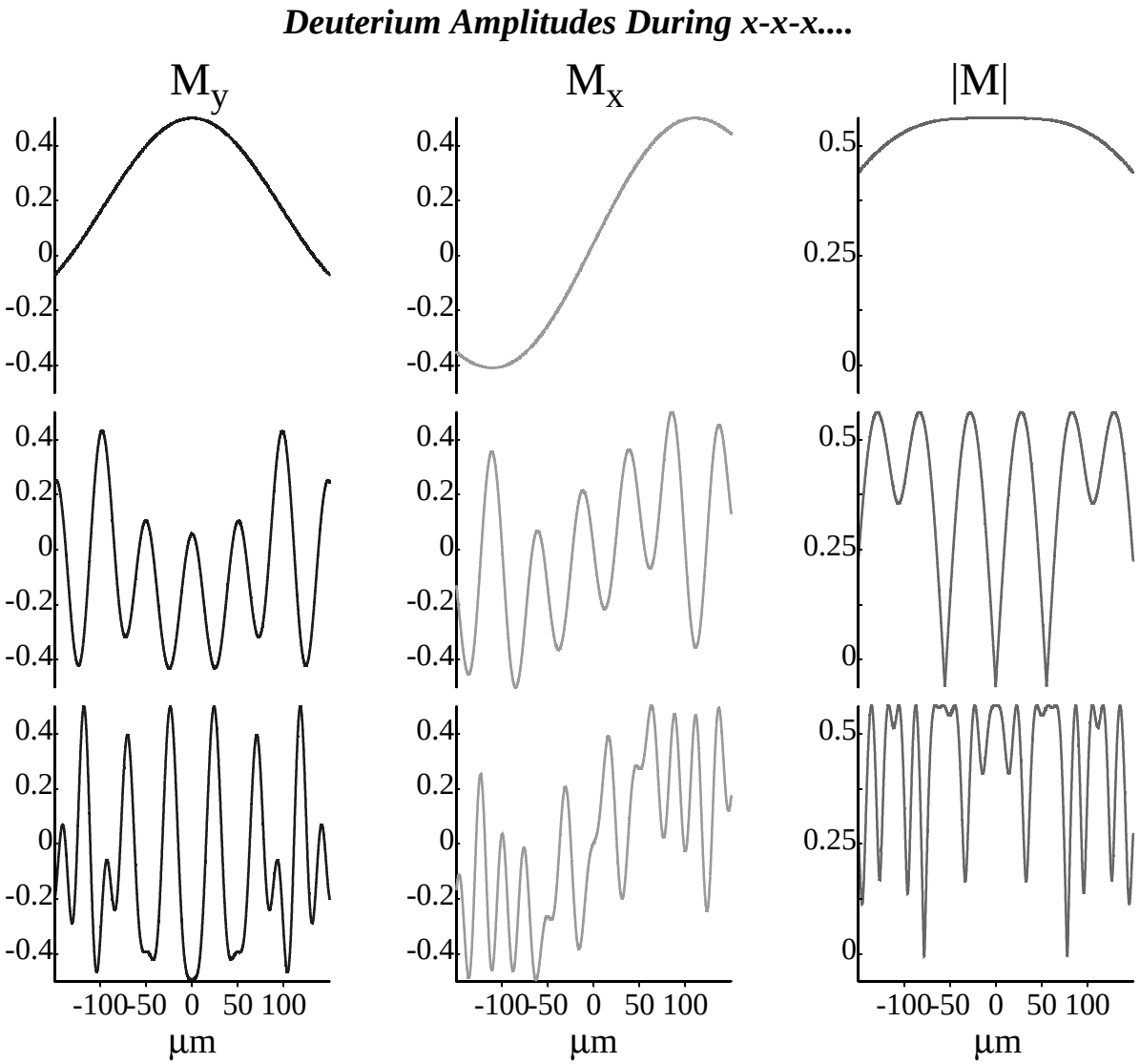

Figure 2-5 **Simulated magnetization during STRAFI sequence 90_x-(τ-90_x-τ–)$_n$. Row 1 are the values immediately after the pulse. Row 2 are those at the 1st STRAFI echo and Row 3 are those at the 2nd STRAFI echo. Here t_p = 10 μs, tau = 40 μs.**

This figure used parameters similar to those in the Benson & McDonald citation except that a single deuteron was used instead of a proton. Note that the **lower** gyromagnetic ratio of deuterium makes the gradient **less effective at defocusing** the spins over the detected sample. Thus these profiles are "stretched out" versions of the profiles obtained from the proton calculation. **Simulations on 2H spanning +/- 900 μm (6x to compensate for 1/6 γ) closely match 1H profiles!** Although they are (anti-)symmetric here, the echo profiles are often asymmetric, as were the proton echoes.

2.6.3 Mag. vs. Distance, Gaussian Pulse

It is possible (albeit unlikely) that shaped pulses might produce better magnetization profiles over the sample. In this section we will examine whether Gaussian pulses have any (dis-)advantages over rectangular pulses. For typical pulse lengths (~10 μs) and field gradients (~50 T/m) it would seem that shift dispersion effects due to the gradient will far outweigh any compensation from the Gaussian pulse profile superiority.

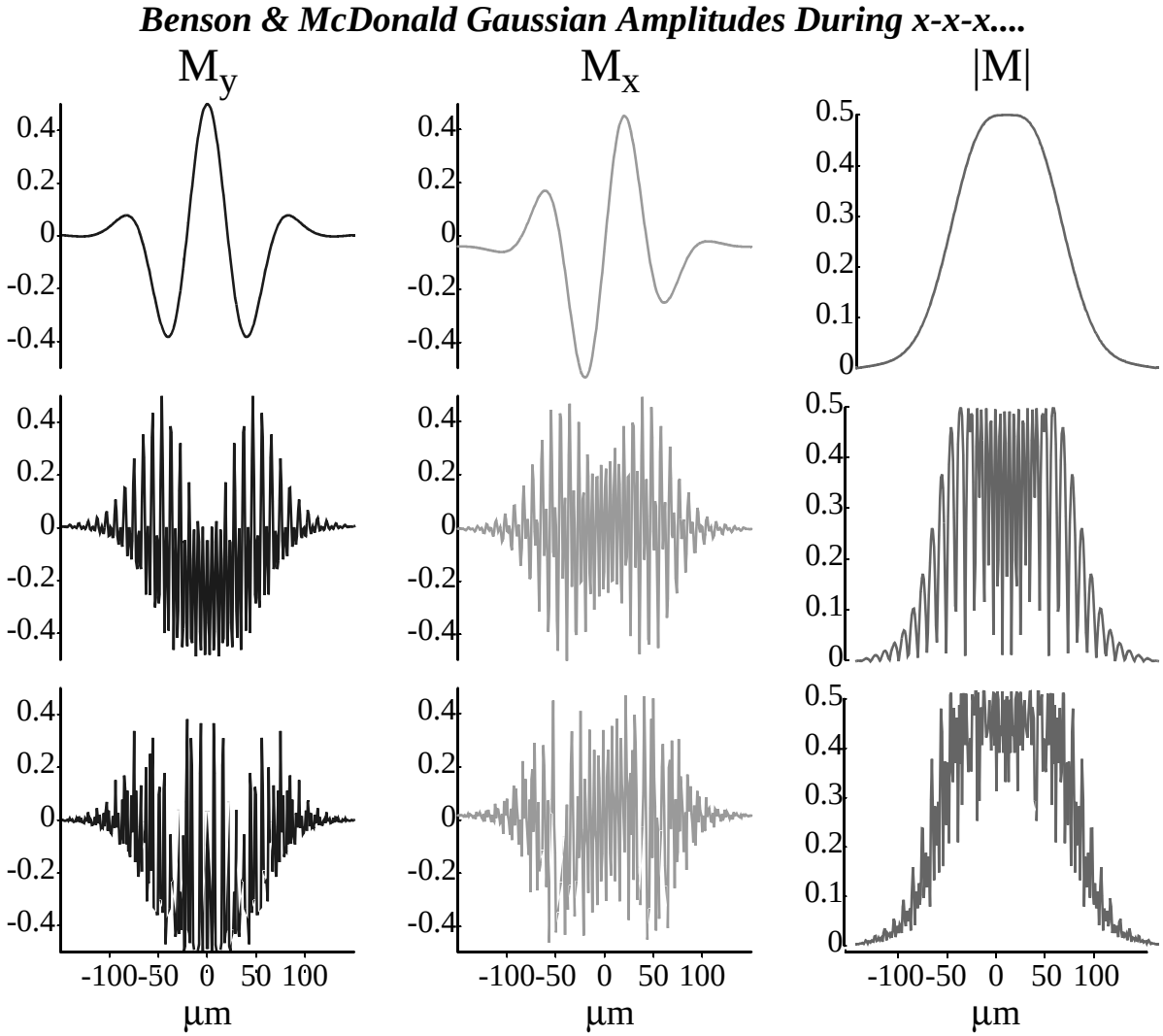

Figure 2-6 **Simulated magnetization during STRAFI sequence 90_x-(τ-90_x-τ–)$_n$. Row 1 are values immediately after the pulse. Row 2 are those at the 1st STRAFI echo and Row 3 are those at the 2nd STRAFI echo. Here the applied pulse was Gaussian that used 101 steps and a 2% endpoint cutoff. Additionally t_p = 10 μs and tau = 50 μs.**

These are indeed different from the results using rectangular pulses (compare with Figure 2-3). The change in magnetization with distance is much less dramatic using Gaussian pulses, at least immediately following the first pulse. Subsequent echoes also seem to keep the magnetization within a more well defined region as well! Here, signals come from a broader sample region.

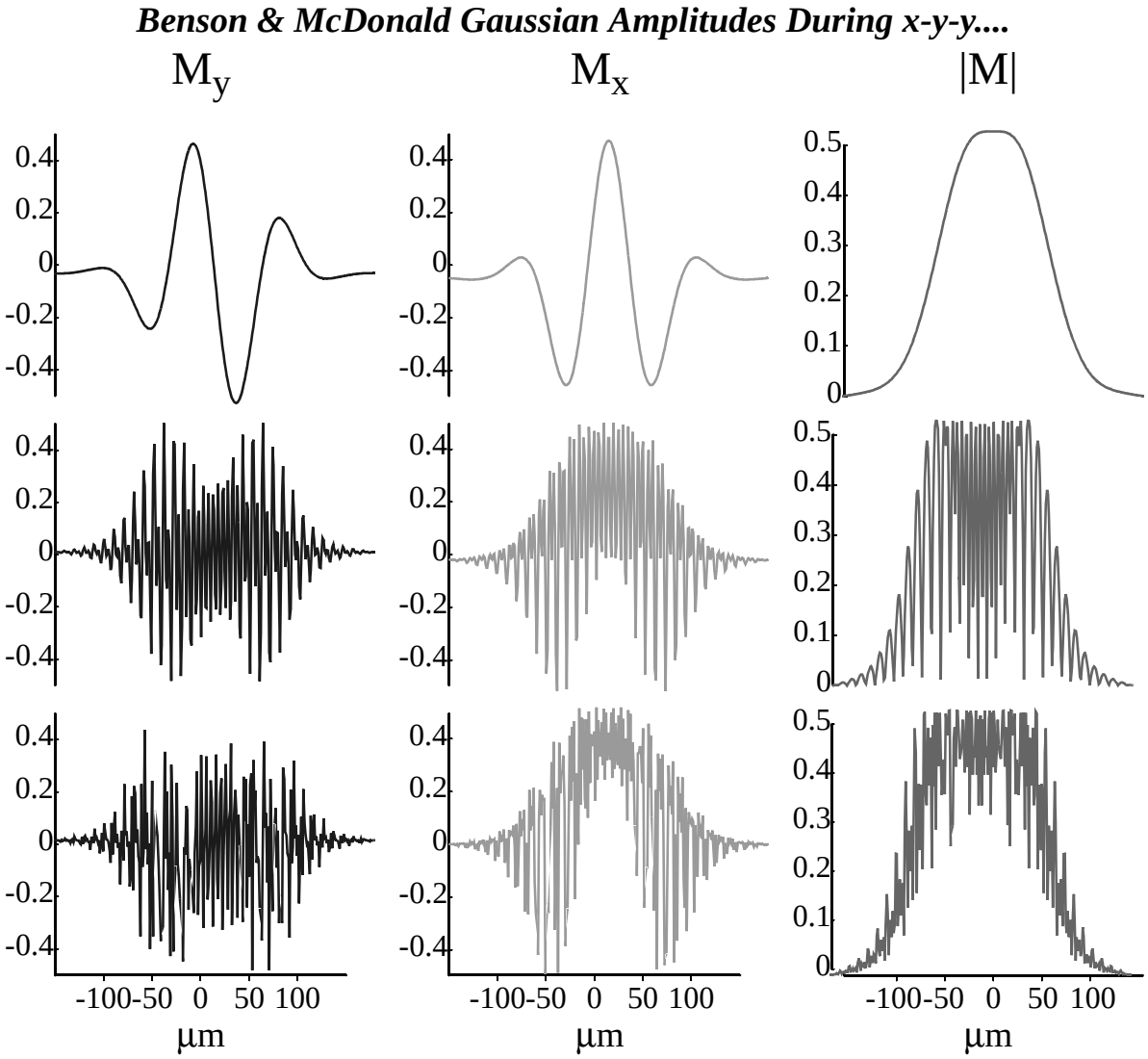

Figure 2-7 **Simulated magnetization during STRAFI sequence 90_x-(τ-90_y-τ–)$_n$. Row 1 are values immediately after the pulse. Row 2 are those at the 1st STRAFI echo and Row 3 are those at the 2nd STRAFI echo. Here the applied pulse was Gaussian that used 101 steps and a 2% endpoint cutoff. Additionally t_p = 10 μs and tau = 50 μs.**

Again these results are still different from the rectangular pulse simulation (compare with Figure 2-4). Again we obtain a more well defined region over which the magnetization is the same. That implies that one may be able to use longer Gaussian pulses to produce nice well defined regions of excitation for STRAFI echoes. In turn, that could be used advantageously in imaging.

2.6.4 Dipolar Spin Pair, I=1 with I=1/2

The following figure is a repeat of the Benson & McDonald computation ala GAMMA, except that now a proton-deuteron spin pair with 100 KHz dipolar coupling is being treated.

Benson & McDonald Dipolar Amplitudes During x-x-x....

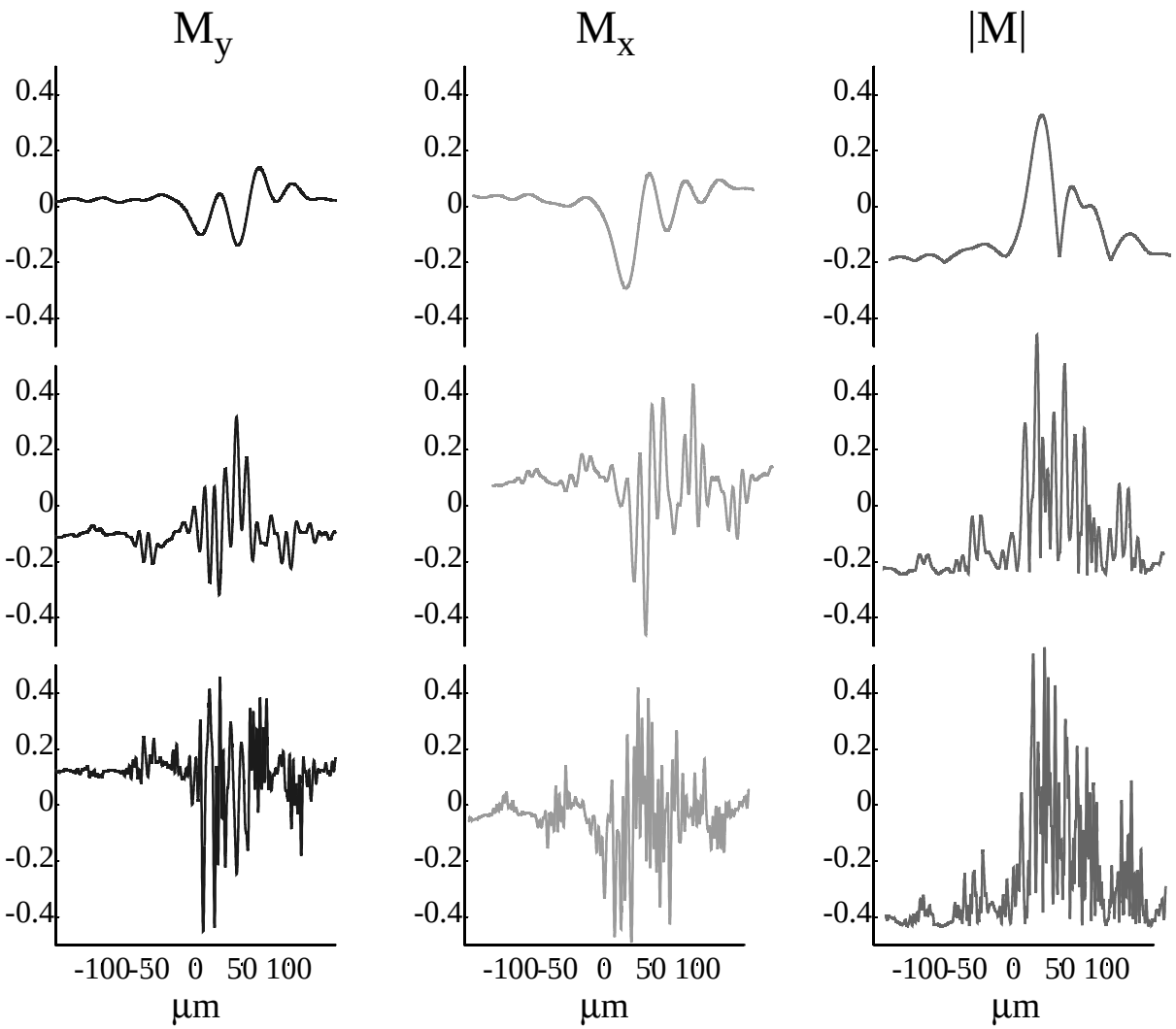

Figure 2-8 **Simulated magnetization during STRAFI sequence 90_x-(τ-90_x-τ–)$_n$. Row 1 are the value immediately after the pulse. Row 2 are those at the 1st STRAFI echo and Row 3 are those at the 2nd STRAFI echo.**

This figure used parameters similar to those in the Benson & McDonald citation except that a deuteron coupled to a proton was used. Note that the lower gyromagnetic ratio of deuterium makes the gradient less effective at defocusing the spins over the detected sample. Thus these profiles are "stretched out" version of the profiles obtained from the proton calculation.

2.7 Bain & Randall Simulation

2.7.1 Ideal Echo Amplitudes, Single 1H

For a second check of GAMMA's ability to treat STRAFI problems we will reproduce the echo amplitudes published by Bain and Randall[1]. In this article a single spin 1/2 system was used to follow the echo amplitudes, no gradient was applied and ideal pulses were used.

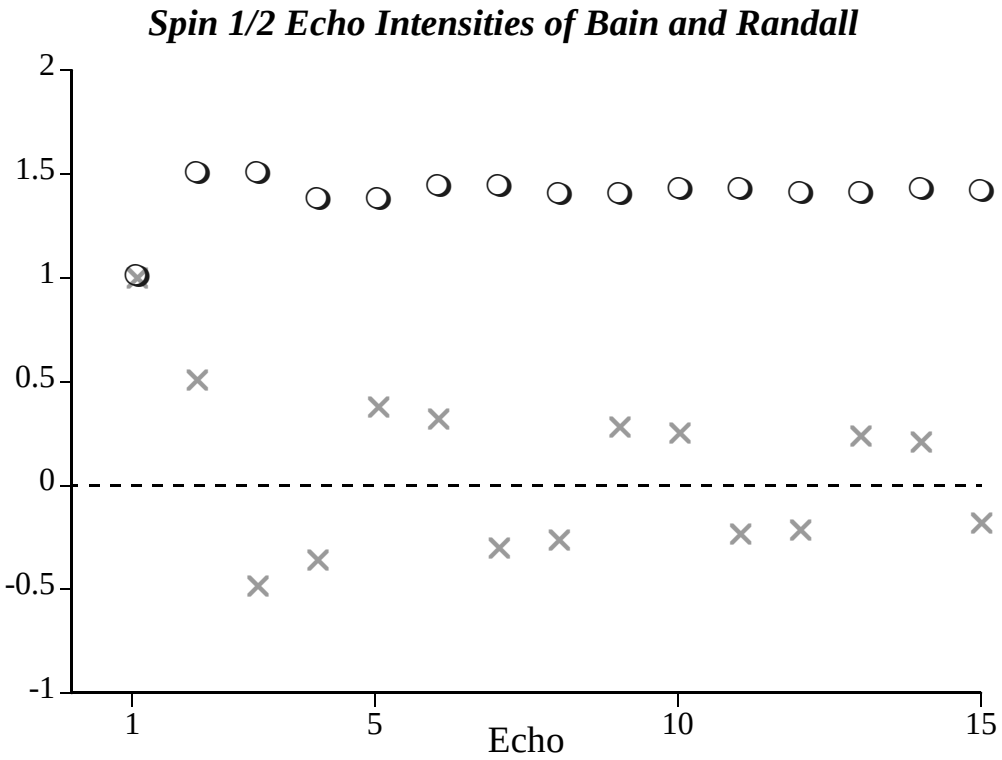

Figure 2-9 **Simulated magnetization during STRAFI spin echo sequence $90_{x,y}$-(τ-90_{y}-τ–)$_{n}$. The blue circles are echo intensities following the x-y-y-y-.... sequence while the green exes are echo intensities following the y-y-y-y-.... sequence. Ideal 90 pulses were used for excitation and the system was a single proton. The gradient was set to 50 T/m and the effective system length to 10 μm. The spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz.**

In this instance, ideal pulses were used and the intensities reflect magnetization exactly at the echo times. A field gradient was applied and the system was a single proton. Note that there are no pulse offset effects in this calculation nor are there any scalar, dipolar, or quadrupolar couplings involved. The intensities match the published ones exactly. The program which produced this plot, SFIBain.cc, which is found at the end of this Chapter, page 68. The input spin system is also provided on the same page, Bain.sys. The program will output the plot into FrameMaker format for direct incorporation into this document.

1. "Spin Echoes in Static Gradients Following a Series of 90 Degree Pulses", Alex D. Bain and E.W. Randall, *J. Magn. Reson. Ser. A, **123**,* 49-55 (1996).

2.7.2 Rectangular Pulse Echo Amplitudes

At this point we shall make the jump to use of non-ideal pulses, specifically rectangular pulses, and have a look at the echo amplitudes from slightly more precise computations. With a proper choice of pulse length we should be able to reproduce the ideal pulse echo intensities in Figure 2-9.

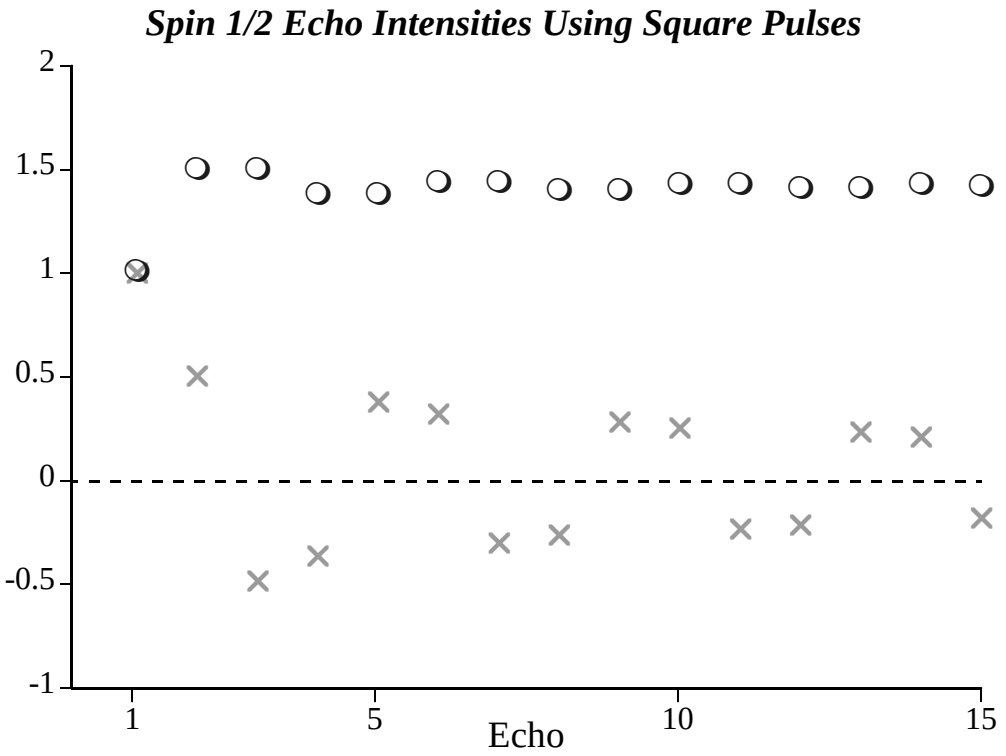

Figure 2-10 **Simulated echoes during the STRAFI spin echo sequence $90_{x,y}$-(τ-90_y-τ–)$_n$. The blue circles are echo intensities following the x-y-y-y-.... sequence while the green exes are echo intensities following the y-y-y-y-.... sequence. Rectangular 90 pulses of 10μs length were used for excitation on a single proton. The gradient was set to 50 T/m and the effective system length to 5μm. The spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz.**

This figure was generated from the program strafi0.cc found on page 71. Remarkably, results are virtually identical to the ideal pulse intensities. Apparently the echo amplitudes are not dramatically affected by the excitation profile of the pulse. Note that in the ideal case all spins in the gradient were pulsed by 90 degrees, whereas here the effective pulse angle and phase depend upon where in the sample the spin sub-systems lie. Thus, both de-phasing during precession in the gradient and initial phase/magnitude starting conditions prior to precession do good averaging over the sample but leave the echo intensity (on resonance) alone! This situation cannot be generally true of course, but should be applicable to the case when the pulse lengths are compatible with the gradient (and sample length). This does not mean to imply that the echo envelopes are the same. We haven't yet looked at them, here we are only taking the amplitudes at the center of the echo. The envelopes may indeed be affected, but to what extent is as yet unclear. It DOES imply that changing the pulse waveform, i.e. using a Gaussian pulse, will have no affect on the echo amplitudes.

2.7.3 Gaussian Pulse Echo Amplitudes

Now we can have a look at whether shaped pulses have any effect on the echo amplitudes. We'll regenerate the amplitude maxima using Gaussian shaped pulses.

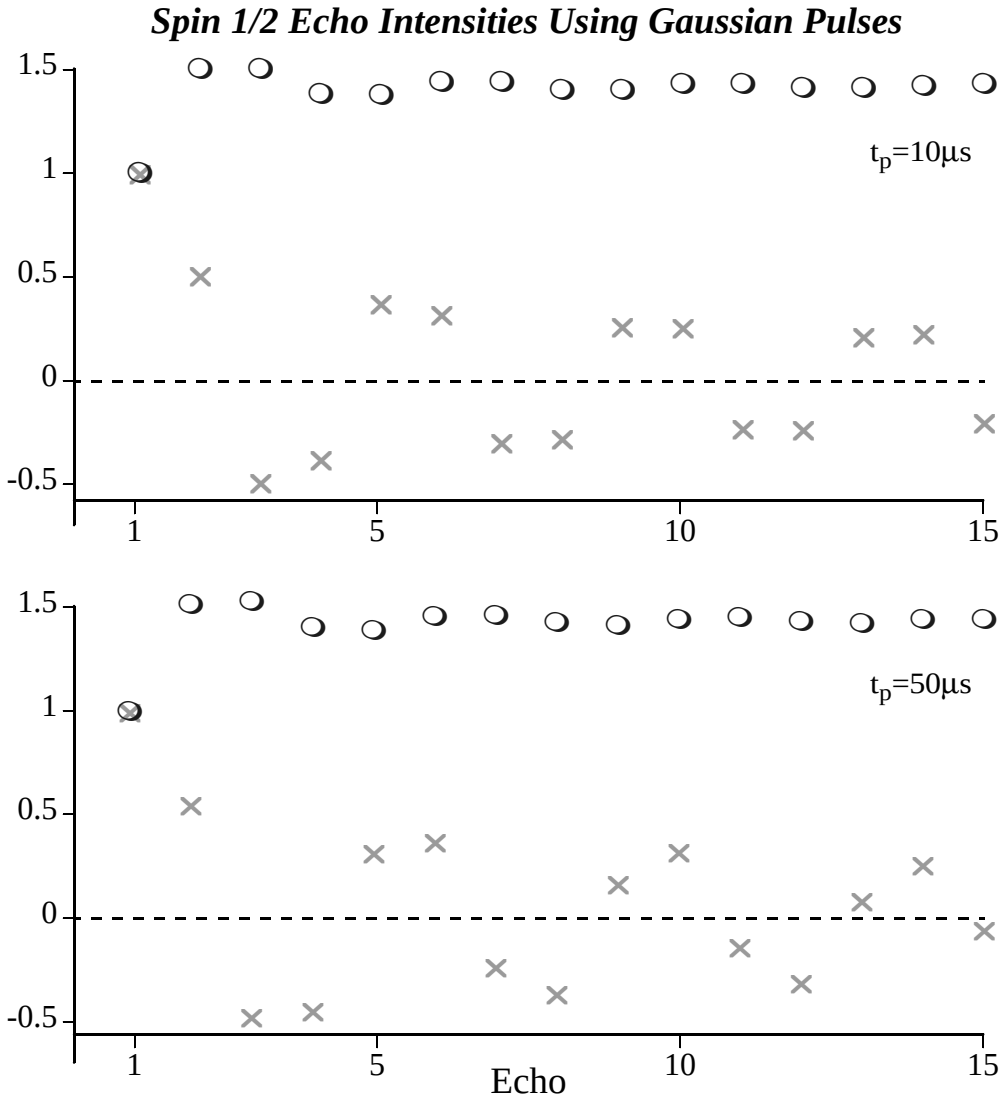

Figure 2-11 **Simulated echoes during the STRAFI spin echo sequence $90_{x,y}$-(τ-90_y-τ–)$_n$. The blue circles are echo intensities following the x-y-y-y-.... sequence while the green exes are echo intensities following the y-y-y-y-.... sequence. Gaussian 90 pulses were used for excitation on a single proton. The gradient was 50 T/m & the system length 5μm. The spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz. The pulses were done in 101 steps with 2% end intensity and length shown.**

This figure was generated from the program Bain2.cc. These results are close, but not identical to the rectangular pulse intensities. In general, the relative echo intensities seem fairly consistent under a strong short pulse.

2.7.4 Ideal Echo Amplitudes, Single 2H

The ideal STRAFI echo amplitudes of Bain and Randall[1] were calculated based on a simple single spin 1/2 treatment. Where there are multiple spins and/or spins with higher I values (not to mention quadrupolar and dipolar interactions) the intensity pattern may well be different. Here we will have a peek at some spins of differing I values.

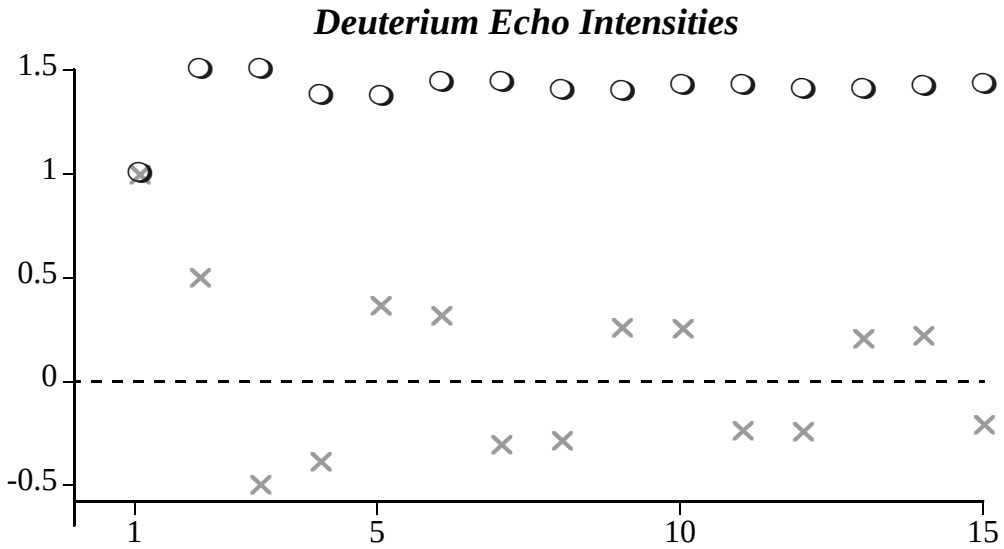

Figure 2-12 **Simulated echoes during the STRAFI spin echo sequence $90_{x,y}$-(τ-90_y-τ–)$_n$. The blue circles are echo intensities following the x-y-y-y-.... sequence while the green exes are echo intensities following the y-y-y-y-.... sequence. Rectangular 90 pulses of 10μs length were used for excitation on a single deuteron. The gradient was 50 T/m & the system length 5μm. The spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz.**

1. "Spin Echoes in Static Gradients Following a Series of 90 Degree Pulses", Alex D. Bain and E.W. Randall, *J. Magn. Reson. Ser. A, **123**,* 49-55 (1996).

2.8 STRAFI Echoes Vs. Pulse Angle

2.8.1 Ideal Echo Amplitudes, Single 1H

In this calculation we should like to see how the "up-down" pattern in the echo amplitudes vary relative to the pulse angle. We will begin using ideal pulses.

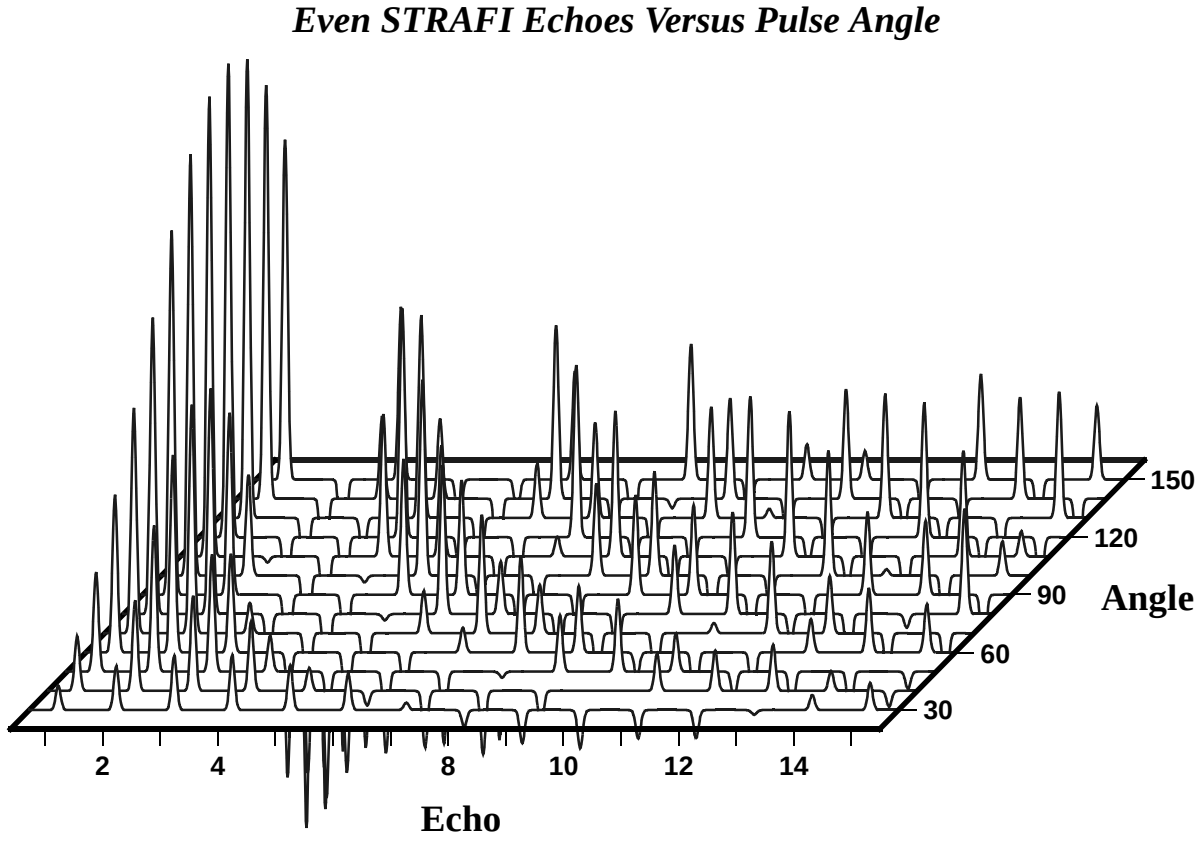

Figure 2-13 **Simulated echoes during the STRAFI spin echo sequence θ_y-(τ-θ_y-τ–)$_n$. Each row contains the 1st 15 echoes in the STRAFI sequence at a particular pulse angle. Ideal 90 pulses were used for excitation and the system was a single proton. The gradient was set to 50 T/m and the effective system length to 10 μm. The spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz.**

This plot was produced by SFIPAngVsEAmp1.cc, which is found at the end of this Chapter, page 69. The spin system used for input was PAng.sys which may be found on page 70. Current thinking is that the 1st echo will attain its maximum intensity for a pulse angle of 120 degrees. This is borne out in this simulation. It is also known that the up-down pattern is different for each pulse angle, also seen in the simulation. These two facts are important because they are used to calibrate 90 pulses in STRAFI experiments. A few of the rows are isolated in the following figure for clarification.

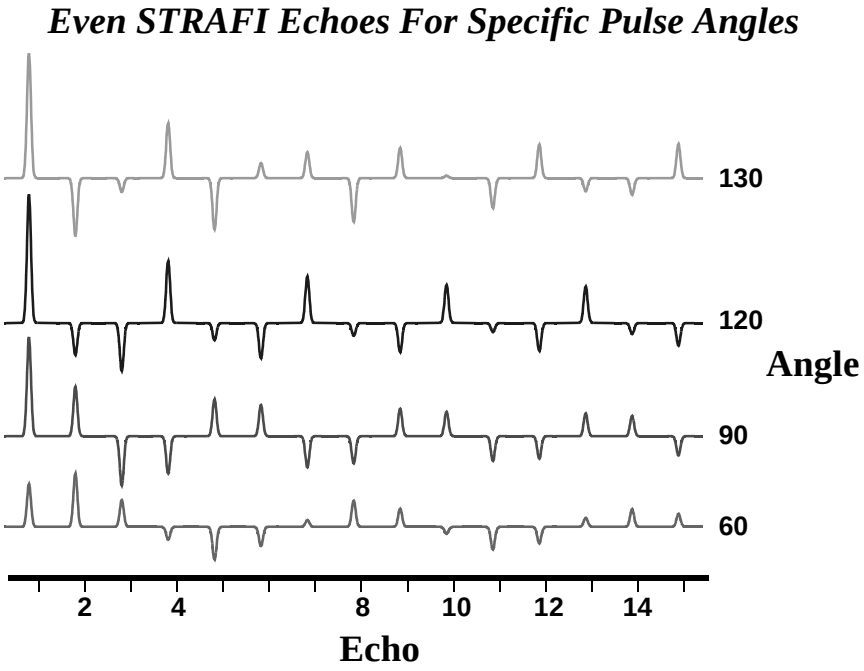

Figure 2-14 **Simulated echoes during the STRAFI spin echo sequence θ_y-(τ-θ_y-τ–)$_n$. Each row contains the 1st 15 echoes in the STRAFI sequence at a particular pulse angle. Ideal 90 pulses were used for excitation and the system was a single proton. The gradient was set to 50 T/m and the effective system length to 10 μm. The spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz.**

The "odd" STRAFI sequence, θ_x-(τ-θ_y-τ–)$_n$, also produces echo intensity maxima at pulse angles of around 120 degrees, but in this case all the echoes are positive.

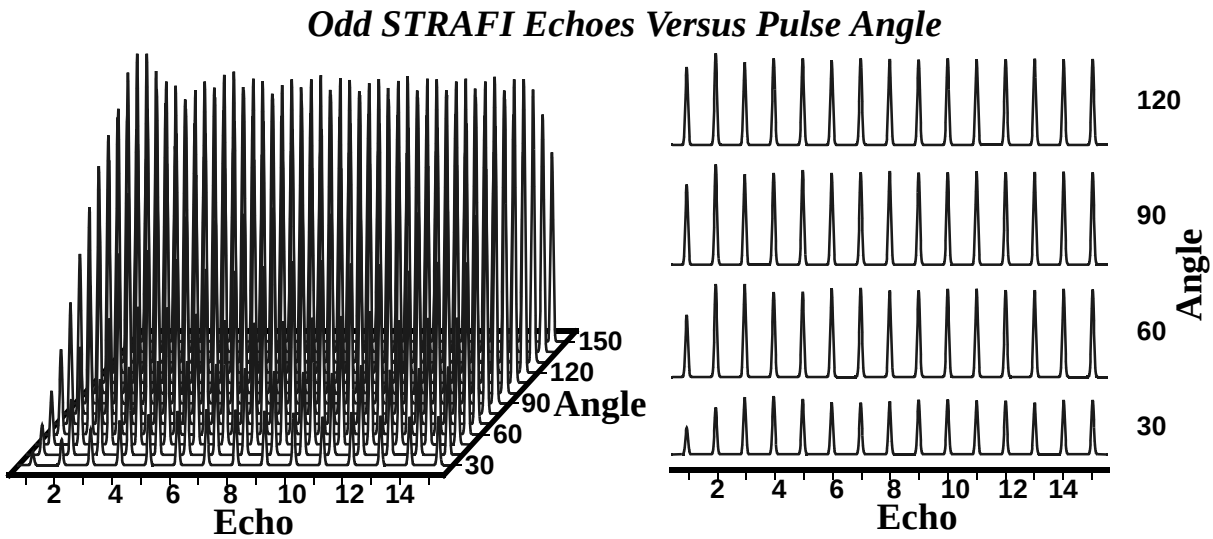

Figure 2-15 **Simulated echoes during the STRAFI spin echo sequence θ_x-(τ-θ_y-τ–)$_n$. Other conditions were set the same as in the previous figure.**

2.8.2 Single Proton: Rectangular Pulses

In this calculation we should like to compare the simulated “up-down” pattern in the echo amplitudes vs. pulse angle to the previous computation using ideal pulses. Our prediction, based on the amplitude pattern following the 90 pulse (Bain & Randall type of simulation), is that there will be little difference in the outcome as long a the pulse lengths are reasonable.

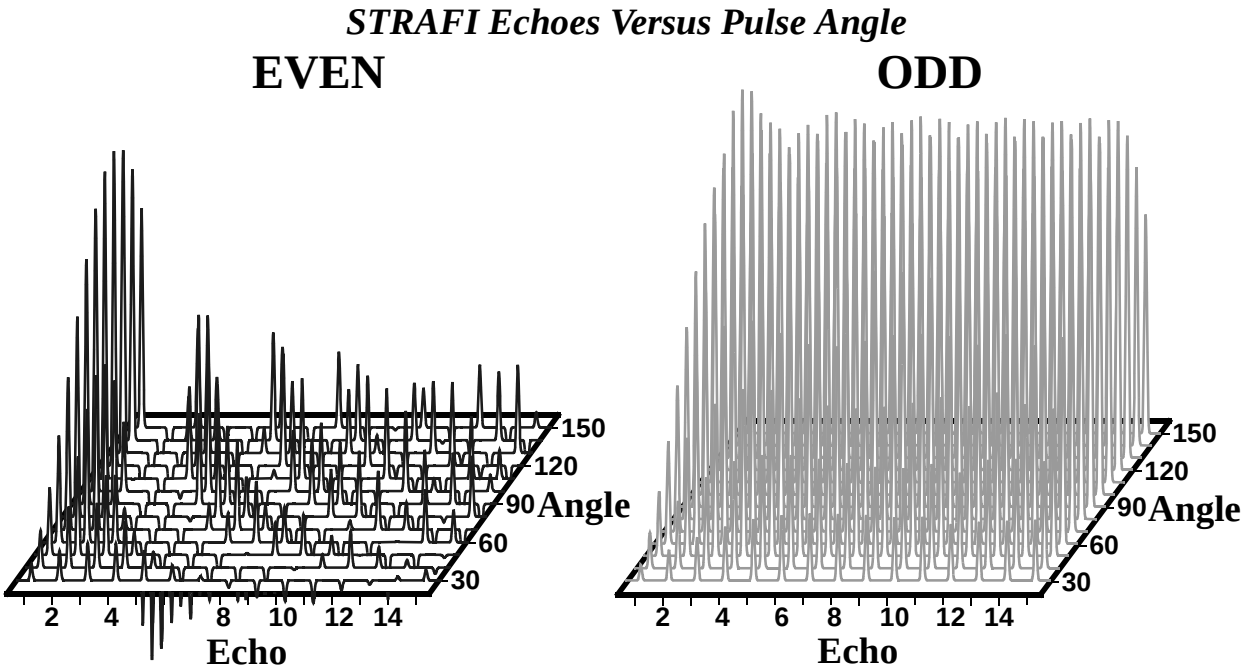

Figure 2-16 **Simulated echoes during STRAFI spin echo sequence $\theta_{x,y}$-(τ-θ_y-τ–)$_n$. Each row contains the 1st 15 echoes in the STRAFI sequence at a particular pulse angle. 10 μs pulses were used for excitation, their intensities adjusted to produce the proper rotation on resonance. The system was a single proton. The gradient was set to 50 T/m and the effective system length to 5μm. The spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz.**

This plot was produced by PAngVsEAmp2.cc, which is found at the end of this Chapter, page 72. The spin system used for input was again PAng.sys which may be found on page 70. There is indeed no significant difference between calculations with ideal vs. rectangular pulses.

2.9 FID Generation Under Gradients

We will now simulate some FIDs following the application of a single pulse. This can be an important aspect of STRAFI calculations because we must know a time scale on which the FID decays in order to set appropriate delay time(s) during the echo sequences. The main STRAFI echoes should appear in the middle of two pulses which are separated by delay time, τ. Time τ is typically set longer than the FID decay length so that the echo is not blended in with any transverse magnetization created by the pulse.

2.9.1 Ideal Pulse FID

To begin, we shall apply a single ideal pulse and examine the resulting magnetization. The pulse will thus affect all spins in the system with the same pulse angle and phase. However, we should still see the FID decay as the result of the superposition of the many signals from sub-systems whose spins will resonate at differing frequencies. The figure below demonstrates how the number of spin sub-systems will affect the resulting FID.

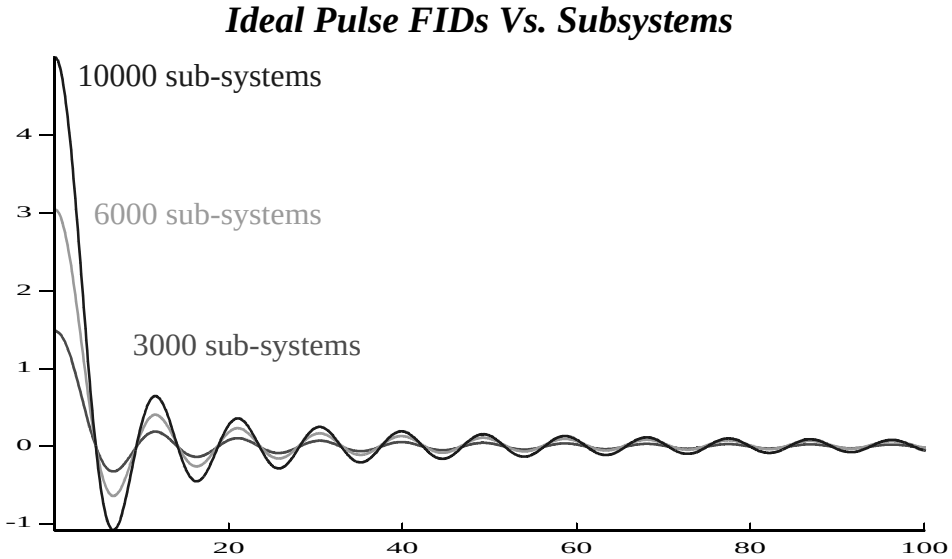

Figure 2-17 **Simulated single proton FIDs following an ideal 90_y pulse. The number of sub-systems computed were varied as indicated. In all cases the effective system length was set to 100μm, the spin was set at 0 PPM, the field gradient was 50 T/m, and the base field strength set to 300 MHz. Plots were produced by GrdFID0.cc found on page 74 and the FID collected for 100 μs.**

The above calculation demonstrates how we can effectively generate decaying FIDs in a field gradient. Note that the FID doesn't change with sub-system increase except for a scaling factor! That implies that we are taking sufficient sub-systems, the scaling being artificial from the computation.

Next we can view the effects of differing field gradients. Assuming that 6000 sub-systems (more than) suffice to effectively sample our system, we can compare FIDs generated under the same conditions except for the applied gradient.

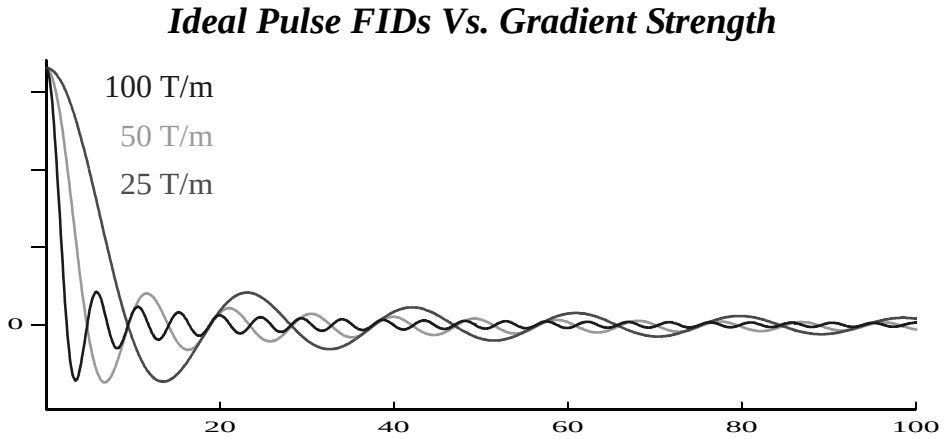

Figure 2-18 **Simulated single proton FIDs following an ideal 90_y pulse. The applied field gradient was varied as indicated. In all cases the effective system length was set to 100μm, the spin was set at 0 PPM, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. Plots were produced by GrdFID0.cc found on page 74 and the FID collected for 100 μs.**

Evidently it is the field gradient which is largely responsible for the oscillation frequencies, at least in this single proton computation. The larger the gradient the faster the oscillations.

The next aspect we shall consider is the effective spin system detected length. Again assuming that 6000 sub-systems suffice to effectively sample our system, we can compare FIDs generated under the same conditions except for the detectable sample size to be considered.

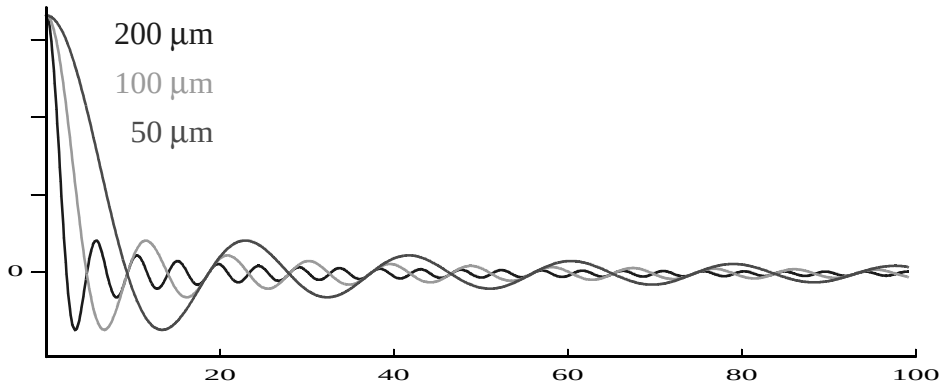

Figure 2-19 **Simulated single proton FIDs following an ideal 90_y pulse. The considered sample length was varied as indicated. In all cases the effective gradient strength was set to 50 T/m, the spin was set at 0 PPM, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. Plots were produced by GrdFID0.cc found on page 74 and the FID collected for 100 μs.**

Oddly, increasing the effective sample length has the identical effect as increasing the field gradient. Again this implies that we are sampling the gradient just fine (enough sub-systems) but that the outer spins, those away from the center of the field, are still contributing. Such should NOT be the case for finite length pulses because these "fringe" spins will not reside in the excitation profile.

2.9.2 Rectangular Pulse FID

To more realistically approach an FID generated with STRAFI we shall switch to using rectangular pulses. Now, depending upon the pulse length and strength, differing sub-systems in the gradient will indeed be affected by the pulse differently. Here are some examples.

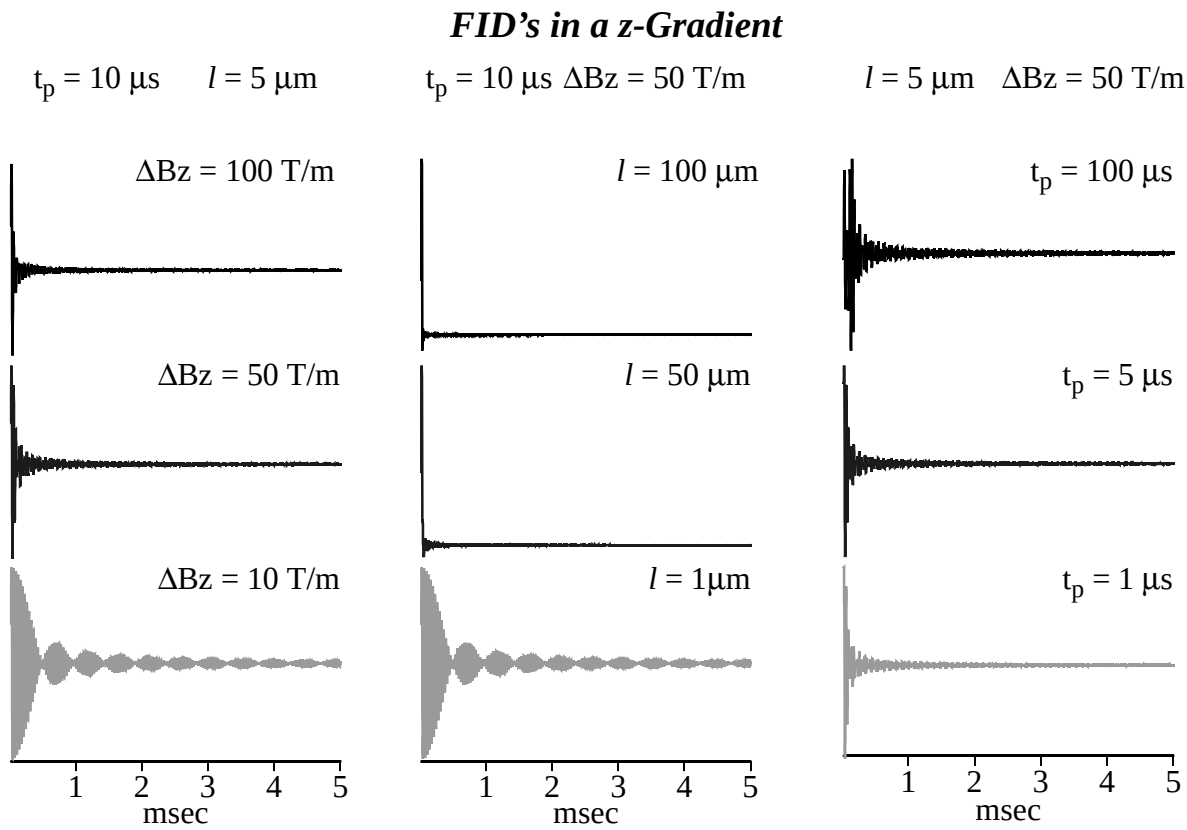

Figure 2-20 **Simulated single proton FIDs following a 90_y pulse. Pulse lengths and gradient strengths were varied as indicated. In all cases the effective system length was set to 5μm, the spin was set at 100 PPM, 3000 sub-systems were used, and the base field strength set to 300 MHz. Plots were produced by GrdFID1.cc found on page 75.**

As seen from the plots, as the pulse length increases (more selectivity, smaller sample excitation) the FID length increases because there are less spins (excited) at differing offsets to cancel net magnetization. The same is true for decreasing gradient strengths using the same explanation. This is roughly true for decreasing sample length too, although there should be a length that spans the total pulse excitation over the sample and any additional sample length should not contribute additional FID effects. One must keep in mind that since we are using only a limited number of sub-systems to represent a continuum through the gradient, the magnetization averaging may be better by increasing the number of sub-systems taken into account. Fewer sub-systems will also lengthen the FID due to less superposition of multiple frequencies.

In order to directly compare the effects of a finite length pulse to that of an ideal pulse in simulations we will now reproduce the three figures generated in the previous section. The first simply

looks at FID generation versus spin system sampling. It is expected that, give a sufficient number of sub-systems in the calculation there will be little change in the generated FID.

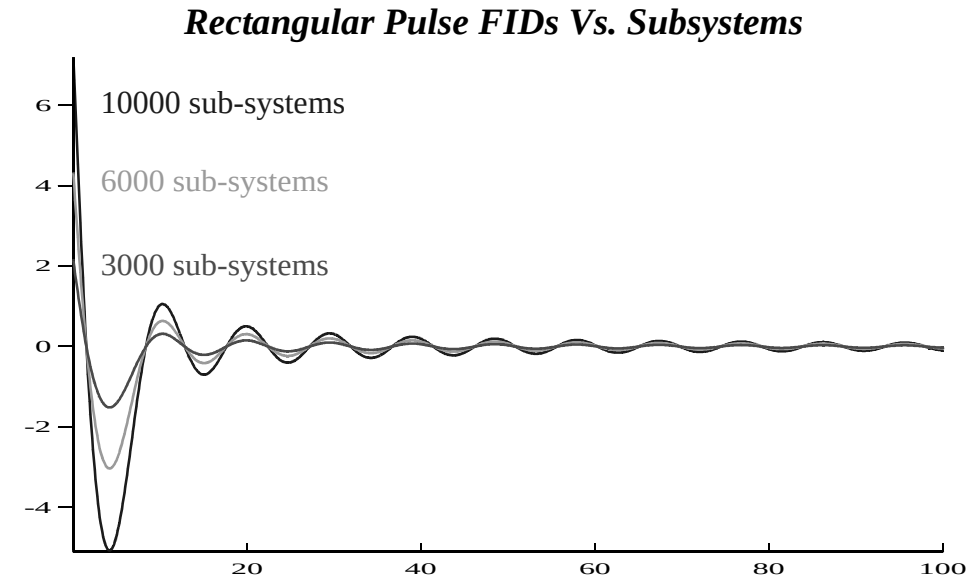

Figure 2-21 **Simulated single proton FIDs following a 10μs 90_y rectangular pulse. The number of sub-systems computed were varied as indicated. In all cases the effective system length was set to 100μm, the spin was set at 0 PPM, the field gradient was 50 T/m, and the base field strength set to 300 MHz. Plots were produced by GrdFID1.cc found on page 75 and the FID collected for 100 μs.**

Comparison with Figure 2-17 shows that the only effect of using a finite length pulse is to produce some phasing in the resulting FIDs, undone by a zeroth order phase correction. Below is a similar comparison to Figure 2-19, but now we see the effects of the pulse excitation envelope.

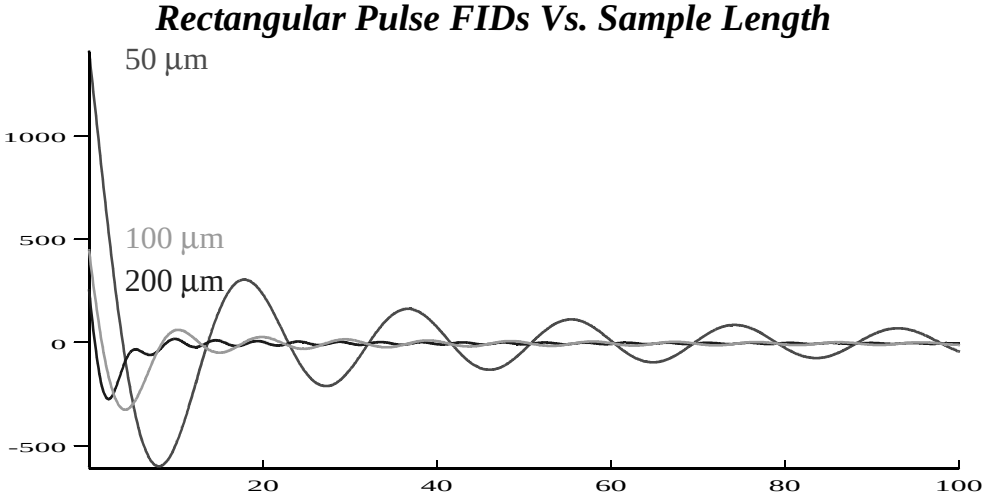

Figure 2-22 **Simulated single proton FIDs following a 10μs 90_y rectangular pulse. The considered sample length was varied as indicated. In all cases the effective gradient strength was set to 50 T/m, the spin was set at 0 PPM, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. Plots were produced by GrdFID1.cc found on page 75 and the FID collected for 100 μs.**

2.9.3 Gaussian Pulse FID

As our last venture into FID simulations in a z-gradient we will look at how Gaussian pulses influence the results. Previously we saw that finite length pulses would 1.) introduce some phasing into the FID and 2.) the length of sample excitation was reduced. These effects should also be apparent using Gaussian pulses. However, since the pulse excitation profile should now be more uniform, we may see some interesting effects.

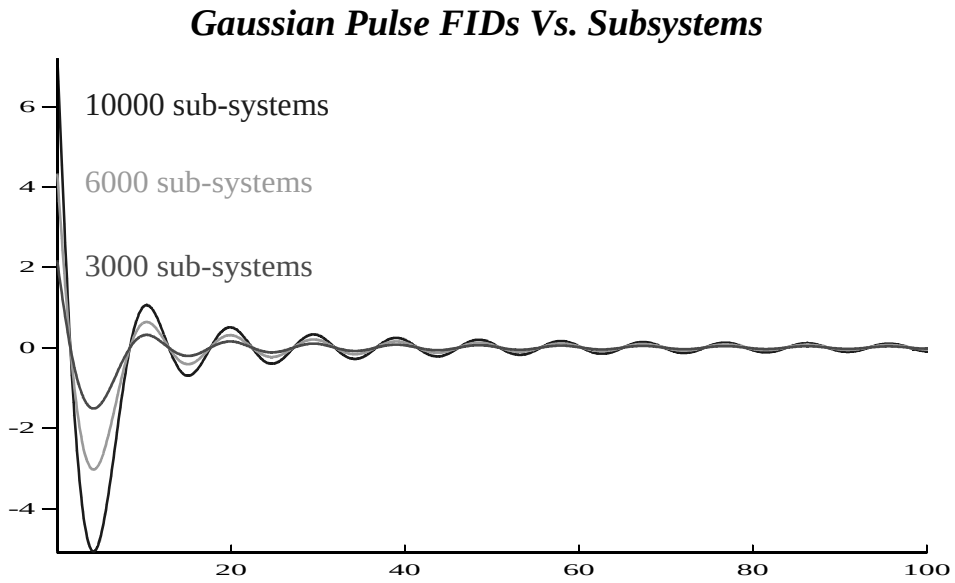

Figure 2-23 **Simulated single proton FIDs following a 10μs 90_y Gaussian pulse. The pulse was broken up into 101 steps with end intensities of 2%. The number of sub-systems computed were varied as indicated. In all cases the effective system length was set to 100μm, the spin was set at 0 PPM, the field gradient was 50 T/m, and the base field strength set to 300 MHz. Plots were produced by GrdFID1.cc found on page 75 and the FID collected for 100 μs.**

2.10 Echo Generation Under Gradients

We will now simulate echoes following the application of a 90-τ-180 sequence. The generated echo from this sequence should, in an ideal case, appear centered at time τ following the 180 pulse. Remember, the time τ is typically set longer than the FID decay length so that the echo is not blended in with any transverse magnetization created by the pulse.

2.10.1 Ideal Pulse Spin Echoes

The next figures demonstrate how various factors affect the resulting echo.

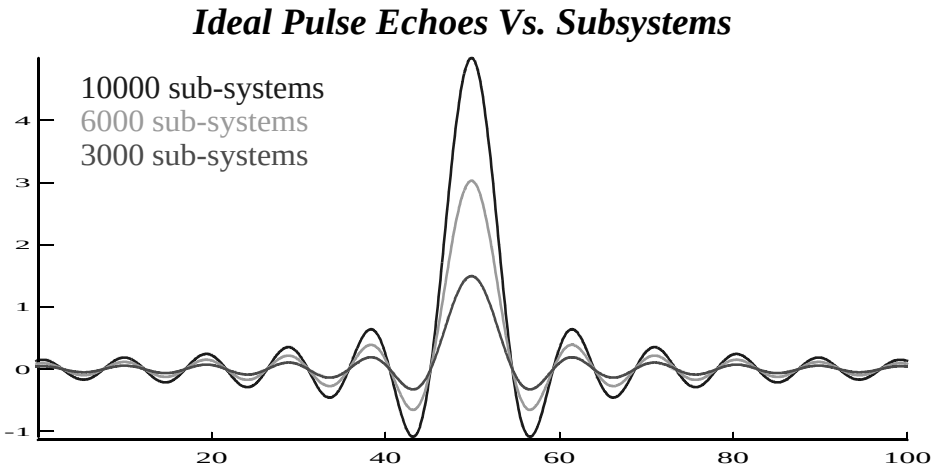

Figure 2-24 **Single proton echoes following a 90_y-τ-180_x sequence. The number of sub-systems were varied as indicated. The effective system length was set to 100μm, the spin was set at 0 PPM, the field gradient was 50 T/m, and the base field strength set to 300 MHz. Plots were produced by GrdSpinEcho0.cc, page 76 and the echo collected for 100 μs.**

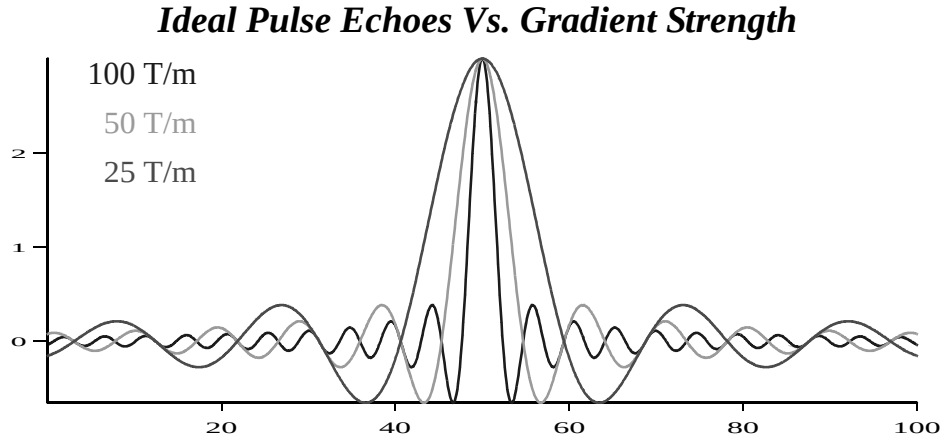

Figure 2-25 **Simulated single proton spin echo following a 90_y-τ-180_x sequence. The applied field gradient was varied as indicated. In all cases the effective system length was set to 100μm, the spin was set at 0 PPM, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. Plots were produced by GrdSpinEcho0.cc found on page 76 and the echo collected for 100 μs immediately following the π pulse.**

Were the echoes in the first figure normalized to the number of subsystems the three echoes would be identical. That implies (again) that our sub-system sampling is quite sufficient to generate proper results. The second figure details how the echo "sinc" rate increases with increasing gradient strength.

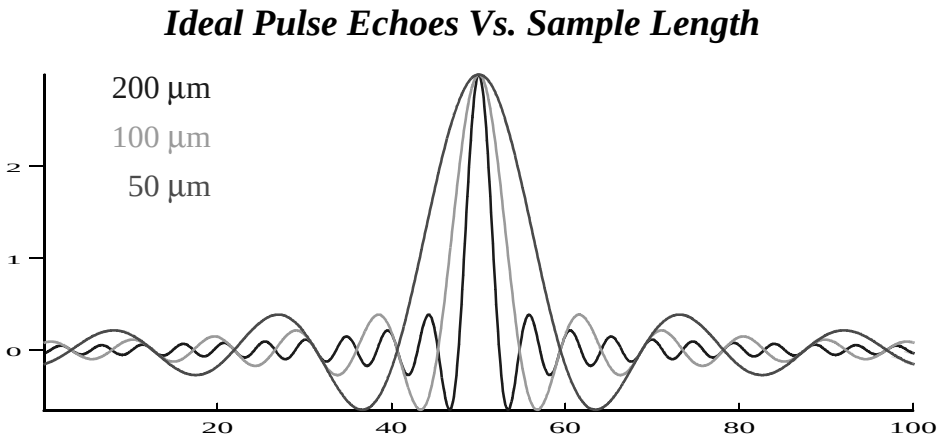

Figure 2-26 **Simulated single proton spin echo following a 90_y-τ-180_x sequence. The considered sample length was varied as indicated. In all cases the effective gradient strength was set to 50 T/m, the spin was set at 0 PPM, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. Plots were produced by GrdSpinEcho0.cc found on page 76 and the echo collected for 100 μs.**

The figure above indicates that using a larger effective system cause more cancellation, so the distant sub-systems under are still contributing to the echo. That isn't so surprising for the ideal case, we simply have more frequencies superimposing.

For the ideal pulse case increasing the effective sample length has the identical effect as increasing the field gradient. Again this implies that we are sampling the gradient just fine (enough sub-systems) but that the outer spins, those away from the center of the field, are still contributing. Such should NOT be the case for finite length pulses because these "fringe" spins will not reside in the excitation profile.

2.10.2 Rectangular Pulse Spin Echoes

Now we'll investigate use of rectangular pulses to make spin echoes following a typical 90-τ-180 sequence. Such simulations should produce echoes with true "sinc" like behavior. We will use our intuition about FID decay rates vs. gradient parameters to insure that our FID has decayed to zero long before our echo(es) will appear. The next figures demonstrate how various factors affect the resulting echo.

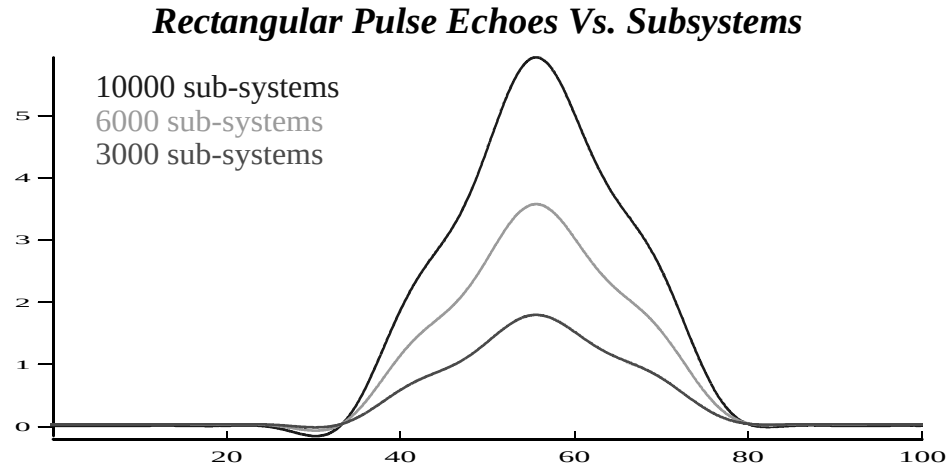

Figure 2-27 **Simulated single proton echoes after a 90_y-τ-180_x sequence. The # of sub-systems were varied as indicated. The system length was set to 100μm, the field gradient was 50 T/m, and the base field strength set to 300 MHz. The echo spans 100 μs centered at τ after the π pulse. The 90 pulse length was 10μs & the delay τ was 5 ms.**

Different from the ideal pulse echoes! Note that the echo width (in comparison with Figure 2-24) here is about 20μs compare to 7μs in the ideal case. Repeated calculations using a shorter τ (50 μs) produced the identical echo shape although there were slightly more noticeable baseline wobbles. Neither had any of the nice "sinc" type of behavior such as that seen in the ideal case.

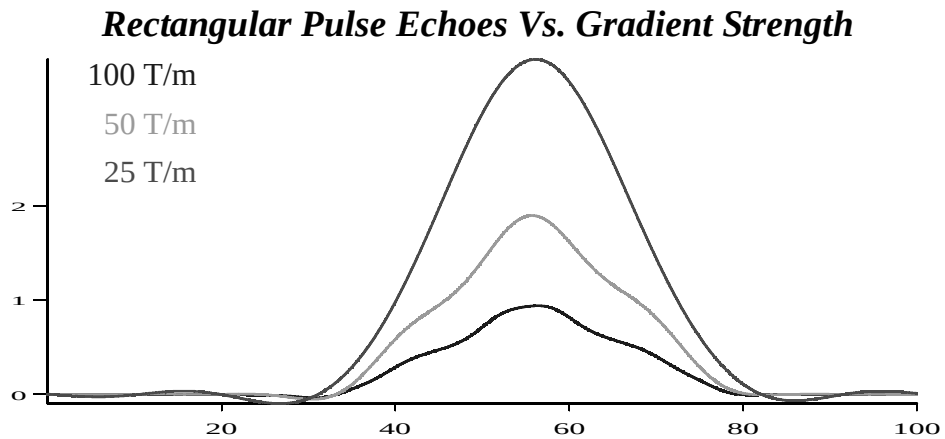

Figure 2-28 **Simulated single proton spin echo following a 90_y-τ-180_x sequence. The gradient was varied as indicated. In all cases the system length was 100μm, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. The echo spans 100 μs centered at τ after the π pulse. The 90 pulse length was 10μs & the delay τ was 5 ms.**

Again we obtain quite different results when compared to ideal pulse generated echoes (compare with Figure 2-25)! In this case the echo widths remain constant as the field gradient is altered yet now the amplitude actually increases with a decreasing gradient. What must be occurring now is a gradient profile/pulse profile match up. With the smaller gradient more spins in the effective sample are being positively affected by the pulse. Thus, the observed behavior is a clear reflection of how much of the sample is being detected.

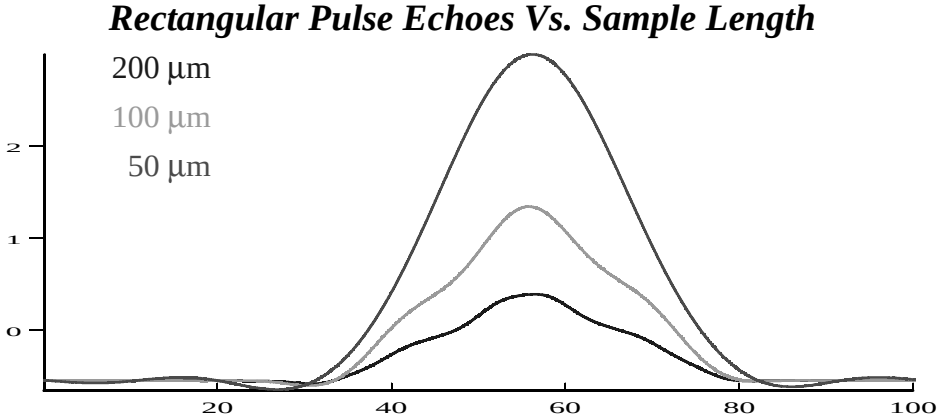

Figure 2-29 **Simulated single proton spin echo following a 90_y-τ-180_x sequence. The considered sample length was varied as indicated. In all cases the effective gradient strength was set to 50 T/m, the spin was set at 0 PPM, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. Plots were produced by GrdSpinEcho0.cc found on page 76 and the echo collected for 100 μs.**

The results above can be directly compared with Figure 2-26. A computation using a 10 μm effective sample length produces a single broad echo spanning the full 100 μs range (this is too small a sample). A 10 μs pulse should span a shift range corresponding to values in the mm rather than μm range in a 50 T/m gradient so we might try a larger sample. A 1 mm sample produces a plot akin to the 200 μm run above. These echoes look to be accurate (for a single spin anyhow). We can get nicer appearing echoes by adjusting the parameters if we try...

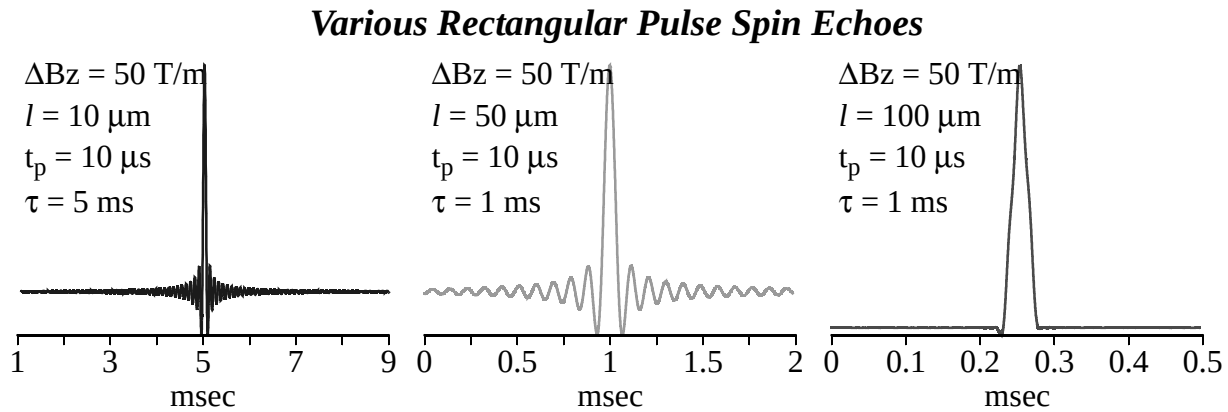

Figure 2-30 **Simulated spin echoes after application of a 90-τ-180 sequence. The echo acquisitions were centered about a time τ from the last pulse. The system was a single proton, 3000 sub-systems were used, and the base field strength set to 300 MHz. Pertinent parameters are indicated on each echo plot. The plots were produced by SpinEcho.cc page 77.**

2.10.3 Gaussian Pulse Spin Echoes

For completeness, we'll also look at some echoes generated using Gaussian pulses. Not sure what to expect under this type of pulse.

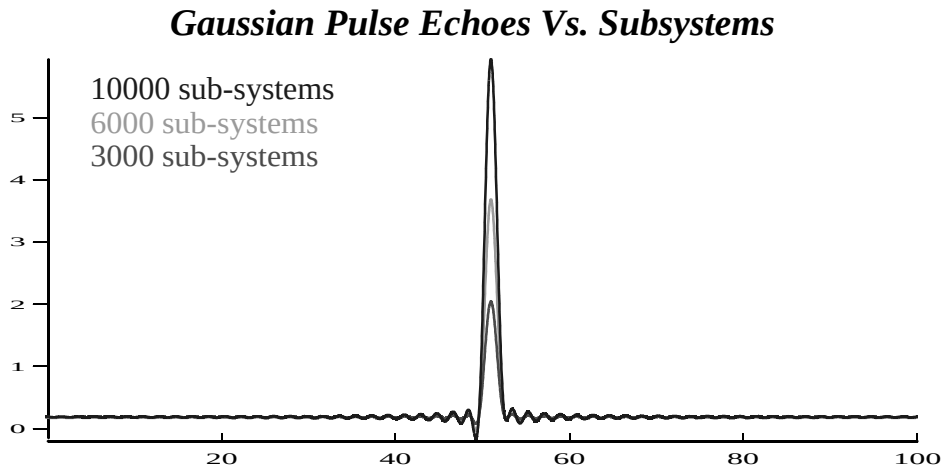

Figure 2-31 **Simulated single proton echoes after a 90_y-τ-180_x sequence. The # of sub-systems were varied as indicated. The system length was set to 100μm, the field gradient was 50 T/m, and the base field strength set to 300 MHz. The echo spans 100 μs centered at τ after the π pulse. The 90 pulse length was 10μs & the delay τ was 5 ms.**

From above it appears that echo generation is much cleaner, more like using ideal rather than rectangular pulses. Note that, were the echoes in the above figure normalized to the number of subsystems, the three echoes would be identical.

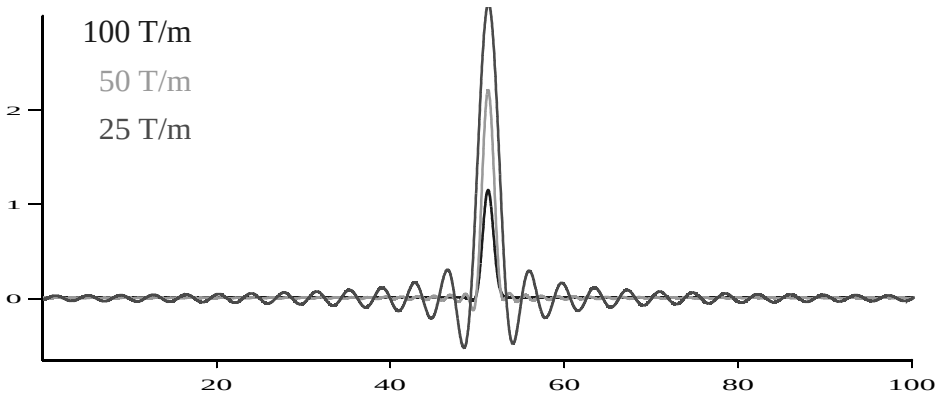

Figure 2-32 **Simulated single proton spin echo following a 90_y-τ-180_x sequence. The applied field gradient was varied as indicated. In all cases the effective system length was set to 100μm, the spin was set at 0 PPM, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. The echo collected for 100 μs immediately following the π pulse.**

This figure (above) is like the results obtained with the ideal pulses, and to a lesser extent like the rectangular pulses. This is a reflection of effective sample size relative to the applied gradient.

When the gradient is small our 100μm sample is more applicable since it is better at spanning the entire range over which the pulse is causing any magnetization changes. Then the echo is fine. With a stronger gradients the echo isn't as "sinc" like but should be if we were to take a larger effective sample.

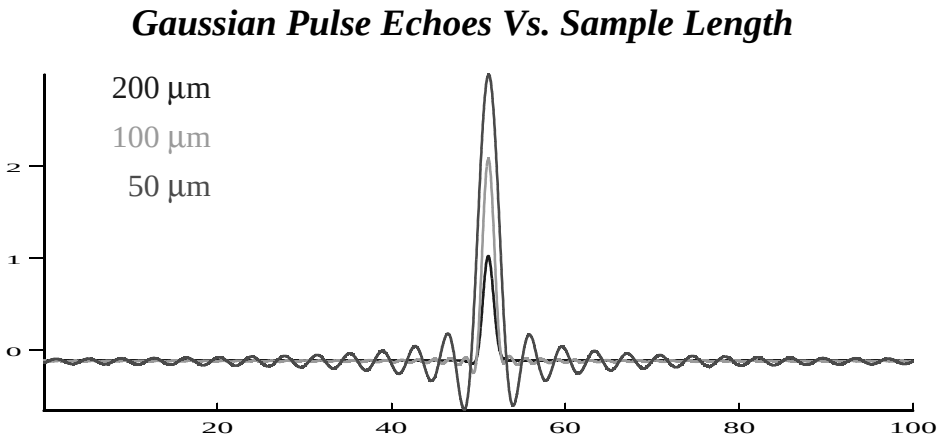

Figure 2-33 **Simulated single proton spin echo following a 90_y-τ-180_x sequence. The considered sample length was varied as indicated. In all cases the effective gradient strength was set to 50 T/m, the spin was set at 0 PPM, 6000 sub-systems were calculated, and the base field strength set to 300 MHz. The echo collected for 100 μs immediately following the π pulse.**

Or perhaps NOT! The above calculations show that as the sample is larger the echo isn't any better. In fact this figure is very similar to the previous one so there definitely is a one-to-one correspondence between the applied gradient strength and the effective sample length. Possibly the 200 mm scan above is the better echo since there certainly aren't as many sinc wobbles on the sides of the echo.... even if the echo amplitude is quite a bit smaller.

So let us summarize. It seems that taking 6000 sub-systems is plenty enough to get good echoes (i.e. sufficient magnetization cancellation) for reasonable field gradient and effective sample distances. That (or maybe 3000) should be used as a generic number is most simulations. Second, we seem to get good echoes for 100 μm sample lengths (or even less) although there are definitely spins past this distance that are excited by a 10 μs pulse in a 50 T/m gradient. We can take larger samples if we wish to insure that the computations mimic experimental conditions. As for the field gradient, that value will normally be set to match the experiment anyway so we needn't consider how to "properly" set it. However we must always take quick consideration of the effective sample size when dealing with gradients much smaller and/or much larger than 50 T/m.

2.11 Square Pulse STRAFI Echoes

We have previously examined STRAFI echo maxima, assigning a Gaussian line shape to them for nice stack plots. Now we're in the position to compute full spin echoes in time following a STRAFI $90\text{-}(\tau\text{-}90\text{-}\tau\text{-})_n$ sequence. These simulations should again produce echoes with the "sinc" behavior.

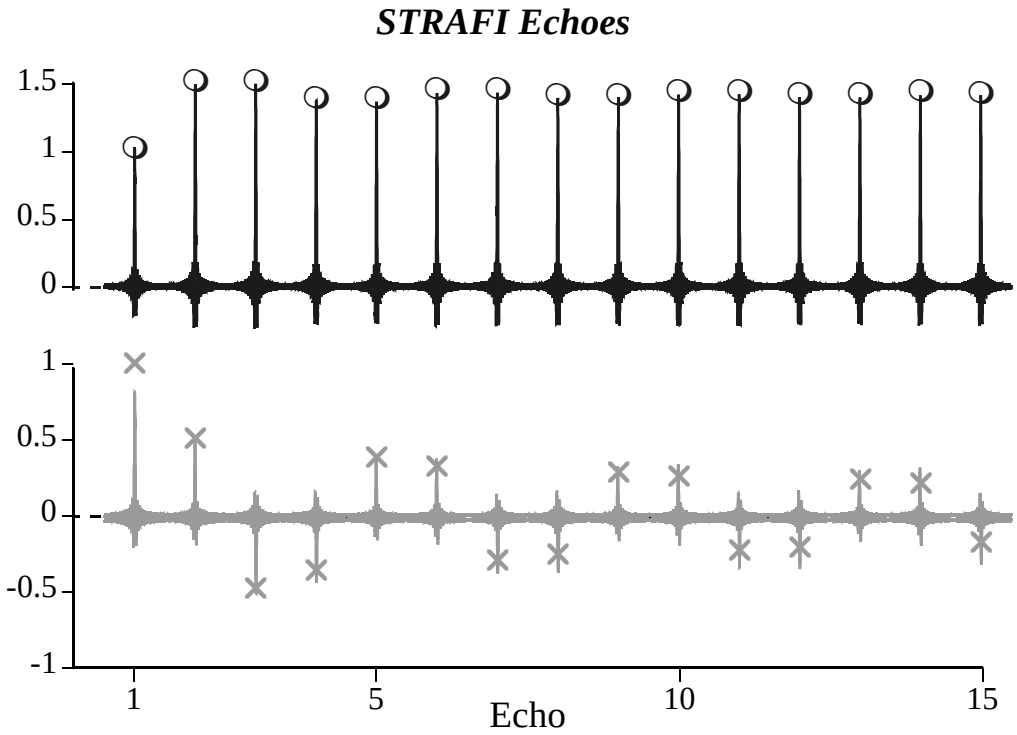

Figure 2-34 **Simulated echoes during the STRAFI spin echo sequence $90_{x,y}\text{-}(\tau\text{-}90_y\text{-}\tau\text{–})_n$. The upper trace are echoes following the x-y-y-y-.... sequence while the lower are echoes intensities following the y-y-y-y-.... sequence. Rectangular 90 pulses of 10μs length were used for excitation.The system was a single proton in a 50 T/m gradient. The effective system length 10μm, the spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz. The "ideal" intensities are overlaid on top of the echo maxima. 2000 points per echo were taken of length 3 ms about the echo center and τ was 5 ms.**

The previous figure was generated with **strafi1.cc** found on **page 78** except for the ideal echo intensities where were copied from a previous simulation.

2.12 Echo Amplitude Vs. Pulse γB_1 & t_p

Next up, it would be nice to investigate a rather cryptic statement by Randall and Bain regarding echo amplitudes. The authors indicate that for a given pulse length the echo intensities maximize for a pulse angle of 120 degrees whereas for a given pulse strength the amplitudes maximize echoes maximize at a pulse angle of 90 degrees. The validity of these statements is buried in the pulse envelope which occurs over the irradiated sample.

2.12.1 Rectangular Pulse Echo Amplitudes

A quick GAMMA computation should verify these facts for rectangular shaped pulses.

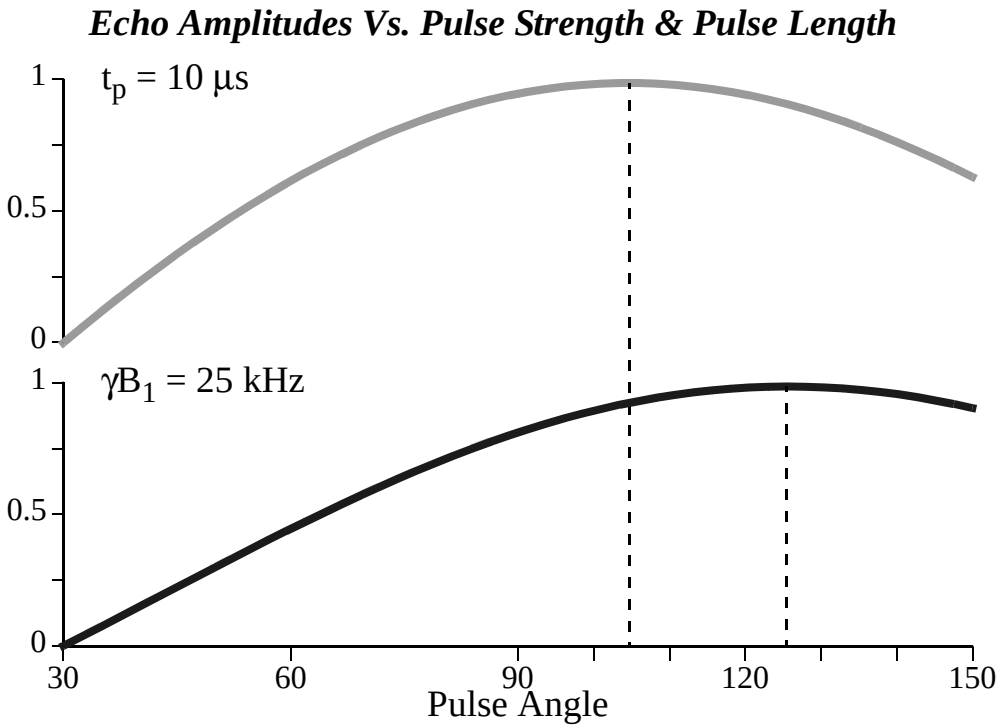

Figure 2-35 **The upper trace contains simulated intensities of the 1st STRAFI echo over differing RF field strengths, keeping the pulse length to 10 μs. The lower trace contains the same echo's intensities under a constant field strength of 25 kHz while allowing the pulse length to vary. Rectangular pulses were used for excitation. The system was a single proton in a 50 T/m gradient. The effective system length 10μm, the spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz. Both even and odd echo sequences were simulated, their intensity profiles were identical.**

The upper plot (constant pulse length) was produced by the program **EAmpVsGB1.cc** found on **page 80** whereas the lower (constant pulse strength) was accomplished with the program **EAmpVsPL.cc** found on **page 81**. Recall that as the pulse length varies the excitation profile (to a 1st approximation) changes dramatically whereas when the pulse strength varies the profile remains the same. At least under the simulation conditions, we do NOT find echo intensity maxima at either 90 or 120 degree pulse angles, *in disagreement with STRAFI literature.*

2.12.2 Gaussian Pulse Echo Amplitudes

Perhaps Gaussian pulses are more consistent in producing the "ideal" maxima. Below are the calculations under a Gaussian pulse.

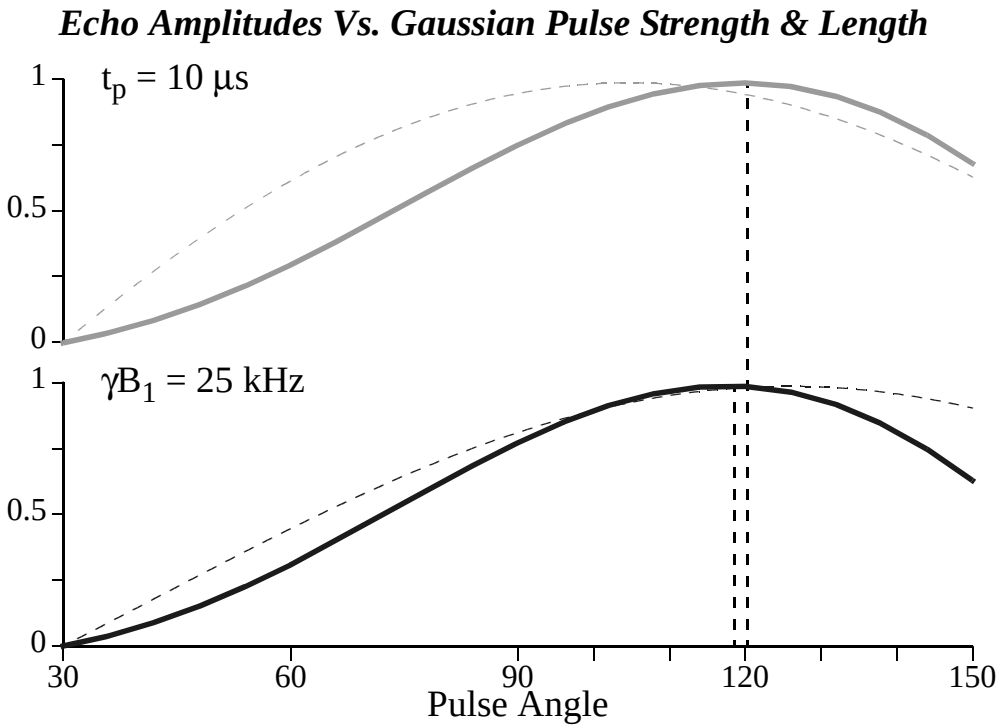

Figure 2-36 **The upper trace contains simulated intensities of the 1st STRAFI echo over differing RF field strengths, keeping the pulse length to 10 μs. The lower trace contains the same echo's intensities under a constant field strength of 25 kHz while allowing the pulse length to vary. Gaussian pulses (101 steps, 2% endpoint cutoff) were used for excitation. The system was a single proton in a 50 T/m gradient. The effective system length 10μm, the spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz. Both even and odd echo sequences were simulated, their intensity profiles were identical. The colored dashed lines are the corresponding rectangular pulse intensities.**

This is the first indication that Gaussian (or shaped) pulses can have any beneficial effects in STRAFI work. From earlier calculations in this chapter it was shown that Gaussian pulses excite a more clearly defined sample range. However, due to self-cancelling magnetization in the sample areas not on resonance, such changes were not thought to be apparent in the STRAFI echo. Here we see that this is not the case, the 1st echo intensity relative to pulse length and strength does not coincide with either the rectangular pulse results (previous section) nor with the 90 angle maximum for constant pulse length. Rather, the maximum seems to reside essentially at 120 degrees under all cases. The inference is that 120 pulses should be used in STRAFI, and that pulse calibration will be a bit easier with Gaussian pulses (probably they are better for use in imaging as well). The upper plot (constant pulse length) was produced by the program SFIEAmpVsGB14.cc whereas the lower (constant pulse strength) was accomplished with the program SFIEAmpVsPL4.cc.

2.12.3 Rectangular Pulse STRAFI Echo Amplitude Contour

It seems that we now have to further investigate the pulse angles at which STRAFI echoes maximize. Our previous simulations indicate that the 1st echo maxima do not occur at either a 90 pulse nor a 120 pulse. We have yet to determine whether this is a function of pulse length, pulse strength, echo number, gradient strength, or perhaps the number of sub-systems used in the computation. So, apparently what is needed is the ability to make a contour map of echo intensity over a range of pulse lengths and pulse strengths.

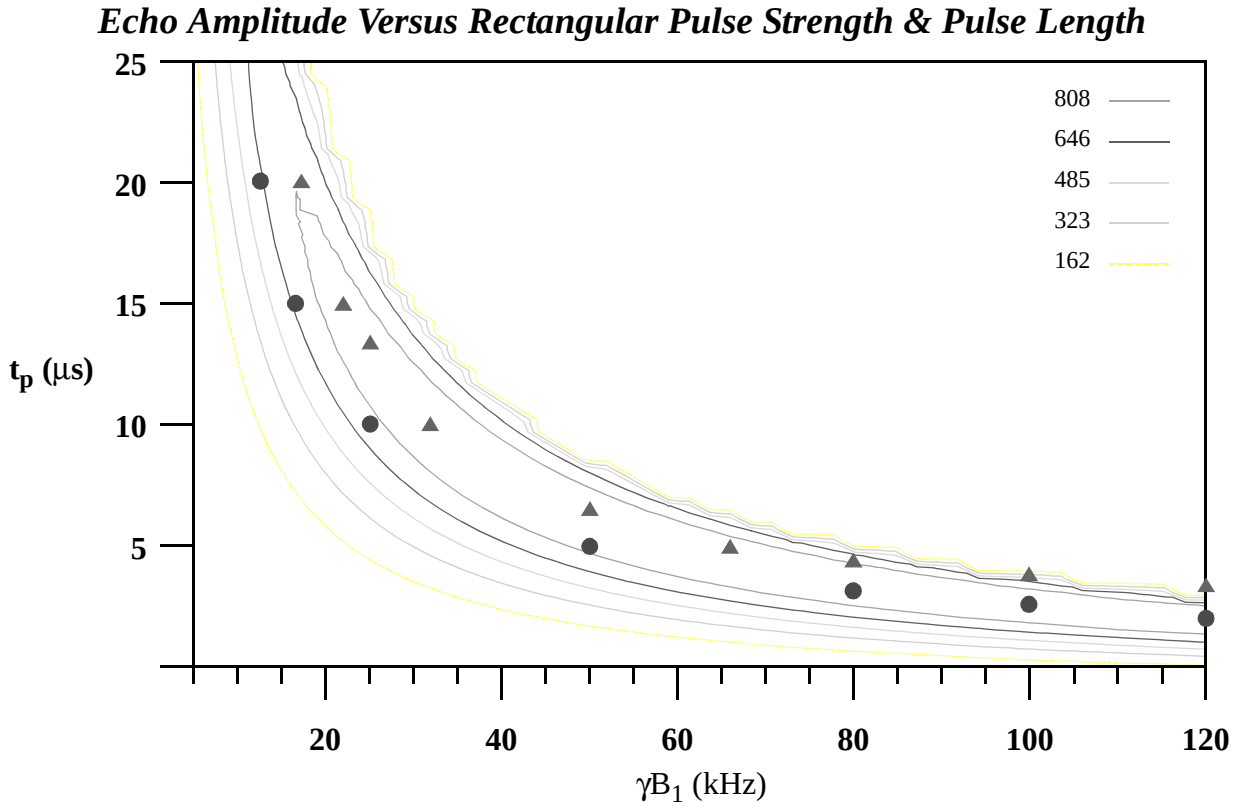

Figure 2-37 **Simulated intensities of the 1st STRAFI echo over differing RF field strengths and pulse lengths for the odd echo sequence. The red dots indicate where 90 pulses would be. Rectangular pulses were used for excitation. The system was a single proton in a 50 T/m gradient. The effective system length 10μm, the spin was set on resonance, 3000 subsystems were used, and the base field strength set to 300 MHz. The dots represent where 90 degree pulses would lie.**

Again we see clear indications that the maximum echoes do not occur at either 90 or 120 degrees except when the pulses become very short and strong. This computation was performed with the program **EAmpVsPL.cc** found on **page 81.**

2.12.4 Gaussian Pulse STRAFI Echo Amplitude Contour

A previous simulation indicated that Gaussian pulses will always maximize the echo amplitude when 120 (on resonance) pulse angles are used. Hopefully we can more precisely clarify under what circumstances this applies by again producing a contour map of echo intensity over a range of pulse lengths and pulse strengths.

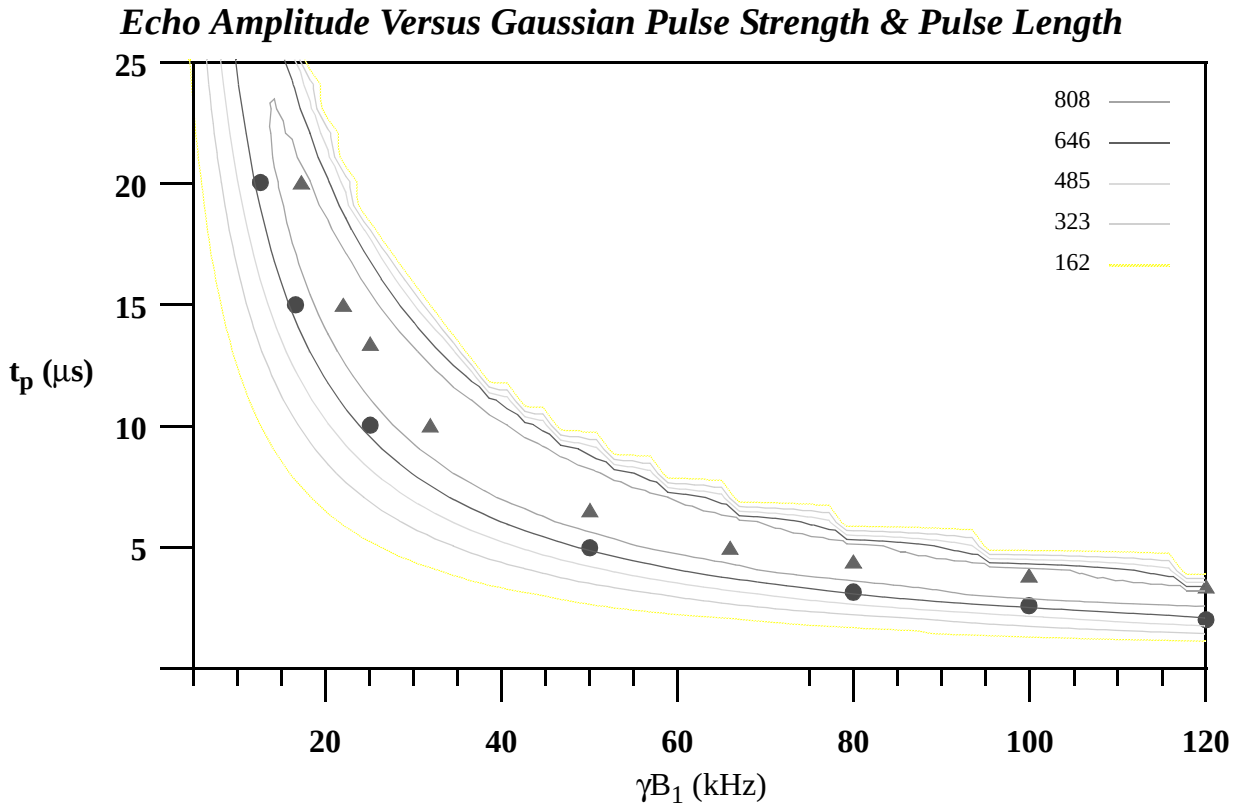

Figure 2-38 **Simulated intensities of the 1st STRAFI echo over differing RF field strengths and pulse lengths for the odd echo sequence. The red dots indicate where 90 pulses would be. Gaussian pulses were used for excitation. The system was a single proton in a 50 T/m gradient. The effective system length 10μm, the spin was set on resonance, 3000 sub-systems were used, and the base field strength set to 300 MHz. The dots represent where 90 degree pulses would lie. The pulses were built using 101 steps with a 2% endpoint cutoff.**

So, it seems that Gaussian pulses to produce maximum echoes using 120 degree pulses rather consistently.

2.13 Dipolar Echoes (Powles & Mansfield)

In this section an attempt will be made to follow one of the early papers on dipolar echoes. The authors, Powles and Mansfield[1], looked at proton echoes in gypsum ($CaSO_4.2H_20$), both the single crystal and the powder. The experiments were performed using a 90 pulse length of 1.0 μs in on protons with a Larmor frequency of 21.5 MHz ^1H. In this instance, no gradients were applied.

2.13.1 Gypsum Single Crystal

We'll begin by trying to simulate the essence of Fig. 1 in the paper. We shall work only with the "single crystal" and ignore the effects of spin diffusion. Thus, we can only get rough reproduction of their first four plots, a.) - d.) and will have to apodize our results to mimic the relaxation.

Powles & Mansfield Gypsum Echoes

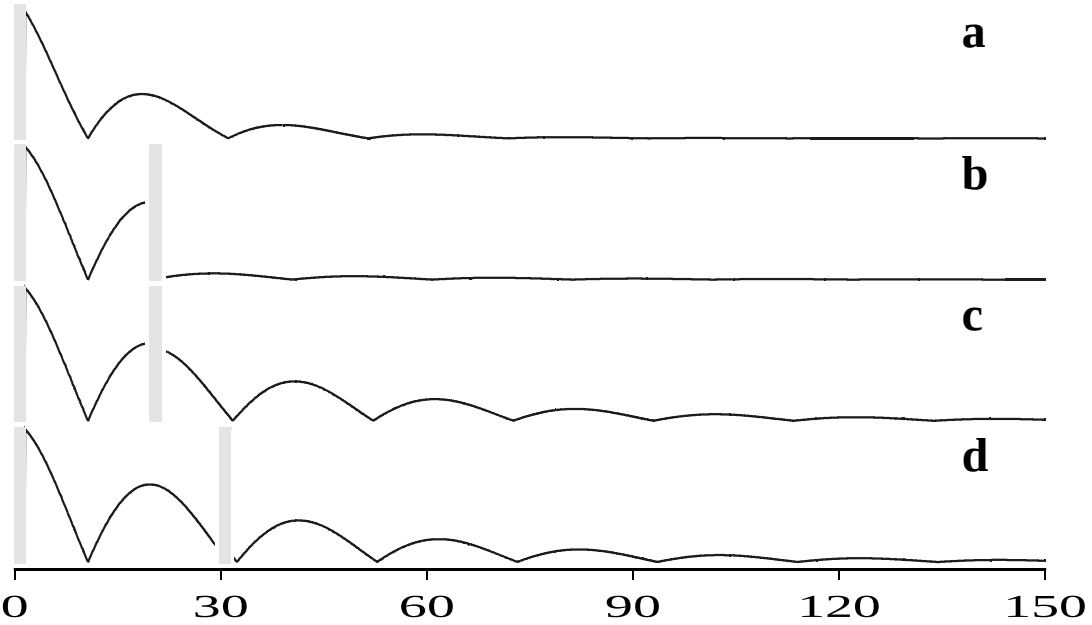

Figure 2-39 **Simulated magnetization (blue) and applied 90 pulses (yellow) on two protons. The two protons were set to resonance at 21.5 MHz and the dipolar coupling was 140 KHz. The pulse length was 1 μs. The second pulse was 90 degrees out of phase from the first except in b. No gradient was applied. Ad hoc damping terms were applied to simulate the strong transverse relaxation effects present in plot a compared to b-d. Note that the echo delay time and dipolar couplings used were approximated from the paper.**

From the previous figure it is evident that we are obtaining the basic behavior of dipolar echoes. In particular, the even sequence (b above) does not refocus magnetization as well as the odd sequence (c and d above) does. The plots were generated by the SFIMansfield3.cc given at the end of this chapter.

1. "Double-Pulse Nuclear-Resonance Transients In Solids", J.B. Powles and P. Mansfield, *Phys. Lett.*, **2**, 58-59, (1962).

2.13.2 Gypsum Single Crystal in Gradient

We'll next have a look at how gradients affect this situation. Again we shall work only with the "single crystal" and ignore the effects of spin diffusion - but still add in some ad hoc apodization so as to mimic relaxation. Now we will add in the effects of a 50 T/m external gradient.

Powles & Mansfield Gypsum Echoes With A Gradient

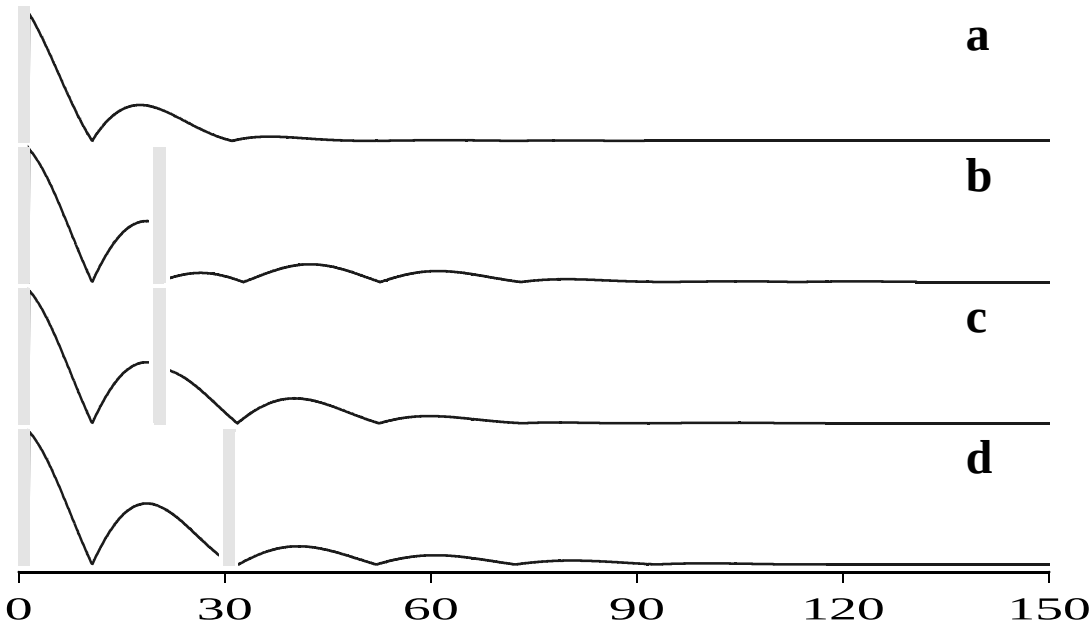

Figure 2-40 **Simulated magnetization (blue) and applied 90 pulses (yellow) on two protons. The two protons were set to resonance at 21.5 MHz and the dipolar coupling was 140 KHz. The pulse length was 1 μs. The second pulse was 90 degrees out of phase from the first except in b. A 50 T/m gradient was applied and the system taken to be 3000 sub-systems spanning 10 μm. Ad hoc damping terms were applied to simulate the strong transverse relaxation effects present in plot a compared to b-d. Note that the echo delay time and dipolar couplings used were approximated from the paper.**

Comparison with Figure 2-39 shows there are two primary effects of the applied gradient. First, all echoes now decay in time **whether or not** apodization is applied. Second. the even sequence (b.) does produce echoes almost on the same scale as the odd sequence. Note that only the echoes around 40 (in b. and c.) are related to STRAFI. Indeed, this is the 1st STRAFI echo and, from single spin calculations, the even and odd sequences should produce an echo of about equal intensity. In this case the even sequence echo is smaller than the odd one and slightly shifted. My guess is that the delay time is too small for STRAFI type of work and the "non" STRAFI intensity is competing. That will probably not be a problem in the powder sample. The plots in the figure were generated by the SFIMansfield4.cc.

2.13.3 Bodart Simulation

Philippe Bodart has done some simulations similar to the ones in the previous two sections. Here is the simulation that he ran with the parameters used (I think).

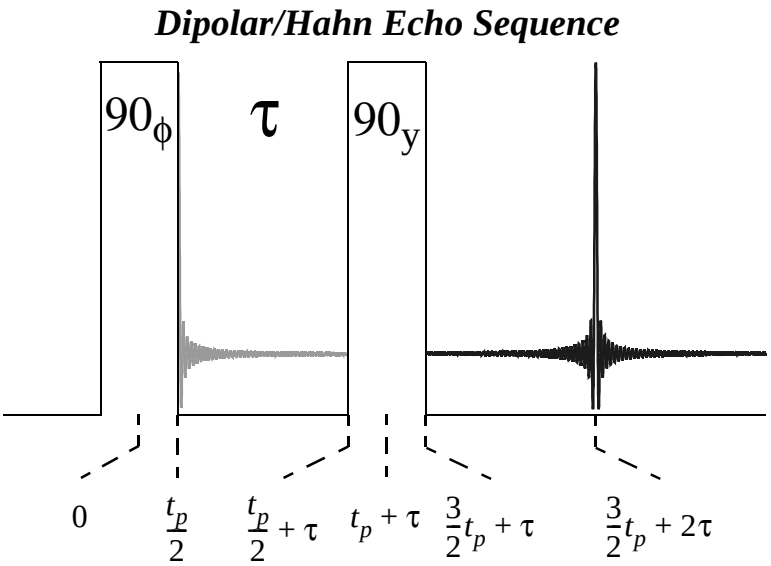

Figure 2-41 **The simulation sequence used by P. Bodart to generate Dipolar and Hahn echoes. The pulse length was 10 μs and the delay time τ set to 50 μs. The dipolar coupling was 50 KHz and simulations run on two spin 1/2 nuclei. Both the even (ϕ=y) and odd (ϕ=x) sequences were run. The effective sample length was 0.5 mm.**

Before attempting to reproduce his results, a few comments are in order.

First, a homonuclear spin pair that has a dipolar coupling of 50 KHz should produce a spectrum with transitions at +/- 75 KHz (1.5 * D). That implies that the resulting FID, if the spins are on resonance, will have an oscillation period of $1.e^{-3}/75$ seconds, or 17.33 μs. But Phillipe's oscillation times are ~30 μs!? Here is what a single crystallite should produce according to GAMMA:

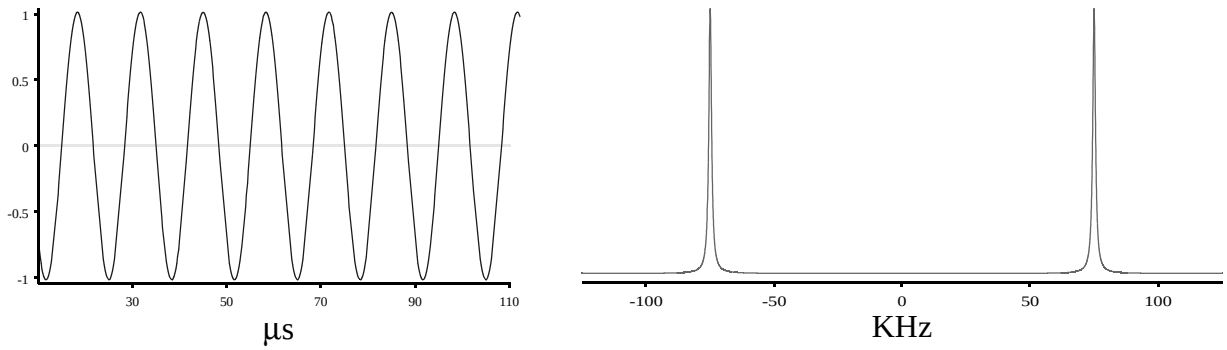

Figure 2-42 **Simulated proton spectra. The protons were set 2.4714 A apart along the field axis to produce a dipolar coupling constant of 50 KHz. Was generated from a 10 μs pulse, the FID begins immediately at the end of the pulse but the pulse beginning is set to 0 μs. The spectrum was generated from an ideal pulse (to avoid phasing effects).**

The frequency domain spectrum is correct and the FID oscillation occurs with a wavelength of ~ 17 μs, also as expected. The previous FID plot was generated with the program SFIDipSpec0.cc and the spectrum was produced using program DipSpec.cc with DipSpecHH.pset.

Philippe either used the wrong dipolar coupling or is performing the simulation on a powder sample? Lets have a look at the latter. To begin, lets have a look at the powder spectrum one would obtain without any gradient applied.

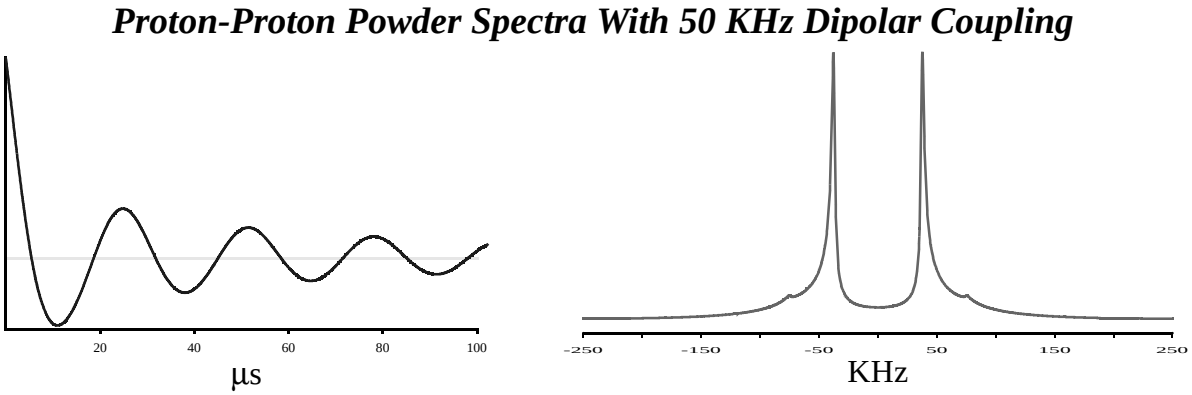

Figure 2-43 **Simulated proton spectra. The protons were set 2.4714 A apart along the field axis to produce a dipolar coupling constant of 50 KHz. Both spectra were generated from a 10 μs pulse, the FID begins immediately at the end of the pulse. A 2 KHz linebroadening was applied. 1800 crystallite orientations from 0 to 90 down the +z axis were taken for the powder average.**

The above spectra were generated from the program SFDipPow0.cc and SFIDipPow.sys by setting the gradient to 0 and taking only 1 sub-system. Note that these spectra will vary significantly when a shorter pulse is taken because the 10 μs pulse doesn't evenly excite over the powder frequency band. Here is what a shorter pulse produces under the same conditions.

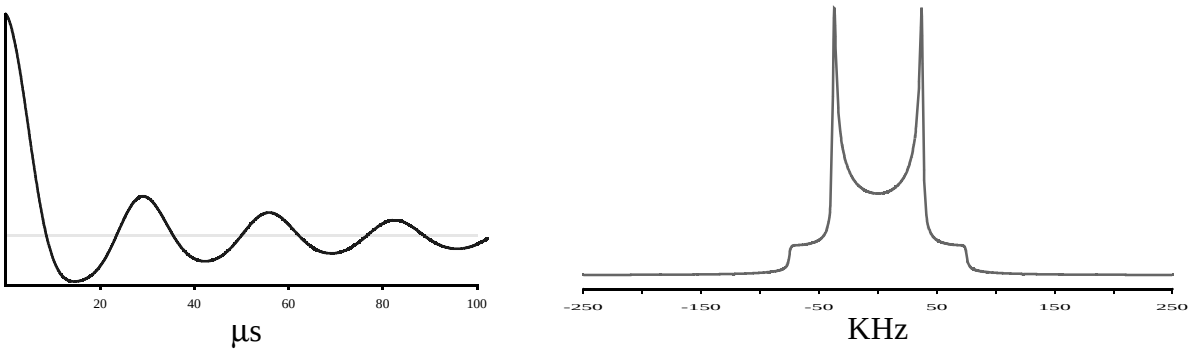

Figure 2-44 **Same as the previous simulation except with a pulse length of 1 μs .**

This alone is evidence that the longer pulse does not evenly excite the dipolar spectrum. The magnitude powder spectrum looks the same in the first figure, so it is very unlikely that phasing can adjust that powder pattern to look normal. The good news is that the FID's in the previous two figures do somewhat match what Philippe obtained (prior to the second 90 pulse) in his simulations, so we are on the right track for reproducing his work.

Next, I find it strange that the initial acquisition points (up to the second pulse) are not identical between even and odd sequences that share the same gradient. Since, in his calculations, both first pulses are 90x, shouldn't they be the same? There should be no difference whatsoever in the respective simulations up to that point.

Finally, about these computations in general. Apparently the gradient will "dephase" the dipolar echo but not the Hahn echo. Thus, the higher the gradient the more quick the dephasing, and the smaller the resulting dipolar echo. This is akin to a relaxation effect and probably has very little to do with the spin pair's dipolar coupling constant.

Now for the GAMMA simulations. First lets have a look at the pulse profile for this system.

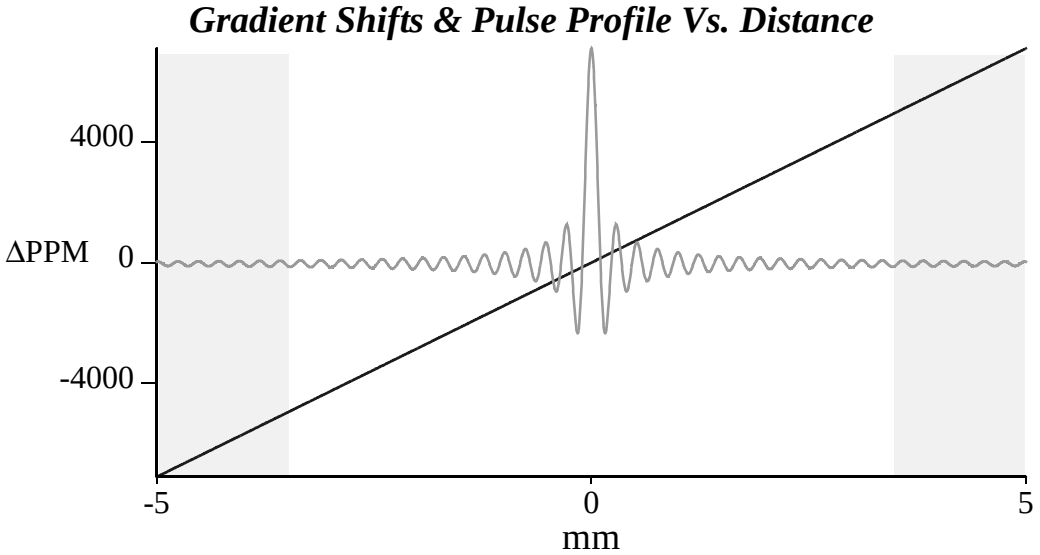

Figure 2-45 **Simulated shift offsets (blue) and excitation profile (green) versus position in a z-Gradient. The gradient was set to 10 T/M and the system was two protons in a 300 MHz field with a 50 KHz dipolar coupling. The pulse angle was set to 90 on resonance and the length to 10 μs. The yellow region is sample areas one can neglect in computations.**

The above plot (single crystallite in a gradient) shows that for a 10 μs pulse there is plenty of excitation over a distance of +/- 5mm in the sample. The pulse is sufficiently short to get appreciable excitation on both dipolar transitions (which lie at +/- 75 KHz) when irradiation occurs on resonance, although this will NOT induce the full rotation angle from the pulse. Remember, the first sinc node following a 10 μs pulse will occur at 1/10 μs or 100 KHz. Thus at 75 KHz there will be significant degredation in tip angle from what is set on resonance. Of course, off resonance the pulse will not excite the two transitions equally. A repeat computation with 50 T/m gradient collapsed the horizontal scale of the excitation profile roughly by a factor of 5 (and the vertical shift scale greatly expanded). At a gradient of 0.1 T/m the excitation plot never goes below half the maximum and at 0.001 T/m it appears constant throughout the sample.

Philippe takes his sample to be 0.5 mm wide (which for me is +/- 0.25) and that will obviously not suffice for gradient averaging at the lower the gradient strengths he uses. His run at 0.001 T/m cannot exhibit any gradient effects and is virtually equivalent to using no gradient. A quick calculation insures this is the case. The additional field contribution at the ends of Phillipes sample (we'll take

that as .5 mm) is

$$(0.5 \times 10^{-3} m)(0.001 T/m) \; = \; 5 \times 10^{-7} T$$

Multiplying by then $^{1}\mathrm{H}\,\gamma$ of $4.26 \times 10^{7} Hz/T$, we obtain a frequency shift of about **21 Hz** over the entire sample - and more like half of that in Philippes calculation. Recall that this change is out of the 75 KHz that the transitions are already resonating at. A check with GAMMA verifies this in giving the system @ 0.25 mm an additional shift of 10.64 in a 0.001 T/m gradient.

So, any decay in that simulated data at 0.001 T/m must be caused from the powder averaging exclusively. It looks to be a bit slow based on the GAMMA calculated FID's presented earlier in this section. Could the second pulse have such an effect on the signal degradation? Perhaps so!

To further show this we can generate some dipolar FID's and spectra at the gradients Philippe used.

Dipolar FID & Spectra Vs. Field Gradient

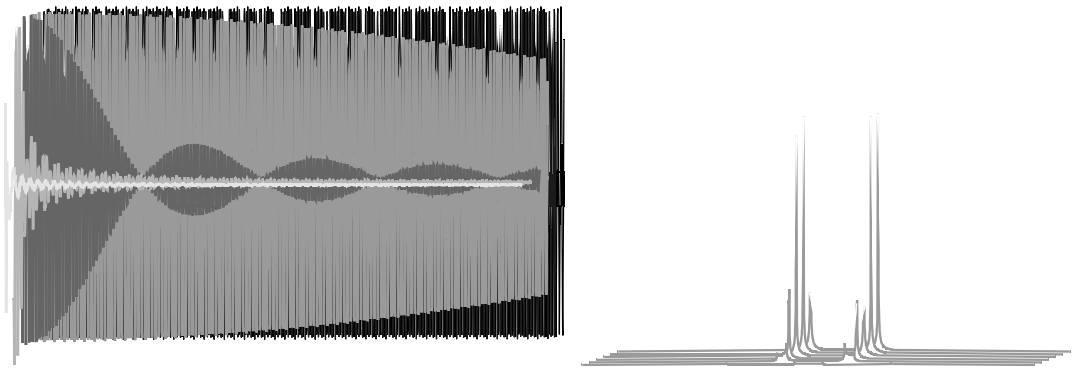

Figure 2-46 **Simulated proton spectra follwing a 10 μs pulse. Two protons at 300 MHz with a dipolar coupling of 50 KHz was input. 3000 sub-systems were used with system length 0.5 mm. The applied gradient varied (back to front) as: 0, 0.001, 0.01, 0.1, 1.0, & 10.0 T/m. This is a single crystal simulation, NOT a powder average.**

A gradient of 0.001 T/m produces spectra that are virtually identical to spectra without a gradient. Does a longer sample size cause this to be different?

Dipolar FID & Spectra Vs. Field Gradient

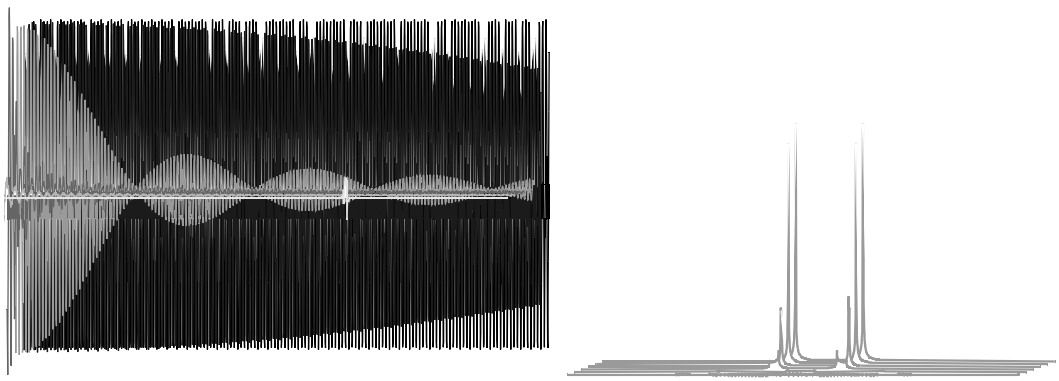

Figure 2-47 **Simulated proton spectra follwing a 10 μs pulse. Two protons at 300 MHz with a dipolar coupling of 50 KHz was input. 3000 sub-systems were used with system length 5 mm. The applied gradient varied (back to front) as: 0, 0.001, 0.01, 0.1, 1.0, & 10.0 T/m.**

The pulse profiles indicated that a 0.5 mm sample length was too small to span the excitation, although that is also limited by the experimental hardware. For fun I will regenerate the plots using a much longer effective sample. Note that the frequency domain spectra shown are in a magnitude mode to avoid having to deal with phasing which results from a finite length pulse. Also the spectral width used was 1000 KHz and the FIDs span 2.048 ms.There is some, but not much, difference in taking a broader sample area. The previous two figures were generated with the program SFDipSpec2.cc using the input file SFIDipSpec.sys, the parameters mentioned, and the gradient list file SFIDipSpecGrd.pset. Now, back to powder samples. Here are some spectra in a gradient.

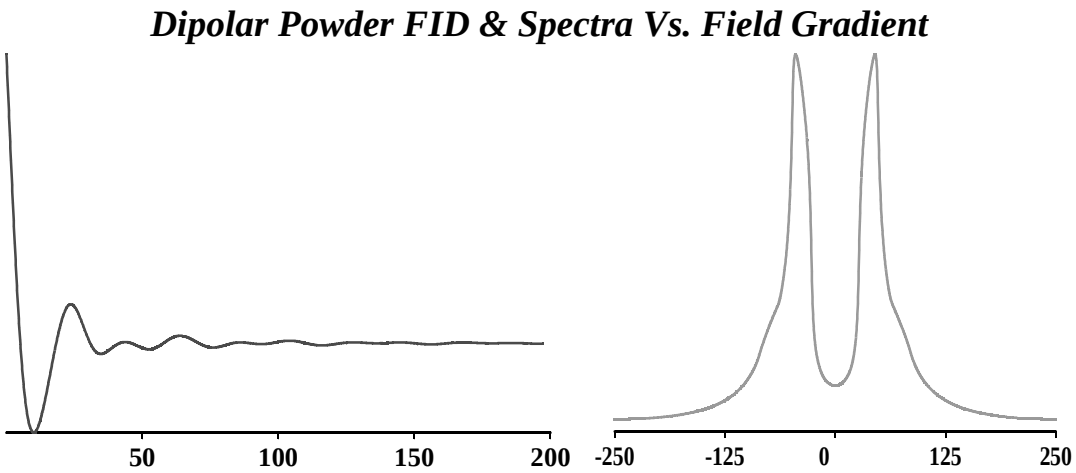

Figure 2-48 **Simulated proton spectra follwing a 10 μs pulse. Two protons at 300 MHz with a dipolar coupling of 50 KHz was input. 1000 sub-systems were used with system length 0.5 mm. The applied gradient was 1.0 T/m and 900 crystal orientations were taken in the powder average. The spectral width was set to 500 KHz and a 2 KHz linebroadening applied.**

The odd powder pattern is caused from an insufficiently short pulse (i.e. 90 pulses aren't really being applied to the transitions). These spectra clean up with a shorter pulse.

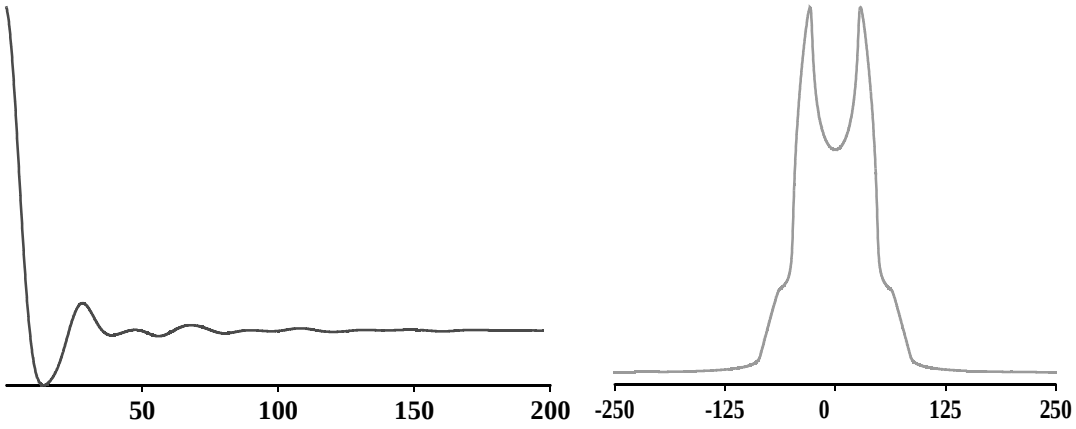

Figure 2-49 **Simulated proton spectra follwing a 1 μs pulse. Two protons at 300 MHz with a dipolar coupling of 50 KHz was input. 300 sub-systems were used with system length 0.5 mm. The applied gradient was 1.0 T/m and 90 crystal orientations were taken in the powder average. The spectral width was set to 500 KHz and a 2 KHz line broadening applied.**

Note that although I am applying an apodization to the data, that alone is not damping the data down. The gradient is largely responsible for that in the above simulations (compare the FID decay with the no gradient powder average simulation).

There are some problems with use of a 10 μs pulse as demonstrated in the previous section. However, that doesn't imply dipolar/Hahn echoes won't be generated under the conditions set in Philippe's simulations. The echoes will probably be generated just fine, only they won't be as intense as the ones generated by a 1 μs pulse.

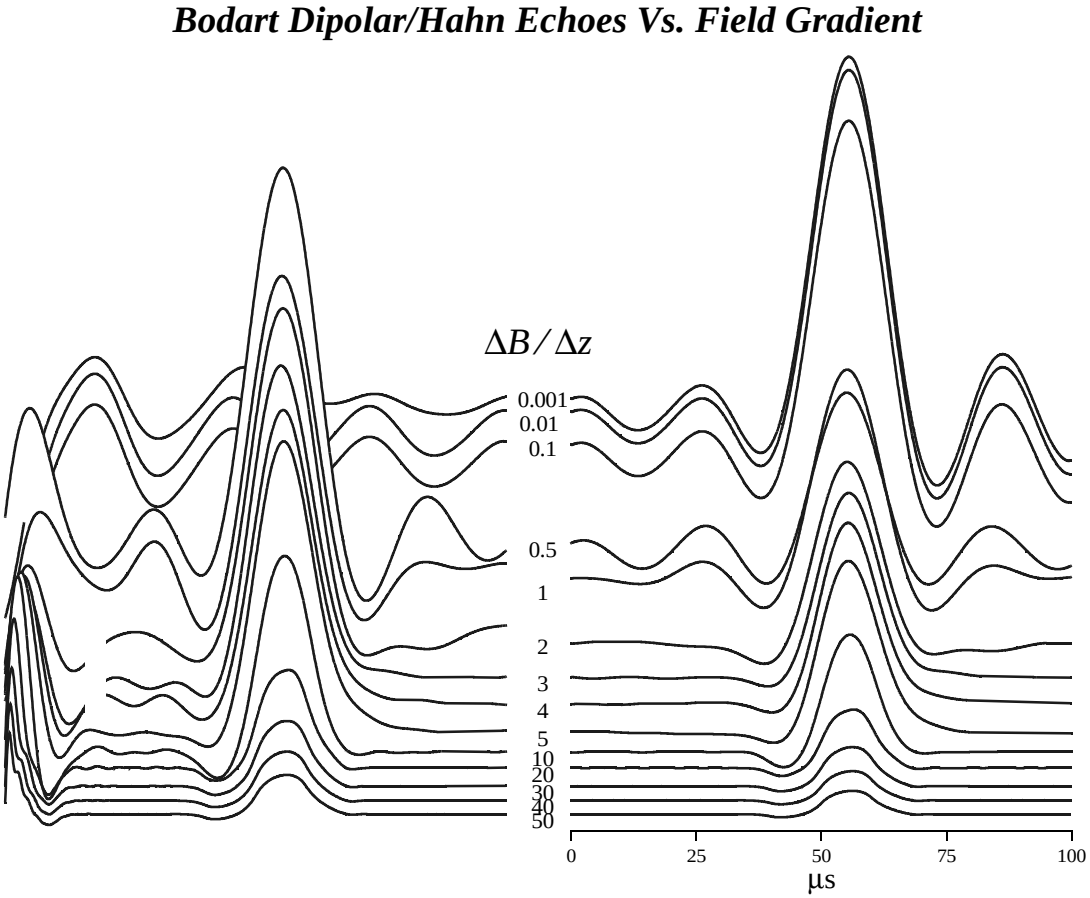

Figure 2-50 **Simulated echo following a 90x-τ-90y sequence. The pulse length was 10 μs and 500 subsystems were considered spanning a 0.5 mm effective sample length. The powder average used 270 angle increments down from +z to the horizontal plane. The two protons were set to resonance at 300 MHz and the dipolar coupling was 50 KHz.**

Bodart Dipolar/Hahn Echoes Vs. Field Gradient

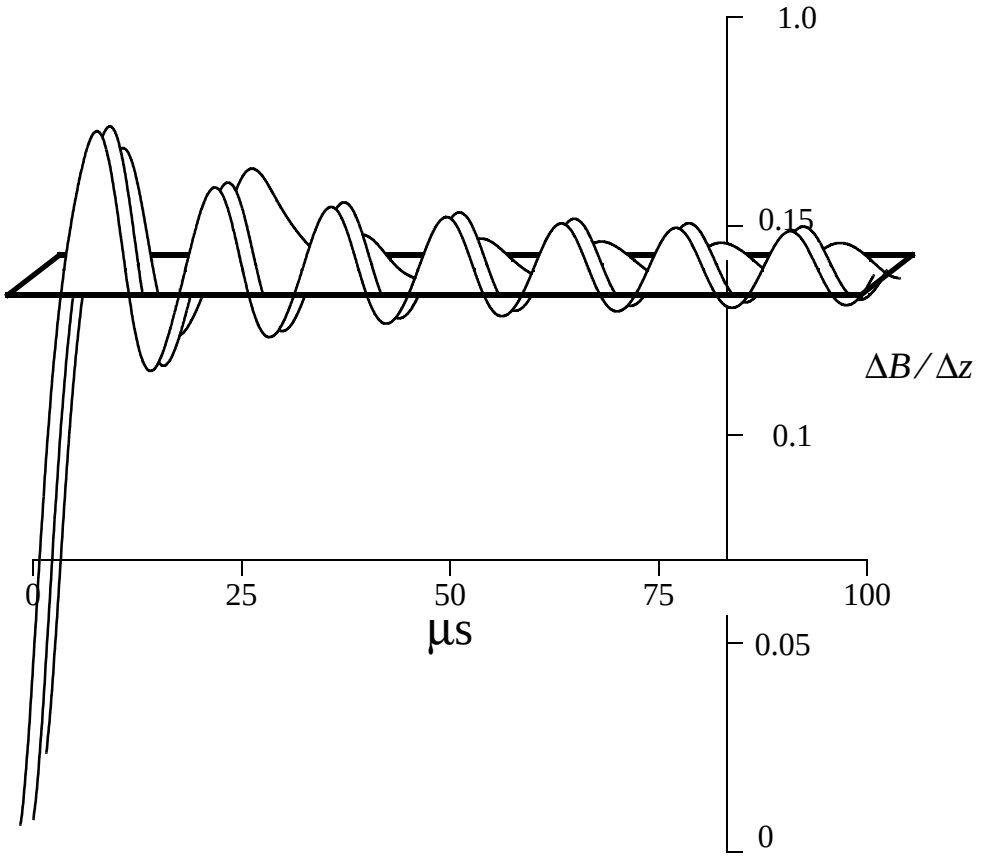

Figure 2-51 **Simulated echo following a 90x-τ-90y sequence. The pulse length was 10 μs and 500 subsystems were considered spanning a 0.5 mm effective sample length. The powder average used 270 angle increments down from +z to the horizontal plane. The two protons were set to resonance at 300 MHz and the dipolar coupling was 50 KHz.**

2.14 Sodium STRAFI (Bodart, et. al.)

In this section an attempt will be made to follow the STRAFI work on ^{23}Na systems done by of Bodart, Nunes, and Randall[1]. This nucleus is of spin I=3/2. The experiments were performed using a pulse length of 4.7 μs in a field gradient of 37.5 T/m on a 2.9T magnet (300 MHz ^{1}H).

2.14.1 ^{23}Na Pulse Profile & Shift *Versus* Distance

To begin, we first examine how a 4.7 μs 90 pulse affects the sodium system. Using the previous program[2] for examination of spin system response versus distance, we obtain the following.

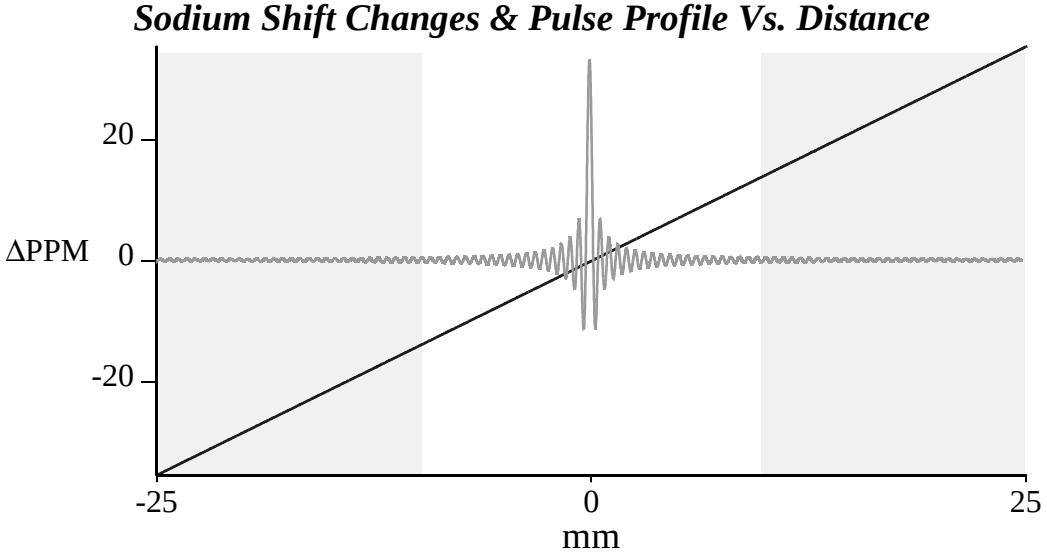

Figure 2-52 **Simulated shift offsets (blue) and excitation profile (green) versus position in a z-Gradient. The gradient was set to 37.5 T/m and the system was a single sodium in a 300 MHz field. The pulse angle was set to 90 on resonance and the length to 4.7 ms. The yellow shaded regions thus represent areas of the sample which do NOT contribute to any simulated result of the pulse. 8192 points were calculated.**

For this short pulse length there are contributions to the signal coming from areas in the sample several millimeters outside of the "on-resonance" distance. We should therefore keep our effective sample length at ~50 mm for 23Na STRAFI calculations at this field gradient and pulse length. Let's check if this is reasonable. We know that the first "sinc" node from the pulse will occur at 1/tp = 1/(4.7μsec) = 0.21x10^{6}/sec. This implies that the first node occurs a a distance where the gradient alters the sample shift by 0.21x10^{6}/sec. A quick calculation reveals that for a 4.7μsec ^{23}Na pulse & a 37.5 T/m gradient, 50 sinc nodes on each side of zero are roughly spanned in

$$z \approx \frac{100}{(2.2x10^{8} radT^{-1}s^{-1})(1cycle/(2\pi rad))(4.7x10^{-6}\text{s})(37.5Tm^{-1})} \approx 16.2mm. \qquad (2\text{-}3)$$

1. "Stray-Field Imaging of Quadrupolar Nuclei of Half Integer Spin in Solids", Philippe Bodart, Teresa Nunes, and Edward W. Randall, *Sol. St. Nucl. Magn. Reson.*,**8**, 257-263, (1999).
2. The program GzPOffset.cc given at the end of this Chapter, page 65, will be utilized.

2.14.2 ^{23}Na Magnetization *Versus* Distance

Using a modified version of a previous program, we can repeat the Benson and McDonald calculation to watch the intensity contributions in the sample during the first few STRAFI echoes.

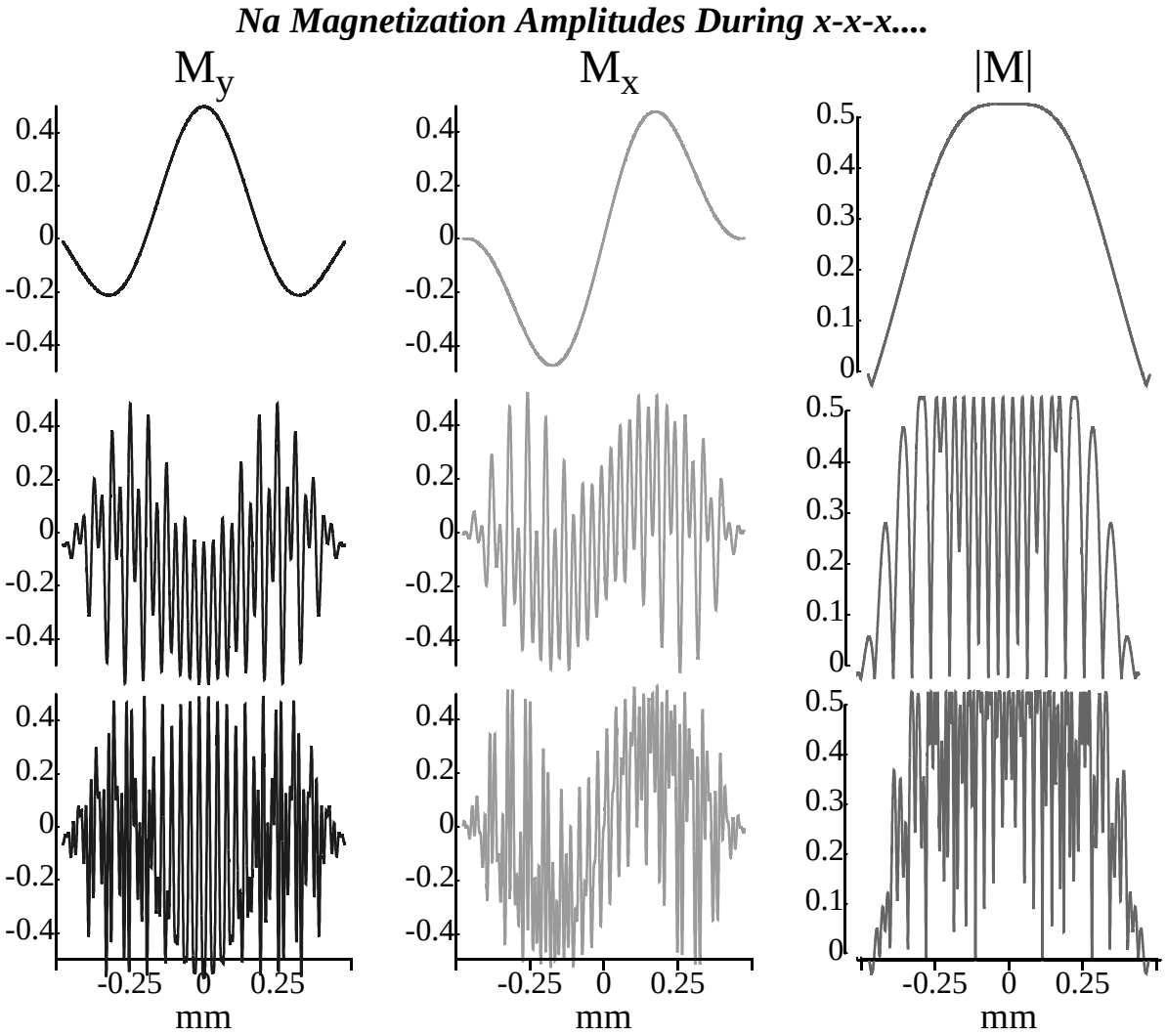

Figure 2-53 **Simulated magnetization during STRAFI sequence 90_x-(τ-90_x-τ–)$_n$. Row 1 are the values immediately after the pulse. Row 2 are those at the 1st STRAFI echo and Row 3 are those at the 2nd STRAFI echo. Here t_p = 4.7μs, tau = 40 μs. The applied gradient was 37.5 T/m, the base field 2.9 T, and 3000 subsystems were used.**

As is evident from these plots, the short pulse (4.7 usec) on 23 Na does a good job at exciting spins well off resonance. At 1 mm in a 37.5 T/m gradient there is still plenty of transverse magnetization generated. This is in agreement with the previous computation of the pulse profile on sodium. In contrast, the authors of the ^{23}Na STRAFI paper argue that the effective sample length is only about 300 μm! These calculations imply that such can be the case only if the detector picks up magnetization exclusively from the sample in that range or the outer magnetization always cancels.

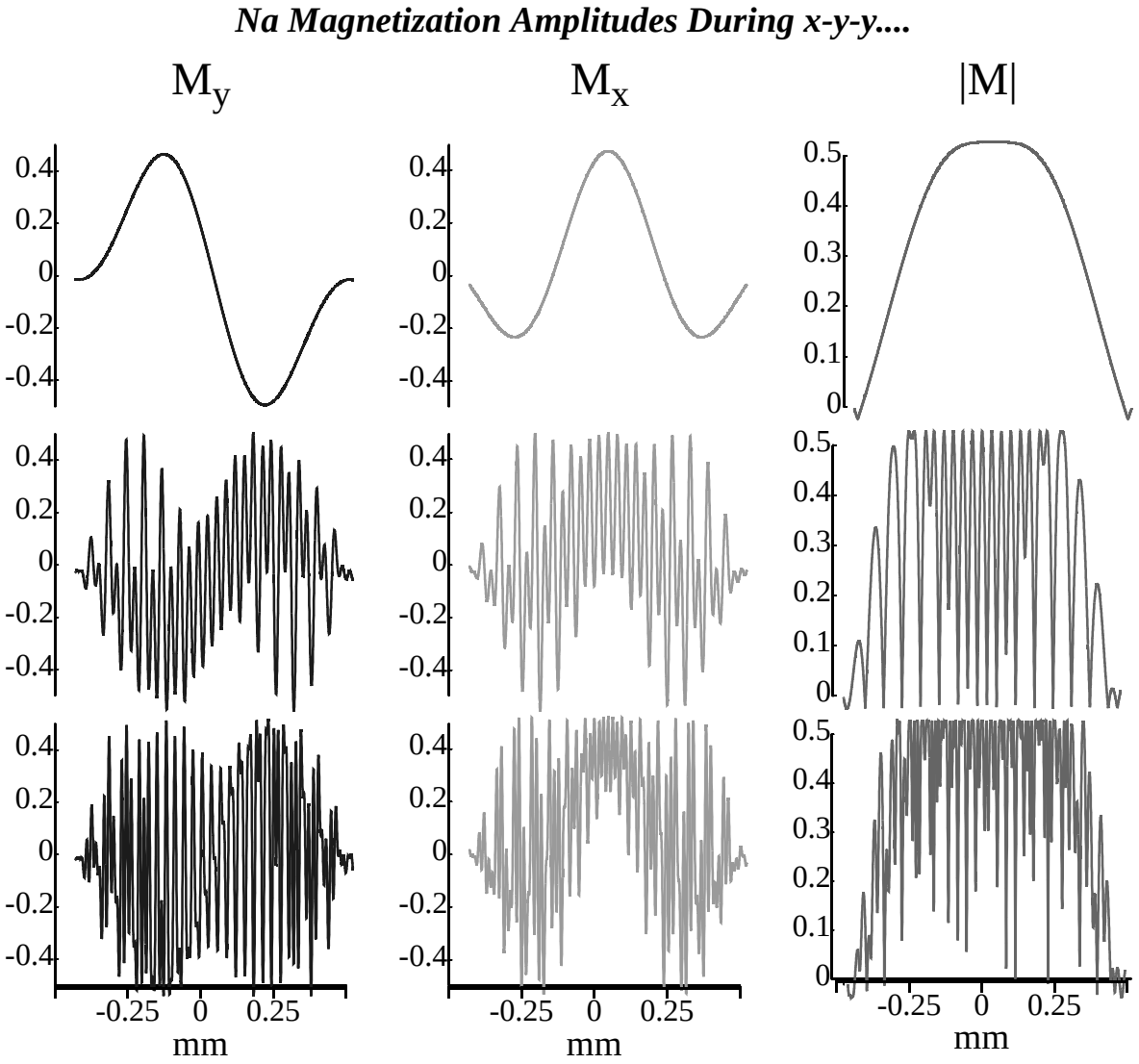

Figure 2-54 **Simulated magnetization during STRAFI sequence 90_x-(τ-90_y-τ–)$_n$. Row 1 are the value immediately after the pulse. Row 2 are those at the 1st STRAFI echo and Row 3 are those at the 2nd STRAFI echo. Here t_p = 4.7 μs, tau = 40 μs. The applied gradient was 37.5 T/m, the base field 2.9 T, and 3000 subsystems were used.**

Both figures were generated from a modification of the GAMMA program SFIBenson.cc which is found at the end of this Chapter, page 66. It will be interesting to see if use of a 300 μm effective sample size will suffice to simulate the STRAFI echoes here.

2.14.3 Sodium FIDs

At this point we have an estimation of the system parameters required to get good simulations of sodium in the desired gradient. In order to check that this is the case, we should first generate some FID's following a nice 90 pulse. If we are correct in our estimations we should get a nice decaying FID from a proper accounting of the gradient effects[1].

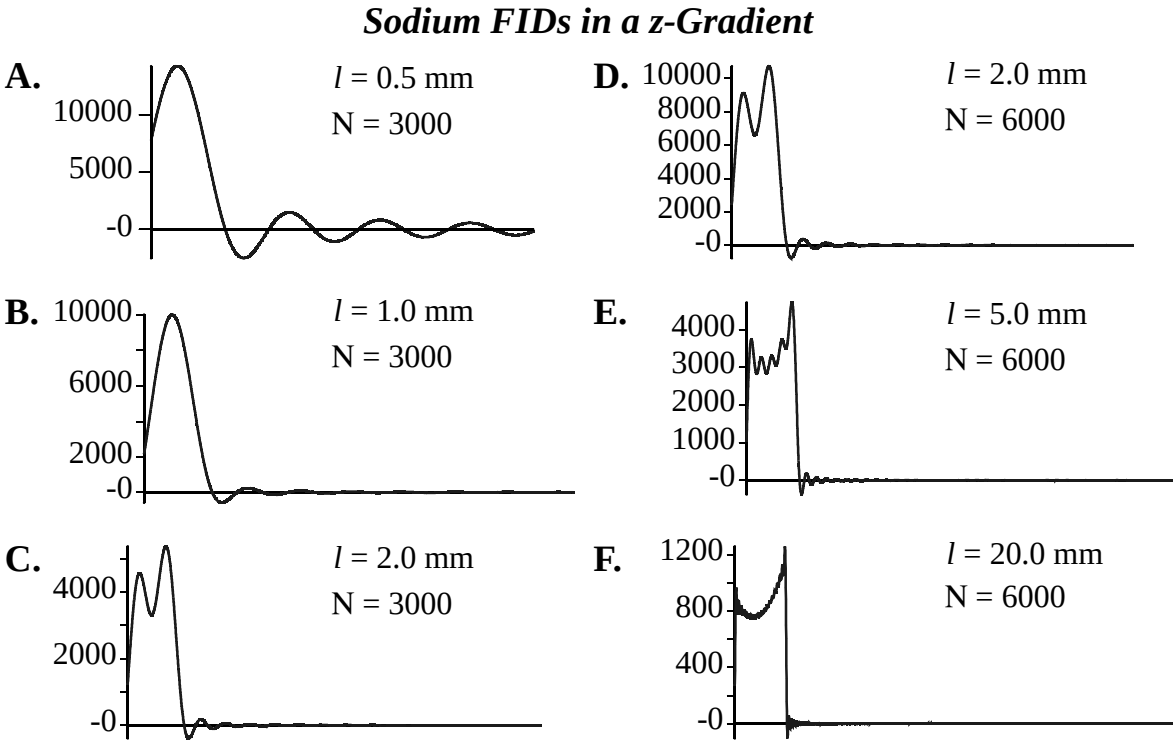

Figure 2-55 **Simulated FIDs following application of a 90 pulse of 4.7 μs. The FIDs were acquired over a 40 μs period, 4096 points were taken. The system was a single sodium in isotropic media. The gradient used was 37.5 T/m.**

As the system length increases from 0.5 mm (A.) to 2 mm (C.) we see a drop in the signal intensity as well significantly diminished sinc lobes. That can either come from more effective averaging over the sample or from an averaging of sub-systems that simply do not contribute to the signal. The inherent FID phase (i.e. not beginning with a max at t=0) is expected (?) since all but the on-resonance spins will precess during the course of the pulse. The increase in sub-systems (D.) has no effect on the FID shape, indicating that we are performing an adequate integration over the system - which we had surmised was an adequate number from earlier simulations in this chapter. In the next graph (E.), the system length is increased to 5 mm, and then to 20 mm (F.) If N is increased to 50000 and or the length to 9 cm, there is no change in the last FID.

Perhaps it is not obvious, but the FID is simply turning into a rough Fourier approximation of the applied pulse! The linewidth at higher and higher sub-systems becomes the pulse linewidth of 4.7

1. The fid simulation uses the program GrdFID1.cc found on page 75.

us. Of course, this is not what is observed experimentally for a number of reasons: relaxation effects, pulse intensity attenuation over the sample, and finite detector width. In the simulation we have neglected relaxation effects and the sample is irradiated with the same rf-power over the entire sample length. Furthermore, we are able to detect over the entire sample volume, which in the last run above (F.) spanned 2 cm, quite a bit further that the 300 μm which the authors state is the effective detected sample length.

Evidently we should keep our sample length down to more closely mimic the ability of our detection coil and perhaps even attenuate the rf-power as we move away from the coil center (the resonance area). It is important to always remember the limitations of the simulation. We are forced to discretely sample systems in the gradient which contribute to the signal, in contrast with the continuum of systems that do so in an experiment. Thus we can only approximate the echo generation (and side lobes) and may see some inconsistencies if we push the simulation parameters to hard. Perhaps taking a 20 mm sample is such an instance, and we are simply loosing to much in the computation when we do so.....

In the Na paper, the authors claim that any odd echoes or FIDs following the pulses are completely gone by the time a STRAFI echo is collected.... which would be around 5 μs. All but the first simulation above would probably suffice in that regard, but that uses an even larger sample than was used in the paper simulations. Either I am leaving out something Philippe used (such as signal damping) or the amount of signal left is not of consequence. We'll see that when we get to some STRAFI simulations.

2.14.4 Sodium Spin Echoes

Next up we have a look at typical spin echoes (rather than the STRAFI type) to see how well sodium behaves[1]. The following figure contains the sodium echo following a 90-tau-180 sequence, where zero time is the refocusing point.

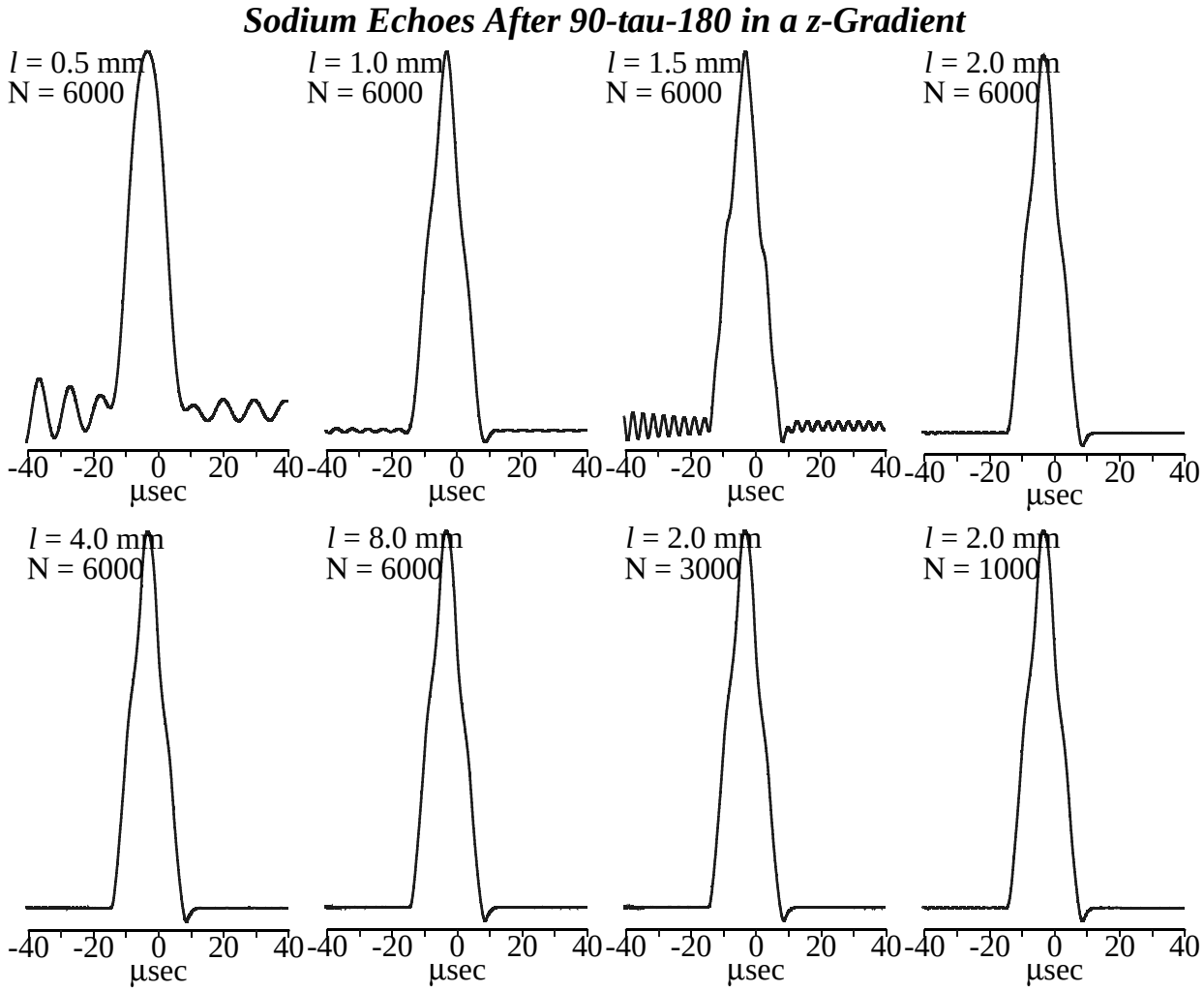

Figure 2-56 **Simulated spin echoes following application of a 90-τ-180 sequence where the 90 pulse length was 4.7 μs and the τ delay 40 μs. The echoes were acquired over a 80 μs period centered at time τ after the 180 pulse, 4096 points were taken. The system was a single sodium in isotropic media. The gradient used was 37.5 T/m.**

After a length of only 2 mm and for as few a 1000 sub-systems the echo lineshapes appear to be the same. This is in spite of that fact that the FID's for the same system are quite different! Another interesting aspect is that the echo width is ~10 ms, roughly twice as broad as the observed Na STRAFI echo widths. Here we refocus with a π pulse compared with π/2 in STRAFI.

1. This echo simulation uses the program SpinEcho.cc found on page 77.

2.14.5 Sodium STRAFI Echoes

At this point we'll begin looking at some STRAFI echoes. Since we're still "exploring" we will just look at the 1st echo with a program very similar to the one that generated the previous figure.

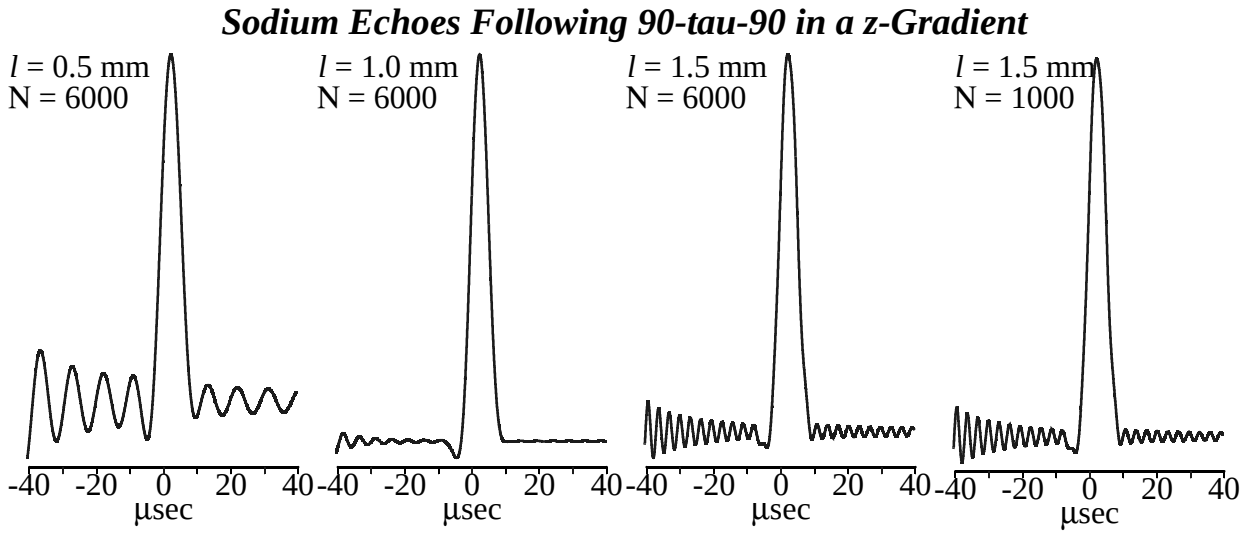

Figure 2-57 **Simulated echoes following application of a 90-τ-90 sequence. The acquisitions were centered about time τ from the last pulse. The system was a single sodium in isotropic media. The number of sub-systems and effective sample size are as indicated. The base field strength was set to 7.04 T (300 MHz ^{1}H, 80 MHz ^{23}Na). The gradient used was 37.5 T/m and the 90 pulse length 4.7 μs. The horizontal axis reflects time about the center of the echo, the echo collected for 80 μs and τ = 50 μs.**

These echoes look to be compatible (comparable widths) to the NaCl simulations in the paper (Fig. 2b and Fig. 3c). The baseline "noise" that appear in Philippe's even sequence simulation is present only in the runs with longer effective sample lengths here. In fact, it seems a bit odd that there is any difference between his even and odd simulations for the first echo. In the publication, both the experimental and simulated echoes show the odd sequence on NaCl produces nice clean echoes whereas the even sequence echoes get progressively more distorted. Perhaps I have noticed some tiny differences in the baseline wobbles.... In any event, the above echoes have ~ 5 μs width which is compatible with the published simulations.

On to the full STRAFI calculations. I'll stick with a 1.5 mm effective sample length since the baseline wobble level looks more like the simulations in the paper there (compared to.3 mm used in the paper). Furthermore, I now must switch to use a field base of 2.9T (Ω^{1H} = 123.4 MHz, Ω^{23Na} = 32.3 MHz). That change should make no difference in these type of calculations. For comfort (versus shorter computation time) I'll keep with a larger number of sub-systems.

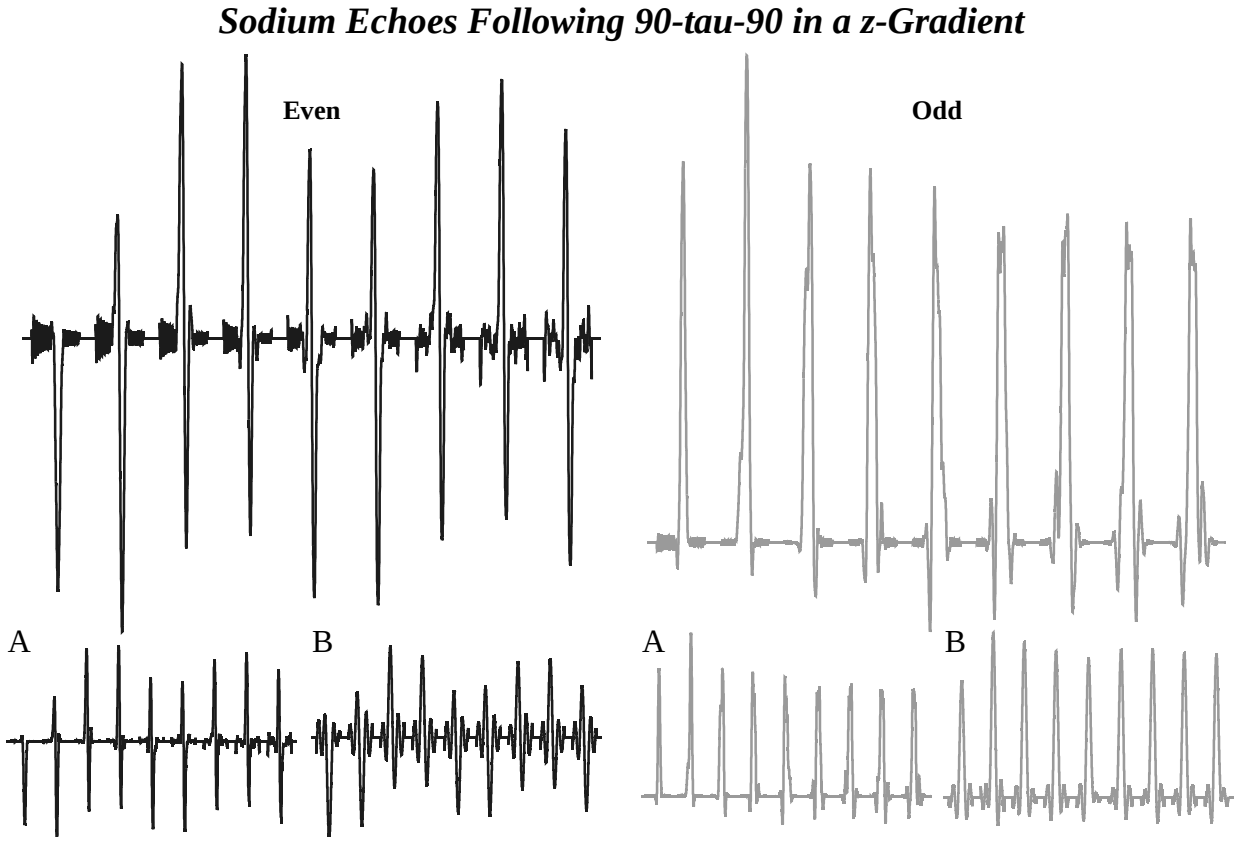

Figure 2-58 **Simulated sodium echoes following application of a 90-τ-90 sequence. The acquisitions were taken in a window 60 μs centered about time τ from pulses. 6000 sub-systems spanning 1.5 mm were taken. The base field strength was set to 2.9 T (123.4 MHz ^{1}H, 32.3 MHz ^{23}Na). The gradient used was 37.5 T/m and the 90 pulse length 4.7 μs. The horizontal axis reflects time about the center of the echo, the echo collected for 80 μs and τ = 50 μs. The insets are repeat calculations using A.) 10000 subsystems and a 3 mm sample length & B.) 1000 subsystems and a 0.3 mm length.**

The above simulations are not nearly as good as those in the publication. Why not? I don't know what can possibly be missing. For example, in both the experiment and simulation the third odd echo is the strongest... not seen in any of these simulations. Nor has this been the maximum echo in any previous calculation performed in this chapter so far, except the one using a 50 μs Gaussian pulse on a proton. The inherent "phasing" in the even echoes doesn't match the paper figures either (after the first two echoes).

All of this is a bit distressing, but perhaps there are other parameters that figure into this stuff or possibly I am just not using the proper ones. Is it possible that the detection method used was different? The only "better" part of the GAMMA simulations above are the peak splitting in the later echoes of the odd sequence (which seem to be in the experimental work too?) and the nice wobble in intensity of the higher odd echoes (also in the experimental).

2.15 Programs and Input Files

GzPOffset.cc

```
/* GzPOffset.cc ***************************************************************-*-c++-*-
**                                                                              **
**                 Test Program for STRAFI Using The GAMMA Library              **
**                                                                              **
**  This program uses the GAMMA class sys_gradz in combination with the         **
**  GAMMA module Gradients2 in dealing with STRAFI calculations. This           **
**  paraticular program is used to investigate the range of shift               **
**  offsets and pulse excitation vs. distance expected in a z-gradient.         **
**                                                                              **
** Author:      S.A. Smith                                                      **
** Date:       2/9/97                                                          **
** Update:      2/9/97                                                         **
** Version:     3.6                                                            **
** Copyright:   S. Smith.  You can modify this program as you see fit          **
**                for personal use, but you must leave the program intact       **
**                if you re-distribute it.                                      **
**                                                                              **
*********************************************************************************/

#include <gamma.h>                          // Include GAMMA

main (int argc, char* argv[])

  {
  sys_gradz sys(1);                         // A working spin system
  sys.Omega(300.0);                         // Field strength
  int qn = 1;                               // Query parameter
  String I;                                 // Isotope type
  query_parameter(argc, argv, qn++,         // Read in the gradient
    "\n\tSpin Isotope Type [1H, 13C, 2H,...]? ", I);
  double Gz;                                // Field gradient
  query_parameter(argc, argv, qn++,         // Read in the gradient
    "\n\tApplied Field Gradient (T/m)? ", Gz);
  sys.BoGrad(Gz);                           // Set the field gradient
  double slen;                              // Effective sample length
  query_parameter(argc, argv, qn++,         // Read in the length
    "\n\tEffective Sample Length (um)? ", slen);
  sys.SysLen(slen*1.e-6);                   // Set the length
  int nss;                                  // Point to plot
  query_parameter(argc, argv, qn++,         // Read in the points
    "\n\tNumber of Points? ", nss);
  sys.NSS(nss);                             // Set the # of points
  double tp;                                // Pulse length
  query_parameter(argc, argv, qn++,         // Read in the length
    "\n\tPulse Length (us)? ", tp);
  tp *= 1.e-6;                              // Set length in usec

  row_vector data1(nss), data2(nss), vxs[2];// Shifts and excitation
  int i;
  for(i=0; i<nss; i++)                      // Loop over points
    data1.put(sys.SubSysPPM(i,0),i);        // and store the shifts
  double sf = data1.getRe(data1.max(1));    // Maximum vector value
  vxs[0] = data1;                           // Store the vector

  gen_op sigma0 = sigma_eq(sys);            // Equilibrim
  gen_op H0 = Ho(sys);                      // Base Hamiltonian
  gen_op D = Fm(sys);                       // Detection operator
  H0.set_EBR();                             // Use the eigenbasis
  D.Op_base(H0);                            // Put D int H eigenbasis
  gen_op Hs[nss];                           // Sub-system Hamiltonians
  Hzgrad(sys, H0, Hs);                      // Fill up Hamiltonians
  gen_op Up90ys[nss];                       // Pulse propagators
  for(int j=0; j<nss; j++)                  // 90y pulse propagators
    Up90ys[j] = Sypuls_U(sys,Hs[j],I,0,tp,90.0);
  gen_op sigmas[nss];                       // Working density ops.
  evolve(nss, sigma0, Up90ys, sigmas);      // System after 90y pulse
  for(i=0; i<nss; i++)                      // Distance profile after
    data2.put(trace(sigmas[i],D), i);       // the first pulse
  sf /= data2.getRe(data2.max(1));          // Vector maximum value
  vxs[1] = sf*data2;                        // Normalize the vector

  GP_1D("shifts.asc",data1,0,-slen/2,slen/2);// Output in ASCII (gnuplot)
  GP_1Dplot("shifts.gnu", "shifts.asc");    // Interactive plot
  GP_1D("poff.asc", data2, 0, -slen/2, slen/2);// Output in ASCII (gnuplot)
  GP_1Dplot("poff.gnu", "poff.asc");        // Interactive plot
  FM_1Dm("poff.mif",2, vxs);                // Both plot into FM
  cout << "\n\n";                           // Keep the screen nice
  }
```

SFIBenson.cc

```
/* SFIBenson.cc ***************************************************-*-c++-*-
**                                                                      **
**          Test Program for STRAFI Using The GAMMA Library             **
**                                                                      **
**  This program uses the GAMMA class sys_gradz in combination with the **
**  GAMMA module Gradients2 in dealing with STRAFI calculations. This   **
**  paraticular program is meant to reproduce Fig 1. in the paper       **
**                                                                      **
**  "Profile Amplitude Modulation in Stray-Field Magnetic-Resonance     **
**   Imaging", T.B. Benson and P.J. McDonald, J. Magn. Reson. Ser. A,    **
**   112, 17-23, (1995).                                                **
**                                                                      **
**  The figure shows the value of y-magnetization vs. distance in a     **
**  gradient following                                                  **
**                                                                      **
** 1.) The initial 90x in the SRAFI echo sequence                       **
** 2.) At the first STRAFI echo (90x-tau-90y-tau).                      **
** 3.) At the second STRAFI echo (90x-tau-90y-2*tau-90y-tau).           **
**                                                                      **
**  Although the paper used a Bloch treatment, we will herein use a     **
**  density operator approach.  The results should be compatible if the **
**  input spin system is a single spin.                                 **
**                                                                      **
**  The following parameters were used in the Benson & McDonald calc-   **
**  ulation and should be used herein as well:                          **
**                                                                      **
** 1.) Effective sample length: 250 um                                  **
** 2.) Applied pulse length: 10 us                                     **
** 3.) Field gradient strength: 50 T/m                                  **
** 4.) A pulse gap of 50 us                                            **
**                                                                      **
** Author:      S.A. Smith                                              **
** Date:       1/28/97                                                  **
** Update:      1/28/97                                                 **
** Version:    3.6                                                     **
** Copyright:   S. Smith.  You can modify this program as you see fit   **
**              for personal use, but you must leave the program intact **
**              if you re-distribute it.                                **
**                                                                      **
**************************************************************************/

#include <gamma.h>                              // Include GAMMA
#include "Gradients2.cc"

main (int argc, char* argv[])

  {
  //                                            Set Up Working Spin System In a
  z-Gradient
    sys_gradz sys(1);                           // A working spin system
    sys.NSS(3001);                              // Use 3001 sub-systems
    sys.BoGrad(50.0);                           // Use field gradient 50T/m
    sys.SysLen(300.0*1.e-6);                    // Use length of 250 um
    double tau = 40.0*1.e-6;                    // Pulse gap of 40 usec
    double tp = 10.0*1.e-6;                     // Pulse length of 10 usec
  //                                            Set Up Operators Active Over All Spin
  Sub-Systems
    String I = sys.symbol(0);                   // Isotope we deal with
    gen_op sigma0 = sigma_eq(sys);              // Equilibrim
    gen_op H0 = Ho(sys);                        // Base Hamiltonian
    gen_op D = Fm(sys);                         // Detection operator
    H0.set_EBR();
    D.Op_base(H0);
  //                                            Set Up Operators Active On Particular
  Spin Sub-Systems
    int nss = sys.NSS();                        // Number of sub-systems
    gen_op Hs[nss];                             // Sub-system Hamiltonians
    Hzgrad(sys, H0, Hs);                        // Fill up Hamiltonians
    gen_op Us[nss];                             // Propagators
    Props(nss, Hs, tau, Us);                    // Fill up propagators
    gen_op Up90ys[nss];
    for(int j=0; j<nss; j++)                    // 90y pulse propagator
      Up90ys[j] = Sypuls_U(sys,Hs[j],I,0,tp,90.0);
    gen_op Up1s[nss];                           // 1st pulse propagators
    String yn;
    query_parameter(argc, argv, 1,
          "\n\nOffset 1st Pulse Phase [y,n]? ", yn);
    if(yn != "y")
      {
      for(int k=0; k<nss; k++)                  // 90y pulse propagator
        Up1s[k] = Up90ys[k];
      }
    else
      {
      for(int k=0; k<nss; k++)                  // 90y pulse propagator
        Up1s[k] = Sxpuls_U(sys,Hs[k],I,0,tp,90.0);
      }
  //                                            Apply The Pulse Sequence
  //                                            (A Single Spin Echo)
  row_vector data(nss), vxs[3];                 // Blocks for spectra
    gen_op sigmas[nss];                         // Working density ops.
    evolve(nss, sigma0, Up1s, sigmas);          // System after initial pulse
    int i;
    for(i=0; i<nss; i++)                        // Distance profile after
      data.put(trace(sigmas[i],D), i);          // the first pulse
    vxs[0] = data;
    evolve(nss, sigmas, Us, sigmas);            // Evolve for time tau
    evolve(nss, sigmas, Up90ys, sigmas);        // Apply the 90 pulse
    evolve(nss, sigmas, Us, sigmas);            // Evolve for time tau
```

```
  for(i=0; i<nss; i++)                    // Distance profile after
      data.put(trace(sigmas[i],D), i);    // the first pulse
  vxs[1] = data;
  evolve(nss, sigmas, Us, sigmas);        // Evolve for time tau
  evolve(nss, sigmas, Up90ys, sigmas);    // Apply the 90 pulse
  evolve(nss, sigmas, Us, sigmas);        // Evolve for time tau
  for(i=0; i<nss; i++)                    // Distance profile after
      data.put(trace(sigmas[i],D), i);    // the first pulse
  vxs[2] = data;
//                                        Interactive Output Of FID
  double d = sys.SysLen()*1.e6/2.0;
  if(yn == "y")
      {
      FM_1Dm("xy.mif",3,vxs,20,20,-d,d,1);
      for(i=0; i<nss; i++)
        data.put(norm(vxs[0].get(i)), i);
      vxs[0] = data;
      for(i=0; i<nss; i++)
        data.put(norm(vxs[1].get(i)), i);
      vxs[1] = data;
      for(i=0; i<nss; i++)
        data.put(norm(vxs[2].get(i)), i);
      vxs[2] = data;
      FM_1Dm("xyn.mif",3,vxs,20,20,-d,d,0);
      }
  else
          {
          FM_1Dm("xx.mif",3,vxs,20,20,-d,d,1);
          for(i=0; i<nss; i++)
            data.put(norm(vxs[0].get(i)), i);
          vxs[0] = data;
          for(i=0; i<nss; i++)
            data.put(norm(vxs[1].get(i)), i);
          vxs[1] = data;
          for(i=0; i<nss; i++)
            data.put(norm(vxs[2].get(i)), i);
          vxs[2] = data;
          FM_1Dm("xxn.mif",3,vxs,20,20,-d,d,0);
          }
  cout << "\n\n";
  }
```

SFIBain.cc

```
/* SFIBain.cc ***********************************************************************-*-c++-*-
**                                                                                     **
**          Test Program for STRAFI Using The GAMMA Library                            **
**                                                                                     **
**  This program uses GAMMA to generate STRAFI spin echo amplitudes in                 **
**  a single spin 1/2 system. This uses a primitive model of STRAFI                    **
**  it neglects the pulse excitaion envelope, assuming that all spins                  **
**  feel a perfect pulse regardless of their position in the gradient.                 **
**  Still there will be a “spin echo” because there will be some                       **
**  cancellation of net transverse magnetization during the acquistions                **
**  due to the many sub-systems evolving at different frequencies.                     **
**                                                                                     **
** Author:      S.A. Smith                                                           **
** Date:        1/29/97                                                              **
** Update:      1/29/97                                                              **
** Version:     3.6                                                                  **
** Copyright:   S. Smith.  You can modify this program as you see fit                **
**              for personal use, but you must leave the program intact              **
**              if you re-distribute it.                                             **
**                                                                                     **
*************************************************************************************/

#include <gamma.h>                              // Include GAMMA

main (int argc, char* argv[])

    {
    sys_gradz sys;                              // A working spin system
    sys.ask_read(argc, argv, 1);                // Get the spin system
    gen_op sigma0 = sigma_eq(sys);              // This is equlibrium
    gen_op H = Ho(sys);                         // Evolution Hamiltonian
    gen_op D = Fm(sys);                         // Detection operator
    gen_op Deven = -D;                          // Even detection
    gen_op Dodd = complexi*D;                   // Odd detection
    gen_op UP90y = lypuls_U(sys, 90.0);         // Prop for 90y pulse
    gen_op UP90x = lxpuls_U(sys, 90.0);         // Prop for 90x pulse

    int nechos = 15;                            // This many echoes
    double tau = 0.333;                         // Basic tau delay
    int nss = sys.NSS();                        // Number of subsystems
    gen_op Hs[nss], Utaus[nss];                 // Prop for time dealy
    Hzgrad(sys, H, Hs);                         // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);                 // Fill up 2*tau props
    row_vector veven(nechos), vodd(nechos);     // Store echo amplitudes
    double Mx, My;

    gen_op sigma1x = evolve(sigma0, UP90x);     // Apply the 90x pulse
    gen_op sigma1y = evolve(sigma0, UP90y);     // Apply the 90y pulse
    gen_op sigmaxs[nss], sigmays[nss];
    evolve(nss, sigma1x, Utaus, sigmaxs);       // Evolve for tau
    evolve(nss, sigma1y, Utaus, sigmays);       // Evolve for tau
    for(int i=0; i<nechos; i++)
      {
      Mx = 0;                                   // Zero transverse mag.
      My = 0;                                   // Zero transverse mag.
      evolve(nss, sigmaxs, UP90y, sigmaxs);     // Apply the 90y pulse
      evolve(nss, sigmays, UP90y, sigmays);     // Apply the 90y pulse
      evolve(nss, sigmaxs, Utaus, sigmaxs);     // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays);     // Evolve for tau
      for(int j=0; j<nss; j++)                  // Loop over sub-systems
        {
        Mx += zIm(trace(D, sigmaxs[j]));        // Get magnetization
        My -= zRe(trace(D, sigmays[j]));        // Get magnetization
        }
      vodd.put(complex(i,Mx), i);               // Store “echo” amplitude
      veven.put(complex(i,My), i);              // Store “echo” amplitude
      evolve(nss, sigmaxs, Utaus, sigmaxs);     // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays);     // Evolve for tau
      }
  row_vector vxs[2];                            // Array of vectors
    vxs[0] = veven;                             // Store normalized evens
    vxs[1] = vodd;                              // Store scaled odds
    FMxy FM;                                    // Set up for FM output
    FM.hsize=10.0;                              // Plot height 10 cm
    FM.vsize=15.0;                              // Plot width 15 cm
    FM.hmin=1.0;                                // Horizontal left axis value
    FM.hmax=nss;                                // Horizontal right axis value
    FM.vmin = -1;                               // Vertical axis bottom value
    FM.vmax = 2.0;                              // Vertical axis top value
    FM.debug = 0;                               // No debugging
    FM_scatterm(“strafi.mif”, 2, vxs, ‘Q’, FM); // Plot in FrameMaker
    cout << “\n\n”;                             // Keep screen nice
    }
```

Bain.sys

This is the spin system file fed to the GAMMA program in the simulation using the program Bain.cc

```
SysName     (1) : Proton        - Name of the Spin System
NSpins      (0) : 1             - Number of Spins in the System
Iso(0)      (2) : 1H            - Spin Isotope Type
PPM(0)      (1) : 0.0           - Set shift of 1st spin
NSubSys     (0) : 3000          - Number of Sub-Systems
BoGrad      (1) : 50.00         - Gradient Strength (T/m)
SysLen      (1) : 10.0          - Effective Sample Length (um)
Omega       (1) : 300.00        - Spectrometer Freq. in MHz (1H based)
```

SFIPAngVsEAmp1.cc

```
/* SFIPAngVsEAmp1.cc ***************************************************-*-c++-*-
**                                                                       **
**         Test Program for STRAFI Using The GAMMA Library               **
**                                                                       **
**  This program uses GAMMA to generate STRAFI spin echo amplitudes in   **
**  a single spin 1/2 system. This uses a primitive model of STRAFI      **
**  it neglects the pulse excitaion envelope, assuming that all spins    **
**  feel a perfect pulse regardless of their position in the gradient.   **
**  Still there will be a "spin echo" because there will be some         **
**  cancellation of net transverse magnetization during the acquistions  **
**  due to the many sub-systems evolving at different frequencies.       **
**                                                                       **
**  This program differs from its predecessor, PAngVsEAmp0.cc in that    **
** a Gaussian function is assigned to each echo intensity.  This         **
**  gives the appearance of an nice echo envelope and makes for stack    **
**  plots with a much nicer appearances.                                 **
**                                  *                                    **
** Author:      S.A. Smith                                               **
** Date:        2/4/97                                                   **
** Update:      2/4/97                                                   **
** Version:     3.6                                                      **
** Copyright:   S. Smith.  You can modify this program as you see fit    **
**              for personal use, but you must leave the program intact  **
**              if you re-distribute it.                                 **
**                                                                       **
***************************************************************************/

#include <gamma.h>                              // Include GAMMA

  main (int argc, char* argv[])
    {
    sys_gradz sys;                              // A working spin system
    int qn = 1;                                 // Query parameter number
    sys.ask_read(argc, argv, qn++);             // Get the spin system
    gen_op sigma0 = sigma_eq(sys);              // This is equlibrium
    gen_op H = Ho(sys);                         // Evolution Hamiltonian
    gen_op D = Fm(sys);                         // Detection operator
    gen_op Deven = -D;                          // Even detection
    gen_op Dodd = complexi*D;                   // Odd detection

    int nech = 15;                              // This many echoes
    double tau = 0.333;                         // Basic tau delay
    int nss = sys.NSS();                        // Number of subsystems
    gen_op Hs[nss], Utaus[nss];                 // Prop for time dealy
    Hzgrad(sys, H, Hs);                         // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);                 // Fill up 2*tau props
    row_vector veven(nech), vodd(nech);         // Store echo amplitudes

    double Mx, My;

    double pangi, pangf;                        // Initial, final pulse angles
    int nang;                                   // Number of pulse angles
    query_parameter(argc,argv,qn++,             // Get the 1st pulse angle
    "\n\tInitial Pulse Angle (Degrees)? ", pangi);
    query_parameter(argc,argv,qn++,             // Get the last pulse angle
      "\n\tFinal Pulse Angle (Degrees)? ", pangf);
    query_parameter(argc,argv,qn++,             // Get the # of pulse angles
          "\n\tNumber of Pulse Angles? ", nang);

    gen_op UPy, UPx;                            // Props for x & y pulses
    gen_op sigma1x, sigma1y;                    // Dens. op after 1st pulse
    gen_op sigmaxs[nss], sigmays[nss];          // Working density operators
    double pang = pangi;                        // Pulse angle
    double delang = (pangf-pangi)/double(nang-1); // Pulse angle increment
    int npts=100;                               // Points per Gaussian
    int ntot = npts*nech;                       // Total points per angle
    matrix mxeven(nang,ntot), mxodd(nang,ntot); // Angle vs echo amplitude

    double sig = 0.1*double(npts)*42.6/100.0;   // Gaussian width (10%)
    row_vector peaks[nech], peak(ntot);         // For Gaussians
    for(int k=0; k<nech; k++)                   // Fill up Gaussians
      peaks[k] = Gaussian(ntot,k*npts+npts/2,sig);
    row_vector ve,vo,v0(ntot,complex0);

    for(int ang=0; ang<nang; ang++)
      {
      UPy = lypuls_U(sys, pang);                // Prop for y pulse
      UPx = lxpuls_U(sys, pang);                // Prop for x pulse
      sigma1x = evolve(sigma0, UPx);            // Apply the x pulse
      sigma1y = evolve(sigma0, UPy);            // Apply the y pulse
      evolve(nss, sigma1x, Utaus, sigmaxs);     // Evolve for tau
      evolve(nss, sigma1y, Utaus, sigmays);     // Evolve for tau
      ve = v0;                                  // Zero the vector
      vo = v0;                                  // Zero the vector
      for(int i=0; i<nech; i++)
        {
        Mx = 0;                                 // Zero transverse mag.
        My = 0;                                 // Zero transverse mag.
        evolve(nss, sigmaxs, UPy, sigmaxs);     // Apply the y pulse
        evolve(nss, sigmays, UPy, sigmays);     // Apply the y pulse
        evolve(nss, sigmaxs, Utaus, sigmaxs);   // Evolve for tau
        evolve(nss, sigmays, Utaus, sigmays);   // Evolve for tau
        for(int j=0; j<nss; j++)                // Loop over sub-systems
          {
          Mx += zIm(trace(D, sigmaxs[j]));      // Get magnetization
          My -= zRe(trace(D, sigmays[j]));      // Get magnetization
          }
        ve += My*peaks[i];                      // Add Gaussian this echo
        vo += Mx*peaks[i];                      // Add Gaussian this echo
```

```
      if(i != nech-1)
        {
        evolve(nss, sigmaxs, Utaus, sigmaxs);        // Evolve for tau
        evolve(nss, sigmays, Utaus, sigmays);        // Evolve for tau
        }
      }
    mxodd.put_block(ang,0,vo);                       // Store echo intensities
    mxeven.put_block(ang,0,ve);                      // Store echo intensities
    cout << "\n\t\tAngle " << pang << " Done.";      // See where we are
    cout.flush();                                    // Insure output is written
    pang += delang;                                  // Increment pulse angle
    }
String Afile("evenstck.asc");                        // ASCII output filename
String Gfile("evenstck.gnu");                        // Gnuplot load filename
double dn = double(nech);                            // Need this as double
GP_stack(Afile,mxeven,0,pangi,pangf,1.,dn);          / Output stack data
GP_stackplot(Gfile, Afile);                          // Interactive stack plot
Afile = String("oddstck.asc");                       // ASCII output filename
Gfile = String("oddstck.gnu");                       // Gnuplot load filename
GP_stack(Afile,mxodd,0,pangi,pangf,1.,dn);           // Output stack data
GP_stackplot(Gfile, Afile);                          // Interactive stack plot

FMStack FMSTK;                                       // FrameMaker controls
FMSTK.DebugLev(1);                                   // Set for no debugging
FMSTK.VPlotSize(10.0);                               // Vert. plot size 10 cm
FMSTK.HPlotSize(15.0);                               // Horiz. plot size 15 cm
FMSTK.VInc(0.25);                                    // Vert. increment 0.25 cm
FMSTK.HInc(0.25);                                    // Horiz. increment 0.25 cm
String Mfile("evenstck.mif");                        // FrameMaker MIF filename
FM_stack(Mfile, mxeven, FMSTK);                      // Output FM stack plot
Mfile = String("oddstck.mif");                       // FrameMaker MIF filename
FM_stack(Mfile, mxodd, FMSTK);                       // Output FM stack plot
cout << "\n\n";                                      // Keep screen nice
}
```

PAng.sys

This is the spin system file fed to the GAMMA prograin in the simulation using the program Bain.cc

SysName	(1) : Proton	- Name of the Spin System
NSpins	(0) : 1	- Number of Spins in the System
Iso(0)	(2) : 1H	- Spin Isotope Type
PPM(0)	(1) : 0.0	- Set shift of 1st spin
NSubSys	(0) : 3000	- Number of Sub-Systems
BoGrad	(1) : 50.00	- Gradient Strength (T/m)
SysLen	(1) : 5.0	- Effective Sample Length (um)
Omega	(1) : 300.00	- Spec.r Freq. in MHz (1H based)

strafi0.cc

```
/* strafi0.cc ******************************************************************-*-c++-*-
**                                                                               **
**          Test Program for STRAFI Using The GAMMA Library                      **
**                                                                               **
**  This program uses GAMMA to generate STRAFI spin echo amplitudes in           **
**  a single spin 1/2 system. This uses square pulses so as to produce           **
**  a pulse excitaion envelope. Thus, spins in differing sub-systems             **
**  are affected by the applied pulses differently.  This should result          **
**  in nice spin echo generation because there will be cancellation of           **
**  net transverse magnetization during the acquistions due to the many          **
**  sub-systems evolving at different frequencies and intensities.               **
**                                                                               **
**  This program is a modification of the simpler echo amplitude GAMMA           **
**  simulation program, Bain.cc.  This difference is that Bain uses              **
**  ideal pulses whereas this uses a square pulse of user specified              **
**  length.                                                                      **
**                                                                               **
** Author:      S.A. Smith                                                       **
** Date:        2/4/97                                                           **
** Update:      2/4/97                                                           **
** Version:     3.6                                                              **
** Copyright:   S. Smith.  You can modify this program as you see fit            **
**              for personal use, but you must leave the program intact          **
**              if you re-distribute it.                                         **
**                                                                               **
*********************************************************************************/

#include <gamma.h>                          // Include GAMMA

main (int argc, char* argv[])

    {
    sys_gradz sys;                          // A working spin system
    int qn = 1;                             // Query parameter number
    sys.ask_read(argc, argv, qn++);         // Get the spin system
    double tp;                              // Pulse length
    query_parameter(argc, argv, qn++,       // Get the pulse length
            "\n\tPulse Length (usec)? ", tp);
    tp *= 1.e-6;                            // Set pulse length (sec)
    gen_op sigma0 = sigma_eq(sys);          // This is equlibrium
    gen_op H = Ho(sys);                     // Evolution Hamiltonian
    gen_op D = Fm(sys);                     // Detection operator
    gen_op Deven = -D;                      // Even detection
    gen_op Dodd = complexi*D;               // Odd detection
    String I = sys.symbol(0);               // Work on this spin

    int nechos = 15;                        // This many echoes
    double tau = 0.333;                     // Basic tau delay
    int nss = sys.NSS();                    // Number of subsystems

    gen_op Hs[nss], Utaus[nss];             // Prop for time dealy
    Hzgrad(sys, H, Hs);                     // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);             // Fill up 2*tau props
    gen_op Upys[nss], Upxs[nss];            // Pulse evolution props
    for(int j=0; j<nss; j++)                // The y pulse propagators
      {
      Upys[j] = Sypuls_U(sys,Hs[j],I,0,tp,90.0);
      Upxs[j] = Sxpuls_U(sys,Hs[j],I,0,tp,90.0);
      }

    row_vector veven(nechos), vodd(nechos); // Store echo amplitudes
    double Mx, My;                          // Specific echo amplitudes
    gen_op sigmaxs[nss], sigmays[nss];      // Working density operators
    evolve(nss, sigma0, Upxs, sigmaxs);     // Apply the 90x pulse
    evolve(nss, sigma0, Upys, sigmays);     // Apply the 90y pulse
    evolve(nss, sigmaxs, Utaus, sigmaxs);   // Evolve for tau
    evolve(nss, sigmays, Utaus, sigmays);   // Evolve for tau
    for(int i=0; i<nechos; i++)
      {
      Mx = 0;                               // Zero transverse mag.
      My = 0;                               // Zero transverse mag.
      evolve(nss, sigmaxs, Upys, sigmaxs);  // Apply the 90y pulse
      evolve(nss, sigmays, Upys, sigmays);  // Apply the 90y pulse
      evolve(nss, sigmaxs, Utaus, sigmaxs); // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays); // Evolve for tau
      for(int j=0; j<nss; j++)              // Loop over sub-systems
        {
        Mx += zIm(trace(D, sigmaxs[j]));    // Get magnetization
        My -= zRe(trace(D, sigmays[j]));    // Get magnetization
        }
      vodd.put(complex(i,Mx), i);           // Store "echo" amplitude
      veven.put(complex(i,My), i);          // Store "echo" amplitude
      evolve(nss, sigmaxs, Utaus, sigmaxs); // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays); // Evolve for tau
      }

    row_vector vxs[2];                      // Array of vectors
    vxs[0] = veven;                         // Store normalized evens
    vxs[1] = vodd;                          // Store scaled odds
    FMxy FM;                                // Set up for FM output
    FM.hsize=10.0;                          // Plot height 10 cm
    FM.vsize=15.0;                          // Plot width 15 cm
    FM.hmin=1.0;                            // Horizontal left axis value
    FM.hmax=nss;                            // Horizontal right axis value
    FM.vmin = -1;                           // Vertical axis bottom value
    FM.vmax = 2.0;                          // Vertical axis top value
    FM.debug = 0;                           // No debugging
    FM_scatterm("strafi.mif", 2, vxs, 'Q', FM); // Plot in FrameMaker
    cout << "\n\n";                         // Keep screen nice
    }
```

PAngVsEAmp2.cc

```
/* PAngVsEAmp2.cc ***************************************************-*-c++-*-
**                                                                       **
**         Test Program for STRAFI Using The GAMMA Library               **
**                                                                       **
**  This program uses GAMMA to generate STRAFI spin echo amplitudes in   **
**  a single spin 1/2 system over a series of pulse angles.  The output  **
**  is a stack plot of the echoes (horizontal axis) vs. pulse angle      **
**  (vertical axis).  The plots are output interactively using gnuplot   **
**  as well as to FrameMaker MIF files.                                  **
**                                                                       **
**  This program differs from its predecessor, PAngVsEAmp1.cc in that    **
**  the rectancular pulses are used instead of ideal ones.  For          **
**  reasonable values for the gradient vs. pulse length and sample       **
**  length one would expect the results to be virtually equivalent       **
**  to those from PAngVsEAmp1.cc.                                        **
**                                                                       **
** Author:      S.A. Smith                                               **
** Date:       2/15/97                                                   **
** Update:      2/15/97                                                  **
** Version:     3.6                                                      **
** Copyright:   S. Smith.  You can modify this program as you see fit    **
**              for personal use, but you must leave the program intact  **
**              if you re-distribute it.                                 **
**                                                                       **
***************************************************************************/

#include <gamma.h>                          // Include GAMMA

main (int argc, char* argv[])
    {
    sys_gradz sys;                          // A working spin system
    int qn = 1;                             // Query parameter number
    sys.ask_read(argc, argv, qn++);         // Get the spin system
    gen_op sigma0 = sigma_eq(sys);          // This is equlibrium
    gen_op H = Ho(sys);                     // Evolution Hamiltonian
    gen_op D = Fm(sys);                     // Detection operator
    gen_op Deven = -D;                      // Even detection
    gen_op Dodd = complexi*D;               // Odd detection

    int nech = 15;                          // This many echoes
    double tau = 0.333;                     // Basic tau delay
    int nss = sys.NSS();                    // Number of subsystems
    gen_op Hs[nss], Utaus[nss];             // Prop for time dealy
    Hzgrad(sys, H, Hs);                     // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);             // Fill up 2*tau props
    row_vector veven(nech), vodd(nech);     // Store echo amplitudes
    double Mx, My;

    double tp;                              // 90 pulse length
    query_parameter(argc,argv,qn++,         // Get the 90 pulse length
        "\n\t90 Pulse Length (usec)? ", tp);
    tp *= 1.e-6;                            // Set pulse length in sec
    double pangi, pangf;                    // Initial, final pulse angles
    int nang;                               // Number of pulse angles
    query_parameter(argc,argv,qn++,         // Get the 1st pulse angle
    "\n\tInitial Pulse Angle (Degrees)? ", pangi);
    query_parameter(argc,argv,qn++,         // Get the last pulse angle
      "\n\tFinal Pulse Angle (Degrees)? ", pangf);
    query_parameter(argc,argv,qn++,         // Get the # of pulse angles
        "\n\tNumber of Pulse Angles? ", nang);

    gen_op Upys[nss], Upxs[nss];            // Pulse evolution props
    gen_op sigmaxs[nss], sigmays[nss];      // Working density operators
    double pang = pangi;                    // Pulse angle
    double delang = (pangf-pangi)/double(nang-1); // Pulse angle increment
    int npts=100;                           // Points per Gaussian
    int ntot = npts*nech;                   // Total points per angle
    matrix mxeven(nang,ntot), mxodd(nang,ntot); // Angle vs echo amplitude

    double sig = 0.1*double(npts)*42.6/100.0; // Gaussian width (10%)
    row_vector peaks[nech], peak(ntot);     // For Gaussians
    for(int k=0; k<nech; k++)               // Fill up Gaussians
      peaks[k] = Gaussian(ntot,k*npts+npts/2,sig);
    row_vector ve,vo,v0(ntot,complex0);

    int i, j;
    double pl;                              // For adjusted pulse length
    String I = sys.symbol(0);               // Work on this spin
    for(int ang=0; ang<nang; ang++)
      {
      pl = (pang/90.0)*tp;                  // Pulse length angle pang
      for(j=0; j<nss; j++)                  // The pulse propagators
        {                                   // for this pulse angle
        Upys[j] = Sypuls_U(sys,Hs[j],I,0,pl,pang); // (same pulse strength)
        Upxs[j] = Sxpuls_U(sys,Hs[j],I,0,pl,pang);
        }
      evolve(nss, sigma0, Upxs, sigmaxs);   // Apply the x pulse
      evolve(nss, sigma0, Upys, sigmays);   // Apply the y pulse
      evolve(nss, sigmaxs, Utaus, sigmaxs); // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays); // Evolve for tau
      ve = v0;                              // Zero the vector
      vo = v0;                              // Zero the vector
      for(i=0; i<nech; i++)
        {
        Mx = 0;                             // Zero transverse mag.
        My = 0;                             // Zero transverse mag.
        evolve(nss, sigmaxs, Upys, sigmaxs); // Apply the y pulse
        evolve(nss, sigmays, Upys, sigmays); // Apply the y pulse
        evolve(nss, sigmaxs, Utaus, sigmaxs); // Evolve for tau
        evolve(nss, sigmays, Utaus, sigmays); // Evolve for tau
```

```
   for(j=0; j<nss; j++)                    // Loop over sub-systems
     {
     Mx += zIm(trace(D, sigmaxs[j]));      // Get magnetization
     My -= zRe(trace(D, sigmays[j]));      // Get magnetization
     }
   ve += My*peaks[i];                      // Add Gaussian this echo
   vo += Mx*peaks[i];                      // Add Gaussian this echo
   if(i != nech-1)
     {
     evolve(nss, sigmaxs, Utaus, sigmaxs); // Evolve for tau
     evolve(nss, sigmays, Utaus, sigmays); // Evolve for tau
     }
   }
  mxodd.put_block(ang,0,vo);               // Store echo intensities
  mxeven.put_block(ang,0,ve);              // Store echo intensities
  cout << "\n\t\tAngle " << pang << " Done."; // See where we are
  cout.flush();                            // Insure output is written
  pang += delang;                          // Increment pulse angle
  }
String Afile("evenstck.asc");              // ASCII output filename
String Gfile("evenstck.gnu");              // Gnuplot load filename
double dn = double(nech);                  // Need this as double
GP_stack(Afile,mxeven,0,pangi,pangf,1.,dn); // Output stack data
GP_stackplot(Gfile, Afile);                // Interactive stack plot
Afile = String("oddstck.asc");             // ASCII output filename
Gfile = String("oddstck.gnu");             // Gnuplot load filename
GP_stack(Afile,mxodd,0,pangi,pangf,1.,dn); // Output stack data
GP_stackplot(Gfile, Afile);                // Interactive stack plot

FMStack FMSTK;                             // FrameMaker controls
FMSTK.DebugLev(1);                         // Set for no debugging
FMSTK.VPlotSize(10.0);                     // Vert. plot size 10 cm
FMSTK.HPlotSize(15.0);                     // Horiz. plot size 15 cm
FMSTK.VInc(0.25);                          // Vert. increment 0.25 cm
FMSTK.HInc(0.25);                          // Horiz. increment 0.25 cm
String Mfile("evenstck.mif");              // FrameMaker MIF filename
FM_stack(Mfile, mxeven, FMSTK);            // Output FM stack plot
Mfile = String("oddstck.mif");             // FrameMaker MIF filename
FM_stack(Mfile, mxodd, FMSTK);             // Output FM stack plot
cout << "\n\n";                            // Keep screen nice
}
```

GrdFID0.cc

```
/* GrdFID0.cc ***************************************************************-*-c++-*-
**                                                                             **
**            Field Gradient Test Program for the GAMMA Library                **
**                                                                             **
**  This program uses GAMMA to generate an FID following a 90 pulse            **
**  applied to a single spin 1/2 system. The program uses ideal pulses         **
**  thus ingnoring the produce a pulse excitation envelope. Thus, spins        **
**  in differing sub-systems are NOT affected by the applied pulses            **
**  differently.  This should result in generation of a nice decaying          **
**  FID because of good cancellation of net transverse magnetization           **
**  during the acquistions due to the many sub-systems evolving at             **
**  different frequencies and intensities.                                     **
**                                                                             **
**  Note that here we'll just use a 90y pulse so our FID will be in            **
**  phase.  Users may vary the gradient strength and the number of             **
**  sub-systems considered in the computation by altering the input            **
**  spin system file.  In addtion, one may interactively set the FID           **
**  acquisition size and length.                                               **
**                                                                             **
**  In a 50 T/m gradient, you'll get a nice FID over a 1 ms acquisiton.        **
**                                                                             **
** Author:      S.A. Smith                                                     **
** Date:       2/15/97                                                        **
** Update:      2/15/97                                                       **
** Version:     3.6                                                           **
** Copyright:   S. Smith.  You can modify this program as you see fit         **
**              for personal use, but you must leave the program intact        **
**              if you re-distribute it.                                       **
**                                                                             **
*********************************************************************************/

#include <gamma.h>                          // Include GAMMA
#include <Strafi/Gradients2.cc>

main (int argc, char* argv[])

  {
//                  Set Up Working Spin System In a z-Gradient
  sys_gradz sys;                            // A working spin system
  int qn=1;                                 // Input query index
  sys.ask_read(argc, argv, qn++);           // Construction via file
  cout << sys;
//                       Set Up Evolution Hamitonians
  gen_op H = Ho(sys);                       // Evolution Hamiltonian
  int nss = sys.NSS();                      // Number of subsystems
  gen_op Hs[nss];                           // Hamiltonians
  Hzgrad(sys, H, Hs);                       // Fill up Hamiltonians
//                          Set Up For 90 Pulse
  String I = sys.symbol(0);                 // Work on this spin
  gen_op UP90y = lypuls_U(sys,90.0);        // 90y pulse prop
//              Apply The Pulse Sequence
  gen_op sigmas[nss];                       // Working density operators
  gen_op sigma0 = sigma_eq(sys);            // This is equlibrium
  gen_op sigma1 = evolve(sigma0, UP90y);    // Apply the 90y pulse
//                          Ask For FID Specifics
  int t2pts;
  query_parameter(argc, argv, qn++,         // Get the block size
            "\n\tBlock size? ", t2pts);
  double time;
  query_parameter(argc, argv, qn++,         // Get the block size
          "\n\tFID Length (usec)? ", time);
  time *= 1.e-6;                            // Set the time in sec
  double td = time/double(t2pts-1);         // Time between echo pts

  gen_op D = Fm(sys);                       // Detection operator
  row_vector data(t2pts);                   // Stuff for spectrum
  cout << "\n\tSetup Complete, "
       << "Beginning Acquisition\n";
  cout.flush();
  for(int i=0; i<nss; i++)
    {
    acquire1D ACQ(D, Hs[i]);                // Prepare for acquisition
    data += ACQ.T(sigma1,t2pts,td);         // Sum up sub-system FID's
    }
//                       Interactive Output Of FID
  GP_1D("FID.asc", data, 0, 0, time);       // Output FID in ASCII (gnuplot)
  GP_1Dplot("FID.gnu", "FID.asc");          // Interactive 1D gnuplot
  FM_1D("FID.mif", data, 0, 0, time);       // Output FID in MIF (Frame)
  cout << "\n\n";
  }
```

GrdFID1.cc

```
/* FID.cc ***************************************************************************-*-c++-*-
**                                                                                      **
**            STRAFI Test Program for the GAMMA Library                                 **
**                                                                                      **
**  This program uses GAMMA to generate an FID following a 90 pulse                     **
**  applied to a single spin 1/2 system. The program uses a rectangular                 **
**  pulse of a set length to produce a pulse excitation envelope. Thus,                 **
**  spins in differing sub-systems are affected by the applied pulses                   **
**  differently.  This should result in generation of a nice decaying                   **
**  FID because of good cancellation of net transverse magnetization                    **
**  during the acquistions due to the many sub-systems evolving at                      **
**  different frequencies and intensities.                                              **
**                                                                                      **
** Author:      S.A. Smith                                                             **
** Date:        2/15/97                                                                **
** Update:      2/15/97                                                                **
** Version:     3.6                                                                    **
** Copyright:   S. Smith.  You can modify this program as you see fit                  **
**              for personal use, but you must leave the program intact                 **
**              if you re-distribute it.                                                **
**                                                                                      **
******************************************************************************************/

#include <gamma.h>                              // Include GAMMA

main (int argc, char* argv[])

    {
//                  Set Up Working Spin System In a z-Gradient

    sys_gradz sys;                              // A working spin system
    int qn=1;
    sys.ask_read(argc, argv, qn++);             // Construction via file
//                          Set Up Evolution Hamitonians
    gen_op H = Ho(sys);                         // Evolution Hamiltonian
    int nss = sys.NSS();                        // Number of subsystems
    gen_op Hs[nss];                             // Hamiltonians
    Hzgrad(sys, H, Hs);                         // Fill up Hamiltonians
//                              Set Up For 90 Pulse
    String I = sys.symbol(0);                   // Work on this spin
    double tp;                                  // Pulse length
    query_parameter(argc, argv, qn++,           // Get the pulse length
          "\n\t90 Pulse Length (usec)? ", tp);
    tp *= 1.e-6;                                // Set pulse length (sec)
    gen_op UP90ys[nss];                         // Pulse evolution props
    for(int j=0; j<nss; j++)                    // The y pulse propagators
      UP90ys[j]  = Sypuls_U(sys,Hs[j],I,0,tp,90.0);
//                          Apply The Pulse Sequence
    gen_op sigmas[nss];                         // Working density operators
    gen_op sigma0 = sigma_eq(sys);              // This is equlibrium
    evolve(nss, sigma0, UP90ys, sigmas);        // Apply the 90y pulse
   //                                          Ask For FID Specifics
    int t2pts;
    query_parameter(argc, argv, qn++,           // Get the block size
                "\n\tBlock size? ", t2pts);
    double time;
    query_parameter(argc, argv, qn++,           // Get the block size
              "\n\tFID Length (usec)? ", time);
    time *= 1.e-6;                              // Set the time in sec
    double td = time/double(t2pts-1);           // Time between echo pts

    gen_op D = Fm(sys);                         // Detection operator
    row_vector data(t2pts);                     // Stuff for spectrum
    for(int i=0; i<nss; i++)
      {
      acquire1D ACQ(D, Hs[i]);                  // Prepare for acquisition
      data += ACQ.T(sigmas[i],t2pts,td);        // Sum up sub-system FID's
      }
//                          Interactive Output Of FID
    GP_1D("FID.asc", data, 0, 0, time);         // Output FID in ASCII (gnuplot)
    GP_1Dplot("FID.gnu", "FID.asc");            // Interactive 1D gnuplot
    FM_1D("FID.mif", data, 0, 0, time);         // Output FID in MIF (FrameMaker)
    cout << "\n\n";
    }
```

GrdSpinEcho0.cc

```
/* GrdSpinEcho0.cc ***************************************************-*-c++-*-
**                                                                       **
**          STRAFI Test Program for the GAMMA Library                    **
**                                                                       **
**  This program uses GAMMA to generate an echo following a spin echo    **
**  sequence applied to a single spin 1/2 system. The program uses       **
**  ideal pulses, ignoring the pulse excitation envelope. Thus, spins    **
**  in differing sub-systems are not affected by the applied pulses      **
**  differently.  This should result in nice spin echo generation        **
**  because there will be cancellation of net transverse magnetization   **
**  during the acquistions due to the many sub-systems evolving at       **
**  different frequencies and intensities.                               **
**                                                                       **
**                                                                       **
** Author:      S.A. Smith                                              **
** Date:        2/15/97                                                 **
** Update:      2/15/97                                                 **
** Version:     3.6                                                     **
** Copyright:   S. Smith.  You can modify this program as you see fit   **
**              for personal use, but you must leave the program intact  **
**              if you re-distribute it.                                 **
**                                                                       **
***************************************************************************/

#include <gamma.h>                              // Include GAMMA
#include <Strafi/Gradients2.cc>

main (int argc, char* argv[])
    {
//                  Set Up Working Spin System In a z-Gradient

    sys_gradz sys;                              // A working spin system
    int qn=1;                                   // Input query index
    sys.ask_read(argc, argv, qn++);             // Construction via file
    cout << sys << "\n";
//                          Set Up For Tau Time Delay

    double tau;                                 // Delay length
    query_parameter(argc, argv, qn++,           // Tau delay length (usec)
          "\n\tTau Delay Length (usec)? ", tau);
    tau *= 1.e-6;                               // Set tau length (sec)
    gen_op H = Ho(sys);                         // Evolution Hamiltonian
    int nss = sys.NSS();                        // Number of subsystems
    gen_op Hs[nss], Utaus[nss];                 // Prop for time delay
    Hzgrad(sys, H, Hs);                         // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);                 // Fill up tau props
//                      Set Up For 90 and 180 Pulses

    String I = sys.symbol(0);                   // Work on this spin
    gen_op UP90y  = Iypuls_U(sys,90.0);         // 90y pulse prop
    gen_op UP180x = Ixpuls_U(sys,180.0);        // 180x pulse prop
//                          Ask For Echo Specifics
//                  (Defaults To Acquire For 2*tau After 2nd Pulse)

    int t2pts;
    query_parameter(argc, argv, qn++,           // Get the block size
            "\n\tEcho Block Size? ", t2pts);
    double td = (2.0*tau)/double(t2pts-1);      // Time between echo pts (sec)
    double flen = td*double(t2pts-1);           // Total acquisition length(sec)
    if(!td)                                     // If no tau length set, ask
        {                                       // for acquisition length
        query_parameter(argc, argv, qn++,       // (we can't use 2*tau now)
          "\n\tAcquisition Length (usec)? ", flen);
        flen *= 1.e-6;
        td = flen/double(t2pts-1);
        }
//                  A                   pply The Pulse Sequence

    gen_op sigma1, sigmas[nss];                 // Working density operators
    gen_op sigma0 = sigma_eq(sys);              // This is equlibrium
    sigma1 = evolve(sigma0, UP90y);             // Apply the 90y pulse
    evolve(nss, sigma1, Utaus, sigmas);         // Evolve for tau
    evolve(nss, sigmas, UP180x, sigmas);        // Apply the 180x pulse
    gen_op D = Fm(sys);                         // Detection operator
    row_vector data(t2pts, complex0);           // Stuff for spectrum
    cout << "\n\tBeginning Acquisition\n";
    cout.flush();
    for(int i=0; i<nss; i++)
        {
        acquire1D ACQ(D, Hs[i]);                // Prepare for acquisition
        data += ACQ.T(sigmas[i],t2pts,td);      // Sum up sub-system FID's
        }
//                          Interactive Output Of FID

    GP_1D("echo.asc", data, 0, 0, flen*1.e6);   // Output FID in ASCII (gnuplot)
    GP_1Dplot("echo.gnu", "echo.asc");          // Interactive 1D gnuplot
    FM_1D("echo.mif", data, 0, 0, flen*1.e6);   // Output FID in MIF (FrameMaker)
    cout << "\n\n";
    }
```

SpinEcho.cc

```
/* SpinEcho.cc ***************************************************************-*-c++-*-
**                                                                              **
**                    STRAFI Test Program for the GAMMA Library                 **
**                                                                              **
**  This program uses GAMMA to generate an echo following a spin echo           **
**  sequence applied to a single spin 1/2 system. The program uses              **
**  square pulses so as to produce a pulse excitation envelope. Thus,           **
**  spins in differing sub-systems are affected by the applied pulses           **
**  differently.  This should result in nice spin echo generation               **
**  because there will be cancellation of net transverse magnetization          **
**  during the acquistions due to the many sub-systems evolving at              **
**  different frequencies and intensities.                                      **
**                                                                              **
** Author:      S.A. Smith                                                     **
** Date:        2/15/97                                                        **
** Update:      2/15/97                                                        **
** Version:     3.6                                                            **
** Copyright:   S. Smith.  You can modify this program as you see fit          **
**              for personal use, but you must leave the program intact        **
**              if you re-distribute it.                                       **
**                                                                              **
******************************************************************************/

#include <gamma.h>                              // Include GAMMA

main (int argc, char* argv[])
    {
//                      Set Up Working Spin System In a z-Gradient

    sys_gradz sys;                              // A working spin system
    int qn=1;
    sys.ask_read(argc, argv, qn++);             // Construction via file
//                          Set Up For Tau Time Delay

    double tau;                                 // Delay length
    query_parameter(argc, argv, qn++,           // Get the delay length
          "\n\tTau Delay Length (usec)? ", tau);
    tau *= 1.e-6;                               // Set tau length (sec)
    gen_op H = Ho(sys);                         // Evolution Hamiltonian
    int nss = sys.NSS();                        // Number of subsystems
    gen_op Hs[nss], Utaus[nss];                 // Prop for time delay
    Hzgrad(sys, H, Hs);                         // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);                 // Fill up tau props
//                      Set Up For 90 and 180 Pulses

    String I = sys.symbol(0);                   // Work on this spin
    double tp;                                  // Pulse length
    query_parameter(argc, argv, qn++,           // Get the pulse length
          "\n\t90 Pulse Length (usec)? ", tp);
    tp *= 1.e-6;                                // Set pulse length (sec)
    gen_op UP90ys[nss], UP180ys[nss];           // Pulse evolution props
    for(int j=0; j<nss; j++)                    // The y pulse propagators
      {
      UP90ys[j]  = Sypuls_U(sys,Hs[j],I,0,tp,90.0);
      UP180ys[j] = Sxpuls_U(sys,Hs[j],I,0,2*tp,180.0);
      }
//                          Ask For Echo Specifics

    int t2pts;
    query_parameter(argc, argv, qn++,           // Get the block size
          "\n\tEcho Block Size? ", t2pts);
    double time;
    query_parameter(argc, argv, qn++,           // Get the block size
       "\n\tLength About Echo Center (usec)? ", time);
    time *= 1.e-6;                              // Set the time in sec
    double td = time/double(t2pts-1);           // Time between echo pts
    double te = tau - (time/2.0);               // Echo acq. start
    if(te < 0) te = 0;
    gen_op Utes[nss];                           // Prop for time delay
    Props(nss, Hs, te, Utes);                   // Fill up te props
//                          Apply The Pulse Sequence

    gen_op sigma1s[nss], sigmas[nss];           // Working density operators
    gen_op sigma0 = sigma_eq(sys);              // This is equlibrium
    evolve(nss, sigma0, UP90ys, sigma1s);       // Apply the 90y pulse
    evolve(nss, sigma1s, Utaus, sigmas);        // Evolve for tau
    evolve(nss, sigmas, UP180ys, sigmas);       // Apply the 180y pulse
    evolve(nss, sigmas, Utes, sigmas);          // Evolve up to echo start

    gen_op D = Fm(sys);                         // Detection operator
    row_vector data(t2pts);                     // Stuff for spectrum
    for(int i=0; i<nss; i++)
      {
      acquire1D ACQ(D, Hs[i]);                  // Prepare for acquisition
      data += ACQ.T(sigmas[i],t2pts,td);        // Sum up sub-system FID's
      }
//                      Interactive Output Of FID

    GP_1D("echo.asc", data, 0, te, time+te);    // Output FID in ASCII (gnuplot)
    GP_1Dplot("echo.gnu", "echo.asc");          // Interactive 1D gnuplot
    FM_1D("echo.mif", data, 0, te, time+te);    // Output FID in MIF (FrameMaker)
    cout << "\n\n";
    }
```

strafi1.cc

```
/* strafi1.cc ************************************************************-*-c++-*-
**                                                                          **
**          Test Program for STRAFI Using The GAMMA Library                 **
**                                                                          **
**  This program uses GAMMA to generate STRAFI spin echoes in a single      **
**  spin 1/2 system. This accounts for the pulse excitation envelope        **
**  by applying rectangular pulses.  For a specified 90 pulse length,       **
**  the program will produce a series of echoes for bot the even and        **
**  odd STRAFI sequence.                                                    **
**                                                                          **
**  This program differs from its predecessor, strafi0.cc in that           **
**  full oscillatory echoes are produced rather than simply their           **
**  amplitudes at the exact echo time.                                      **
**                                                                          **
** Author:      S.A. Smith                                                 **
** Date:        2/15/97                                                    **
** Update:      2/15/97                                                    **
** Version:     3.6                                                        **
** Copyright:   S. Smith.  You can modify this program as you see fit      **
**              for personal use, but you must leave the program intact    **
**              if you re-distribute it.                                   **
**                                                                          **
******************************************************************************/

#include <gamma.h>                              // Include GAMMA

main (int argc, char* argv[])
    {
//                  Set Up Working Spin System In a z-Gradient
    sys_gradz sys;                              // A working spin system
    int qn = 1;                                 // Query parameter number
    sys.ask_read(argc, argv, qn++);             // Get the spin system
  //              Set Up For Tau Time Delay
    double tau;                                 // Basic tau delay
    query_parameter(argc,argv,qn++,             // Get the tau length
            "\n\tTau Length (usec)? ", tau);
    tau *= 1.e-6;                               // Set tau in sec
    int nss = sys.NSS();                        // Number of subsystems
    gen_op H = Ho(sys);                         // Evolution Hamiltonian
    gen_op Hs[nss], Utaus[nss], U2taus[nss];    // Prop for time delay
    Hzgrad(sys, H, Hs);                         // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);                 // Fill up tau props
    Props(nss, Hs, 2.0*tau, U2taus);            // Fill up 2*tau props
//                      Set Up For 90 Pulses
    String I = sys.symbol(0);                   // Work on this spin
    double tp;                                  // 90 pulse length
    query_parameter(argc,argv,qn++,             // Get the 90 pulse length
        "\n\t90 Pulse Length (usec)? ", tp);
    tp *= 1.e-6;                                // Set pulse length in sec
    gen_op Upys[nss], Upxs[nss];                // Pulse evolution props
    int i;
    for(i=0; i<nss; i++)                        // The pulse propagators
      {                                         // for 90x and 90y pulses
      Upys[i] = Sypuls_U(sys,Hs[i],I,0,tp,90.);
      Upxs[i] = Sxpuls_U(sys,Hs[i],I,0,tp,90.);
      }
//                      Ask For Echo Specifics
    int nech;                                   // This many echoes
    query_parameter(argc, argv, qn++,           // Get the # of echoes
            "\n\tNumber Of Echoes? ", nech);
    int npts;                                   // Echo block size
    query_parameter(argc, argv, qn++,           // Get the block size
            "\n\tEcho Block Size? ", npts);
    double time;
    query_parameter(argc, argv, qn++,           // Get the block size
      "\n\tLength About Echo Center (usec)? ", time);
    time *= 1.e-6;                              // Set the time in sec
    double td = time/double(npts-1);            // Time between echo pts
    double te = tau - (time/2.0);               // Echo acq. start
    if(te < 0) te = 0;
    gen_op Utes[nss];                           // Prop for time delay
    Props(nss, Hs, te, Utes);                   // Fill up te props
//                      Set Parameters For STRAFI
    gen_op D = Fm(sys);                         // Detection operator
    gen_op Dy = -D;                             // Even detection
    gen_op Dx = complexi*D;                     // Odd detection
    int ntot = npts*nech;                       // Total points acquired
    row_vector veven(ntot, 0), vodd(ntot,0);    // Blocks to store echoes
    row_vector echox, echoy, vx0(npts, complex0);
//                      Apply The STRAFI Pulse Sequence
    gen_op sigma0 = sigma_eq(sys);              // This is equlibrium
    gen_op sigmaxs[nss], sigmays[nss];          // Working density ops
    gen_op sigmaxas[nss], sigmayas[nss];        // Working density ops
    evolve(nss, sigma0, Upxs, sigmaxs);         // Apply the 90x pulse
    evolve(nss, sigma0, Upys, sigmays);         // Apply the 90y pulse
    evolve(nss, sigmaxs, Utaus, sigmaxs);       // Evolve for tau
    evolve(nss, sigmays, Utaus, sigmays);       // Evolve for tau
    int j, k=0;                                 // Working indices
    for(i=0; i<nech; i++)
      {
      evolve(nss, sigmaxs, Upys, sigmaxs);      // Apply the y pulse
      evolve(nss, sigmays, Upys, sigmays);      // Apply the y pulse
      evolve(nss, sigmaxs, Utes, sigmaxas);     // Evolve to echo start
      evolve(nss, sigmays, Utes, sigmayas);     // Evolve to echo start
      echox = vx0;                              // Zero the echo
```

```
  echoy = vx0;                                  // Zero the echo
  for(j=0; j<nss; j++)                          // Loop subsystem
   {
   acquire1D ACQx(Dx, Hs[j]);                   // Prepare for acquisition
   echox += ACQx.T(sigmaxas[j],npts,td);        // Sum up sub-system FID's
   acquire1D ACQy(Dy, Hs[j]);
   echoy += ACQy.T(sigmayas[j],npts,td);
   }
  veven.put_block(0,k,echoy);                   // Store the block
  vodd.put_block(0,k,echox);                    // Store the block
  cout << "\n\tEcho " << i+1 << " Complete";
  cout.flush();
  evolve(nss, sigmaxs, U2taus, sigmaxs);        // Evolve to end of 2*tau
  evolve(nss, sigmays, U2taus, sigmays);        // Evolve to end of 2*tau
  k+=npts;
  }
GP_1D("echox.asc", vodd);
GP_1Dplot("echox.gnu", "echox.asc");
FM_1D("echox.mif", vodd);
GP_1D("echoy.asc", veven);
GP_1Dplot("echoy.gnu", "echoy.asc");
FM_1D("echoy.mif", veven);
cout << "\n\n";                                // Keep screen nice
}
```

EAmpVsGB1.cc

```
/* EAmpVsGB1.cc ***********************************************************-*-c++-*-
**                                                                           **
**          Test Program for STRAFI Using The GAMMA Library                  **
**                                                                           **
**  This program uses GAMMA to generate STRAFI spin echo amplitudes in a     **
** single spin 1/2 system. This uses square pulses.                          **
**                                                                           **
**  Herein a loop is made over differing RF field strengths while keeping the **
** effective pulse length constant.  A plot is then made of echo amplitude vs. **
** pulse angle (field strength).  This is to verify astatement by Randall & Bain **
** that, for a given pulse length, echoes maximize for 120 degree pulses.    **
**                                                                           **
** Author:      S.A. Smith                                                   **
** Date:        2/18/97                                                      **
** Version:     3.6                                                          **
** Copyright:   S. Smith.  You can modify this as you see fit for private use, **
**                  but you must leave the program intact if you re-distribute it. **
**                                                                           **
*******************************************************************************/

#include <gamma.h>                          // Include GAMMA
main (int argc, char* argv[])

    {
    sys_gradz sys;                          // A working spin system
    int qn = 1;                             // Query parameter number
    sys.ask_read(argc, argv, qn++);         // Get the spin system
    double tp;                              // Pulse length
    query_parameter(argc, argv, qn++,       // Get the pulse length
                    "\n\tPulse Length (usec)? ", tp);
    tp *= 1.e-6;                            // Set pulse length (sec)
    int echo;                               // Echo number
    query_parameter(argc, argv, qn++,       // Get the echo
            "\n\tEcho to Amplitude Track? ", echo);
    int nang;                               // Number of Angles
    query_parameter(argc, argv, qn++,       // Get the # of angles
          "\n\tNumber of Angles To Sample? ", nang);
    gen_op sigma0 = sigma_eq(sys);          // This is equlibrium
    gen_op H = Ho(sys);                     // Evolution Hamiltonian
    gen_op Deven = -Fm(sys);                // Even detection
    gen_op Dodd = -complexi*D;              // Odd detection
    double tau = 3.e-3;                     // Basic tau delay (3 ms)
    int nss = sys.NSS();                    // Number of subsystems
    gen_op Hs[nss], Utaus[nss];             // Prop for time dealy
    Hzgrad(sys, H, Hs);                     // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);             // Fill up 2*tau props
    gen_op Upys[nss], Upxs[nss];            // Pulse evolution props
    int i, j;
    row_vector veven(nang), vodd(nang);     // Store echo amplitudes
    double Mx, My;                          // Specific echo amplitudes
    gen_op sigmaxs[nss], sigmays[nss];      // Working density operators
    evolve(nss, sigma0, Upxs, sigmaxs);     // Apply the 90x pulse
    evolve(nss, sigma0, Upys, sigmays);     // Apply the 90y pulse
    evolve(nss, sigmaxs, Utaus, sigmaxs);   // Evolve for tau
    evolve(nss, sigmays, Utaus, sigmays);   // Evolve for tau
    double angi = 30.0, angf=150.0;         // Start at 30 deg, end at 150.
    double anginc = (angf-angi)/double(nang-1);  // Angle increment
    double ang = angi;
    for(i=0; i<nang; i++)                   // Loop the angles
      {
      for(j=0; j<nss; j++)                  // Set pulse props
        {
        Upys[j] = Sypuls_U(sys,Hs[j],"1H",0,tp,ang);
        Upxs[j] = Sxpuls_U(sys,Hs[j],"1H",0,tp,ang);
        }
      evolve(nss, sigma0, Upxs, sigmaxs);   // Apply the x pulse
      evolve(nss, sigma0, Upys, sigmays);   // Apply the y pulse
      evolve(nss, sigmaxs, Utaus, sigmaxs); // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays); // Evolve for tau
      for(j=0; j<echo; j++)                 // Set pulse props
        {
        evolve(nss, sigma0, Upxs, sigmaxs); // Apply the x pulse
        evolve(nss, sigma0, Upys, sigmays); // Apply the y pulse
        evolve(nss, sigmaxs, Utaus, sigmaxs); // Evolve for tau
        evolve(nss, sigmays, Utaus, sigmays); // Evolve for tau
        if(j<echo-1)
          {
          evolve(nss, sigmaxs, Utaus, sigmaxs); // Evolve for tau
          evolve(nss, sigmays, Utaus, sigmays); // Evolve for tau
          }
        }
      Mx = 0;                               // Zero transverse mag.
      My = 0;                               // Zero transverse mag.
      for(j=0; j<nss; j++)                  // Loop over sub-systems
        {
        Mx += zIm(trace(D, sigmaxs[j]));    // Get magnetization
        My -= zRe(trace(D, sigmays[j]));    // Get magnetization
        }
      vodd.put(complex(i,Mx), i);           // Store "echo" amplitude
      veven.put(complex(i,My), i);          // Store "echo" amplitude
      ang += anginc;                        // Increment pulse angle
      }
    String asc = "IvsGamB1.asc";
    String gnu = "IvsGamB1.gnu";
    GP_xy(asc, vodd);
    GP_xyplot(gnu,asc);
    FM_xyPlot("IvsGamB1o.mif",vodd);        // Plot xyyy... in FrameMaker
    FM_xyPlot("IvsGamB1e.mif",veven);       // Plot yyyy... in FrameMaker
    cout << "\n\n";                         // Keep screen nice
    }
```

EAmpVsPL.cc

```
/* EAmpVsPL.cc ************************************************************-*-c++-*-
**                                                                           **
**                 Test Program for STRAFI Using The GAMMA Library           **
**                                                                           **
**  This program uses GAMMA to generate STRAFI spin echo amplitudes in       **
**  a single spin 1/2 system. This uses square pulses so as to produce       **
**  a pulse excitaion envelope. Thus, spins in differing sub-systems         **
**  are affected by the applied pulses differently.  This should result      **
**  in nice spin echo generation because there will be cancellation of       **
**  net transverse magnetization during the acquistions
due to the many                                                             **
**  sub-systems evolving at different frequencies and intensities.           **
**                                                **
**  Herein a loop is made over differing pulse lengths while keeping         **
**  the effective pulse strength constant.  A plot is then made of echo      **
**  amplitude vs. pulse angle (pulse length).  This is to verify the         **
**  statement by Randall and Bain that, for a given pulse strength,          **
**  echoes will have a maximum amplitude for 120 degree pulses.              **
**                                                                           **
** Author:      S.A. Smith                                                  **
** Date:        2/18/97                                                     **
** Update:      2/18/97                                                     **
** Version:     3.6                                  **
** Copyright:   S. Smith.  You can modify this program as you see fit   **
**              for personal use, but you must leave the program intact **
**              if you re-distribute it.                    **
**                                                **
*****************************************************************************/

#include <gamma.h>                              // Include GAMMA
  main (int argc, char* argv[])

    {
    sys_gradz sys;                              // A working spin system
    int qn = 1;                                 // Query parameter number
    sys.ask_read(argc, argv, qn++);             // Get the spin system
    double gamB1;                               // Pulse strength
    query_parameter(argc, argv, qn++,           // Get the pulse strength
          "\n\tField Strength (kHz)? ", gamB1);
    gamB1 *= 1.e3;                              // Set pulse strength (Hz)
    int echo;                                   // Echo number
    query_parameter(argc, argv, qn++,           // Get the echo
        "\n\tEcho to Amplitude Track? ", echo);
    int nang;                                   // Number of Angles
    query_parameter(argc, argv, qn++,           // Get the # of angles
       "\n\tNumber of Angles To Sample? ", nang);
    gen_op sigma0 = sigma_eq(sys);              // This is equlibrium
    gen_op H = Ho(sys);                         // Evolution Hamiltonian
    gen_op D = Fm(sys);                         // Detection operator
    gen_op Deven = -D;                          // Even detection
    gen_op Dodd = complexi*D;                   // Odd detection
    String I = sys.symbol(0);                   // Work on this spin

    double tau = 3.e-3;                         // Basic tau delay (3 ms)
    int nss = sys.NSS();                        // Number of subsystems
    gen_op Hs[nss], Utaus[nss];                 // Prop for time dealy
    Hzgrad(sys, H, Hs);                                 // Fill up Hamiltonians
    Props(nss, Hs, tau, Utaus);                 // Fill up 2*tau props
    gen_op Upys[nss], Upxs[nss];            // Pulse evolution props
    int i, j;

    row_vector veven(nang), vodd(nang);         // Store echo amplitudes
    double Mx, My;                              // Specific echo amplitudes
    gen_op sigmaxs[nss], sigmays[nss];          // Working density operators
    evolve(nss, sigma0, Upxs, sigmaxs);         // Apply the 90x pulse
    evolve(nss, sigma0, Upys, sigmays);         // Apply the 90y pulse
    evolve(nss, sigmaxs, Utaus, sigmaxs);       // Evolve for tau
    evolve(nss, sigmays, Utaus, sigmays);       // Evolve for tau
    double angi = 30;                           // Start at 30 deg.
    double angf = 150;                          // End at 150 deg.
    double anginc = (angf-angi)/double(nang-1);     // Angle increment
    double ang = angi;                          // Initial angle
    double tp = (ang/360.0)/gamB1;              // Initial pulse length

    for(i=0; i<nang; i++)                       // Loop the angles
     {
     for(j=0; j<nss; j++)                       // Set pulse props
      {
      Upys[j] = Sypuls_U(sys,Hs[j],I,0,tp,ang);
      Upxs[j] = Sxpuls_U(sys,Hs[j],I,0,tp,ang);
      }
     evolve(nss, sigma0, Upxs, sigmaxs);        // Apply the x pulse
     evolve(nss, sigma0, Upys, sigmays);        // Apply the y pulse
     evolve(nss, sigmaxs, Utaus, sigmaxs);      // Evolve for tau
     evolve(nss, sigmays, Utaus, sigmays);      // Evolve for tau
     for(j=0; j<echo; j++)                      // Set pulse props
      {
      evolve(nss, sigma0, Upxs, sigmaxs);       // Apply the x pulse
      evolve(nss, sigma0, Upys, sigmays);       // Apply the y pulse
      evolve(nss, sigmaxs, Utaus, sigmaxs);     // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays);     // Evolve for tau
      if(j<echo-1)
       {
       evolve(nss, sigmaxs, Utaus, sigmaxs);    // Evolve for tau
       evolve(nss, sigmays, Utaus, sigmays);    // Evolve for tau
       }
      }
     Mx = 0;                                    // Zero transverse mag.
     My = 0;                                    // Zero transverse mag.
     for(j=0; j<nss; j++)                       // Loop over sub-systems
```

```
    {
    Mx += zIm(trace(D, sigmaxs[j]));          // Get magnetization
    My -= zRe(trace(D, sigmays[j]));          // Get magnetization
    }
  vodd.put(complex(i,Mx), i);                 // Store “echo” amplitude
  veven.put(complex(i,My), i);                // Store “echo” amplitude
  cout << “\n\tAngle “ << ang
       << “ Complete, Pulse Length “
       << tp*1.e6 << “ usec.”;
  cout.flush();
  ang += anginc;                              // Increment pulse angle
  tp = (ang/360.0)/gamB1;                     // Increment pulse length
  }

 String asc = “IvsGamB1.asc”;
 String gnu = “IvsGamB1.gnu”;
 GP_xy(asc, vodd);
 GP_xyplot(gnu,asc);
 row_vector vxs[2];                           // Array of vectors
 vxs[0] = veven;                             // Store normalized evens
 vxs[1] = vodd;                              // Store scaled odds
 FMxy FM;                                    // Set up for FM output
 FM.hsize=10.0;                              // Plot height 10 cm
 FM.vsize=15.0;                              // Plot width 15 cm
 FM.hmin=1.0;                                // Horizontal left axis value
 FM.hmax=nss;                                // Horizontal right axis value
 FM.vmin = -1;                               // Vertical axis bottom value
 FM.vmax = 2.0;                              // Vertical axis top value
 FM.debug = 0;                               // No debugging
 FM_xyPlot(“IvsGamB1o.mif”,vodd);            // Plot xyyy... in FrameMaker
 FM_xyPlot(“IvsGamB1e.mif”,veven);           // Plot yyyy... in FrameMaker
//  FM_scatterm(“IvsGamB1s.mif”,2,vxs,’Q’,FM); // Plot in FrameMaker
 cout << “\n\n”;                             // Keep screen nice
 }
```

EAmpVsPul1.cc

```
/* EAmpVsPul1.cc ***********************************************-*-c++-*-
**                                                                      **
**          Test Program for STRAFI Using The GAMMA Library             **
**                                                                      **
**  This program uses GAMMA to generate STRAFI spin echo amplitudes in  **
**  a single spin 1/2 system. This uses square pulses so as to produce  **
**  a pulse excitaion envelope. Thus, spins in differing sub-systems    **
**  are affected by the applied pulses differently.  This should result **
**  in nice spin echo generation because there will be cancellation of  **
**  net transverse magnetization during the acquistions due to the many **
**  sub-systems evolving at different frequencies and intensities.      **
**                                                                      **
**  Herein a loop is made over differing RF field strengths & lengths.  **
**  A plot is then madeof a choosen echo's amplitude over the user      **
**  specified pulse lengths and strengths.  The purpose of this is to   **
**  look into the validity of statements in the literature which state  **
**  that 1.) For a given pulse length, echoes will have a maximum       **
**  amplitude for 120 degree pulses, and 2.) For a given pulse strength **
**  echoes will have a maximum amplitude for 90 degree pulses.          **
**                                                                      **
** Author:      S.A. Smith                                             **
** Date:        2/18/97                                                **
** Update:      2/18/97                                                **
** Version:     3.6                                                    **
** Copyright:   S. Smith.  You can modify this program as you see fit  **
**              for personal use, but you must leave the program intact **
**              if you re-distribute it.                                **
**                                                                      **
**************************************************************************/

#include <gamma.h>                                  // Include GAMMA
#include "Gradients2.cc"

  void SPulProps(const sys_gradz& sys, gen_op H[], const String& iso,
                 double tp, double fact, gen_op* Us, char axis)

      // Input          sys   : spin system
      //                H     : active static Hamiltonian
      //                iso   : isotope type
      //                freq  : pulse frequency (offset)
      //                tp    : soft pulse length
      //                fact  : either gamma*B1 or pulse angle scaled
      //                axis  : axis along which the pulse is applied
      // Output         U     : pulse propagator for the application
      //                        of a soft pulse.

  {
  int i, nss = sys.NSS();                           // Number of subsystems
  if(tp == 0.0)
    {
    int hs = H[0].size();                           // Hilbert space
    matrix mx(hs,hs,i_matrix_type);
    gen_op U(mx);                                   // An zero pulse if tp=0
    for(i=0; i<nss; i++) Us[i] = U;                 // so all props are I
    return;
    }
  gen_op H1, Heff;
  switch(axis)                                      // RF-field rot.frame Ham.
    {
    case 'x':
    default:
      H1 = fact*Fx(sys,iso);                        // Field along the x-axis
      break;
    case 'y':
      H1 = fact*Fy(sys,iso);                        // Field along the y-axis
      break;
    }
  for(i=0; i<nss; i++)
    {
    Heff = H[i] + H1;
    Us[i] = prop(Heff, tp);
    }
  return;
  }


void SyPulProps(const sys_gradz& sys, gen_op* H, const String& iso,
                double tp, double theta, gen_op* Us)

      // Input          sys   : Spin system in z-Gradient
      //                H     : Acting static Hamiltonians
      //                iso   : Isotope type
      //                tp    : Soft pulse length
      //                theta : Pulse angle (for a spin at resonance)
      // Output         U     : pulse propagator soft pulse

  {
  if(tp == 0.0)
    {
    int nss = sys.NSS();                            // Number of subsystems
    gen_op U = Ry(sys, iso, theta);                 // An ideal pulse if tp=0
    for(int i=0; i<nss; i++) Us[i] = U;             // so all props the same
    return;
    }
  double fact = theta/(360.*tp);
  SPulProps(sys, H, iso, tp, fact, Us, 'y');
  }


void SxPulProps(const sys_gradz& sys, gen_op* H, const String& iso,
```

```
                    double tp, double theta, gen_op* Us)

    // Input          sys   : Spin system in z-Gradient
    //               H     : Acting static Hamiltonians
    //               iso   : Isotope type
    //               tp    : Soft pulse length
    //               theta : Pulse angle (for a spin at resonance)
    // Output          U     : pulse propagator soft pulse

{
if(tp == 0.0)
  {
  int nss = sys.NSS();                          // Number of subsystems
  gen_op U = Rx(sys, iso, theta);               // An ideal pulse if tp=0
  for(int i=0; i<nss; i++) Us[i] = U;           // so all props the same
  return;
  }
double fact = theta/(360.*tp);
SPulProps(sys, H, iso, tp, fact, Us, 'x');
}


main (int argc, char* argv[])

{
sys_gradz sys;                                  // A working spin system
int qn = 1;                                     // Query parameter number
sys.ask_read(argc, argv, qn++);                 // Get the spin system
double tpi;                                     // Initital pulse length
query_parameter(argc, argv, qn++,               // Get the 1st pulse length
   "\n\tInitial Pulse Length (usec)? ", tpi);
tpi *= 1.e-6;                                   // Set pulse length (sec)
double tpf;                                     // Final pulse length
query_parameter(argc, argv, qn++,               // Get the last pulse length
   "\n\tFinal Pulse Length (usec)? ", tpf);
tpf *= 1.e-6;                                   // Set pulse length (sec)
int ntp;
query_parameter(argc, argv, qn++,               // Get the # of pulse lengths
          "\n\tNumber of Lengths? ", ntp);
double gB1i;                                    // Initital pulse strength
query_parameter(argc, argv, qn++,               // Get the 1st pulse strength
  "\n\tInitial Pulse Strength (kHz)? ", gB1i);
gB1i *= 1.e3;                                   // Set pulse strength (Hz)
double gB1f;                                    // Final pulse strength
query_parameter(argc, argv, qn++,               // Get the last pulse strength
   "\n\tFinal Pulse Strength (kHz)? ", gB1f);
gB1f *= 1.e3;                                   // Set pulse strength (Hz)
int ngB1;
query_parameter(argc, argv, qn++,               // Get the # of pulse strengths
        "\n\tNumber of Strengths? ", ngB1);
int echo;                                       // Echo number
query_parameter(argc, argv, qn++,               // Get the echo
     "\n\tEcho to Amplitude Track? ", echo);
double tau;                                     // Delay time
query_parameter(argc, argv, qn++,               // Get the tau value
 "\n\tTau Delay in Echo Sequence(msec)? ", tau);
tau *= 1.e-3;                                   // Set tau in seconds

gen_op sigma0 = sigma_eq(sys);                  // This is equlibrium
gen_op H = Ho(sys);                             // Evolution Hamiltonian
gen_op D = Fm(sys);                             // Detection operator
gen_op Deven = -D;                              // Even detection
gen_op Dodd = complexi*D;                       // Odd detection
String I = sys.symbol(0);                       // Work on this spin

int nss = sys.NSS();                            // Number of subsystems
gen_op Hs[nss], Utaus[nss];                     // Prop for time dealy
Hzgrad(sys, H, Hs);                             // Fill up Hamiltonians
Props(nss, Hs, tau, Utaus);                     // Fill up 2*tau props
gen_op Upys[nss], Upxs[nss];                     // Pulse evolution props

matrix mxeven(ntp, ngB1,complex0);              // Array for even amplitudes
matrix mxodd(ntp, ngB1,complex0);               // Array for odd amplitudes
double tpinc = (tpf-tpi)/double(ntp);           // Pulse length increment
double gB1inc = (gB1f-gB1i)/double(ngB1);       // Pulse strength increment
double angi = 30;                               // Mimimum angle allowed
double angf = 160;                              // Maximum angle allowed
double tp = tpi;                                // Initialize pulse length
double gamB1 = gB1i;                            // Initialize pulse strength
double ang;                                     // For pulse angle (degrees)
double Mx, My;                                  // Specific echo amplitudes
gen_op sigmaxs[nss], sigmays[nss];              // Working density operators
int i, j, k;
for(i=0; i<ntp; i++)                            // Loop the pulse lengths
  {
  cout << "\n\n\tCalculating Amplitudes for "
       << tp*1.e6 << " usec Pulse.\n\t\t";
  cout.flush();
  for(j=0; j<ngB1; j++)                         // Loop pulse strengths
    {
    ang = tp*gamB1*360.0;                       // Pulse angle (degrees)
    if(ang>=angi && ang<=angf)                  // Don't calculate for big
      {                                         // or small pulse angles
      SyPulProps(sys, Hs, I, tp, ang, Upys);    // Fill up 2*tau props
      SxPulProps(sys, Hs, I, tp, ang, Upxs);    // Fill up 2*tau props
      evolve(nss, sigma0, Upxs, sigmaxs);       // Apply the x pulse
      evolve(nss, sigma0, Upys, sigmays);       // Apply the y pulse
      evolve(nss, sigmaxs, Utaus, sigmaxs);     // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays);     // Evolve for tau
      for(k=0; k<echo; k++)                     // Set pulse props
        {
        evolve(nss, sigmaxs, Upys, sigmaxs);    // Apply the y pulse
```

```
      evolve(nss, sigmaxs, Upys, sigmays);        // Apply the y pulse
      evolve(nss, sigmaxs, Utaus, sigmaxs);       // Evolve for tau
      evolve(nss, sigmays, Utaus, sigmays);       // Evolve for tau
      if(k<echo-1)
       {
       evolve(nss,sigmaxs,Utaus,sigmaxs);         // Evolve for tau
       evolve(nss,sigmays,Utaus,sigmays);         // Evolve for tau
       }
      }
     Mx = 0;                                      // Zero transverse mag.
     My = 0;                                      // Zero transverse mag.
     for(k=0; k<nss; k++)                         // Loop over sub-systems
      {
      Mx += zIm(trace(D, sigmaxs[k]));            // Get magnetization
      My -= zRe(trace(D, sigmays[k]));            // Get magnetization
      }
     mxodd.put(complex(Mx), i, j);                // Store “echo” amplitude
     mxeven.put(complex(My), i, j);               // Store “echo” amplitude
     }
    cout << ang << “ “;
    cout.flush();
    gamB1 += gB1inc;                              // Increment pulse strength
    }
   gamB1 = gB1i;                                  // Reset pulse strength
   tp += tpinc;                                   // Increment pulse length
   }

String asce = “IvsPulEven.asc”;
String asco = “IvsPulOdd.asc”;
String gnu = “IvsPul.gnu”;
GP_contour(asco, mxodd);
GP_contplot(gnu,asco);
GP_contour(asce, mxeven);
GP_contplot(gnu,asce);
cout << “\n\n”;                                   // Keep screen nice
}
```