High Resolution Fast MRI and MRSI using Spiral Data Acquisition

by

He Zhu

B.S. Physics, Peking University, 1998

M.S. Physics, University of Pittsburgh, 2002

Submitted to the Graduate Faculty of

School of Art and Science in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2007

UNIVERSITY OF PITTSBURGH

School of Art and Science

This thesis was presented

by

He Zhu

It was defended on

November 20th, 2006

and approved by

Dr. Qiuhong He, Dept. of Radiology and Dept. of Bioengineering

Dr. Yi Wang, Dept. of Radiology and Dept. of Bioengineering

Dr. Ralph Roskies, Dept. of Physics and Astronomy

Dr. David Snoke, Dept. of Physics and Astronomy

Dr. James Mueller, Dept. of Physics and Astronomy

Dr. Irving Lowe, Dept. of Physics and Astronomy

High Resolution Fast MRI and MRSI using Spiral Data Acquisition

He Zhu, PhD

University of Pittsburgh, 2007

In this thesis, spiral k-space sampling was configured differently for two applications in Magnetic Resonance Spectroscopic Imaging (MRSI) and time resolved 3D Magnetic Resonance Imaging (MRI). Selective Multiple Quantum Coherence Transfer (Sel-MQC) technique was implemented in *in vivo* MRSI in combination with Spiral data acquisition. The Spiral Sel-MQC technique enabled fast mapping of Polyunsaturated Fatty Acids (PUFA) in human breast *in vivo* compared to Chemical Shift Imaging (CSI). The Sel-MQC technique utilizes a scalar coupling to excite PUFA signal while suppressing other resonances in lipid and water. An *in vivo* 2D Sel-MQC sequence was first implemented to optimize the performance of the Sel-MQC excitation and to investigate the compositions of PUFA as well as Monounsaturated Fatty Acids (MUFA). Spiral data acquisition was then implemented to image PUFA signal exclusively. An image can be acquired in 1-2 min with 16 or 32 scans. Time resolved 3D MRI was also developed with high spatial and temporal resolutions with spiral data acquisition. Off-resonance correction was performed using inhomogeneity field maps. View sharing and sliding window reconstruction were utilized to generate high temporal resolution. High resolution 3D angiograms were generated at 1-2 seconds per frame *in vivo*. A quantitative method was developed to evaluate the performance of spiral parameters in these applications. This method approximates spiral imaging as a process of linear estimation. The optimal spiral parameters can be determined by finding the least estimation error. Two *in vivo* applications of spiral data acquisition discussed in this thesis shows that spiral data acquisition can shorten scan times by a factor of up to 10. It can, therefore, enable clinics to include advanced MR techniques such as Sel-MQC and time resolved 3D MRI to enhance diagnosis of cancer or vascular diseases.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This thesis describes two original experiments in both Magnetic Resonance Spectroscopy (MRS) and Magnetic Resonance Imaging (MRI). Fast spiral data acquisition is applied to both MRS and MRI to significantly reduce the scan time so that advance MR techniques can be included in MR exams in clinics. Spiral data acquisition is combined with Sel-MQC spin editing technique for the first time in MRS. The resulting Spiral Sel-MQC sequence enables fast mapping of Polyunsaturated Fatty Acids (PUFA) as a possible index for tumour development. Spiral data acquisition is also configured for time resolved 3D acquisition in MRI for the first time. The resulting 3D stack-of-spiral sequence enables dynamic 3D visualization of blood vessels with high spatial and temporal resolutions.

Chapter 1 overviews the physical principles of Magnetic Resonance Spectroscopy and Magnetic Resonance Imaging with the classical theory and the quantum mechanical theory. Chapters 2 discusses Magnetic Resonance Spectroscopy in two spectral dimensions including two variations of the Correlated Spectroscopy (COSY) experiment and the Selective Multiple Quantum Coherence (Sel-MQC) Transfer experiment. Chapter 2 is concluded with an optimized version of the Sel-MQC experiment for *in vivo* applications that enables its application in Magnetic Resonance Spectroscopic Imaging (MRSI), which is the subject of Chapter 3. The spiral Sel-MQC experiment discussed in Chapter 3 is a combination of the spectroscopic editing technique (Sel-MQC) and spiral data acquisition for fast MRSI. Chapter 4 discusses a different application of spiral data acquisition in Time Resolved 3D Angiography. Chapter 5 includes a quantitative analysis of applying spiral imaging to different applications in MRI and MRSI. Matlab programs and C source codes are included in the Appendix.

ACKNOWLEDGEMENTS

This thesis documents the research I participated in from 2002 to 2006 in Magnetic Resonance Research Center (MRRC) of University of Pittsburgh Medical Center for my Ph.D. research program in the Department of Physics and Astronomy in University of Pittsburgh. I worked with Prof. Yi Wang from 2002 to 2004 on Time Resolved 3D MR Angiography and Prof. Qiuhong He from 2004 to 2006 on 2D MR spectroscopy and spiral MRSI with Selective Multiple Quantum Coherence Transfer. I am sincerely grateful to both Profs. Qiuhong He and Yi Wang for the opportunities to be part of their exciting research projects. They generously supported me with graduate research assistantship, foreign student status and mentored my projects without any reservation. This thesis is the result of my participation in their scientific achievements. We designed our experimental methods based on the spiral sequence and reconstruction programs from by Profs. Andew Stenger and Douglas Noll. We are grateful to their help.

Prof. Irving Lowe has been my academic advisor in the Department of Physics and Astronomy since 2002. I am deeply grateful to Prof. Irving Lowe for his continued support throughout the process of conducting research activities in the Dept. of Radiology outside of the Dept. of Physics and Astronomy. This thesis would not be possible without his help and advice. I also deeply appreciate the time and efforts of my other committee members: Profs. James Mueller, Ralph Roskies and David Snoke. Their comments and suggestions helped me improve this thesis and understand scientific principles. Profs. Michael Gach and Fernando Boabda also gave me generous help during my work at MRRC, for which I am deeply thankful. Dr. Kaung-Ti Yung offered me detailed and extensive editorial suggestions throughout the writing process. I am very much grateful to his efforts.

1.0 INTRODUCTION

Nuclear Magnetic Resonance (NMR) experiments are based on an interaction between paramagnetic moments of nucleons and an external magnetic field (1,2). The applications of NMR include solid state physics and structural chemistry (3). The invention of NMR imaging techniques in 1973 expanded its applications to biology and medicine (4,5), with new techniques in both imaging and spectroscopy. A discussion of new developments of NMR should start with an overview of its physical principles, which are best explained with a combination of classical picture and quantum mechanical treatment (6,7).

1.1 CLASSICAL TREATMENT OF NMR

1.1.1 Spin Precession and the Rotating Frame

Consider a magnetic moment $\vec{\mu}$ of a nuclear spin-½ in a constant magnetic field $\vec{B}_0$. Since the electric energy of a spin-½ vanishes, the total energy of the spin is its magnetic energy:

$$E_{mag} = -\vec{\mu} \cdot \vec{B}_0$$

Equation 1-1

The minus sign in front of the dot product indicates that the energy is the lowest when the magnetic moment vector $\vec{\mu}$ and the external field vector $\vec{B}_0$ are parallel. In an NMR experiment, this relation applies to a group of nuclear spins each with angular momentum $\frac{1}{2}\hbar$. Because of thermal excitation, each spin may not be aligned with the external magnetic field $\vec{B}_0$. The external magnetic field causes spins to precess around the direction of the field (Fig. 1-1). The

frequency of the precession ω_0, called the Larmor frequency, is proportional to the strength of the external magnetic field $\vec{B}_0$:

$$\vec{\omega}_0 = -\gamma \vec{B}_0 .$$

Equation 1-2

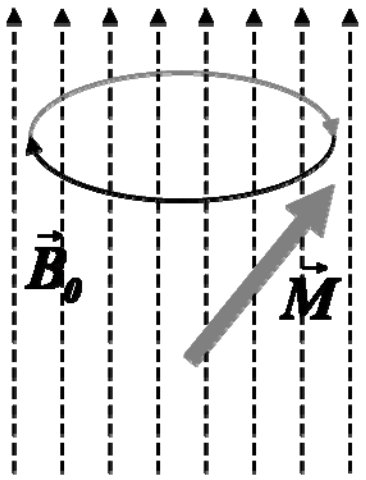

Figure 1-1 Spin Precession in an External Magnetic Field

An external magnetic field $\vec{B}_0$ causes spins to rotate around the direction of B_0 at Larmor frequency.

The proportionality constant γ is called the gyromagnetic ratio. The value of γ for hydrogen protons is 2.68×10^8 rad/s/Tesla (8), which translates to proton's resonance frequency of 63.9MHz at 1.5 Tesla and 127.8MHz at 3.0 Tesla. At thermal equilibrium the total net magnetic moment of a group of spins is aligned with the external field $\vec{B}_0$ so that the net magnetic moment does not precess. When the net magnetic moment is rotated by an alternating magnetic field $\vec{B}_1$ away from the direction of $\vec{B}_0$, a precessing moment is created. The alternating field that causes that rotation is a Radio Frequency (RF) pulse. The precessing magnetic moments collectively generate a small induced signal in an RF coil nearby. This NMR signal is amplified and recorded for post-processing.

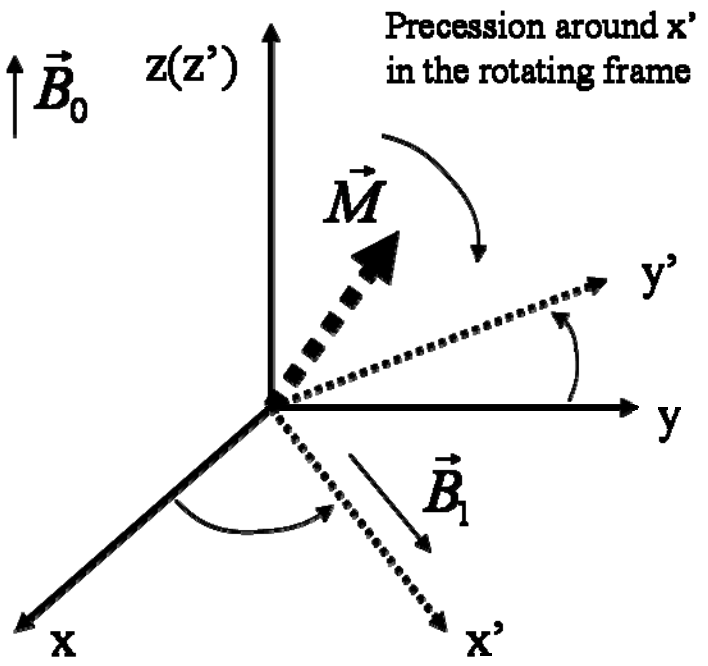

Figure 1-2 Spin Tipping in the Rotating Frame

The reference frame (x', y', z') rotates around z (z') axis in laboratory frame (x, y, z). A rotating field $\vec{B}_1$ is perpendicular to the direction of $\vec{B}_0$. $\vec{B}_1$ creates another precession in the rotating frame to tip the spins away from $\vec{B}_0$ direction as the basis for MR signal excitation.

The precession of the spins around the direction of $\vec{B}_0$ and their motion of "tipping" away from the direction of $\vec{B}_0$ are best explained in a rotating reference frame (9). This is a reference frame rotating at the Larmor Frequency around the direction of $\vec{B}_0$. The axes of the rotating frame is denoted by x', y' and z'. A $\vec{B}_1$ field is in the (*x*-*y*) plane perpendicular to the $\vec{B}_0$ field along z(z') direction. It is rotating at the Larmor frequency. The $\vec{B}_1$ field in the rotating frame (x', y', z') is a constant field perpendicular to the direction of $\vec{B}_0$ field, and it creates another spin precession around the direction of $\vec{B}_1$. The spin motion in the laboratory frame is called nutation. Another way to explain the rotating frame is that $\vec{B}_1$ has to be alternating at the Larmor frequency in order to exert a constant effect on precessing spins so they can be "tipped" away from $\vec{B}_0$ direction, as

shown in Fig. 1-2. The latter explanation is also considered the resonance condition. Spins may rotate at a different frequency due to inhomogeneity of the $\vec{B}_0$ field.

1.1.2 Flip-Angle Formula of an RF Pulse

An RF pulse in an NMR experiment is often characterized as an on-resonance $\vec{B}_1$ field activated for a small period of time, while off-resonance RF excitations are occasionally used for special purposes (10-12). If the magnitude of on-resonance $\vec{B}_1$ field is constant during its active period τ, the flip angle θ experienced by the magnetication $\vec{M}$ in the rotating frame is, therefore,

$$\theta = \gamma B_1 \tau \ .$$

Equation 1-3

If the magnitude of $\vec{B}_1(t)$ is time dependent, starting at t_0 and ending at t_1, the accumulated rotation angle θ is

$$\theta = \int_{t_0}^{t_1} dt \cdot \gamma B_1(t) \ .$$

Equation 1-4

An RF pulse is often labeled by its flip angle θ so that a 90° pulse refers to a $\vec{B}_1$ field with θ=90°. The magnitude and duration of an RF pulse can be determined according to the power of the RF transmition amplifier or the desired pulse bandwidth. When an RF pulse is plotted as a function of magnitude versus time (Figure 1-3), i.e. $B_1=B_1(t)$, the area under the plot is proportional to the desired flip angle.

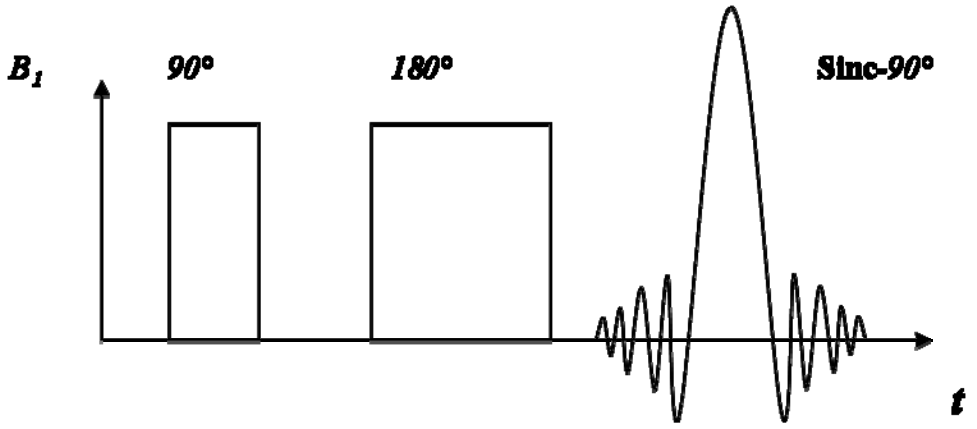

Figure 1-3 Examples of RF pulses.

Three RF pulses are plotted above as functions of the amplitudes of B_1 versus time. The area under each plot is proportional to its desired flip angle so that the 180° pulse shares the same amplitude with the first 90° pulse but the duration is twice as long. The pulses with a constant amplitude perform the corresponding rotation to spins across the entire spectrum. The 90° pulse with the shape of a sinc function performs a 90° rotation on spins in a limited frequency bandwidths.

The relative phase of an RF pulse is distinguished by the axis of the desired rotation in the rotating frame. For example, a $90°_x$ refers to a 90° rotation around the x axis while a $180°_y$ pulse refers to a 180° rotation around the y axis in the rotating frame.

1.1.3 Relaxation and The Bloch Equation

For magnetic moments $\vec{\mu}$ of nuclear spins in a macroscopic object, a property of the object, magnetization $\vec{M}$, can be defined as the average magnetic dipole moment density

$$\vec{M}(\vec{r}) = \frac{1}{V}\sum_{V(\vec{r})} \vec{\mu}_i \ .$$

Equation 1-5

The volume $V(\vec{r})$ at location $\vec{r}$ is small enough so that $\vec{B}_0$ across $V(\vec{r})$ can be treated as a constant but it is also big enough so that there are enough dipole moments in it. In the absence of an RF field, the equation of motion for $\vec{M}(\vec{r})$ neglecting all sources of relaxation and inhomogeneous dephasing is (13,14)

$$\frac{d\vec{M}(\vec{r})}{dt} = \gamma \vec{M}(\vec{r}) \times \vec{B}_0 .$$

Equation 1-6

Since the $\vec{B}_0$ field is a static field in the *z* direction ($\vec{B}_0 = B_0 \hat{z}$), the above equation of motion can be decoupled into two equations. The component of $\vec{M}(\vec{r})$ that is parallel to the direction of $\vec{B}_0$ is the longitudinal magnetization $M_{\parallel}=M_z$. The perpendicular components to $\vec{B}_0$ are the transverse magnetization and $\vec{M}_{\perp} = M_x \hat{x} + M_y \hat{y}$, where M_x and M_y are the *x* and *y* components of the transverse magnetization. The decoupled equations are

$$\frac{dM_z}{dt} = 0 ,$$

Equation 1-7

and

$$\frac{d\vec{M}_{\perp}}{dt} = \gamma \vec{M}_{\perp} \times \vec{B}_0 .$$

Equation 1-8

The equilibrium state of the magnetization occurs when the net magnetization is parallel to $\vec{B}_0$. As spins precess around the direction of $\vec{B}_0$ after RF tipping by a $\vec{B}_1$ field, the energy of the precession will dissipate gradually. The precessing magnetization will eventually be aligned parallel to $\vec{B}_0$ and become longitudinal when spins reach to the equilibrium state. This process of losing precession energy and re-growing the magnetization $\vec{M}$ along the longitudinal magnetization is called Spin-Lattice relaxation, as illustrated in Fig.1-4. Its time constant T_1 is introduced into Equation [1-7] such that

$$\frac{dM_z}{dt} = \frac{1}{T_1}(M_0 - M_z).$$

Equation 1-9

M_0 is the total magnetization at equilibrium. The solution to the equation above is

$$M_z(t) = M_z(0)e^{-t/T_1} + M_0\left(1 - e^{-t/T_1}\right).$$

Equation 1-10

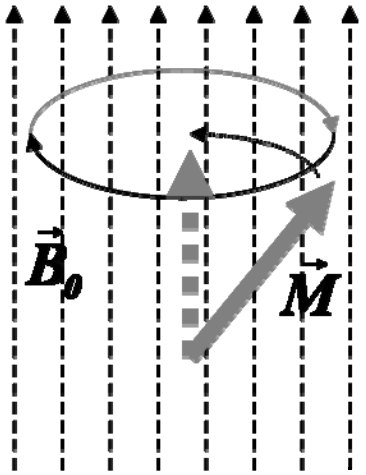

Figure 1-4 Spin-Lattice Relaxation

Spin-Lattice relaxation causes the non-equilibrium magnetization gradually realign to the direction of $\vec{B}_0$ as they precess around the direction of $\vec{B}_0$.

Spin-Lattice relaxation, or T_1 relaxation, describes how the transverse magnetization $\vec{M}_\perp$ is reduced as $\vec{M}$ is being aligned to $\vec{B}_0$ direction. In addition, the magnitude of the transverse magnetization is also reduced gradually as spins precess more and more out-of-phase. This process is called Spin-Spin relaxation. Therefore, after spins are rotated into the transverse plane by an RF pulse, the spins spread out and realign to $\vec{B}_0$ at the same time (Fig. 1-5).

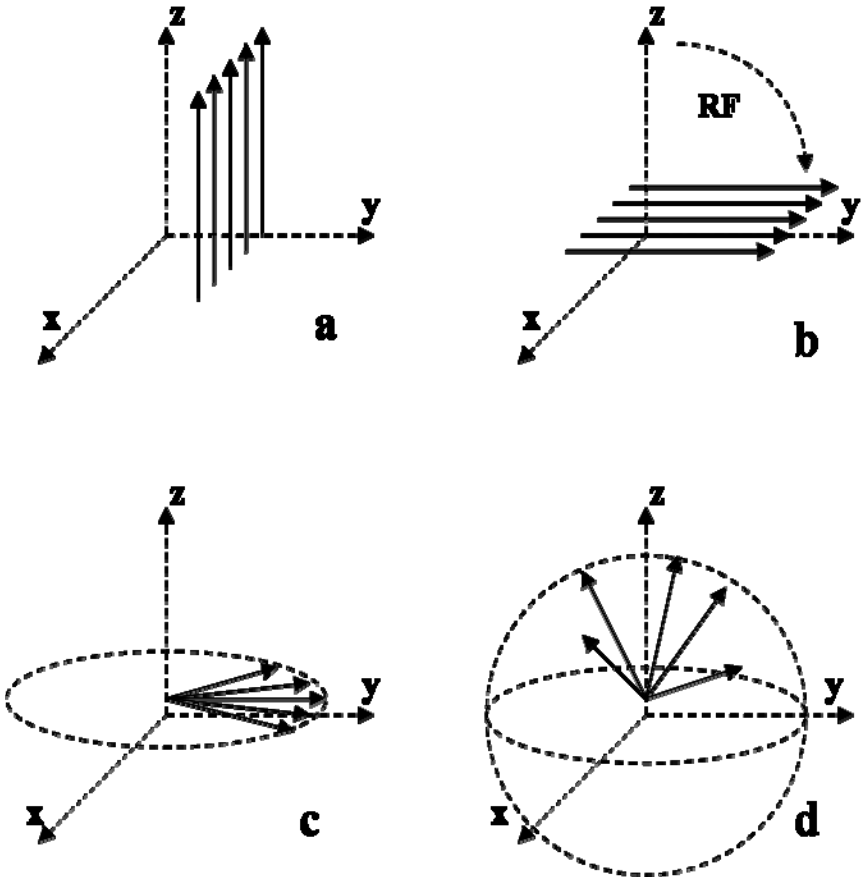

Figure 1-5 Combined Effect of Spin-Lattice Relaxation and Spin-Spin Relaxation

(a) Magnetization is aligned with $\vec{B}_0$ at thermal equilibrium. (b) An RF pulse generates transverse magnetization by rotating spins away from $\vec{B}_0$ direction. (c) Magnetic precession loses coherence and spreads out as they precess around the direction of $\vec{B}_0$ in the T_2 relaxation process. (d) As the result of T_1 and T_2 relaxations, transverse magnetization in precessing decays gradually to zero.

The loss of spin coherence of the precession caused by spin-spin interaction is called T_2 relaxation (14). The process of T_2 relaxation is described by introducing a relaxation coefficient T_2 into Equations [1-8] so that

$$\frac{d\vec{M}_\perp}{dt} = \gamma \vec{M}_\perp \times \vec{B} - \frac{1}{T_2}\vec{M}_\perp .$$

Equation 1-11

The solution of $\vec{M}_\perp$ in the rotating frame is

$$\vec{M}_{\perp} = \vec{M}_{\perp}(0)e^{-t/T_2},$$

Equation 1-12

where $\vec{M}_{\perp}(0)$ is the initial value of $\vec{M}_{\perp}$ immediately after the RF pulse. T_1 and T_2 are intrinsic properties of the spin system.

The inhomogeneity of the external field $\vec{B}_0$ is another source for spins to precess out of phase. The combined effect on the exponential decay of transverse magnetization can be described by Equation [1-12] if the decay coefficient T_2 is replaced by T_2^*, where

$$\frac{1}{T_2^*} = \frac{1}{T_2} + \frac{1}{T_2'}.$$

Equation 1-13

The additional source of dephasing caused by field inhomogeneity has a decay constant T_2'. T_2^* is an observed coefficient in a particular NMR experimental set up. Since $T_2^* < T_2$, the actual decay due to T_2^* dephasing is faster than T_2 relaxation. Spin-Lattice relaxation and spin-spin relaxation are included in a combined equation of motion for spin magnetization. It is the Bloch equation (2,13-15)

$$\frac{d\vec{M}}{dt} = \gamma\vec{M} \times \vec{B}_0 + \frac{1}{T_1}(M_0 - M_z)\hat{z} - \frac{1}{T_2}\vec{M}_{\perp}.$$

Equation 1-14

The solutions to the Bloch equation in the rotating frame are:

$$M_x(t) + iM_y(t) = \left(M_x(0) + iM_y(0)\right) \cdot e^{-t/T_2}$$

$$M_z(t) = M_z(0)e^{-t/T_1} + M_0\left(1 - e^{-t/T_1}\right).$$

Equation 1-15

1.1.4 Spin Echo Experiment

Comparing the two sources responsible for dephasing of transverse magnetization, the dephasing caused by spin-spin interaction is irreversible while dephasing caused by field inhomogeneity can be rephased by a 180° refocusing pulse, as in a spin echo experiment (1) that consists of a $90°_x$ pulse followed by a $180°_y$ pulse (Fig. 1-6).

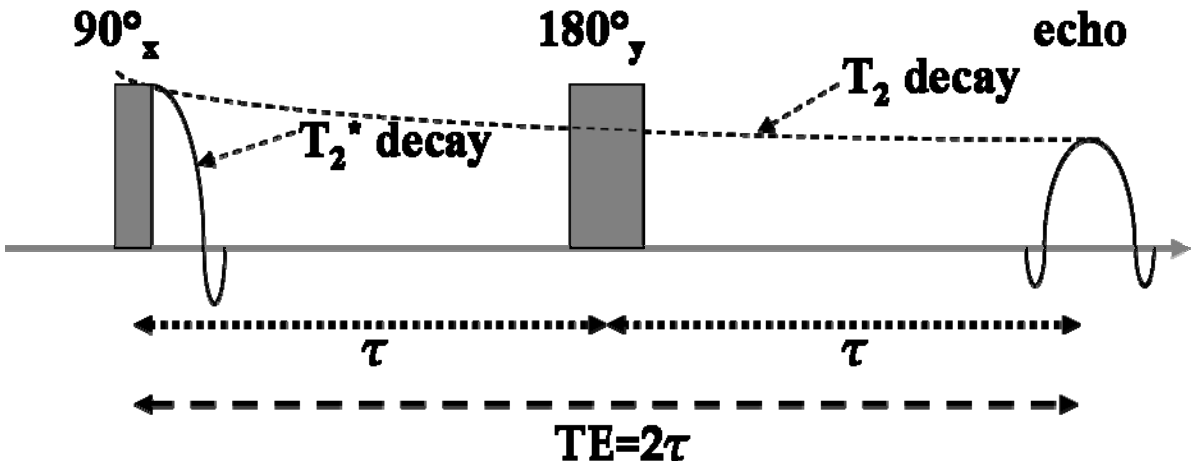

Figure 1-6 Diagram of a Spin Echo Experiment

A spin echo experiment consists of a 90° pulse and 180° pulse separated by a delay τ. The 180° pulse refocuses the spin dephasing caused by magnetic field inhomogeneity so that the transverse magnetization reappears as an echo at τ after the 180° degree pulse. The fast decay of transverse magnetization after the 90° pulse is caused by T_2* decay while the reduction of the transverse magnetization at the echo maximum is due to T_2 decay.

The $90°_x$ pulse flips $\vec{M}$ in the object into the transverse plane and they precess at their local Larmor Frequency $\omega_0+\delta(r)$. ω_0 is the expected Larmor Frequency determined by $\vec{B}_0$ and $\delta(r)$ is the spatially dependent change of magnetic fields caused by the field inhomogeneity. The precession due to the $\delta(r)$ term accumulates a spatially dependent phase term $\delta(r)\tau$ when the 180° pulse is applied at a time τ after the 90° pulse. The $180°_y$ pulse rotates $\vec{M}$ around the y axis so that the phase term $\delta(r)\tau$ becomes $-\delta(r)\tau$ at all locations. Thus, at time 2τ after the 90° pulse, the $\delta(r)\tau$ phase factor is refocused. The amplitude of the precessing magnetization $\vec{M}_\perp$ decays after the 90° pulse, but it grows back after the 180° pulse and reaches a maximum (an echo) at τ after the 180° pulse. The time interval 2τ between the first 90° pulse and the signal maximum is called echo time (TE = 2τ). The loss of transverse magnetization after the 90° pulse to echo time is spin-spin interaction or T_2 relaxation, while the decay of magnetization immediately after the 90° pulse is caused by T_2* relaxation (Eq. 1-13). Spin echo experiments can be used to measure T_2 relaxation time in an inhomogeneous magnetic field.

1.1.5 Principles of MRI and K-space Mapping

The basis of generating a spatial image from a precessing $\vec{M}_{\perp}$ is to form a correlation between signals and the locations of their sources (4). Although spins are never aligned to the direction of $\vec{B}_0$ in a thermal equilibrium, the magnetization ($\vec{M}_0$) as the average of spin magnetic moments is aligned to $\vec{B}_0$ at equilibrium so that there are no detectable signal from spin precession. An RF pulse rotates a portion of the magnetization into the transverse plane across the entire object and all spins precess in phase regardless of their spatial location. Any signal detected at this stage is a sum of signals from all locations. A first step to Magnetic Resonance Imaging (MRI) is to vary the frequencies of spin precession according to the locations of the spins. An MRI scanner is equipped with three independent sets of gradient coils to achieve this. Each gradient coil produces an additional magnetic field that is in the same direction as $\vec{B}_0$, but the magnitude varies linearly along a corresponding axis. The amplitude of a gradient $G_x(t)$ is determined by a current from a power amplifier as a function of time, and the net magnetic field along the x axis as a function of time and space is

$$B_x(x,t) = B_0 + xG_x(t).$$

Equation 1-16

Fig. 1-7 below is an example of a gradient field along *x* axis $G_x(t)$.

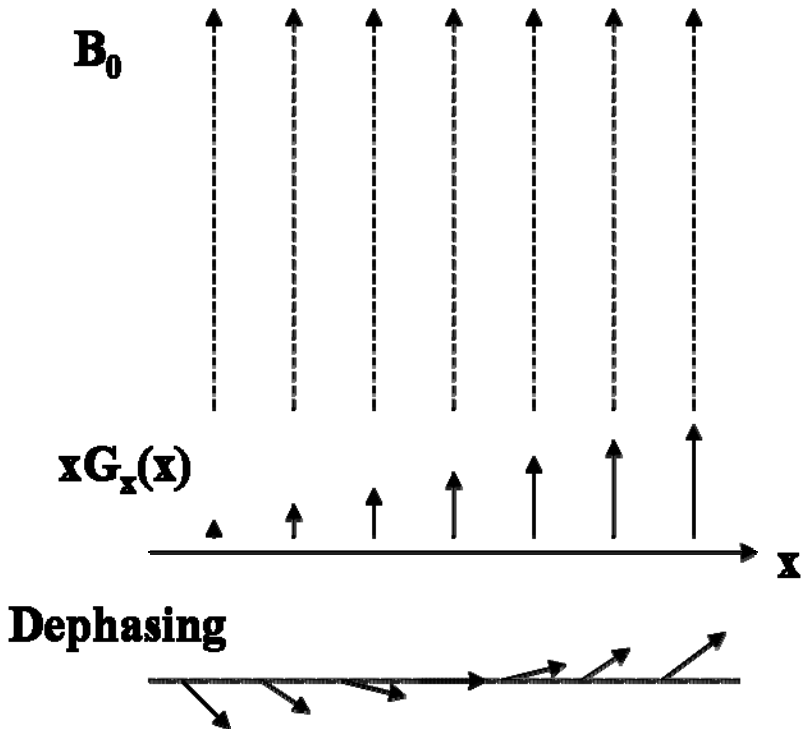

Figure 1-7 Effects of a Linear Magnetic Field Gradient on Magnetization Components

The magnetic field of gradient field $xG_x(t)$ is in the same direction as $\vec{B}_0$. Its amplitude $B_0+xG_x(t)$, however, varies linearly along the *x* axis. Spin precession frequencies along *x* axis depend on the gradient field so that spins precess out of phase. As a result, the spins are dephased by $xG(x)$, giving a reduced transverse magnetization.

The sum of the constant $\vec{B}_0$ field and the linearly varying gradient field $xG_x(t)$ determine the spatially dependent precession frequencies. The spatial dependence is only caused by the gradient so that $\omega(x,t)= \omega_0+\gamma xG_x(t)$. The spins distributed along the *x*-axis will precess at different frequencies when the gradient field along that axis is activated. The shape of $G_x(t)$ can be manipulated to deliver any gradient waveform required by a MR pulse sequence.

On the other hand, the signal $s(t)_{emf}$ generated by $\vec{M}_\perp$ precession in a nearby detection coil is proportional to the rate of the changing magnetic flux Φ through the detection coil (6):

$$s(t)_{emf} = -\frac{d\Phi}{dt}, \quad \Phi = \int_{coil\,area} \vec{B} \bullet d\vec{S}\,.$$

Equation 1-17

The magnetic field $\vec{B}$ is generated by the precessing magnetization at the location of the detection coil. The flux Φ integrated around the area of the coil is calculated using the principle of reciprocity. It states that the flux through the detection coil due to the precessing magnetization is equal to the magnetic flux generated per unit current from the detection coil to the same region of the magnetization. This transformation leads to a formula for the flux that is more convenient for calculation in NMR experiments:

$$\Phi(t) = \int_{sample} d^3 r \vec{B}^{receive}(r) \cdot \vec{M}(\vec{r},t).$$

Equation 1-18

The imagined magnetic field $\vec{B}^{receive}$ is generated from the detection coil and $\vec{M}(r,t)$ is the spatial distribution of magnetization. The dependence of flux Φ(t) on time includes a fast spin precession at the Larmor frequency (ω_0) and a slow exponential decay of the precession amplitude. The signal *s(t)* is proportional to the derivative of the flux, $d\Phi(t)/dt$. The derivative $d\Phi(t)/dt$ is approximated by taking into account of only the derivative of the precession that leads to the ω_0 in the signal equation

$$s(t) \propto \omega_0 \int d^3 r \cdot e^{-t/T_2} B_\perp(\vec{r}) M_\perp(\vec{r},0) e^{i\left[(\omega_0+\omega(\vec{r}))t+\phi_0(\vec{r},t)-\theta_B(\vec{r})\right]}.$$

Equation 1-19

The perpendicular components of $\vec{B}^{receive}$ and $\vec{M}(\vec{r},t)$ in Eq. 1-18 are denoted by $B_\perp(\vec{r})$ and $M_\perp(\vec{r},0)$, respectively. The time dependency of $\vec{M}(\vec{r},t)$ is expressed separately in the exponential decay term with a decay constant T_2. This constant may be replaced by T_2* when the gradient fields are active. The signal decay due to T_1 relaxation during the signal acquisition period is neglected. An initial phase factor $\theta_B(\vec{r})$ is caused by the spatial variations of the coil sensitivity. In the scope of this thesis, the coil sensitivity is assumed to be spatially independent so the terms resulted from the coil $B_\perp(\vec{r})$ and $\theta_B(\vec{r})$ become constants and they are combined in one constant *A* below. The shift of Larmor frequency $\omega(\vec{r},t)$ is caused by the gradient fields. The time and space dependent phase factor $\phi_0(\vec{r},t)$, accumulated by $\omega(\vec{r},t)$, is also caused by the gradient fields. The measured sum of all the magnetization as the result of the gradient is an integral across the area of the magnetization. The accumulated effect of the gradient fields from the time of the RF excitation (t'=0) to the time of signal acquisition (t'= t) is

$$\phi(\vec{r},t) = \int_0^t dt' \omega(\vec{r},t').$$

Equation 1-20

Setting the phase factor $\phi(\vec{r},t)$ to the sum of $\omega(\vec{r},t)t$ and $\phi_0(\vec{r},t)$, Eq. 1-19 becomes

$$s(t) = \omega_0 A e^{-t/T_2} \int d^3 r M_\perp(\vec{r},0) e^{i\left[\omega_0 t + \phi(\vec{r},t)\right]}.$$

Equation 1-21

If the assumption is made again that only the gradient field along *x* axis is active, the frequency ω is proportional to the gradient field strength $xG_x(t)$ so that $\omega(x,t)=\gamma x G_x(t)$. A new parameter often called *spatial frequency*, $k=k(t)$, is defined as

$$k(t) = \frac{\gamma}{2\pi} \int_0^t dt' G_x(t').$$

Equation 1-22

For Cartesian k-space sampling, the value of k is determined by an integration of a linear gradient. Signal demodulation[1] removes the constant modulation at the Larmor frequency ω_0 in the acquired signal so that the demodulated signal equation can be finally reduced to

$$s(k(t)) = e^{-t/T_2} \int dx M(x) e^{-i2\pi k x}.$$

Equation 1-23

In Eq. 1-23, $M(x)$ is the one dimensional magnetization distribution. The equation above shows that the measured MR signal is the Fourier Transform (FT) of the spatial magnetization distribution $M(x)$ with spatial frequency k. When conducting a multi-dimensional imaging experiment the integrations of orthogonal gradients $G_x(t)$, $G_y(t)$ and $G_z(t)$ in the form of Eq.[1-22] produce three spatial frequencies k_x, k_y and k_z. The signal as a function of time is a series of samples in spatial frequency domain (k-space) with coordinates k_x, k_y and k_z. MRI image acquisition, therefore, can be viewed as sampling k-space point by point using three gradient fields. The time dependent locations of the sample points are controlled by the time dependent waveforms of three gradient fields.

The acquired k-space data can be inverse Fourier Transformed (iFT) to obtain the desired image. The transformed MR image is based upon a discrete sampling in the k-space. The process

[1] A process of demodulation removes the modulation at the Larmor frequency from the RF signal to get the signal corresponding to the magnitude of the transverse magnetization.

of the discrete k-space sampling can be expressed mathematically as $s(k)\cdot u(k)$, where $s(k)$ is a continuous FT of the image that is being sampled and $u(k)$ is a sampling function, i.e., $u(k)$= 1 if $s(k)$ is sampled and $u(k)$= 0 if it is not. The inverse Fourier transform of the sampled data is

$$F^i[s(k)\cdot u(k)] = F^i[s(k)] * F^i[u(k)]$$

Equation 1-24

The ideal image of the object is denoted by $F^i[s(k)]$. The star symbol * denotes a convolution and $F^i[u(k)]$ is the Point Spread Function (PSF) of the sampling pattern. The most frequently used sampling pattern is a 2D or 3D Cartesian grid because the Inverse Fast Fourier Transform(iFFT) can be directly applied. The PSF of a 2D Cartesian grid is another Cartesian grid so that when k-space is sampled by a 2D Cartesian grid, the reconstructed image $F^i[s(k)]*F^i[u(k)]$ is a 2D matrix of replicas of the image. Only the image at the center of the image space is needed. All the other replicas in Fig. 1-8 are aliased images. If the object extends outside the FOV, its image will appear or "wrap-around" in the opposite side of the image. The parameters of the Cartesian grid determine the size and resolution of the reconstructed image as indicated in the figure below. The sample spacing in the k-space is denoted by Δk. The range of sampling in k-space is K_{max}. The field-of-View (FOV=1/Δk) is the size of the image.

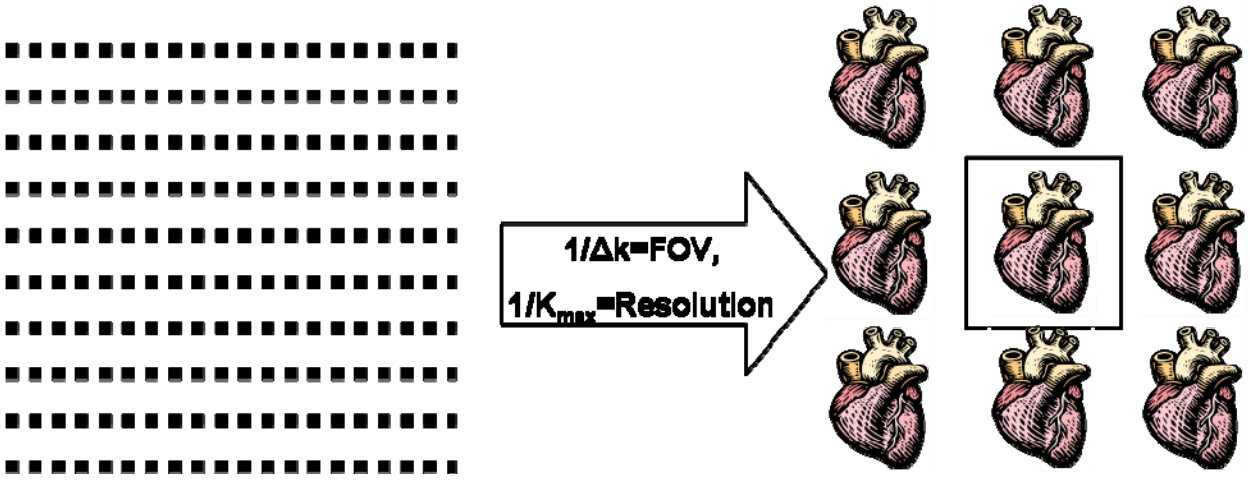

Figure 1-8 Principles of 2DFT MRI

A 2D matrix of k-space sample points are Fourier Transformed into the target image and its aliased images. The spacing of sample points in k-space determines the size of the image and the range of k-space coverage determines the resolution of the image.

Spiral Magnetic Resonance Imaging has been successfully implemented in numerous areas of MRI including cardiovascular MRI (16) and functional MRI (17). In this thesis, two new

techniques discussed are Time Resolved 3D MR Angiography and Spectroscopic Imaging with Multiple Quantum editing. Each application leads to a distinct configuration of the spiral imaging parameters. Spiral imaging uses an alternative k-space sampling pattern (16). In spiral MRI, a sampling trajectory starts at the center of k-space and ends at the edge in a spiral curve or vice versa. The k-space can be covered by either a single spiral trajectory or multiple trajectories (Fig. 1-9a). The PSF of a spiral trajectory is an impulse function at the center of the image space and a ring around it (Fig.1-9b). The radius of the ring depends on the spacing between neighboring sampling curves in the k-space.

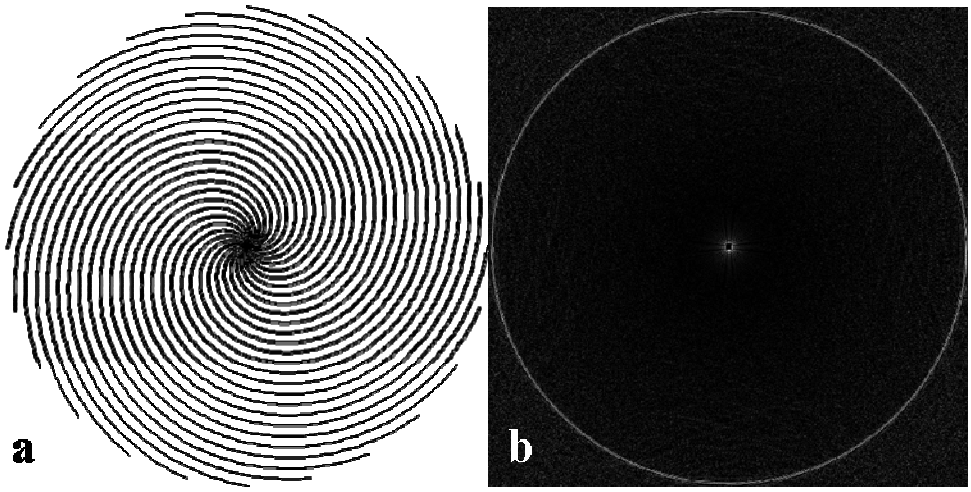

Figure 1-9 Principles of Spiral Imaging

(a): A sampling pattern of 24 spiral interleaves in k-space. (b): The Point Spread Function (PSF) of the sampling pattern shown in (a). The dot at the center corresponds to the image and the ring is the source of spiral aliasing. The area inside the ring is the field-of-view (FOV). 24 interleaves are used of which, constant density spiral [$k_c(\tau)=k_{max}\,\tau\,e^{i\omega\tau}$] is used at center of the k-space and variable density spiral [$k_v(\tau)=k_{max}\,(a\,\tau^2+b\,\tau+c\,)e^{i\omega\tau}$] is used for the rest of the trajectory.

1.2 MOTIVATION OF SPIRAL DATA ACQUISITION

This thesis discusses two new technical developments of fast spiral imaging. They are Magnetic Resonance Spectroscopic Imaging (MRSI) and Time Resolved 3D Magnetic Resonance Imaging. The developments in both areas improve the ability of diagnosis and monitoring of diseases in clinics. MRSI is used to acquire spatial distributions of tissue composition. MRSI

techniques are applied to assist in cancer diagnosis and to monitor cancer treatment. Unfortunately, most MRSI techniques for cancer require long scan times. We applied spiral imaging to the echo of a Sel-MQC sequence to acquire a sub-centimeter map of Polyunsaturated Fatty Acid (PUFA) in 2 minutes in the breast. PUFA may be a possible index for early tumor development (18-21). Due to its short acquisition time, this specific technique of PUFA imaging can be conveniently combined with clinical MR breast scans to enhance diagnosis specificity without a significant increase in scan time.

Time resolved spiral MRI acquires 2D or 3D MR images rapidly to form a temporal series of images. As shown in Chapter 4, our new application of stack-of-spiral data acquisition technique has pushed the performance of time resolved imaging to a new level (22). For the first time, a 3D image set of sub-millimeter resolution can be updated every two seconds using only a one-channel receiver. The technique provides the ability to generate a high resolution time resolved angiogram on virtually any 1.5 tesla MR scanner with only a new software upload. The result can be used to study dynamic body functions like cardiac motion or blood flow in human vasculature as demonstrated in Chapter 4.

The goal of fast imaging in MR is to acquire more or better data in a given time period or to acquire adequate data in shorter time. Spiral acquisition can be configured for either approach. At the moment spiral acquisition in most applications is still a research option because of the complexity of spiral sequence design. The spiral data sampling can improve fast MR imaging in a fashion that is more powerful than Cartesian sampling.

1.2.1 MRSI

MRSI is used to map concentrations of one or more chemical species. Chemical Shift Imaging (CSI) (23,24) is the current standard method for MRSI. Each data acquisition of CSI acquires a readout line in a 3D space determined by (k_x, k_y) as shown in Fig.1-10. k_x-k_y values are determined by phase encoding gradients after signal excitation. A receiver records one FID along the t axis. A CSI sequence acquires $N_x \times N_y$ FIDs to form a three dimensional data matrix. Data reconstruction of CSI consists of Fourier Transformations along all three axis and the result is a matrix of spectra. Each matrix element is a spectrum at the location of a pixel in the image space.

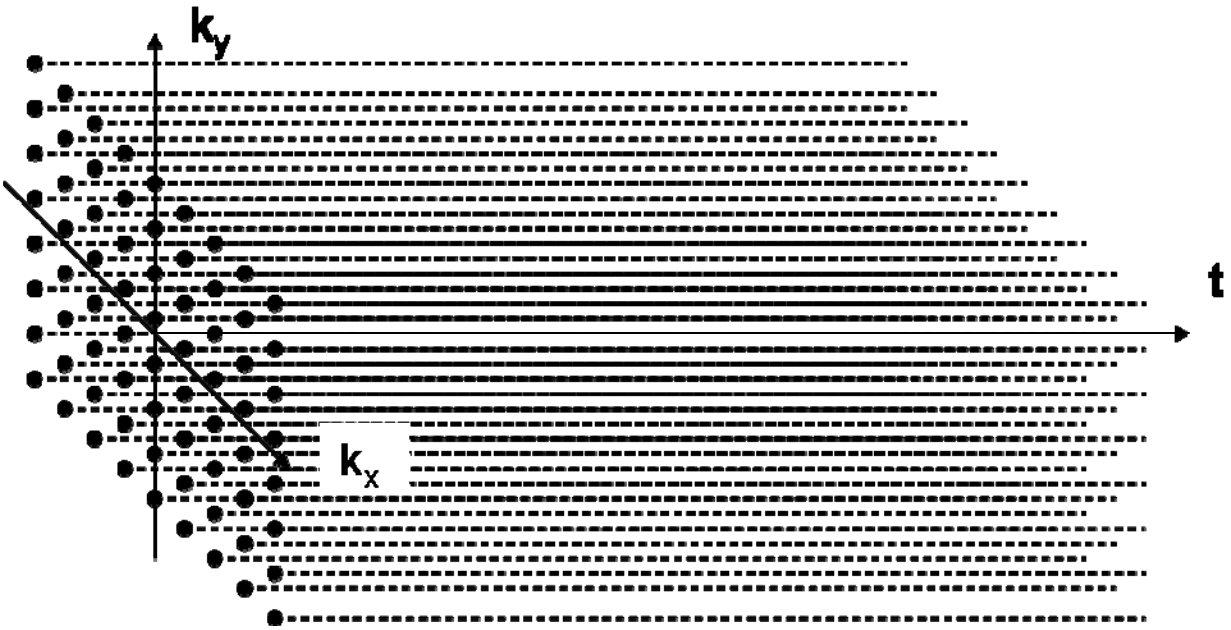

Figure 1-10 K-space Coverage of Chemical Shift Imaging

Each sample point in k_x-k_y plane is sampled continuously along time axis. A 2D data matrix in k_x-k_y is extended into a 3D matrix in k_x-k_y-t that makes up a data set of Chemical Shift Imaging.

Standard CSI acquisition for MRSI leads to long scan time that limits its clinical applications. New developments like EPSI (25,26), spectroscopic RARE (27) have been introduced to acquire more data to cover more of k-space than CSI or Cartesian mapping in one acquisition. Spiral imaging can be applied to MRSI to achieve the similar goal. Water and fat suppress is another issue in MRSI. A typical water/fat suppression procedure consists of a number of CHESS (28) suppression pulses in the beginning of an MRSI sequence. The Sel-MQC sequence (29) to be discussed in Chapter 2 serves as a filter to simultaneously suppress water and lipid signals in a single scan. Sel-MQC's optimal suppression of water and lipid signals enables fast imaging to be applied to MRSI.

1.2.2 Time Resolved MRI

Earlier efforts of Time Resolved MRI were developed from 2D or 3D Cartesian sampling. Time Resolved Imaging with Contrast kinetics (TRICKS) and its later derivations (30,31) are well

known examples. They represent a basic principle of improving sampling efficiency that is to oversample the center of the k-space where most energy is concentrated and undersample the edge of k-space to save time.

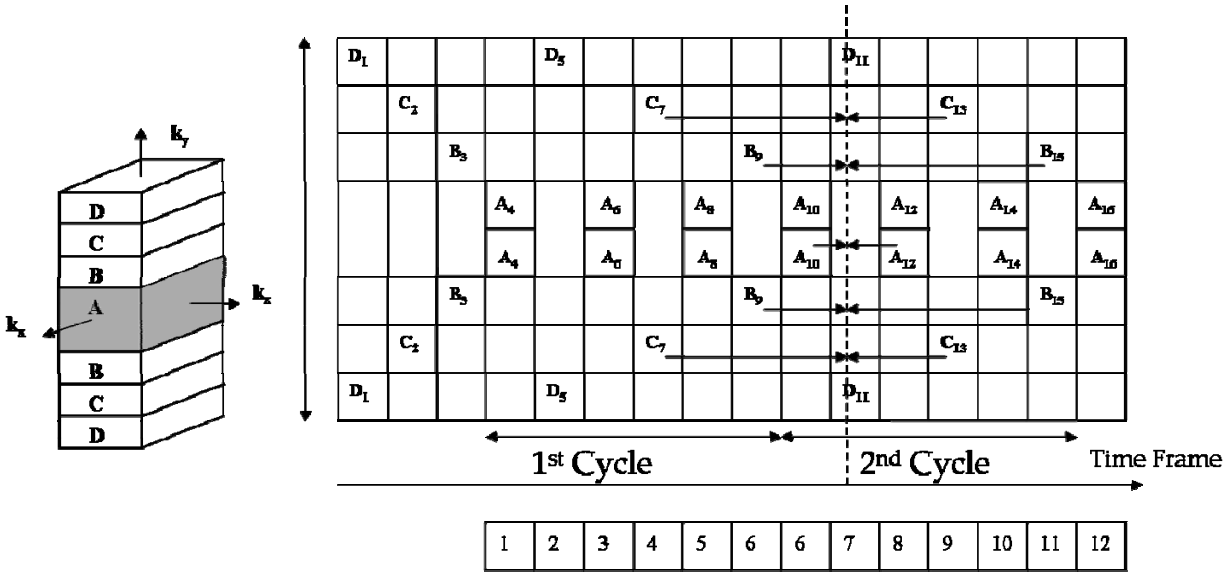

Figure 1-11 Principles of TRICKS

A k-space volume is divided into several blocks that are sampled individually. Each column of blocks represents the k-space data for a time frame. The center block is sampled every other time frame and the other blocks are sampled in cycles. The acquisition order is, therefore, $2D_1 \rightarrow 2C_2 \rightarrow 2B_3 \rightarrow 2A_4 \rightarrow 2D_5$… The data in the missing blocks in each column are interpolated using the data acquired before and after that time as shown with the arrows. For example, in the 11th column marked by the dashed line, D_{11} is directly acquired. C_{11} is obtained by interpolating the values of C_7 and C_{13} so that $C_{11}=(33\%C_7+67\%C_{13})$. The weighting coefficients are determined by the temporal distance to C_{11}. A temporal series of images are constructed as shown in the bottom (30,31).

The above diagram shows the principles of TRICKS. An entire k-space volume is divided into several blocks according to their distance to the center of k-space. Each temporal frame of image is constructed with one full volume but only a part of that volume is actually sampled during that period. The center block is sampled every other time frame and the other blocks are sampled in turn. The data in the missing blocks in each volume are interpolated using the data acquired before and after that time. Later the TRICKS method evolves into PR-TRICKS (32,33) where Cartesian sampling in k_x-k_y plane is replaced with projection sampling and TRICKS scheme is used in phase encoding direction. Time-Resolved 3D spiral sampling we introduced in (Chapter 4) moves one step further from TRICKS with variable density spiral sampling in k_x-k_y plane that

oversamples the center of the k-space and undersamples the outer areas. The sliding window reconstruction technique of the time-resolved spiral sampling divides a 3D volume in the k-space into four subsets. The four subsets are rotationally symmetric and they are sampled in turn as the Cartesian blocks in TRICKS (Fig.1-11). Every consecutive four subsets provide a full 3D data set that corresponds to the linear interpolation procedures in TRICKS. The technical details of the time-resolved spiral MRI and its clinical applications in Contrast Enhanced Magnetic Resonance Angiography are described in Chapter 4.

1.3 QUANTUM MECHANICAL TREATMENT OF NMR

The classical treatment of NMR outlined in the previous section is adequate for understanding most MRI experiments for isolated spins-1/2 systems. Spectroscopy and Spectroscopic Imaging, on the other hand, are better explained using Quantum Mechanics when spin coupling is considered (7,34). This section contains an introduction to the quantum mechanical descriptions of NMR.

1.3.1 Angular Momentum

Before defining the angular momentum operators, let's consider a free particle moving in three dimensions. The wave function ψ representing the state of this particle is

$$\psi = \psi(x, y, z, t).$$

Equation 1-25

The three Cartesian components of the angular momentum of this particle are given by three operators $\hat{l}_x$, $\hat{l}_y$ and $\hat{l}_z$, which act on the wave function ψ of this particle. These angular momentum operators are defined using coordinate operators $\hat{x}$, $\hat{y}$ and $\hat{z}$ and derivative operators $\hat{D}_x = \partial/\partial x$, $\hat{D}_y = \partial/\partial y$ and $\hat{D}_z = \partial/\partial x$ along the three axes. The operation of a coordinate operator along one axis is to multiply the wave function with the coordinate. The

operation of a derivative operator along one coordinate is to take a partial derivative with respect to that coordinate. The three angular momentum operators are

$$\hat{l}_x = -i\left(\hat{y}\hat{D}_z - \hat{z}\hat{D}_y\right)$$
$$\hat{l}_y = -i\left(\hat{z}\hat{D}_x - \hat{x}\hat{D}_z\right),$$
$$\hat{l}_z = -i\left(\hat{x}\hat{D}_y - \hat{y}\hat{D}_x\right)$$

Equation 1-26

with i^2= -1. An important property of the angular momentum operators is that they satisfy three commutation relations that form a cyclic permutation rule:

$$\left[\hat{l}_x, \hat{l}_y\right] = i\hat{l}_z$$
$$\left[\hat{l}_y, \hat{l}_z\right] = i\hat{l}_x$$
$$\left[\hat{l}_z, \hat{l}_x\right] = i\hat{l}_y.$$

Equation 1-27

Rotating a particle around a Cartesian axis is a frequently used physical operation. The rotation operators $\hat{R}_x$, $\hat{R}_y$ and $\hat{R}_z$ acting on a particle rotating around each Cartesian axis with angle θ can consequently be defined using the angular momentum operators

$$\hat{R}_x(\theta) = exp\left(-i\theta\hat{l}_x\right)$$
$$\hat{R}_y(\theta) = exp\left(-i\theta\hat{l}_y\right)$$
$$\hat{R}_z(\theta) = exp\left(-i\theta\hat{l}_z\right).$$

Equation 1-28

The translational motion of the particle in space is described by its coordinate operators whereas the rotational motion of the particle is described by the angular momentum operators. Another degree of freedom of nucleus, spin, can be described by its *intrinsic* spin angular momentum operators denoted by $\hat{I}_x$, $\hat{I}_y$ and $\hat{I}_z$. They form similar cyclic commutation relationships:

$$\left[\hat{I}_x, \hat{I}_y\right] = i\hat{I}_z$$
$$\left[\hat{I}_y, \hat{I}_z\right] = i\hat{I}_x$$
$$\left[\hat{I}_z, \hat{I}_x\right] = i\hat{I}_y.$$

Equation 1-29

The total spin angular momentum operator is defined as

$$\hat{I}^2 = \hat{I}_x{}^2 + \hat{I}_y{}^2 + \hat{I}_z{}^2.$$

Equation 1-30

The operator $\hat{I}^2$ commutes with $\hat{I}_z$ and their respective eigenequations are

$$\hat{I}_z|I,m\rangle = m|I,m\rangle,$$
$$\hat{I}^2|I,m\rangle = I(I+1)|I,m\rangle.$$

Equation 1-31

The state vector $|I,m\rangle$ is the eigenstates of both $\hat{I}^2$ and $\hat{I}_z$. *m* is the eigenvalue of $\hat{I}_z$ and *I(I+1)* is the eigenvalue of $\hat{I}^2$. If the total spin angular momentum has quantum number *I*, then *m* has *2I+1* possible values, *-I*, *-I+1*, *-I+2*,…,*+I*. For example, a nuclear spin-3/2 can have 4 possible values for its *z* component I_z: +3/2, +1/2, -1/2 and -3/2. A spin-½ can only take values +1/2 and -1/2 for its *z* component. Only spins-1/2 are considered in the scope of this thesis. The two eigenstates $|I,m\rangle = \left|\frac{1}{2},+\frac{1}{2}\right\rangle$ or $\left|\frac{1}{2},-\frac{1}{2}\right\rangle$ are orthogonal and are called the Zeeman Eigenstates, or Zeeman Eigenbasis because any spin state can be expressed as a superposition of Zeeman eigenstates. All states and operators can be represented in matrix form using Zeeman eigenbasis.

For spins-1/2, the spin angular momentum operators in matrix form using Zeeman eigenstates are

$$\hat{I}_x = \frac{1}{2}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; \hat{I}_y = \frac{1}{2i}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}; \hat{I}_z = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Equation 1-32

The three rotation operators in matrix form are

$$\hat{R}_x(\theta) = \begin{pmatrix} \cos\frac{1}{2}\theta & -i\sin\frac{1}{2}\theta \\ -i\sin\frac{1}{2}\theta & \cos\frac{1}{2}\theta \end{pmatrix}$$
$$\hat{R}_y(\theta) = \begin{pmatrix} \cos\frac{1}{2}\theta & -\sin\frac{1}{2}\theta \\ \sin\frac{1}{2}\theta & \cos\frac{1}{2}\theta \end{pmatrix}$$
$$\hat{R}_z(\theta) = \begin{pmatrix} \exp\left(-i\frac{1}{2}\theta\right) & 0 \\ 0 & \exp\left(+i\frac{1}{2}\theta\right) \end{pmatrix}.$$

Equation 1-33

1.3.2 Hamiltonian of an Uncoupled Spin ½

In the classical theory, a magnetic moment acts like a magnetic compass that interacts with an external magnetic field. In the quantum theory, the treatment starts by inspecting the Hamiltonian of a spin-½ interacting with external electromagnetic fields (35,36). The electric interaction between an external field and the orientation of a spin- ½ vanishes because all electric multipole moments are zero for spins-½. This means that the Hamiltonian contains only magnetic interaction. The magnetic moment of a spin-½ is proportional to the spin angular momentum and the proportionality constant is the gyromagnetic ratio γ. The Hamiltonian $\hat{\mathcal{H}}$ as the dot product of the external magnetic field $\vec{B}_0 = B_0\hat{z}$ and the angular momentum $\hat{I}_z$ is therefore

$$\hat{\mathcal{H}} = -\gamma B_0 \hat{I}_z .$$

Equation 1-34

Equation 1-34 suggests that all spins are subjected to an external magnetic field of the same amplitude. This is not exactly the case in reality. Inside a molecule, the local magnetic fields at different nuclear sites are different because of different shielding effects from the surrounding electrons. At a particular location the actual magnetic field is

$$\vec{B}^{loc} = \vec{B}_0 + \delta\vec{B}_0 .$$

Equation 1-35

The magnetic field $\delta\vec{B}_0$ is caused by the surrounding electrons and $\delta\vec{B}_0$ is linearly proportional to $\vec{B}_0$ with proportionality constant δ. This shift of local magnetic field is called chemical shift. The Hamiltonian taking account of the chemical shift becomes (7)

$$\hat{\mathcal{H}} = -\gamma B_0(1+\delta)\hat{I}_z = \omega_0\hat{I}_z, \quad \omega_0 = -\gamma B_0(1+\delta).$$

Equation 1-36

The frequency ω_0 is called the chemically shifted Larmor frequency. The value (1+ δ) is often defined as shielding constant σ = (1+ δ). It is an important quantity in NMR spectroscopy. Later chapters of this thesis contain detailed discussions of lipid proton spins with different chemically shifted Larmor frequencies corresponding to their local molecular environments.

For simplicity, the first example of the quantum mechanical discussion of NMR starts by considering the two equivalent protons in water of spins-½ at the same Larmor frequency ω_0. The Hamiltonian above leads to 2 energy eigenstates that are the two eigenstates for its spin angular momentum with quantum numbers +1/2 and -1/2. The two eigenstates are denoted $|\alpha\rangle$ and $|\beta\rangle$. $|\alpha\rangle$ labels the state where the spin is parallel to the external field while $|\beta\rangle$ labels the anti-parallel state. The eigenequations of $|\alpha\rangle$ and $|\beta\rangle$ are

$$\hat{I}_z|\alpha\rangle = +\tfrac{1}{2}|\alpha\rangle, \ |\alpha\rangle = \left|\tfrac{1}{2},+\tfrac{1}{2}\right\rangle$$
$$\hat{I}_z|\beta\rangle = -\tfrac{1}{2}|\beta\rangle, \ |\beta\rangle = \left|\tfrac{1}{2},-\tfrac{1}{2}\right\rangle.$$

Equation 1-37

The energy eigenvalues are $\pm\frac{1}{2}\omega_0$. The splitting of these two energy states is the Zeeman splitting. A general state $|\psi\rangle$ of a spin-½ is a superposition of its two eigenstates

$$|\psi\rangle = c_\alpha|\alpha\rangle + c_\beta|\beta\rangle \quad \text{or} \quad |\psi\rangle = \begin{pmatrix} c_\alpha \\ c_\beta \end{pmatrix}.$$

Equation 1-38

Coefficients c_α and c_β satisfy $|c_\alpha|^2 + |c_\beta|^2 = 1$. The classical term "transverse magnetization" refers to states that are linear combinations of the two eigenstates such as

$$|+x\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}, |-x\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}-i\\+i\end{pmatrix}, |+y\rangle = \frac{1}{2}\begin{pmatrix}1-i\\1+i\end{pmatrix}, |-y\rangle = \frac{1}{2}\begin{pmatrix}1+i\\1-i\end{pmatrix}.$$

Equation 1-39

In a classical picture, the four states above are all in the *x*-*y* plane. They are related to each other by different phase factors.

An uncoupled spin-½ in a constant magnetic field forms a two level quantum system. The Bloch's sphere is a tool often used to visualize a two level quantum system (37). This method starts by reformulating Equation 1-38.

$$|\psi\rangle = c_\alpha |\alpha\rangle + c_\beta |\beta\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|\alpha\rangle + e^{i\varphi} \sin\frac{\theta}{2}|\beta\rangle \right).$$

Equation 1-40

The first exponential term $e^{i\gamma}$ can be ignored because it has no observable effects (37). The two angular coefficients θ, φ correspond to the direction of a three dimensional vector or the location of a point on an unit sphere. This sphere is the Bloch sphere. Operations on the state vector of this two level system can be described as operations on the point on the Bloch sphere.

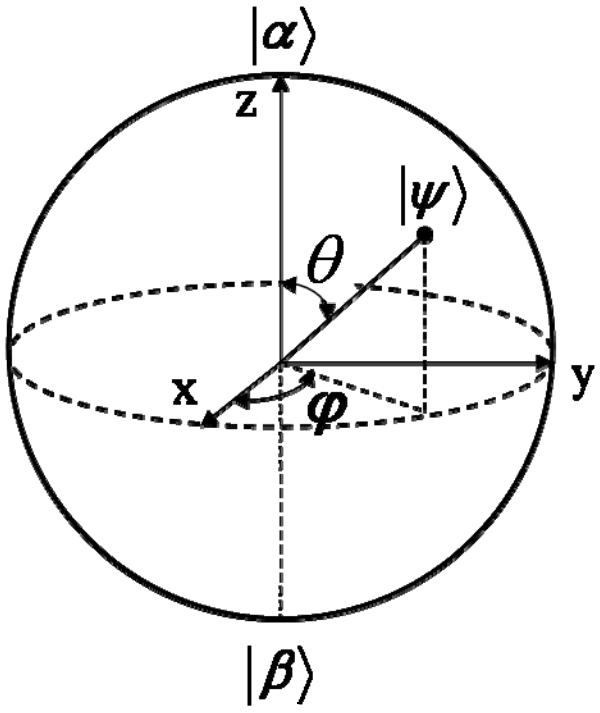

Figure 1-12 The Bloch Sphere

The Bloch sphere is often used to visualized the state of an isolated two level quantum system. Two coefficients c_α and c_β of the two eigenstates are replaced by θ and φ. Any superposition $|\psi\rangle$ of the eigenstates corresponds to a point on the sphere labeled by two angles θ and φ.

In the quantum mechanical treatment of NMR, the application of the Bloch sphere is similar to the magnetization vector model in the classical treatment of NMR. The vector model is the classical basis of understanding the precession of magnetization, RF excitation and spin relaxation. Quantum mechanically, precession is derived with the quantum mechanical equation of motion. RF excitation is formulated as matrix operations on the density matrix in the following sections and relaxation is discussed in terms of transition probabilities in the third section of this chapter.

Only the spin degree of motion is considered in NMR experiments. The quantum mechanical equation of motion of a spin-½ is a time dependent Schrödinger equation:

$$\frac{d}{dt}|\psi(t)\rangle = -i\hat{\mathcal{H}}|\psi\rangle = -i\omega_0\hat{I}_z|\psi(t)\rangle .$$

Equation 1-41

The solution for the spin-1/2 is

$$|\psi(t)\rangle = e^{-i\omega_0 t I_z}|\psi_0\rangle .$$

Equation 1-42

This solution is, in a quantum picture, the equivalent of a "magnetic compass precessing around the direction of $\vec{B}_0$" in the classical picture. The evolution of the energy eigenstates $|\alpha\rangle$ and $|\beta\rangle$ results only in an evolving phase factor multiplied to the same energy eigenstates so the energy eigenstates are stationary. The superposition states such as the ones in Eq.1-39 evolve in the following fashion at the Larmor Frequency

$$|+x\rangle \rightarrow |+y\rangle \rightarrow |-x\rangle \rightarrow |-y\rangle \rightarrow |+x\rangle .$$

The treatment of the rotating frame adopted in the previous section can be applied in the quantum treatment in a similar fashion. However, the concept of density operator discussed in the next section has no classical equivalent for multi level spin systems.

1.3.3 Density Operator

In order to consider the NMR spectroscopic properties of a sample such as a test tube of water, the description of an individual spin-½ in the previous section needs to be extended to a description of a collection of independent and identical spins-½. That constitutes a system called

an ensemble. Any spin in the ensemble can be in a superposition state of its two eigenstates. The superposition state expressed in vector form using the Zeeman eigenstates is

$$|\psi\rangle = \begin{pmatrix} c_\alpha \\ c_\beta \end{pmatrix}.$$

Equation 1-43

The density operator $\hat{\rho}$ of this ensemble is defined as

$$\hat{\rho} = N^{-1}\sum_{i=1}^{N}|\psi_i\rangle\langle\psi_i| = \overline{|\psi\rangle\langle\psi|}.$$

Equation 1-44

N is the number of spins in this ensemble. $|\psi_i\rangle$ is the wave function of the ith spin in the ensemble and the summation is over all spins of the ensemble. The bar over $|\psi\rangle\langle\psi|$ represents an average over the entire ensemble. Therefore, the density operator $\hat{\rho}$ for an ensemble of spins-½ without interaction is

$$\hat{\rho} = \begin{pmatrix} \rho_{\alpha\alpha} & \rho_{\alpha\beta} \\ \rho_{\beta\alpha} & \rho_{\beta\beta} \end{pmatrix} = \begin{pmatrix} \overline{c_\alpha c_\alpha^*} & \overline{c_\alpha c_\beta^*} \\ \overline{c_\beta c_\alpha^*} & \overline{c_\beta c_\beta^*} \end{pmatrix}.$$

Equation 1-45

Two diagonal elements $\rho_{\alpha\alpha}$ and $\rho_{\beta\beta}$ are called the two populations at states $|\alpha\rangle$ and $|\beta\rangle$, respectively, while two off-diagonal elements $\rho_{\alpha\beta}$ and $\rho_{\beta\alpha}$ are called the two coherences between states $|\alpha\rangle$ and $|\beta\rangle$.

At thermal equilibrium, the coherence elements in the ensemble are zero and the two eigenstates are occupied according to the Boltzmann distribution

$$\rho_{rr}^{eq} = \frac{exp\left(-\hbar\omega_r / k_B T\right)}{\sum_s exp\left(-\hbar\omega_s / k_B T\right)} = \frac{exp\left(-\hbar\omega_r / k_B T\right)}{exp\left(-\hbar\omega_\alpha / k_B T\right) + exp\left(-\hbar\omega_\beta / k_B T\right)}. \quad r,s \in \left[|\alpha\rangle, |\beta\rangle\right]$$

Equation 1-46

Eq. 1-46 suggests that the two population states are occupied according to their energy levels and the state with higher energy is occupied less densely than the state with lower energy. Specifically, the Boltzmann distribution of the spins in the ensemble indicates that more spins are aligned parallel to the external field $\vec{B}_0$ than those that are anti-parallel to $\vec{B}_0$, so that the net magnetization is parallel to $\vec{B}_0$. Since the energy values of the two spin states are $+\frac{1}{2}\hbar\omega_0$ and

$-\frac{1}{2}\hbar\omega_0$, Equation 1-46 is evaluated for two energy eigenstates $|\alpha\rangle$ and $|\beta\rangle$ where *rr* takes the values of $\alpha\alpha$ or $\beta\beta$. The exponentials are expanded in a Taylor series and approximated by the first two terms, so that

$$\rho_{\alpha\alpha}^{eq} = \tfrac{1}{2} + \tfrac{1}{4}\frac{\hbar\gamma B_0}{k_B T}$$

$$\rho_{\beta\beta}^{eq} = \tfrac{1}{2} - \tfrac{1}{4}\frac{\hbar\gamma B_0}{k_B T}.$$

Equation 1-47

This approximation is the high-temperature approximation. With all four elements in Eq. 1-45 evaluated, the density operator at thermal equilibrium at temperature *T* is

$$\hat{\rho}^{eq} = \begin{pmatrix} \frac{1}{2} + \frac{1}{4}\mathbb{B} & 0 \\ 0 & \frac{1}{2} - \frac{1}{4}\mathbb{B} \end{pmatrix}, \mathbb{B} = \frac{\hbar\gamma B_0}{k_B T}.$$

Equation 1-48

The symbol $\mathbb{B}$ is defined as the Boltzmann Factor. The density operator at thermal equilibrium describes the spin ensemble at the starting point of an NMR experiment. Now consider the effect of a 90° pulse applied along *x*-axis as an operator $\hat{R}_x\left(\frac{\pi}{2}\right)$. The transformation of the wave function from $|\psi\rangle$ before the 90° pulse to $|\psi'\rangle$ after the 90° pulse is

$$|\psi'\rangle = \hat{R}_x\left(\frac{\pi}{2}\right)|\psi\rangle.$$

Equation 1-49

So the density operator after the 90° pulse is

$$\hat{\rho}' = \overline{|\psi'\rangle\langle\psi'|} = \overline{\hat{R}_x\left(\frac{\pi}{2}\right)|\psi\rangle\langle\psi|\hat{R}_x\left(-\frac{\pi}{2}\right)} = \hat{R}_x\left(\frac{\pi}{2}\right)\hat{\rho}^{eq}\hat{R}_x\left(-\frac{\pi}{2}\right).$$

Equation 1-50

where $\hat{\rho}^{eq} = |\psi\rangle\langle\psi|$ is the density operator before the 90° pulse. From Equation 1-33 the rotation matrix for +90° and -90° pulses around *x*-axis are

$$\hat{R}_x\left(\frac{\pi}{2}\right) = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \\ \frac{-i}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}; \hat{R}_x\left(-\frac{\pi}{2}\right) = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}.$$

Equation 1-51

So the density operator $\hat{\rho}'$ after the 90° pulse is

$$\hat{\rho}' = \tfrac{1}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}\begin{pmatrix} \tfrac{1}{2}+\tfrac{1}{4}\mathbb{B} & 0 \\ 0 & \tfrac{1}{2}-\tfrac{1}{4}\mathbb{B} \end{pmatrix}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} = \begin{pmatrix} \tfrac{1}{2} & +\dfrac{i\mathbb{B}}{4} \\ -\dfrac{i\mathbb{B}}{4} & \tfrac{1}{2} \end{pmatrix}.$$

Equation 1-52

The final matrix expression for the state of the ensemble after the 90° pulse has two equal elements 1/2 for two populations. That means that the 90° pulse rotates the net magnetization into the transverse plane so that the average populations of two energy eigenstates are equal. The two off-diagonal elements in Eq. 1-52 are transformed from 0 to $\pm i\mathbb{B}/4$, which means that coherences are generated by the 90° pulse.

What is shown above is the quantum mechanical formulation of a simple experiment, a 90° rotation. The formulation for a 180° pulse is similar. The rotation matrices for +180° and -180° are

$$\hat{R}_x(\pi) = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix}; \hat{R}_x(-\pi) = \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix}.$$

Equation 1-53

The density operator after the 180° pulse is therefore

$$\hat{\rho}' = \overline{|\psi'\rangle\langle\psi'|} = \hat{R}_x(\pi)\hat{\rho}^{eq}\hat{R}_x(-\pi)$$
$$= \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix}\begin{pmatrix} \tfrac{1}{2}+\tfrac{1}{4}\mathbb{B} & 0 \\ 0 & \tfrac{1}{2}-\tfrac{1}{4}\mathbb{B} \end{pmatrix}\begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix} = \begin{pmatrix} \tfrac{1}{2}-\tfrac{1}{4}\mathbb{B} & 0 \\ 0 & \tfrac{1}{2}+\tfrac{1}{4}\mathbb{B} \end{pmatrix}.$$

Equation 1-54

This experiment represents a phenomenon called population inversion as the two population elements in the density matrix are exchanged. It means that after the 180° pulse the energy eigenstates with higher energy is more densely occupied than the low energy state which is the opposite distribution to that of the equilibrium state.

1.3.4 Homonuclear AX Spin System

NMR spectroscopy distinguishes different spins that resonate at different frequencies. A resulting spectrum is represented as a one dimensional plot with the resonance frequency as the

index and the amplitude of each frequency as the value at each index point. Two spectral dimensions can be generated when the couplings between spins in a molecule are investigated. In this thesis, only weak scalar coupling between two homonuclear spins is considered.

A weakly *J*-coupled homonuclear (AX) spin system[2] exists in a variety of compounds that are relevant in biomedical research (20,38-41). If a molecule consists of two weakly coupled proton spins (AX), their the respective chemically shifted Larmor frequencies in the magnetic field $\vec{B}_0$ are ω_1 and ω_2 for spins A and X, respectively, the spin Hamiltonian can be expressed as

$$\hat{\mathcal{H}} = \omega_1 \hat{I}_{1z} + \omega_2 \hat{I}_{2z} + 2\pi J_{12} \hat{I}_{1z} \hat{I}_{2z},$$

Equation 1-55

where J_{12} is the coupling constant representing the magnitude of the coupling. The eigenstates of the AX system are $|\alpha\alpha\rangle$, $|\alpha\beta\rangle$, $|\beta\alpha\rangle$ and $|\beta\beta\rangle$. Their eigenvalues of this Hamiltonian are (7)

$$\begin{aligned}
\hat{\mathcal{H}}|\alpha\alpha\rangle &= \omega_{\alpha\alpha}|\alpha\alpha\rangle, \quad \omega_{\alpha\alpha} = +\tfrac{1}{2}\omega_1 + \tfrac{1}{2}\omega_2 + \tfrac{1}{2}\pi J_{12} \\
\hat{\mathcal{H}}|\alpha\beta\rangle &= \omega_{\alpha\beta}|\alpha\beta\rangle, \quad \omega_{\alpha\beta} = +\tfrac{1}{2}\omega_1 - \tfrac{1}{2}\omega_2 - \tfrac{1}{2}\pi J_{12} \\
\hat{\mathcal{H}}|\beta\alpha\rangle &= \omega_{\beta\alpha}|\beta\alpha\rangle, \quad \omega_{\beta\alpha} = -\tfrac{1}{2}\omega_1 + \tfrac{1}{2}\omega_2 - \tfrac{1}{2}\pi J_{12} \\
\hat{\mathcal{H}}|\beta\beta\rangle &= \omega_{\beta\beta}|\beta\beta\rangle, \quad \omega_{\beta\beta} = -\tfrac{1}{2}\omega_1 - \tfrac{1}{2}\omega_2 + \tfrac{1}{2}\pi J_{12}.
\end{aligned}$$

Equation 1-56

The four eigenstates of the Hamiltonian form four stationary energy states in the coupled spin system as shown in Figure 1-13.

[2] Two weakly coupled spins are denoted with two letters far away in the alphabet such as AX. AB, on the other hand, denotes two strongly coupled spins.

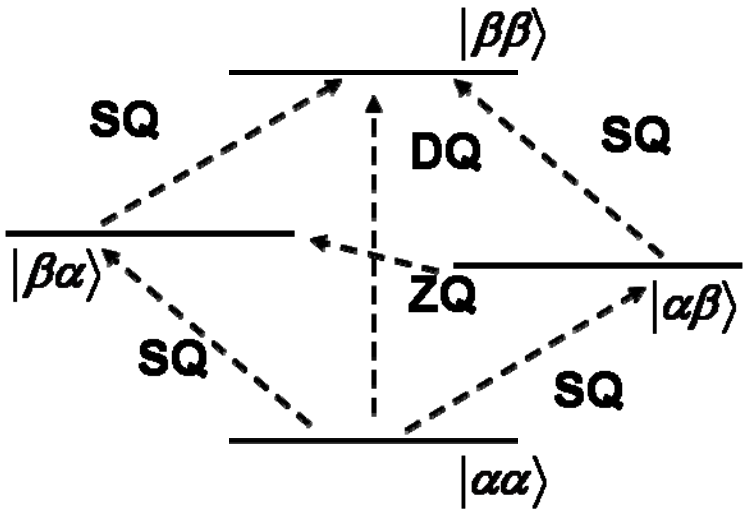

Figure 1-13 Four Energy States in a Homonuclear AX spin system

Four energy eigenstates are formed in a system of two coupled homonuclear spins (AX). Coherences between energy states are represented by dashed arrows. Single quantum coherences (SQ) produce detectable NMR signals. Zero quantum (ZQ) and double quantum (DQ) coherences are not detectable in standard NMR experiments.

The four eigenstates of the Hamiltonian are the Zeeman product states of the coupled spins-½ system. A general expression of the state $|\psi\rangle$ of the two coupled spins is a linear combination of those four eigenstates.

$$|\psi\rangle = c_{\alpha\alpha}|\alpha\alpha\rangle + c_{\alpha\beta}|\alpha\beta\rangle + c_{\beta\alpha}|\beta\alpha\rangle + c_{\beta\beta}|\beta\beta\rangle .$$

Equation 1-57

In an ensemble of AX spin-pairs, the density operator is

$$\begin{aligned}\hat{\rho} &= \overline{|\psi\rangle\langle\psi|} \\ &= \overline{\begin{pmatrix} c_{\alpha\alpha} \\ c_{\alpha\beta} \\ c_{\beta\alpha} \\ c_{\beta\beta} \end{pmatrix} \left(c_{\alpha\alpha}^{*}, \quad c_{\alpha\beta}^{*}, \quad c_{\beta\alpha}^{*}, \quad c_{\beta\beta}^{*}, \right)} \\ &= \begin{pmatrix} \overline{c_{\alpha\alpha}c_{\alpha\alpha}^{*}} & \overline{c_{\alpha\alpha}c_{\alpha\beta}^{*}} & \overline{c_{\alpha\alpha}c_{\beta\alpha}^{*}} & \overline{c_{\alpha\alpha}c_{\beta\beta}^{*}} \\ \overline{c_{\alpha\beta}c_{\alpha\alpha}^{*}} & \overline{c_{\alpha\beta}c_{\alpha\beta}^{*}} & \overline{c_{\alpha\beta}c_{\beta\alpha}^{*}} & \overline{c_{\alpha\beta}c_{\beta\beta}^{*}} \\ \overline{c_{\beta\alpha}c_{\alpha\alpha}^{*}} & \overline{c_{\beta\alpha}c_{\alpha\beta}^{*}} & \overline{c_{\beta\alpha}c_{\beta\alpha}^{*}} & \overline{c_{\beta\alpha}c_{\beta\beta}^{*}} \\ \overline{c_{\beta\beta}c_{\alpha\alpha}^{*}} & \overline{c_{\beta\beta}c_{\alpha\beta}^{*}} & \overline{c_{\beta\beta}c_{\beta\alpha}^{*}} & \overline{c_{\beta\beta}c_{\beta\beta}^{*}} \end{pmatrix},\end{aligned}$$

Equation 1-58

where all averages are calculated to include spins in the entire ensemble. The sixteen elements of the matrix include four population elements (diagonal) and twelve coherence elements (off-diagonal). The coherences are divided into Zero Quantum coherences (ZQ×2) *($c_{\beta\alpha}c_{\alpha\beta}^*$, $c_{\alpha\beta}c_{\beta\alpha}^*$)*, Double Quantum coherences (DQ×2) *($c_{\beta\beta}c_{\alpha\alpha}^*$, $c_{\alpha\alpha}c_{\beta\beta}^*$)* and Single Quantum coherences (SQ×8) in the form of $c_{\alpha\beta}c_{\alpha\alpha}^*$, as shown in Figures 1-13 and 1-14. Coherence $c_{\alpha\beta}c_{\alpha\alpha}^*$ is labeled as α- because it corresponds to a transition from state $|\alpha\beta\rangle$ to $|\alpha\alpha\rangle$ where spin angular momentum on the second spin is decreased.

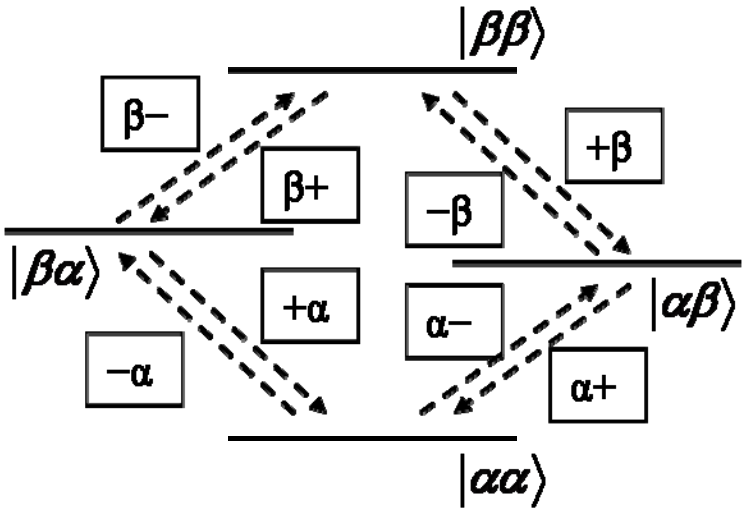

Figure 1-14 Single Quantum Coherences in an AX Spin System

There are 8 Single Quantum (SQ) coherences in an AX spin system. Each is labeled by a symbol in the form of *α*- and each evolves at its own characteristic frequency.

In a reference frame rotating around the $\vec{B}_0$ direction at frequency ω_{ref}, the two spins of an AX system have their respective offset resonance frequencies as

$$\Omega_1 = \omega_1 - \omega_{ref},$$
$$\Omega_2 = \omega_2 - \omega_{ref}.$$

Equation 1-59

where, ω_1 and ω_2 are defined as the chemically shifted Larmor frequencies of spin A and X (Eq. 1-55). The rotating frame Hamiltonian is

$$\hat{\mathcal{H}} = \Omega_1\hat{I}_{1z} + \Omega_2\hat{I}_{2z} + 2\pi J_{12}\hat{I}_{1z}\hat{I}_{2z}.$$

Equation 1-60

The energy eigenvalues in the rotating frame are

$$\Omega_{\alpha\alpha} = +\tfrac{1}{2}\Omega_1 + \tfrac{1}{2}\Omega_2 + \tfrac{1}{2}\pi J_{12}$$
$$\Omega_{\alpha\beta} = +\tfrac{1}{2}\Omega_1 - \tfrac{1}{2}\Omega_2 - \tfrac{1}{2}\pi J_{12}$$
$$\Omega_{\beta\alpha} = -\tfrac{1}{2}\Omega_1 + \tfrac{1}{2}\Omega_2 - \tfrac{1}{2}\pi J_{12}$$
$$\Omega_{\beta\beta} = -\tfrac{1}{2}\Omega_1 - \tfrac{1}{2}\Omega_2 + \tfrac{1}{2}\pi J_{12}.$$

Equation 1-61

Since only the single quantum coherences can produce detectable signals in NMR experiments, we first consider single quantum coherences. The frequency of a signal from a single quantum coherence is proportional to the energy difference between the two corresponding states shown in Fig. 1-14. For example, if the density operator $\rho^0_{+\beta}$ stands for a starting state of single quantum coherence [$+\beta$], its evolution as a function of τ is expressed as

$$\rho_{+\beta}(\tau) = \rho^0_{+\beta}\, exp\left(i\Omega_{+\beta}\tau\right).$$

Equation 1-62

and the characteristic frequency of the evolution is the energy difference of the two corresponding states $\Omega_{\alpha\beta}$ and $\Omega_{\beta\beta}$

$$\Omega_{+\beta} = -\left(\Omega_{\alpha\beta} - \Omega_{\beta\beta}\right) = -\Omega_1 + \pi J_{12}.$$

Equation 1-63

The evolution of Zero Quantum coherences and Double Quantum coherences are important for later discussions in Chapter 2 even though they do not produce directly detectable signals. The characteristic frequencies Ω_{-+} and Ω_{+-} of the two ZQs, and Ω_{--} and Ω_{++} of the two DQs in an AX system are

$$\Omega_{-+} = +\Omega_1 - \Omega_2, (\mathrm{ZQ})$$
$$\Omega_{+-} = -\Omega_1 + \Omega_2, (\mathrm{ZQ})$$
$$\Omega_{--} = +\Omega_1 + \Omega_2, (\mathrm{DQ})$$
$$\Omega_{++} = -\Omega_1 - \Omega_2, (\mathrm{DQ}).$$

Equation 1-64

The frequencies above imply that the 2 Zero Quantum coherences correspond to two offset resonant frequencies Ω_1 and Ω_2 with opposite signs in the rotating frame, while the Double Quantum coherences same signs in the rotating frame. So when ZQs and DQs are in the presence of a gradient field that varies the local Larmor frequency linearly, ZQs do not dephase while DQs dephase at twice the rate of a SQ (42,43).

In lactate a *J*-coupling exists between protons with resonance frequencies at 1.3ppm[3] and 4.2ppm, respectively. In Polyunsaturated Fatty Acids (PUFA) and Monounsaturated Fatty Acids (MUFA), two *J*-couplings exist at (2.8ppm, 5.4ppm) and (2.1ppm, 5.4ppm), respectively. Each *J*-coupling can be represented as a cross peak in a Correlated Spectroscopy (COSY) experiment which is discussed in the next section.

1.3.5 COSY

One of the simplest two-dimensional NMR spectroscopic experiments that shows spin coupling information is the COSY experiment. The pulse sequence consists of two 90° pulses that are strong and short. The bandwidth of the pulses cover two spins equally (Fig. 1-15). The first 90° pulse acting on both spins generates all possible single quantum coherences. These coherences evolve with their individual characteristic frequencies for a period t_1 before experiencing a second 90° pulse. The signal is acquired after the second 90° pulse.

[3] "ppm" stands for "part per million". The resonance frequency of proton spins at 3T is 127.8MHz. Thus, 1ppm in proton spectroscopy corresponds to 127.8Hz.

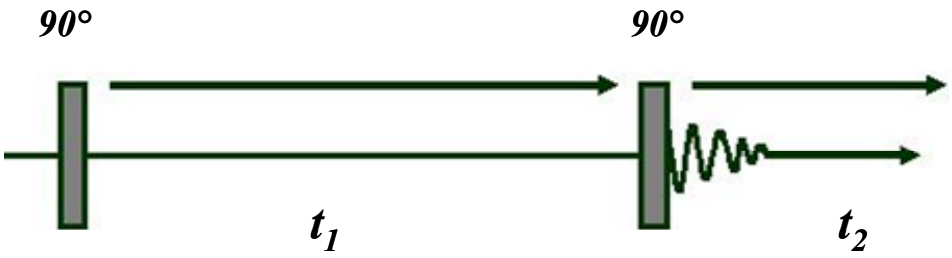

Figure 1-15 COSY Sequence Diagram

A COSY experiment consists of two 90° pulses separated by an incremental delay Δt_1 during t_1 evolution period. The signals are recorded after the second 90° pulse during the t_2 acquisition period.

The state of the spin ensemble after the first 90° pulse is a linear combination of all SQ's

$$\rho^0 = \sum_{SQ} \rho_{SQ},\ \rho_{SQ} \in \left[\rho_{\alpha-}, \rho_{\alpha+}, \rho_{\beta-}, \rho_{\beta+}, \rho_{-\alpha}, \rho_{+\alpha}, \rho_{-\beta}, \rho_{+\beta} \right].$$

Equation 1-65

Since all SQ's are equivalent in this homonuclear AX system, we consider $\rho_{-\alpha}$ as an example. After the first 90° pulse, $\rho_{-\alpha}$ is created and evolves for a period of t_1. The exponential term in Eq. 1-66 represents the evolution at its characteristic frequency $\Omega_{-\alpha}$.

$$\rho_{-\alpha}(t_1) = \tfrac{1}{2i} \exp\left(i\Omega_{-\alpha} t_1\right) \rho_{-\alpha},\quad \Omega_{-\alpha} = -\left(\Omega_{\beta\alpha} - \Omega_{\alpha\alpha}\right) = \Omega_1 + \pi J_{12}.$$

Equation 1-66

The effect of the second 90° pulse on $\rho_{-\alpha}$ can be written as

$$\hat{R}_x(\pi/2)\rho_{-\alpha}(t)\hat{R}_x(-\pi/2).$$

Equation 1-67

In the matrix notation using the four Zeeman eigenbasis, the density operator $\rho_{-\alpha}$ of the SQ is

$$\rho_{-\alpha} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

Equation 1-68

and the rotation matrix for the 90° rotation on the coupled spin system is the dyadic product ("⊗") of two rotation matrices for the two spins A and X

$$\hat{R}_x(\pi/2) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$$

$$= \frac{1}{2}\begin{pmatrix} 1 & -i & -i & -1 \\ -i & 1 & -1 & -i \\ -i & -1 & 1 & -i \\ -1 & -i & -i & 1 \end{pmatrix} \quad \text{and} \quad \hat{R}_x(-\pi/2) = \hat{R}_x(\pi/2)^* .$$

Equation 1-69

The result of the matrix multiplication in Eq. 1-67 is similar to the calculations in previous section for the density operators after the 90° and 180° pulses (Eqs. 1-52, 1-54) so that

$$\rho_{-\alpha}(t_1) = \tfrac{1}{2i}\exp\left(i\Omega_{-\alpha}t\right)\frac{1}{2}\begin{pmatrix} 1 & -i & -i & -1 \\ -i & 1 & -1 & -i \\ -i & -1 & 1 & -i \\ -1 & -i & -i & 1 \end{pmatrix} \times \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \times \frac{1}{2}\begin{pmatrix} 1 & i & i & -1 \\ i & 1 & -1 & i \\ i & -1 & 1 & i \\ -1 & i & i & 1 \end{pmatrix}$$

$$= \tfrac{1}{2i}\exp\left(i\Omega_{-\alpha}t\right)\frac{1}{4}\begin{pmatrix} -i & 1 & 1 & i \\ -1 & -i & i & 1 \\ 1 & i & i & -1 \\ -i & 1 & 1 & i \end{pmatrix}.$$

Equation 1-70

The rotation of Eq.1-70 transforms one matrix element of a single quantum coherence into all 16 elements. The time evolution term $exp(i\Omega_{-\alpha}t_1)$ in Eq.1-66 is carried along with the matrix multiplication containing the characteristic frequency $\Omega_{-\alpha}$ of that particular single quantum coherence [$-\alpha$]. Physically, any one coherence generated by the first 90° pulse is transformed by the second 90° into a combination of all populations and coherences. Each resulting coherence evolves with their characteristic frequency after the second 90° pulse as the t_2 evolution. The t_2 evolution term for coherence [$-\beta$], for example, is $exp(i\Omega_{-\beta}t_2)$. The coherence pathway from [$-\alpha$] to [$-\beta$] results in a signal as the function of t_1 and t_2

$$s(t_1, t_2) = \tfrac{1}{2i}\exp\left(i\Omega_{-\alpha}t_1\right)\exp\left(i\Omega_{-\beta}t_2\right).$$

Equation 1-71

When all single quantum coherences in Eq. 1-65 are taken into account, the resulting coefficients of all populations and coherences are determined by the evolution time t_1, the characteristic frequencies that are dependent on the chemically shifted Larmor frequencies Ω_1 and Ω_2, and the J coupling constant originated in the Hamiltonian $\hat{\mathcal{H}}$ (Eq.1-60).

As the COSY sequence is repeated with incremental t_1 time with a step size Δt, signals along two temporal axes are sampled (Eq. 1-71). A 2D Fourier Transformation of the data results in a 2D spectrum of all the characteristic frequencies. A J coupling between two proton spins at resonance frequencies Ω_1 and Ω_2 are represented by a cross peak in the spectrum at location (Ω_1, Ω_2). Each resonance, i.e. (Ω_1) regardless of coupling also leads to a diagonal peak at (Ω_1, Ω_1), which is the same spectrum obtained in a one-dimensional NMR experiment using one 90° pulse.

1.4 INTRODUCTION TO T_1 AND T_2 RELAXATIONS

There are two types of spin relaxations included in the Bloch equations in Section 1.1.3. Spin-lattice relaxation, also called longitudinal relaxation, refers to the process of spin populations returning to their Boltzmann distribution values. Its time constant is T_1. Spin-spin relaxation, also called transverse relaxation, refers to the process of losing coherence as spins precess about the direction of the external field $\vec{B}_0$. The time constant of this process is T_2. The Bloch equations are a phenomenological description of the relaxation processes. Relaxation processes have been studied with other methods, too. This section introduces some of these microscopic descriptions of the relaxation processes in NMR.

The first method of studying relaxation overviewed in this section is the Random Field Relaxation theory. Autocorrelation functions and spectral density are used to calculate transition probabilities between spin states. The kinetics of spin populations in a two level system is then obtained by evaluating the transition probabilities of uncoupled spins-1/2 (7). T_1 relaxation constant is consequently determined as the result of the Random Field Relaxation theory. Another treatment of relaxation is introduced later in which the dipole-dipole interaction between two spins is taken into account (7,8,34). The time dependent dipole-dipole coupling acts as a perturbation that induces transitions between energy levels in the coupled spin system. This picture provides an insight of the mechanisms of spin relaxations. Both T_1 and T_2 relaxation constants can be expressed in terms of transition probabilities.

1.4.1 Random Field Relaxation

An isolated spin-½ system is subjected to two magnetic fields: a strong uniform field $\vec{B}_0$ along *z* direction that is constant for all spins, and a small random field (e.g. $B_x(t)$ in *x* direction) that is fluctuating in time. Each spin experiences a different fluctuating field $B_x(t)$, but the average time and amplitude of $B_x(t)$ experienced by different spins are equal. The amplitude of $B_x(t)$ is defined as $\sqrt{\langle B_x^2(t)\rangle}$ with the assumption that the mean value of $B_x(t)$, $\langle B_x(t)\rangle = \overline{B_x(t)} = 0$. The time scale of the fluctuation is evaluated using the autocorrelation function $G(\tau)$ of the fluctuating field (7,34)

$$G(\tau) = \langle B_x(t) B_x \langle t+\tau \rangle \rangle \neq 0,$$

Equation 1-72

τ is a time interval between functions $B_x(t)$ and $B_x(t+\tau)$. The autocorrelation function $G(\tau)$ shows a decay curve that characterizes the time scale of $B_x(t)$. If $B_x(t)$ fluctuates rapidly the autocorrelation function $G(\tau)$ decays quickly. If $B_x(t)$ fluctuates slowly, $G(\tau)$ decays slowly.

There are some assumptions for the autocorrelation function. First, $G(\tau)$ is independent of the location of the reference time point *t*, i.e. the fluctuation are stationary in time. Second, $G(\tau)$ is independent of the particular spin it is defined on, i.e. the interaction between neighboring spins is neglected. This greatly simplifies the description of spin relaxation mechanisms like dipole-dipole interaction. One simple model for the autocorrelation function is in the form of an exponential decay with decay constant τ_c

$$G(\tau) = \langle B_x^2 \rangle e^{-|\tau|/\tau_c}.$$

Equation 1-73

Spectral density is an important concept in relaxation theory. It is defined as twice the Fourier Transform of the autocorrelation function.

$$J(\omega) = 2\int_0^\infty d\tau G(\tau) e^{-i\omega\tau}.$$

Equation 1-74

A broad spectral function corresponds to short autocorrelation time and rapid fluctuations while a narrow spectral density function corresponds to long autocorrelation time and slow fluctuations.

Now consider a time-dependent Hamiltonian $\hat{\mathcal{H}}_1(t)$ acting on a spin system, such as the random field $B_x(t)$. $\hat{\mathcal{H}}_1(t)$ can be treated as a small perturbation compared to $\hat{\mathcal{H}}_0$ resulting from $\vec{B}_0$ in a total Hamiltonian $\hat{\mathcal{H}}$ so that

$$\hat{\mathcal{H}} = \hat{\mathcal{H}}_0 + \hat{\mathcal{H}}_1(t).$$

Equation 1-75

The eigenstates of $\hat{\mathcal{H}}_0$ are $|\alpha\rangle$ and $|\beta\rangle$. $\hat{\mathcal{H}}_1(t)$ induces transitions between $|\alpha\rangle$ and $|\beta\rangle$. Using formulations of the perturbation theory (44), the transition probability per unit time $\overline{W_{\alpha\beta}}$ averaged in an ensemble is calculated to be (34)

$$\overline{W_{\alpha\beta}} = \int_0^t \overline{\langle\alpha|\mathcal{H}_1(t')|\beta\rangle\langle\beta|\mathcal{H}_1(t)|\alpha\rangle} e^{i\omega_{\alpha\beta}(t'-t)} dt' + c.c.,$$

Equation 1-76

where $\langle\alpha|\hat{\mathcal{H}}_1(t)|\beta\rangle$ is a random function because it is an element of the random perturbation operator $\hat{\mathcal{H}}_1(t)$. $\omega_{\alpha\beta} = \omega_\alpha - \omega_\beta$, is the Larmor frequency ω_0. Complex conjugate is abbreviated by "*c.c.*". The average under the bar in Eq. 1-76 is consequently the autocorrelation function $G_{\alpha\beta}$ of the random function $\langle\alpha|\hat{\mathcal{H}}_1(t)|\beta\rangle$. Assuming $\langle\alpha|\hat{\mathcal{H}}_1(t)|\beta\rangle$ is stationary, the integral in Eq. 1-76 depends only on the time interval τ between t and t', where $\tau = t - t'$. Therefore, Eq. 1-76 becomes

$$\overline{W_{\alpha\beta}} = \int_{-t}^{t} G_{\alpha\beta}(\tau) e^{-i\omega_{\alpha\beta}\tau} d\tau.$$

Equation 1-77

If we consider $\overline{W_{\alpha\beta}}$ to be much longer than $1/\omega_{\alpha\beta}$ at the end of time t (34), the limits of the integral above can be replaced with $\pm\infty$ so that the transition probability $\overline{W_{\alpha\beta}}$ is the spectral density of the random perturbation evaluated at ω_0

$$W_{\alpha\beta} = J_{\alpha\beta}(\omega_0).$$

Equation 1-78

The average over the ensemble is assumed for all transition probabilities. The details of this treatment can be found in References (8,34).

The transition probability calculation outlined above neglects the effect of thermal equilibrium. In an actual spin ensemble, the two states are occupied according to the Boltzmann distribution of populations so that the state with lower energy is more populated than the higher energy state. With that consideration, the conservation of total spin population in two states requires that

$$\rho_\alpha^{eq} W_- = \rho_\beta^{eq} W_+ .$$

Equation 1-79

The populations of states $|\alpha\rangle$ and $|\beta\rangle$ at equilibrium are ρ_α^{eq} and ρ_β^{eq}, where double scripts used in density matrices $\alpha\alpha$ and $\beta\beta$ are replaced with single scripts in Eq. 1-79 because only populations are considered. W_- represents the transition probability from state $|\alpha\rangle$ to $|\beta\rangle$, in which process angular momentum is decreased through the transition; W_+ is the transition probability from $|\beta\rangle$ to $|\alpha\rangle$, in which process angular momentum is increased. Consequently, $\rho_\alpha^{eq} > \rho_\beta^{eq}$ so $W_- < W_+$. The expressions for W_- and W_+ based upon $W_{\alpha\beta}$ obtained from Eq. 1-78 are

$$W_- = W_{\alpha\beta}\left(1 - \frac{\mathbb{B}}{2}\right)$$
$$W_+ = W_{\alpha\beta}\left(1 + \frac{\mathbb{B}}{2}\right),$$

Equation 1-80

and $\mathbb{B}$ is again the Boltzmann factor

$$\mathbb{B} = \frac{\hbar\gamma B_0}{k_B T}.$$

With the transition probabilities corrected for thermal equilibrium, the kinetics of the spin population ρ_α at state $|\alpha\rangle$ is represented by the following differential equation

$$\frac{d}{dt}\rho_\alpha = -W_-\rho_\alpha + W_+\rho_\beta ,$$

Equation 1-81

where the first term represents the loss of spins from $|\alpha\rangle$ to $|\beta\rangle$ and the second term represents the gain of spins from $|\beta\rangle$ to $|\alpha\rangle$. Similarly, the differential equation for spin population ρ_β of $|\beta\rangle$ is

$$\frac{d}{dt}\rho_\beta = +W_-\rho_\alpha - W_+\rho_\beta .$$

Equation 1-82

The z component of the magnetization M_z is proportional to the difference of the spin populations at two states

$$M_z = 2\mathbb{B}^{-1}\left(\rho_\alpha - \rho_\beta\right).$$

Equation 1-83

The differential equation of M_z is therefore

$$\begin{aligned}\frac{d}{dt}M_Z &= 2\mathbb{B}^{-1}\left(\frac{d}{dt}\rho_\alpha - \frac{d}{dt}\rho_\beta\right)\\ &= 4\mathbb{B}^{-1}\left(-W_-\rho_\alpha + W_+\rho_\beta\right)\\ &= -2W_{\alpha\beta}\left(M_z - 1\right).\end{aligned}$$

Equation 1-84

By comparing above the result with the Bloch equation for z component of the magnetization $dM_z/dt = 1/T_1\left(M_0 - M_z\right)$, we obtain

$$T_1 = \frac{1}{2W_{\alpha\beta}}.$$

Equation 1-85

The simple relation above suggests that the rate of spin-lattice relaxation is a direct result of the transition probability and hence the spectral density of the random perturbation at Larmor frequency ω_0, because $W_{\alpha\beta} = J_{\alpha\beta}\left(\omega_0\right)$. The semi-classical treatment of spin-lattice relaxation can be derived directly from Fermi's golden rule in quantum mechanics (44).

1.4.2 Dipole-Dipole Relaxation

The random field treatment of relaxation is only a first attempt at understanding the microscopic processes of relaxation. It treats all sources of relaxation as an external perturbation to an isolated spin. Any further discussion of relaxation including an introduction of the spin-spin relaxation constant T_2 will be obtained by taking account of the actual mechanisms of relaxation. For spins-½, the dominant relaxation mechanism is the dipole-dipole interaction

between two spins (7,45), including intra-molecular interaction when the dipoles are in the same molecule and inter-molecular interaction when dipoles are in different molecules. Inter-molecular interaction can be neglected if large molecules are considered such as proteins. In the cases of water relaxation, both inter and intra molecular interactions are present and the magnetic interactions between two dipoles are modulated by the random tumbling motions of the molecules. A thorough treatment of relaxation by dipole-dipole interaction was described by McConnell (45). The first important result of the treatment is that the spectral density of the random thermal motion of the two spin system $J(\omega)$ is expressed in terms of spherical harmonic functions Y_{20} (46)

$$J(\omega)=\frac{3\pi}{5}\gamma^4\hbar^2\int_{-\infty}^{\infty}\left\langle\frac{Y_{20}(\theta(0),\phi(0))}{r(0)^3}\frac{Y_{20}(\theta(t),\phi(t))}{r(t)^3}\right\rangle e^{-i\omega t}dt,$$

Equation 1-86

where $r(t)$ is the distance between spin I and spin S, whose dipole interaction is considered. Y_{20} is a spherical angular function of $\theta(t)$ and $\phi(t)$. It is determined by the relative orientation or $r(t)$ between I and S. As the treatment is specifically applied to water molecules, the intra-molecular spin interaction between the two proton spins are then explicitly evaluated using their Cartesian components: I_x, I_y and I_z for spin I and S_x, S_y and S_z for spin S. Two differential equations are obtained for the longitudinal component $< I_z+S_z>$ and transverse component $< I_x+S_x>$ of the combined two spin system

$$\frac{d\langle I_z+S_z\rangle}{dt}=-I(I+1)\left[\tfrac{4}{3}J(\omega_0)+\tfrac{16}{3}J(2\omega_0)\right]\langle I_z+S_z\rangle,$$
$$\frac{d\langle I_x+S_x\rangle}{dt}=-I(I+1)\left[2J(0)+\tfrac{10}{3}J(\omega_0)+\tfrac{4}{3}J(2\omega_0)\right]\langle I_x+S_x\rangle.$$

Equation 1-87

The relaxation constants T_1 and T_2 for the mechanism of dipole-dipole relaxation are consequently determined to be

$$\frac{1}{T_1}=I(I+1)\left[\tfrac{4}{3}J(\omega_0)+\tfrac{16}{3}J(2\omega_0)\right],$$
$$\frac{1}{T_2}=I(I+1)\left[2J(0)+\tfrac{10}{3}J(\omega_0)+\tfrac{4}{3}J(2\omega_0)\right].$$

Equation 1-88

Another mechanism causing spin relaxation is the Chemical Shift Anisotropy (CSA) (45). As previously mentioned in the discussion of the spin Hamiltonian, the actual magnetic field experienced by a spin in a molecule is shifted from the value of the external field $\vec{B}_0$ because of the shielding effects of the surrounding electrons. This shielding effect depends on the orientation of the molecule with respect to the direction of $\vec{B}_0$. So as molecules tumble randomly, the magnitude of the Chemical Shift varies randomly as well. CSA relaxation is only comparable to dipole-dipole relaxation in magnetic fields higher than 3T (7).

2.0 *IN VIVO* 2D MULTIPLE QUANTUM COHERENCE TRANSFER SPECTROSCOPY

The Correlated Spectroscopy (COSY) sequence introduced in the first chapter is a good example of two dimensional Magnetic Resonance Spectroscopy (MRS). Two variations of the COSY sequence, Localized COSY and Double Quantum Filtered COSY are discussed in this chapter as an introduction to implementing 2D MRS *in vivo*. An advanced and specialized 2D spectroscopic sequence, Selective Multiple Quantum Coherence Transfer (Sel-MQC) (29), is the primary subject of this chapter.

2.1 LOCALIZED COSY

The Correlated Spectroscopy (COSY) sequence is an important technical tool in chemistry and material science. A localized variation of COSY (L-COSY) was developed by Thomas et. al. (20,41) (Fig. 2-1). The 2D COSY spectrum can be acquired by a pulse train of 90°, 180° and 90° pulses that are slice-selective along three orthogonal planes. The final spectrum comes from the intersection of 3 planes that define a small volume. The spin echo formed by the 90° pulse and the 180° pulse is equivalent to the first 90° pulse in COSY. The delay between that echo and the second 90° pulse is incremental to provide sampling along t_1 direction. This method can be applied *in vivo* to acquired a 2D spectrum from a selected volume (20).

The L-COSY spectrum displays all spectral peaks without editing as in the Sel-MQC sequence discussed later in this chapter. Therefore, the L-COSY spectrum may serve as a reference for the Sel-MQC sequence in assigning spectral peaks according to their resonant frequencies. The L-COSY sequence was implemented on the GE 3T platform.

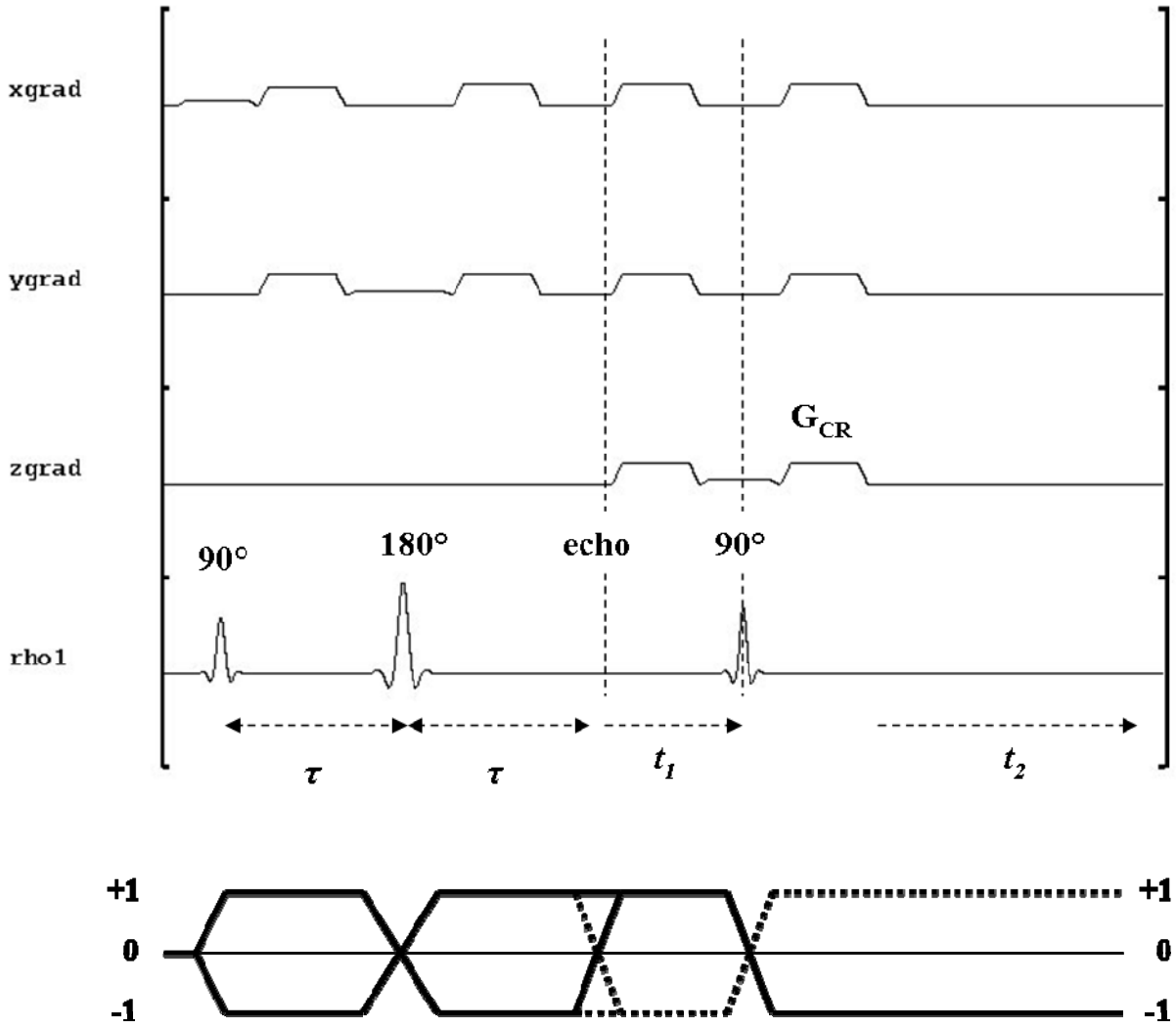

Figure 2-1 L-COSY Sequence Diagram and Coherence Transfer Pathways

L-COSY sequence was implemented on the GE 3T platform. The 90° and 180° pulses make the signal to form a spin echo. τ=10.9ms, which is the minimal value as determined by the gradients. The second 90° pulse follows the echo with an incremental delay t_1. Gradient crushers G_{CR} are placed around the pulses to remove leakage signals. t_1 is incremented in 64 steps of 1ms. 64 FIDs are acquired during t_2 periods (20,41).

Figure 2-1 is a diagram of the L-COSY sequence. After the first 90° pulse and the following 180° pulse, the spins form a spin echo. The second 90° pulse and its coherence selection gradients are placed after the spin echo with a variable delay t_1. The length of the delay increases in 64 increments of 1ms. An FID is recorded with 2048 data points for every increment. The discrete increments and the FID acquisition provide data sampling along t_1 and t_2

directions. A 2D FFT produces a spectrum in two frequency directions. The L-COSY results will be included as references for results of the Sel-MQC sequence.

2.2 DOUBLE QUANTUM FILTERED COSY

The Localized COSY sequence (L-COSY) and other spectroscopic sequences such as Point Resolved Spectroscopy (PRESS) (47) or Stimulated Echo Spectroscopy (STEAM) (48,49) are often combined with Chemical Shift Selective (CHESS) (28) pulses in *in vivo* applications to suppress unwanted water or lipid signals. The inclusion of CHESS pulses in MRS often results in complicated sequence designs and poor suppression performance. When the L-COSY sequence is combined with CHESS water suppression pulses for *in vivo* implementations, the spectral peaks resulting from *J*-coupled resonances can be distinguished but are overwhelmed by residual water/fat signals. Double Quantum Filtered Correlated Spectroscopy (DQF-COSY) (50,51) sequence is improved upon the original COSY sequence in water suppression performance (39). The DQF-COSY sequence bridges the gap between the COSY sequence discussed in the previous section and Sel-MQC in the rest of this chapter. Both DQF-COSY and Sel-MQC use a homonuclear *J*-coupling as the basis for water suppression. Their difference is that DQF-COSY sequence excites signals from all *J*-couplings in the sample to produce a 2D COSY spectrum with inherent water suppression. The Sel-MQC sequence, on the other hand, selects one particular *J*-coupling for spectral editing to suppress both water and unwanted fat signals. The discussion of the DQF-COSY sequence also introduces the concept of using pulsed gradients for coherence order selection, which is crucial to the understanding of the Sel-MQC experiments.

2.2.1 Sequence Design of DQF-COSY

The first 90° pulse of the COSY or L-COSY sequence excites a combination of all Single Quantum (SQ) Coherences. The second 90° pulse following a t_1 evolution period transforms any SQ into a combination of all coherences and populations. This procedure excites signals from all

the proton spins in the sample with or without *J*-coupling. The DQF-COSY sequence includes a third 90° pulse and three gradient pulses to select signals from *J*-coupled protons that form Double Quantum Coherence (50,51). Other coherences are dephased by the coherence selection gradient pulses. A molecule not containing any *J*-coupled spins, such as water, will not generate DQ and ZQ after the second 90° pulse and its signals will be dephased by the coherence selection gradients.

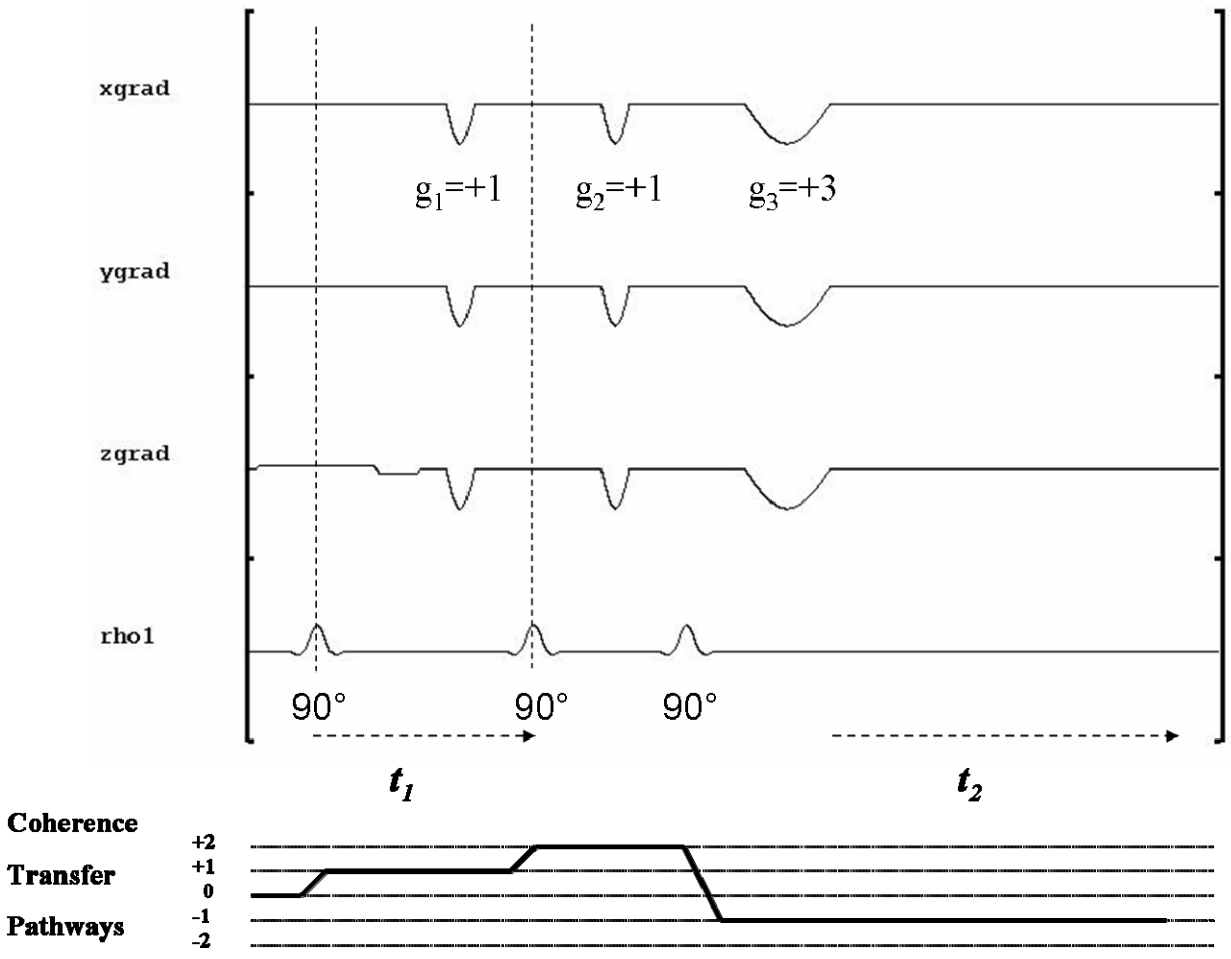

Figure 2-2 Sequence Diagram of Double Quantum Filtered COSY (DQF-COSY) and Coherence Transfer Pathways (CTP)

The Double Quantum Filtered COSY (DQF-COSY) sequence consists of three 90° pulses and three coherence selection gradients. The ratio of their gradient areas is (1: 1: 3). The coherence pathway is plotted at the bottom, which is [0→ +1→ +2 →-1] (50,52).

Figure 2-2 shows the sequence diagram of the DQF-COSY sequence implemented on both GE 1.5T and 3T platforms. The first 90° pulse and its companion slice selection and

refocusing gradients select a slice in the z direction. The incremental delay between the first and second 90° pulses is the t_1 period. The first coherence selection gradient g_1 dephases all Single Quantum (SQ) coherences. The second 90° pulse transforms all SQ into a combination of all coherences including ZQ and DQ in a *J*-coupled molecule. The DQ are dephased by the second gradient pulse g_2 at twice the precessing rate. The last 90° pulse transforms DQ coherences into another combination of coherences and populations. The SQ coherences are dephased or refocused by the last gradient pulse g_3, which is three times the width of g_1 or g_2. Among all possible coherence pathways generated by the three 90° pulses, only the one from [0→ +1→ +2 →-1] is selected by the 3 gradient pulses because $+1\times g_1+2\times g_2-1\times g_3=0$, given the ratio of $g_1:g_2:g_3=1:1:3$. This coherence transfer pathway (CTP) is plotted in Figure 2-2.

2.2.2 Comparison of L-COSY and DQF-COSY In Phantom Studies

The DQF-COSY and L-COSY sequences were compared in a phantom study. The phantom was a test tube of solution with mixed 100mM of lactate, glucose and glutamate. The 2D spectra of both sequences were summed along the second dimension to show two 1D spectral plots (Figure 2-3 a and b). The spectrum acquired by L-COSY sequence (Fig. 2-3a) shows intense residual water signal and other diminished spectral peaks. The spectrum acquired by the DQF-COSY sequence (Fig. 2-3b) shows residual water peak at 4.7ppm that barely overlapped with the other peaks. Double Quantum Filtered glucose signals are visible. In an *ex vivo* study, the same experiment was also performed on fresh pork to simulate chemical concentrations *in vivo*. The spectrum by DQF-COSY (Fig. 2-3c) shows clean diagonal peaks and cross correlation peaks with improved water suppression.

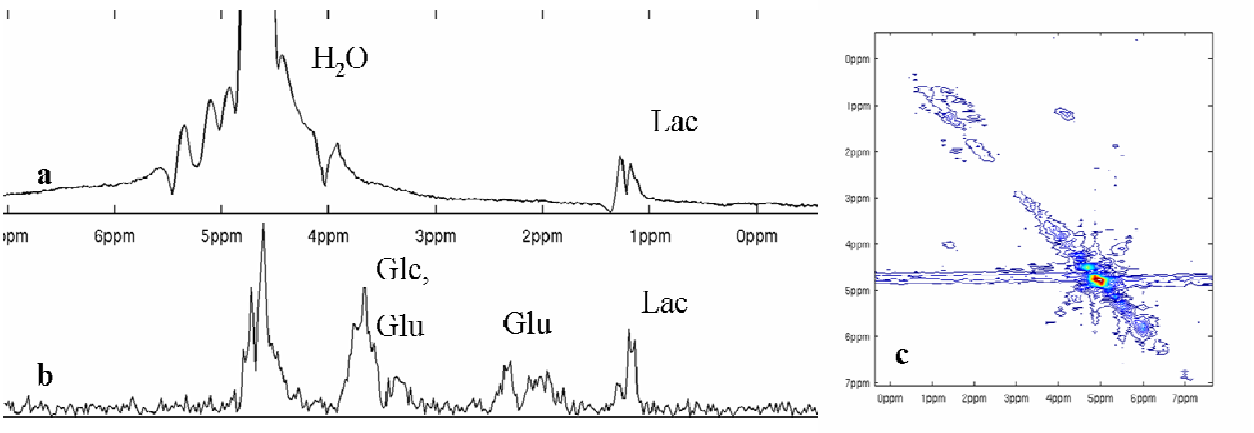

Figure 2-3 Comparison of Water Suppression Performance of L-COSY and DQF-COSY

(a) A spectrum acquired by L-COSY sequence with intense residual water signal. (b) A spectrum acquired by DQF-COSY sequence shows clear spectral peaks of lactate, glucose and two peaks of glutamate. (c) A 2D spectrum by DQF-COSY (fresh pork) shows diagonal peaks and unknown cross correlation peaks with improved water suppression. (The Matlab codes for this figure are included in Appendix C.1)

The comparison of L-COSY and DQF-COSY sequences shows the clear advantage of using multiple quantum coherence filtering techniques to improve water suppression performance in *in vivo* spectroscopy. The Selective Multiple Quantum Coherence Transfer technique discussed in the next section uses frequency selective pulses to enable the selective detection of a specific compound like Polyunsaturated Fatty Acids (PUFA), with excellent suppression of water and unwanted lipid signals.

2.3 2D SPECTROSCOPY WITH SELECTIVE MULTIPLE QUANTUM COHERENCE TRANSFER (SEL-MQC)

2.3.1 Theory

Selective Multiple Quantum Coherence Transfer (Sel-MQC) technique (29) relies on a *J*-Coupling between two protons to generate multiple quantum states of two spins. The coupling term $\hat{I}_{1z}\hat{I}_{2z}$ in Eq. 1-55 satisfies the following commutations with operators $\hat{I}_{1x}$ and $\hat{I}_{2x}$

$$\left[2\hat{I}_{1z}\hat{I}_{2z},\hat{I}_{1x}\right]=i2\hat{I}_{1y}\hat{I}_{2z}$$
$$\left[2\hat{I}_{1z}\hat{I}_{2z},\hat{I}_{2x}\right]=i2\hat{I}_{1z}\hat{I}_{2y}$$

Equation 2-1

The commutations suggest the three operators $\hat{I}_{1z}\hat{I}_{2z}$, $\hat{I}_{1x}$ and $\hat{I}_{1y}\hat{I}_{2z}$ form a cyclic relation similar to that of the three orthogonal angular momentum components. A cyclic relation among those three operators means that a single spin state I_{1x} evolving under the influence of the *J*-coupling will rotate in a plane spanned by $(I_{1y}I_{2z}, I_{1x})$, as shown in Fig. 2-4. An operator $U_{12}(\tau)$ is used to denote this rotation with an angle τ. The rotation represents a periodic creation and destruction of a correlated 2-spin state:

$$U_{12}(\tau)\hat{I}_{1x}U_{12}(\tau)^{-1}=\hat{I}_{1x}\cos\pi J_{12}\tau+2\hat{I}_{1y}\hat{I}_{2z}\sin\pi J_{12}\tau$$

Equation 2-2

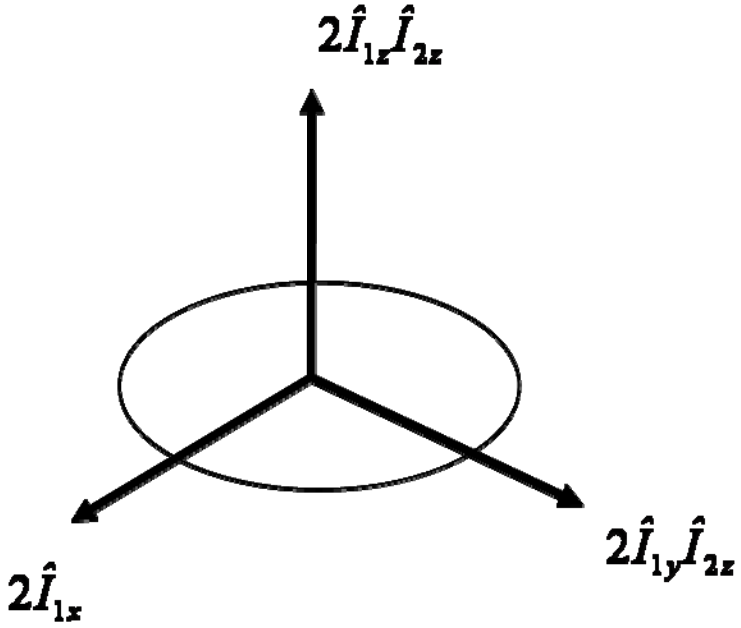

Figure 2-4 Cyclic Relation of *J*-coupled Protons
3 operators $I_{1z}I_{2z}$, I_{1x} and $I_{1y}I_{2z}$ form a cyclic relation.

When the rotation angle is equal to $\pi/2$, I_{1x} state evolves into a correlated state $I_{1y}I_{2z}$. The angle condition determines the desired evolution time: $\tau=1/(2J)$. The value of *J* for lactate and PUFA is 7.4Hz from experimentally measured peak splitting in NMR experiments, and τ = 68ms for *J*-couplings in lactate and lipid signals like PUFA.

2.3.2 Sequence Design

The Sel-MQC sequence was developed by He et al, (29) and was implemented for this study by modifying GE's standard STEAM sequence. The diagram below (NOT to scale) shows all components of this sequence.

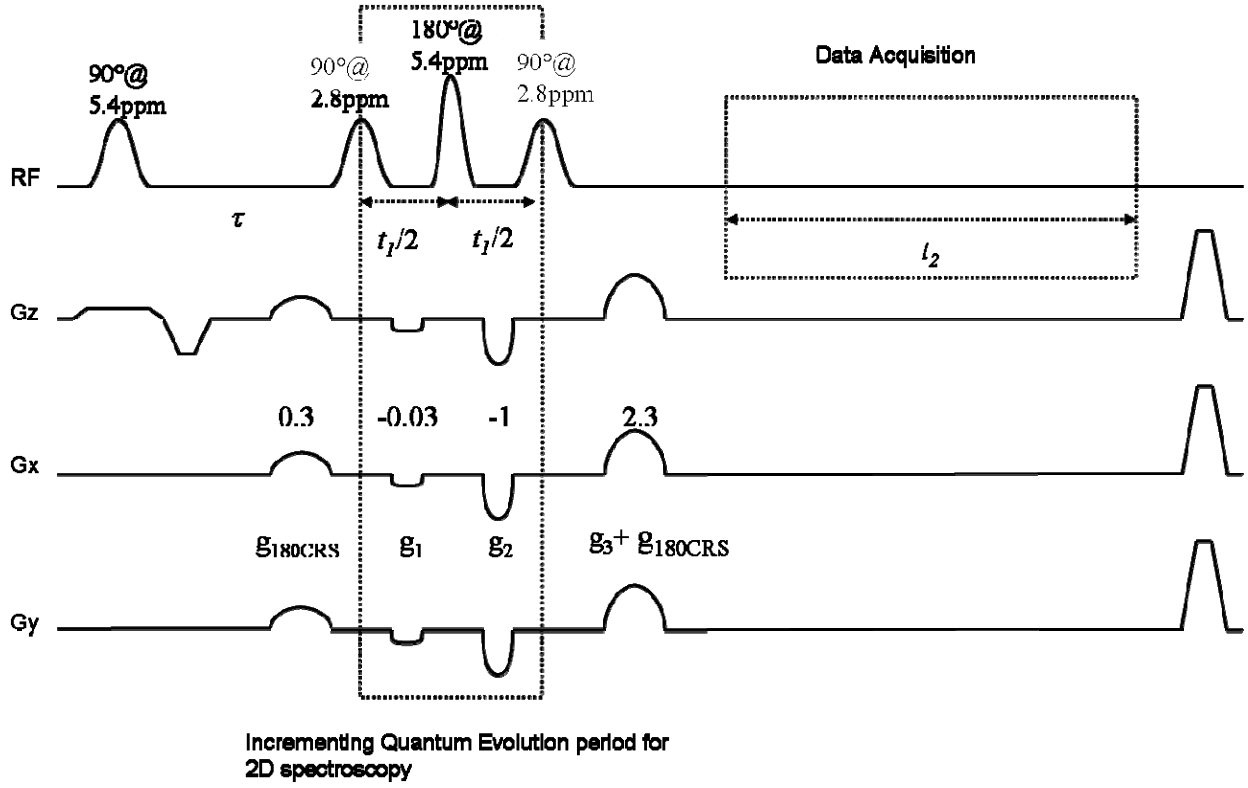

Figure 2-5 Diagram of Sel-MQC 2D Spectroscopy Sequence

The first 90° pulse with slice selection excites the entire proton spectrum. The second and the last 90° pulses are set on the resonance frequency of I_2. The 180° pulse is set on the resonance frequency of I_1. The multiple quantum evolution period between the second and the last 90° pulses is incremented to form a discrete sampling along the t_1 direction. Continuous data acquisition period forms the data sampling along t_2 direction.

2.3.2.1 Excitation

A 180º pulse was inserted between the second and the third 90º pulses in the evolution section of Sel-MQC. The pulse widths of all 90º pulses were extended to 11.7ms and the 180º pulse width was extended to 6.3ms. (The 180º pulse in the Spiral Sel-MQC imaging sequences was extended to 11.7ms for better frequency selectivity.) All pulses were 3-lobe SINC pulses provided by the

GE scanner platform. The first 90° was slice selective and it was set to excite the spins in one slice across the entire proton spectrum. The 180° pulse was set on the frequency of I_1 resonance of 5.4ppm for PUFA and the second and last 90° pulses were set on I_2 resonance of 2.8ppm for PUFA.

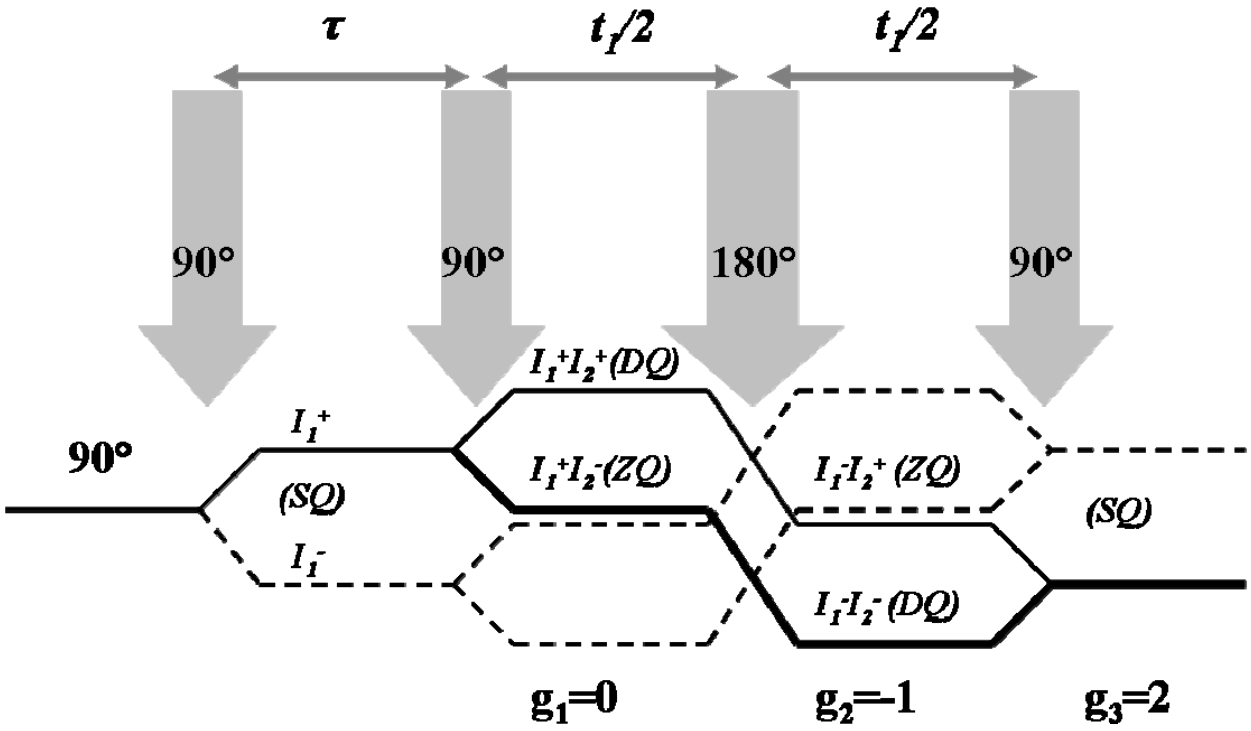

Figure 2-6 Sel-MQC Coherence Pathways

One of two multiple quantum coherence pathways is selected with 3 gradient pulses of area ratio (0: -1: 2). The pathway is zero quantum coherence to double quantum coherence (ZQ→DQ) during the multiple quantum evolution period.

The first 90° pulse creates a single spin state I_{1x}. After a J-evolution time of τ=68ms the single spin state evolves into a correlated spin state $I_{1y}I_{2z}$. The second 90° pulse creates transverse magnetization on I_2 so that the correlated state becomes a superposition of zero-quantum coherence (ZQ) and double-quantum (DQ) coherence. The 180° degree pulse on I_1 interchanges ZQ and DQ coherences. Since only DQ coherences are affected by a dephasing gradient, a gradient pulse before or after the 180° pulse (g_1 or g_2) can dephase the DQ in the superposition of ZQ and DQ. The last 90° pulse returns ZQ or DQ back to SQ. A gradient pulse g_3 after the last 90° pulse refocuses the SQ magnetization that has been dephased. The combination of g_1, g_2 and g_3 can label and select two coexisting coherence pathways. The three possible ratios of the

gradient pulses are: (0: -1: 2) for ZQ→DQ, (-1: 0: 2) for DQ→ZQ and (-1: -1: 2) for selecting both pathways. In the current application the pathway chosen was ZQ→DQ with gradient values at (0: -1: 2). The experimental value of g_1 was set to be non-zero but very small (ε<10% of g_2) as a t_1 crusher.

In a typical experiment, all 90° and 180° pulses include error components, i.e. some spins are not rotated exactly the desired flip angles. These error components are eliminated by extra gradient crushers before and after the 180° degree pulse so that only the magnetization experienced an accurate 180° flip could survive that operation. The pre-180° gradient component is located before the second 90° pulse as shown in Fig. 2-5 and the component of the post-180° gradient is combined into g_3. Their values were 30% of g_1. In Sel-MQC, a total of four crusher gradient pulses are used and their relative values are (+0.3, 0+ε, -1, 2+0.3), as shown in Fig. 2-5.

The diagram below illustrates 2 possible echo positions in the Sel-MQC experiments.

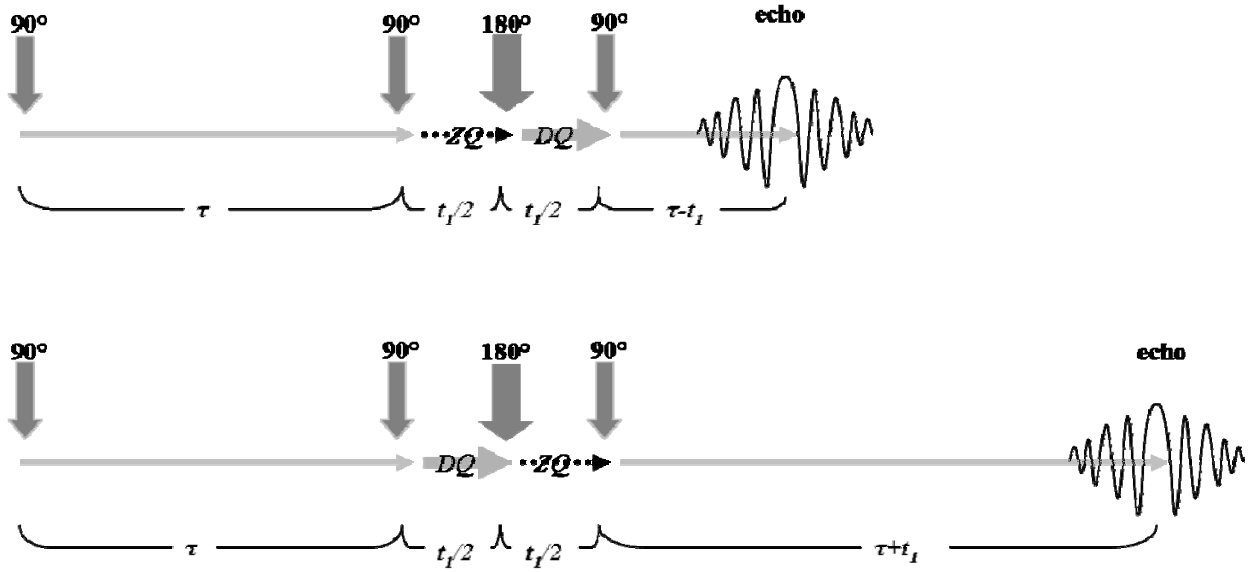

Figure 2-7 Diagrams of Two Multiple Quantum Echoes for Two Pathways
Two coherence pathways result in two echoes formed at different locations.

When ZQ to DQ pathway is selected during t_1, the last 90° pulse changes DQ to SQ and the SQ evolves under J-coupling for a period (τ-t_1) and transfers magnetization back to I_1. The other pathway, DQ to ZQ, would lead to an echo at location (τ+t_1). The echo shifts shown in Fig.2-7 reflect the separation of two echoes for two coherence pathways. When the ZQ→DQ pathway is labeled by gradient g_2 in the second half of t_1, the selection is based on the notion introduced in

the previous section that DQ could be dephased by a gradient pulse at twice the rate and ZQ is not affected. When two spins correlated in DQ mode, they can be dephased and refocused by g_3 of twice the area of g_2 (42,43). This procedure can select DQ when DQ and ZQ coexist during the quantum evolution period.

Along the pathway of ZQ→DQ, the gradient and inhomogeneity dephasing before and after the 180° pulse satisfies $\tau = 2\times(t_1/2)+(\tau-t_1)$. The $2\times(t_1/2)$ term comes from twice the dephasing during DQ after the 180° pulse. An echo for the ZQ→DQ pathway is formed at $(\tau-t_1)$ from the end of the multiple quantum evolution period. Along the pathway of DQ→ZQ, the gradient and inhomogeneity dephasing satisfies $\tau+2\times(t_1/2)=(\tau+t_1)$ because the $2\times(t_1/2)$ term comes from twice the dephasing during DQ before the 180° pulse. The echo along the DQ→ZQ pathway is formed at $(\tau+t_1)$ from the end of the multiple quantum evolution period. In summary, the multiple quantum evolution period splits the total magnetization in two halves and effectively forms two spin echoes from the end of the t_1 period at $(\tau-t_1)$ and $(\tau+t_1)$. In the current application the pathway leading to the shorter echo time was used for data acquisition. The maximum theoretical efficiency of the Sel-MQC pathway selection is 50%.

2.3.2.2 Time Evolution

The 2D Sel-MQC sequence is different from the imaging sequences discussed in Chapters 3 and 4 because of the time evolution period in the sequence. The 2D Sel-MQC sequence consists of a series of pulses that are separated by a variable interval t_1. The 180° pulse divides t_1 into two halves. The minimum of t_1, 24ms, is determined by the pulse widths of 90° and 180° and the durations of g_1 and g_2. Two delays are inserted into two halves of t_1 with a minimum width of 4μs, which is negligible compared to the values of t_1. The durations of the t_1 delays are defined as variables that can be updated in real time after the sequence waveforms are generated. Each half of t_1 is increased with a step size of 250μs for 64 steps. The total delay is increased up to (250μs+250μs)×64=32ms. The maximum t_1 is 24ms+32ms=56ms. TR is set at 2 seconds for all 64 iterations. The 64 repetitions with incrementing t_1 generated discrete sampling points along t_1 direction.

2.3.2.3 Data Acquisition

The FIDs acquired following the last gradient g_3 constitute data acquisition period along the second temporal dimension. The receiver samples one data point every 4μs. The decimation rate is 8, i.e. 8 points acquired in 32μs are recorded and summed in the memory as one data point. Each echo is 2048 points long and the active time of the receiver is 2048×8×4μs=65.5ms.

2.3.3 Phantom Experiments and Sequence Optimization

The 2D Sel-MQC sequence and the L-COSY sequence were tested in phantom studies together to compare edited and un-edited 2D spectra. Lipid spectrum contains a rich collection of peaks thus it is a better choice than the lactate spectrum for phantom studies. There are two separate but similar *J*-coupling pairs of interest: Polyunsaturated Fatty Acids (PUFA) at (2.8ppm, 5.4ppm) and Monounsaturated Fatty Acids (MUFA) at (2.1ppm, 5.4ppm). PUFA and MUFA both contain olefinic protons in a double bond at 5.4ppm (-C**H**=C**H**-), where each double bond is considered an unsaturated site. PUFA refers to a configuration where diallylic methylene protons are adjacent to at least two olefinic protons (=C**H**-CH_2-C**H**=), with diallylic methylene protons at 2.8ppm. MUFA refers to a configuration where allylic methylene protons are adjacent to only one olefinic proton (-CH_2-CH_2-C**H**=), with allylic methylene protons at 2.1ppm. Olefinic protons and methylene protons are coupled with *J* coupling in either configuration. Both PUFA and MUFA are abundant in many kinds of vegetable oils including soybean oil, canola oil and olive oil. The sequences were first tested using a bottle of soybean oil to evaluate the performance of the selective excitation and editing.

Figure 2-8 Soybean Oil Label with Lipid Compositions

In one serving of soybean oil with 14g of fat, 9g is PUFA and 3g is MUFA. The rest is saturated fat.

L-COSY sequence was implemented to acquired an un-edited 2D spectrum (Fig. 2-9). The resulting 2D spectrum below shows prominent diagonal peaks along the line of 1.3ppm and 5.4ppm. Cross peaks are shown at (2.1ppm, 5.4ppm) and (2.8ppm, 5.4ppm).

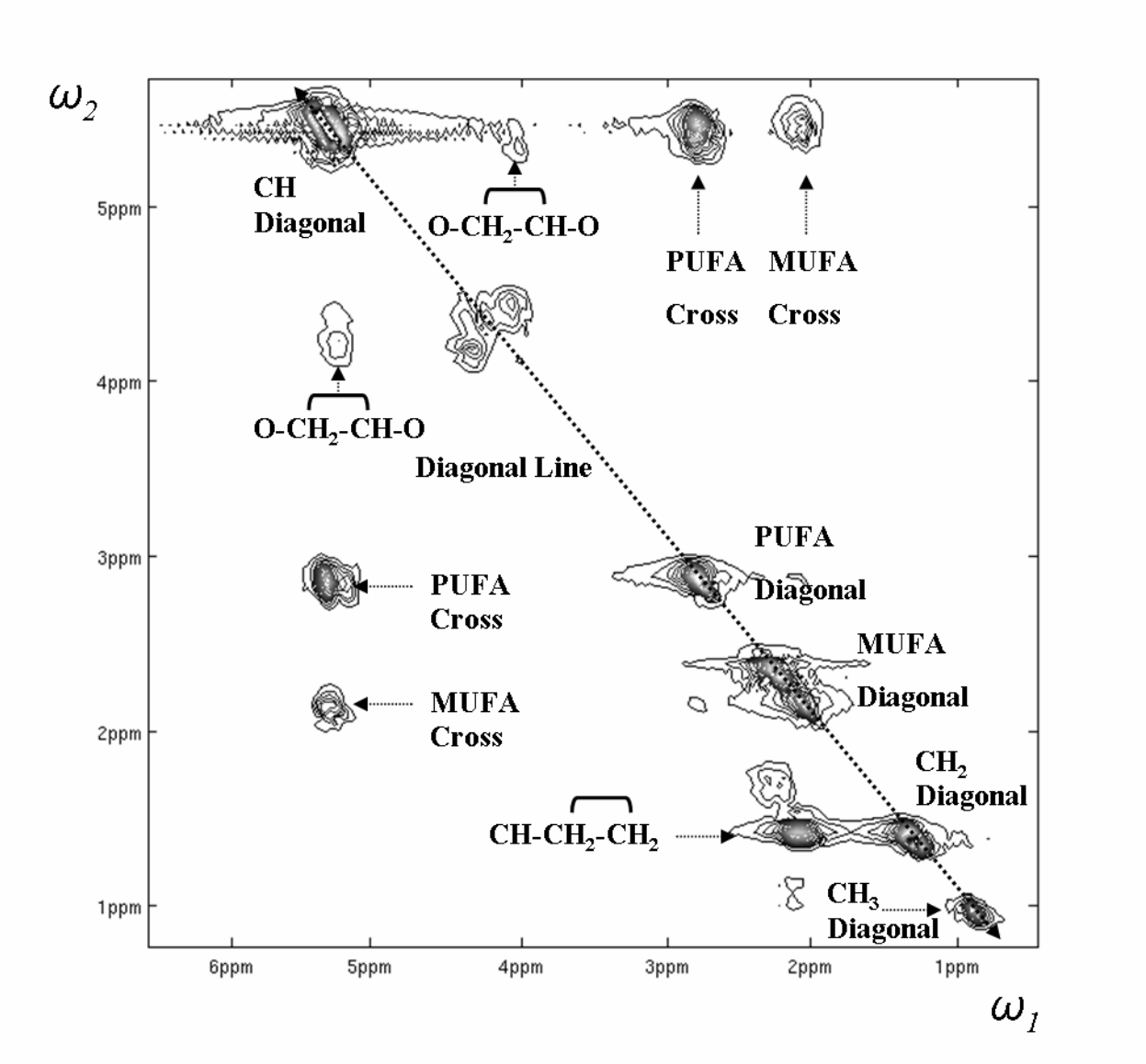

Figure 2-9 An L-COSY Spectrum of the Soybean Oil

Two cross peaks of PUFA and MUFA are displayed in the 2D spectrum acquired by L-COSY. ω_1 is the evolution axis and ω_2 is the chemical shift (readout) axis. The typical lipid diagonal peaks at 0.9ppm, 1.3ppm, 2.1ppm, 2.8ppm and 5.4ppm are shown along the dotted diagonal line. All peaks are assigned according to Ref. (21). (Matlab codes for this figure are included in Appendix. C-3.)

A Sel-MQC 2D spectrum (Fig. 2-10) was then acquired from the same oil phantom with all pulses set on their respective resonance frequencies. Two cross peaks of PUFA and MUFA at (5.4ppm, 2.8ppm) and (5.4ppm, 2.1ppm) are shown clearly in Fig. 2-10 with a number of peaks from residual signals. The intensities of residual peaks in Fig. 2-10 appear to be different from those in the L-COSY spectrum because suppression effects for the residue peaks are different

due to their different spectral locations. Suppression of other unwanted resonances was achieved by adjusting the offset frequencies of some of the frequency selective pulses to optimize the result of spectral editing.

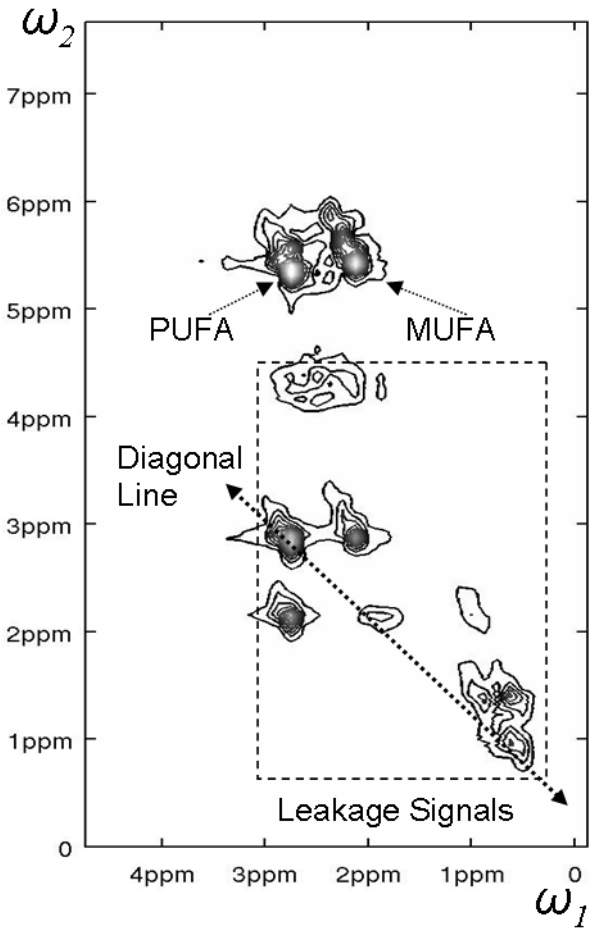

Figure 2-10 Un-edited 2D Sel-MQC Spectrum of Soybean Oil

Two Cross peaks of PUFA at (5.4ppm, 2.8ppm) and MUFA (5.4ppm, 2.1ppm) are acquired with leakage signals shown in the dashed box. (Matlab codes for this figure are included in Appendix. C-4.)

The default RF frequencies of the RF pulses are set on 5.4ppm and 2.8ppm in the Sel-MQC sequence. The first 90º pulse and the 180º pulse were centered on 5.4ppm while the other two 90º pulses were centered on 2.8ppm. Since the first 90º pulse was slice-selective it was not frequency-selective. The 180º pulse was guaranteed by two crushers before and after the Multiple Quantum evolution period. The focus of the optimization was, therefore, placed on the two 90º pulses on 2.8ppm. The frequency adjustment was carried out experimentally by moving

either 90º pulse in either directions independently and comparing the intensities of the residual peaks in a series of 2D spectra. The results were shown below as an array of 2D spectra labeled by their corresponding two frequency offsets of the second and last 90º pulses.

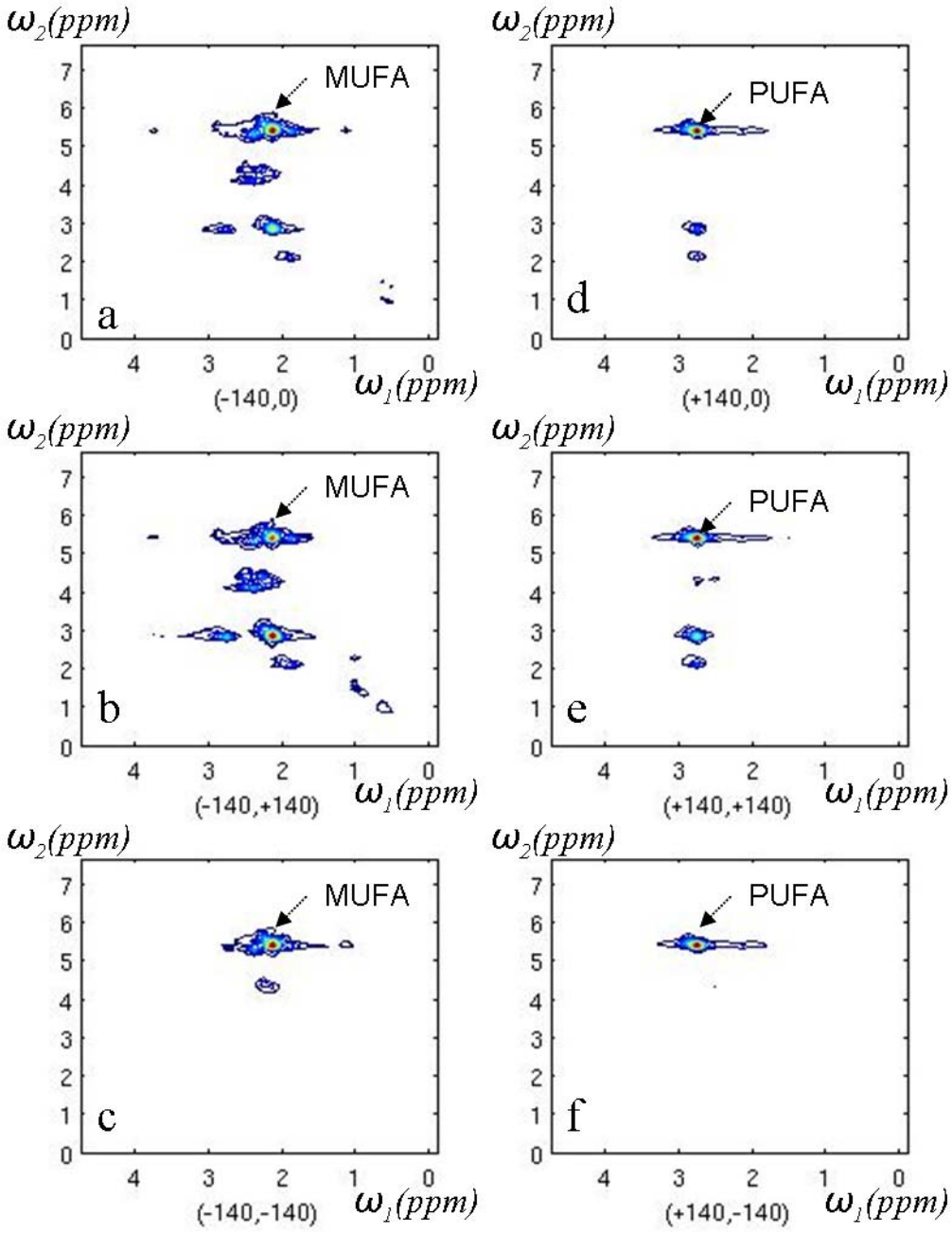

Figure 2-11 Optimization of PUFA selection in a series of 2D experiments

The numbers at the bottom of each 2D spectrum are the values of the off-resonance of the second and last 90° pulses. Resonance peaks of the leakage signals are gradually diminished as the two 90° pulses reached optimal offset frequencies. At the bottom, a single cross peak of PUFA (f) or MUFA (c) was selected. (Matlab codes for this figure are included in Appendix. C-5.)

The column of spectra on the left represents the 2D experiments where the frequency of the second 90º pulse was decreased 140Hz while the frequency of the last 90º pulse was tested with 3 offset values (on resonance or off by ±140Hz). Those three spectra show the MUFA cross peak at (5.4ppm, 2.1ppm) and the residue peaks at 2.1ppm along the readout direction (the vertical direction in Fig. 2-11, also called the chemical shift direction). That means that the second 90º pulse with decreased resonance frequency selects the *J* coupling of lower frequency (2.1ppm). The last 90º pulse affects the intensities of all the residual peaks. When its frequency was reduced by 140Hz, the residual peaks were reduced to a minimum and the cross peak at (5.4ppm, 2.1ppm) was selected.

The right column represents the experiments where the second 90° pulse was moved up by 140Hz and the last 90° pulse was tested with the same three different frequencies. In this case the second 90° pulse with an increased resonance frequency selects one of the two *J* couplings of higher frequency (2.8ppm). The residual peaks were minimized in the same fashion as in the left column when the last 90° pulse frequency was lowered by 140Hz.

At the bottom of two columns are the two cases where two cross peaks of MUFA and PUFA were selected respectively with optimal suppression of all other residual peaks. The selectivity of PUFA (Fig.2-11(f)) was better than that of MUFA (Fig.2-11(c)). These two configurations were both implemented in human subject studies to acquired PUFA and MUFA spectra in human breast tissues. The plot of 2D raw data below demonstrates the performance of the PUFA selection mode. The shape of the echo along readout direction is clean. Most importantly, the modulation of the echo amplitude along t_1 evolution direction clearly shows one frequency component that is responsible for the single cross peak in the resulting 2D spectrum (Fig.2-11(f)).

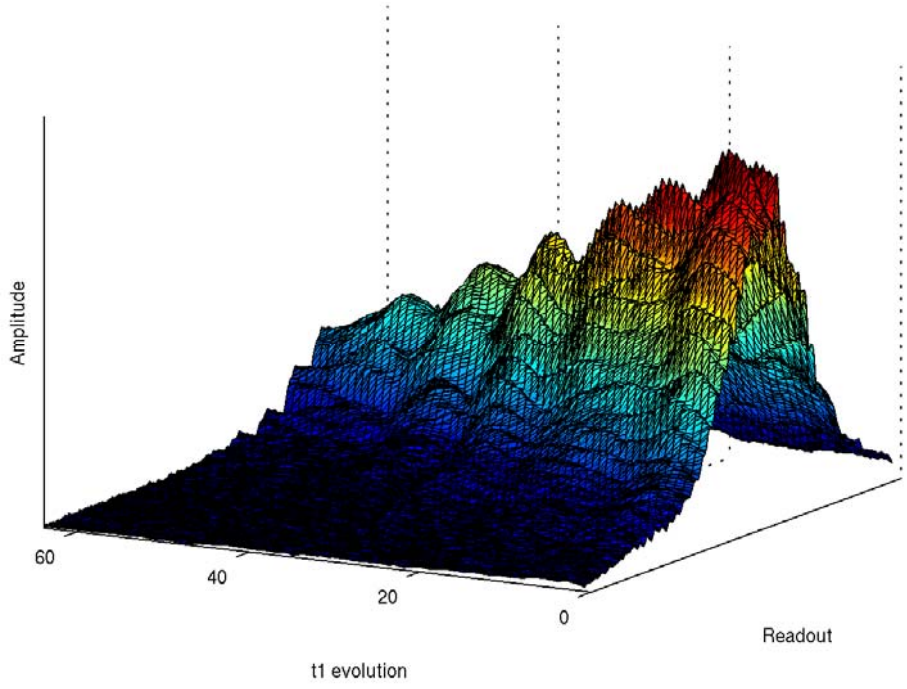

Figure 2-12 Raw Data Acquired by 2D Sel-MQC with PUFA Selection Mode

The signal along readout direction shows a clear shape of the PUFA echo. Along t_1 direction, the sinusoidal modulation of the echo peak shows a single frequency component in the spectrum of Fig.2-11 (f). (Matlab codes for this figure are included in Appendix. C-6.)

2.3.4 *In Vivo* Results

The resonances of PUFA and other signals in lipid were investigated in recent MRS studies (18,19,53). PUFA was considered as an possible indicator of the treatment progress in monitoring drug response in animal models (18,54). We recently reported a Sel-MQC CSI sequence detecting the signal of PUFA in human breast tissues *in vivo* (55). The intensity ratio of the PUFA and MUFA peaks was regarded as a measure of the degree of tissue saturation (21). A higher lipid unsaturation indicates a more fluid phase of lipid membrane.

2D Sel-MQC experiments were performed in 4 *in vivo* studies to acquired 2D spectra using both PUFA and MUFA selection modes of 2D Sel-MQC. The following are spectra of both configurations from two subjects. The difference of PUFA and MUFA concentrations are important tissue properties associated with an individual's metabolism (21).

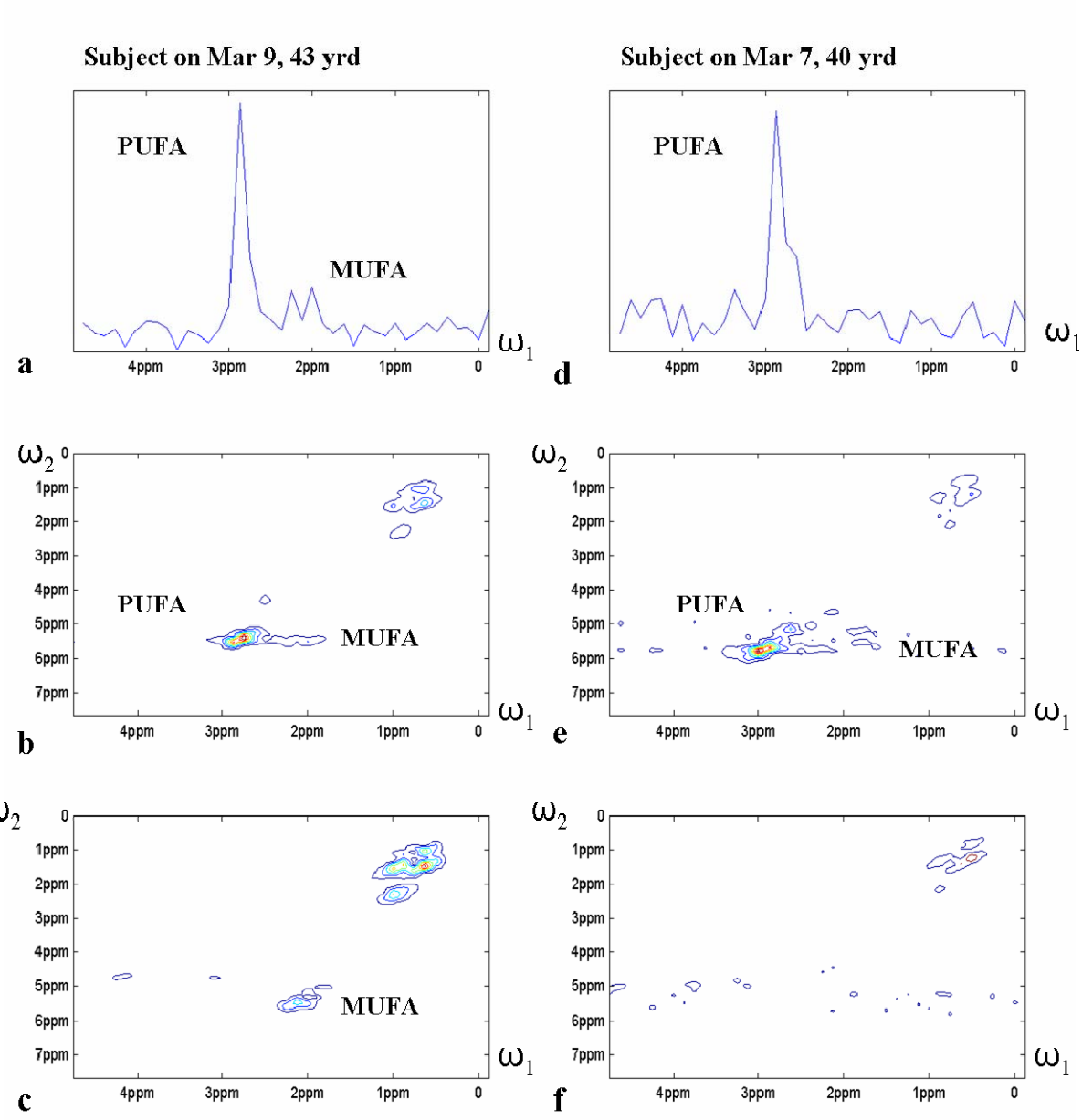

Figure 2-13 *In Vivo* 2D Spectra and 1D projection of PUFA and MUFA Selection from 2 Subjects

The 2D spectra from the subject on Mar. 9 (a, b & c) and the subject on Mar. 7 (d, e & f) show that PUFA (b & e) selection mode suppresses unwanted residual water signals at 4.7ppm and residual lipid signals at 1.3ppm better than the MUFA (c & f) selection mode. MUFA was not detedted in the second subject (f). (a & d) are the 1D projections of the spectra in (b & e) along Multiple Quantum evolution direction. They show that the signal of MUFA was suppressed in PUFA selection mode despite their close spectral locations. (Matlab codes for this figure are included in Appendix. C-7.)

The spectra in Fig. 2-13 show that the method of setting the second and last 90° pulses off-resonance (Fig. 2-11) successfully selected PUFA signal while minimizing MUFA signal contamination *in vivo*. The residual signals at 1.3 ppm and 2.1ppm were high because the 180° pulse was shortened to 6.3ms to make room for incremental Multiple Quantum evolution. 1D Sel-MQC spectroscopy experiments were also performed with fixed Multiple Quantum evolution period at 33ms as opposed to the 24-56ms in the 2D spectroscopy. The width of the 180° pulse was set at 11.7ms in the 1D experiments to enhance frequency selectivity. The 2D Sel-MQC experiments provided the procedure to reach the optimal settings for the 1D Sel-MQC experiments and those settings were used for the Spiral Sel-MQC experiments discussed in Chapter 3. The Sel-MQC 1D experiments were performed in the same *in vivo* subject studies as the 2D experiments. The 1D spectra (Fig. 2-14) were Fourier Transformed from the signal sampling along the readout (t_2) direction of the Sel-MQC experiments, which is different from the 1D spectra in Fig.2-13 (a and d). Those spectra were projected along the Multiple Quantum evolution direction (t_1) to distinguish PUFA at 2.8ppm and possible MUFA at 2.1ppm.

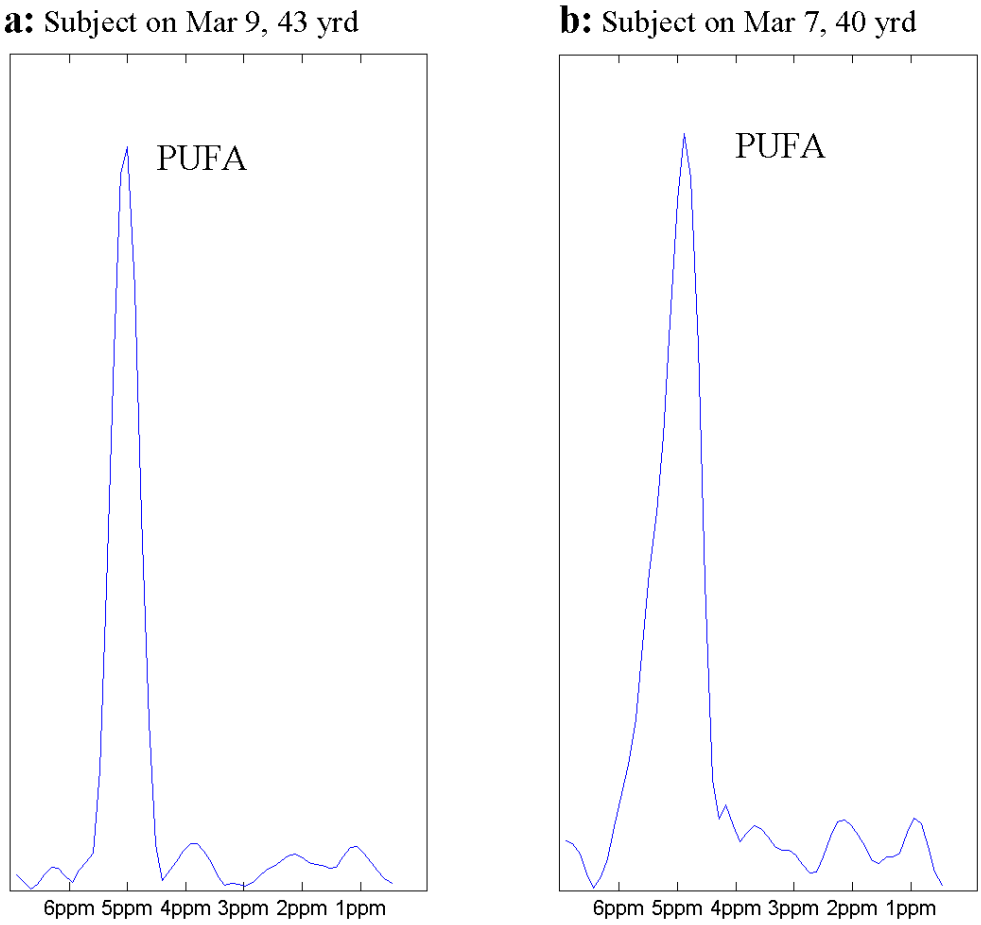

Figure 2-14 *In Vivo* 1D Spectra of PUFA Signal from the Same Two Subjects

Sel-MQC 1D Spectroscopy was performed in the same *in vivo* subject studies as the Sel-MQC 2D spectroscopy (Fig.2-13). The 180° pulse was extended from 6.3ms in the 2D experiments to 11.7ms in the 1D experiments. Two examples are shown above. When the PUFA selection is optimal as in (a), all residual signals are suppressed to the noise level. When the PUFA selection is not ideal (b), the leakage signal is small compared to the desired PUFA signal at 5.4ppm. This signal is the basis of the Spiral Sel-MQC imaging sequence discussed in Chapter 3. (Matlab codes for this figure are included in Appendix. C-8.)

The resulting 1D Sel-MQC echoes were processed through Hann's window at the location of the echoes and 1D spectra are plotted in Figure 2-14. The signal at 1.3 ppm and 2.1ppm was suppressed to noise level in Fig.2-14(a) and the signal at 5.4ppm contained mostly PUFA with small MUFA contamination as proved by the 2D experiments. The inhomogeneous broadening of the signal was observed in Fig.2-14. The location of the PUFA peak shifted from

5.4ppm because selecting the center frequency at 1.3ppm was performed by the MR scanner automatically and it was less accurated *in vivo*. When the selection was less ideal as shown in Fig.2-14(b), the residual signal is small compared to the desired PUFA signal. The effect of this residual signal in the Spiral Sel-MQC image experiments will be discussed in Chapter 3.4.

3.0 FAST SPECTROSCOPY IMAGING WITH SPIRAL SEL-MQC

An MRS sequence for clinical diagnosis (10,48) is often extended into a sequence of Magnetic Resonance Spectroscopic Imaging (MRSI) with the addition of imaging components. The primary benefit of MRSI compared to one voxel MRS is that spatial distributions of one or more chemical compounds can be observed. However, the long scan time associated with MRSI often limits its applications in clinical trials. Fast acquisition techniques originated in MRI are constantly imported into MRSI research in recent years to shorten the scan time. Spectroscopic EPSI (25,26), RARE (27), GRASE (56) and SSFP (57) are all designed to include fast imaging features to acquire multiple data lines in a 3D Cartesian grid to shorten the scan time. Spiral imaging has been implemented in MRI to further speed up acquisition in time critical areas including cardiac imaging (58), time resolved MRA (59), and *f*MRI (17). Exploratory spiral MRSI work at 1.5T demonstrated spiral data acquisition can also improve efficiency in mapping metabolites *in vivo* (60,61). More recently spiral MRSI was combined with *J*-resolved MRS to acquire enhanced spectra at 7T on rats (62,63) and at 3T on humans with CT-PRESS (64)

We previously implemented Sel-MQC CSI (Chemical Shift Imaging) with a two-dimensional phase encoding procedure to acquire PUFA spatial distribution (55). A conventional CSI data acquisition includes spatial encoding gradients and a data acquisition period (23). In each acquisition period the CSI sequence records one FID along the t axis in k-space. The k_x and k_y coordinates of a line are determined by the areas of the spatial encoding gradients in two directions, G_x and G_y. N_x and N_y increments are used respectively to collect $N_x \times N_y$ echoes in k-space. As multiple acquisitions are routinely used to increase SNR, the total scan time is, therefore, $TR \times N_x \times N_y \times NEX$. A typical MRSI scan can take 10~30 minutes, which is rather long for clinical applications. But if numbers of phase encoding steps are reduced to shorten the scan time, spatial resolution will be compromised. Spiral Sel-MQC method uses a multiple quantum filter to excite and select signals from PUFA and use that signal to form a PUFA image using

spiral k-space mapping. Data sampling along the t axis in k-space is no longer necessary, and discrete k_x and k_y mapping is replaced with continuous spiral sampling with increased efficiency.

The advantages of spiral sampling in MR applications result from the flexible design of spiral trajectories according to imaging parameters and signal source. Single shot spiral acquisition is often chosen in *f*MRI because of its ability to generate a low resolution image of the brain in tens of milliseconds. Multishot spiral acquisition, as demonstrated in Chapter 4, can produce high resolution time resolved 3D images combined with contrast enhancement by MR contrast agent Gd. In this Chapter, we discuss a combination of Sel-MQC spectroscopy and spiral imaging, noted as Spiral Sel-MQC. The Spiral Sel-MQC sequence is the first technical approach combining spin editing and fast imaging in *in vivo* MRSI. This approach enables advanced spin editing techniques such as Sel-MQC to be implemented along with standard MRI and MRS sequences in clinics. The focus of this chapter is on how these two technical components are modified and optimized for an application in clinical *in vivo* fast MRSI.

3.1 SPECTROSCOPIC IMAGING

The method of spectroscopic imaging discussed in this chapter is different from the typical method shown in Figure (1-11) so that we do not acquire a spectrum at every pixel location. Instead, we use the Sel-MQC technique to excite the signal of PUFA and form an image directly. When Sel-MQC is implemented as a selective excitation method for imaging, all remaining magnetizations following Sel-MQC pulse train are recorded including the unwanted residual water and lipid. It is, therefore, crucial to maximize the performance of the water and lipid suppression so that no signal from water and lipid is present. The optimization method outlined in the previous chapter serves as an excellent preparation for imaging because a single cross peak of PUFA at (5.4ppm, 2.8ppm) or MUFA at (5.4ppm, 2.1ppm) can be selected as a clean signal source for imaging.

Figure 3-1 demonstrates the preparation for imaging with PUFA signals. When the pulses in the Sel-MQC sequence were set to select PUFA peak and the evolution period was fixed at 33ms, an echo of PUFA magnetization was recorded (Fig. 3-1(a)). A 1D FFT shows a single spectral peak at 5.4ppm, as expected (Fig. 3-1(b)). The residual signals around center frequency

at 1.3ppm is negligible. The magnitude of the PUFA echo was plotted in Fig. 3-1(c) that represented the transverse magnetization available for image acquisition. The center of the echo, about 25ms from the start of the echo, will be the starting point of the spiral image acquisition, corresponding to the center of k-space.

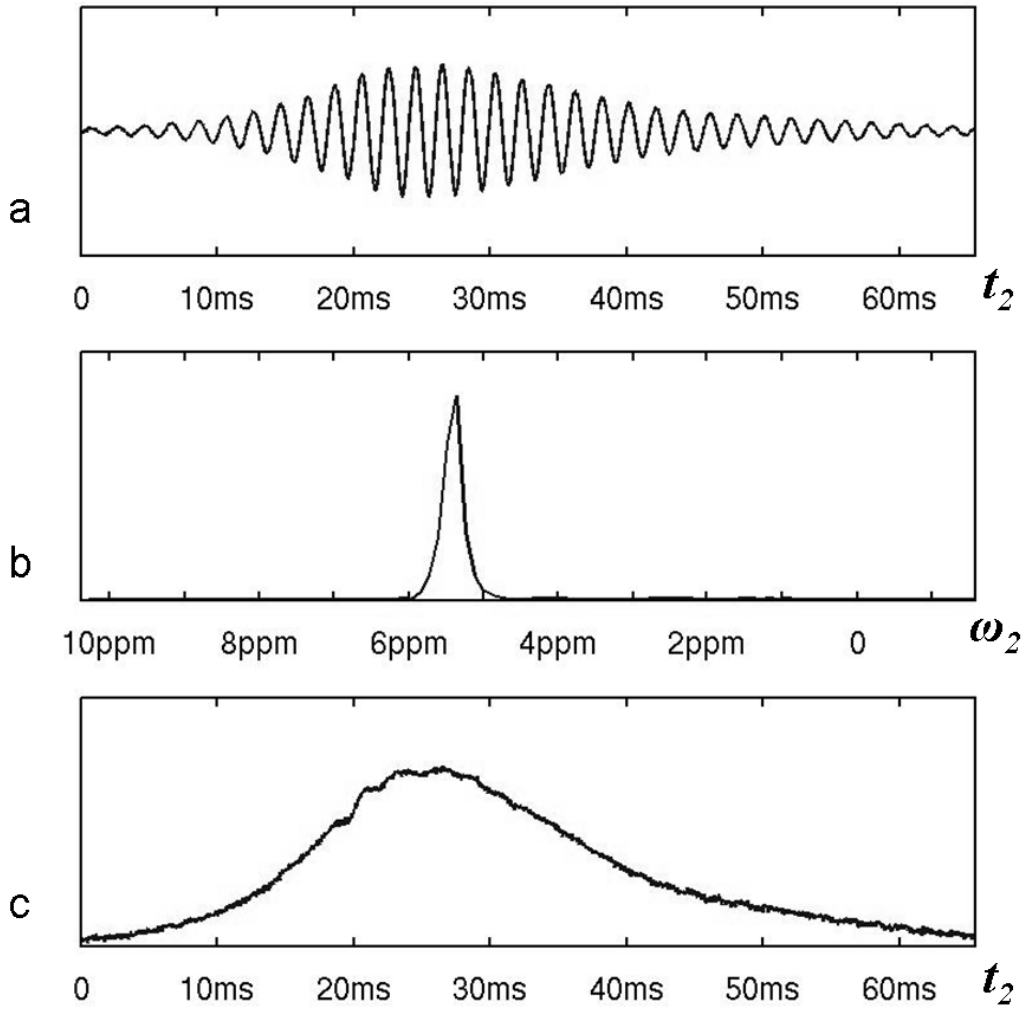

Figure 3-1 Coherence Echo and the Corresponding Spectral Peak of The PUFA Signal in A Phantom Study

One PUFA echo recorded at 1.3ppm (off-resonance) is shown in (a) . Its spectrum in (b) shows one spectral peak at 5.4ppm. The magnitude of the echo recorded is plotted in (c). (Matlab codes for this figure are included in Appendix. C-9.)

3.2 SEQUENCE DESIGN AND OPTIMIZATION

A spiral Sel-MQC sequence for imaging was developed from the Sel-MQC spectroscopic sequence (Figure 3-2). t_1 evolution time was fixed at 33ms. Spiral imaging gradients G_x and G_y were inserted into the sequence from the center of the PUFA echo.

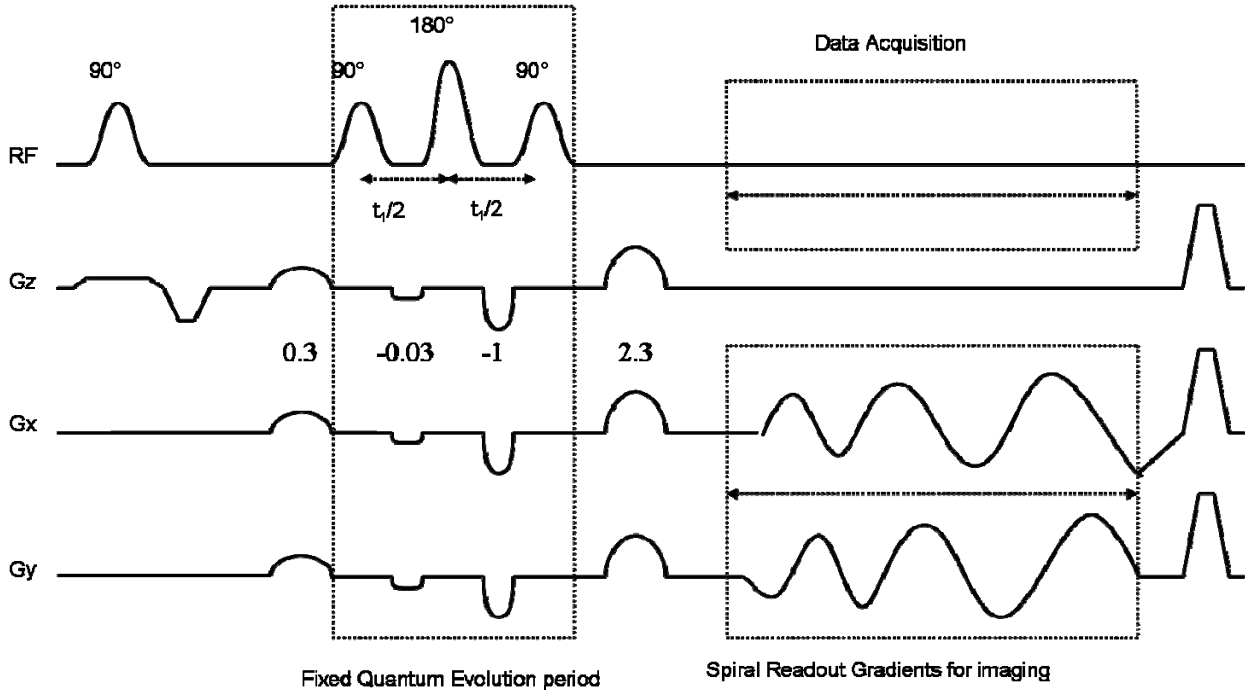

Figure 3-2 Sequence Diagram of Spiral Sel-MQC

Multiple quantum evolution period is fixed at 33ms for image acquisition in Spiral Sel-MQC sequence. Imaging gradients in Gx and Gy direction allows the image acquisition with PUFA magnetization starting at the center of the PUFA echo.

The sequence was first implemented to image a beaker of soybean oil that was a rich source of PUFA signal. Single shot spiral was first chosen to acquire a low resolution image of the beaker of oil (Figure 3-3). FOV was 16cm. Resolution was 24×24 at acquisition. The trajectory was prescribed to be particularly long without significant undersampling to test the limit of single shot configuration. This single shot Spiral Sel-MQC experiment was not designed to resolve any structures in the phantom because its cross section is homogeneous. The length of spiral acquisition was 1887 points with 4μs per point. The duration of the data acquisition was 7.5ms.

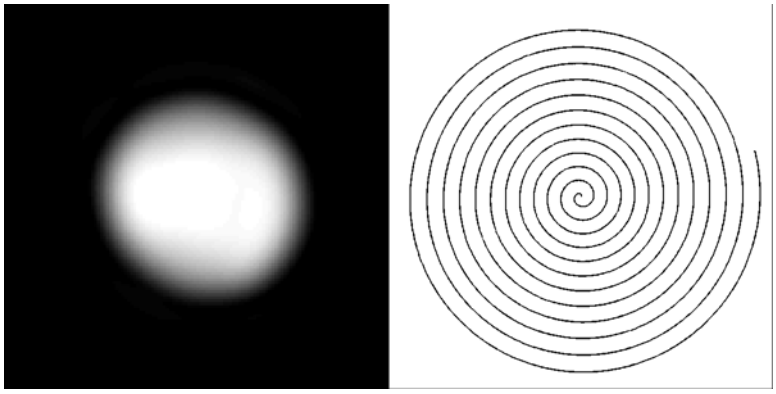

Figure 3-3 Single Shot Spiral Sel-MQC Imaging with PUFA Magnetization

(left) An image of a bottle of oil using a single shot spiral. (right) Single shot trajectory. Constant density spiral [$k_c(\tau)=k_{max}\,\tau\,e^{i\omega\tau}$] is used at center of the k-space and variable density spiral [$k_v(\tau)=k_{max}\,(a\,\tau^2+b\,\tau+c\,)e^{i\omega\tau}$] is used for the rest of the trajectory. (Matlab codes for this figure are included in Appendix. C-10.)

In order to evaluate the choice of spiral parameters the sequence was also executed without the imaging gradients so the decay curve of a PUFA echo was acquired from its echo center as shown in Fig.3-4.

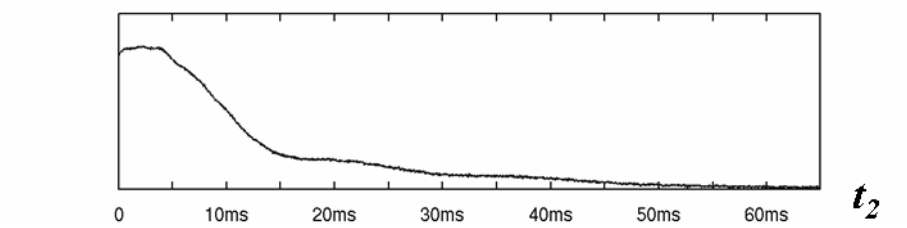

Figure 3-4 The Shape and length of an partial echo of PUFA

The shape and length of a partial echo of PUFA shows that the magnetization decays significantly at 10ms after the echo center. This sets the limit on the length of the spiral trajectory. (Matlab codes for this figure are included in Appendix. C-10.)

The shape of the echo indicates an imaging period as long as 7.5ms reaches the tail of the echo where magnetization decreases considerably because of T_2^* decay. The length of a PUFA echo *in vivo* would be even shorter, which would further limit the parameters of single shot acquisition. Therefore, a multishot spiral acquisition would be needed for imaging because PUFA coherence transfer echo could not provide as much signal as regular signal sources in MRI like water protons.

A test of Spiral Sel-MQC imaging with 2 spiral interleaves was prescribed with adequate undersampling to reduce the lengths of the acquisition. The resulting images contain surprising artifacts as shown below.

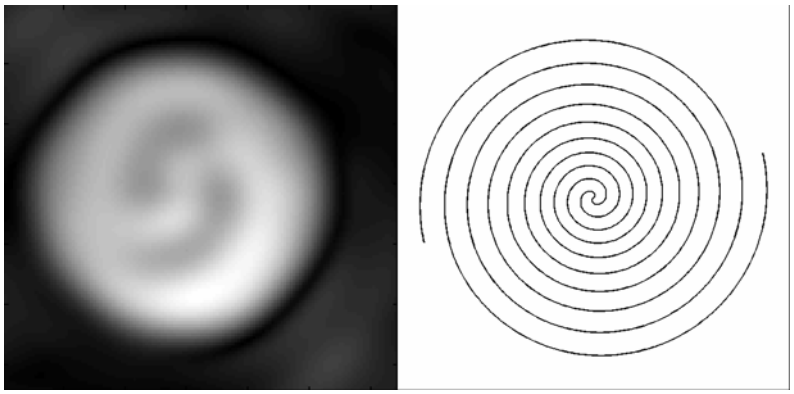

Figure 3-5 Image Artifacts of Two Shot Spiral Sel-MQC with PUFA Magnetization

(left) An image of a bottle of oil shows artifacts caused by a 2-shot spiral acquisition. (right) 2-shot spiral trajectories. Constant density spiral [$k_c(\tau)=k_{max}\,\tau\,e^{i\omega\tau}$] is used at center of the k-space and variable density spiral [$k_v(\tau)=k_{max}\,(a\,\tau^2+b\,\tau+c\,)e^{i\omega\tau}$] is used for the rest of the trajectory. (Matlab codes for this figure are included in Appendix. C-11.)

Further experiments of 2-shot spiral sequence were performed with the imaging gradients disabled so that the experiment was equivalent to acquiring 2 FID's as in a 1D Sel-MQC spectroscopic experiments. The two echoes were plotted in Figure 3-6(a).

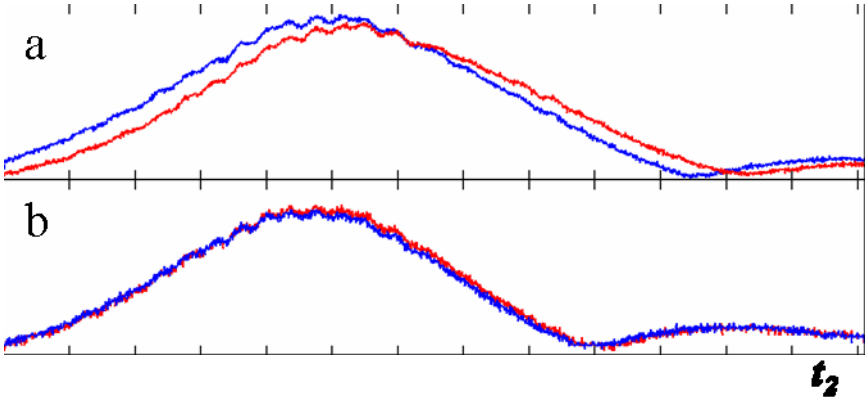

Figure 3-6 Source of Artifacts: Echo Mismatch

(a) Two mismatched echoes were acquired when the 2-shot spiral were executed using the rotation matrix. (b) Two overlapping echoes were acquired when rotation matrix was avoided. (Matlab codes for this figure and an example of the rotation matrix are included in Appendix. C-12.)

Two mismatched coherence transfer echoes were formed by identical excitations because the spiral imaging gradient pulses were disabled. A closer reflection on the structure of multi-shot

spiral image acquisition sequence indicates that the sequence waveforms of all interleaves are identical in the logical frame of the scanner, where the logical frame refers to the frame in which a pulse sequence is designed. In the logical frame, the slice selection direction is always labeled z direction. The physical frame is the frame in which gradient fields are actually executed. So, if an axial slice is selected, the physical slice selection direction is the z direction. If a sagittal slice is selected, the physical slice selection direction is y direction. The transformation of the sequence from the logical frame to the physical frame is achieved by multiplying a 3×3 rotation matrix to all the waveforms from the logical frame to the physical frame. In a spiral imaging sequence, multiple interleaved acquisition are executed by rotating the same pulse sequence around the axis of the slice selection plane. For example, if the slice selection is in the x-y plane, multiple interleaves are rotated arount the z-axis. So the rotation matrix operation

$$\begin{pmatrix} G_x \\ G_y \\ G_z \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} G_x' \\ G_y' \\ G_z' \end{pmatrix}$$

Equation 3-1

transforms the gradient waveforms (G_x', G_y', G_z') from the logical from into the waveforms (G_x, G_y, G_z) in the physical frame. All the gradient waveforms in the sequence are rotated including the spiral imaging gradients and Sel-MQC coherence selection gradients. The rotation of the imaging gradients is the purpose of the change of reference frame but the rotation of the quantum selection gradients in the excitation period would be unnecessary. This difference is shown in Fig.3-7.

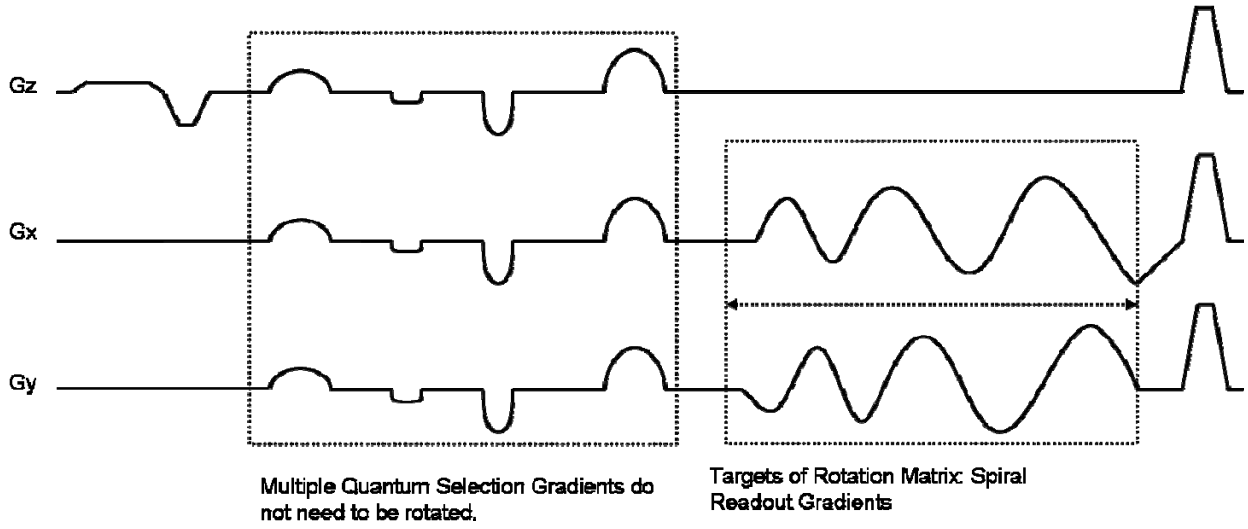

Figure 3-7 Effects of the Rotation Matrix on Gradient Waveforms

Fig.3-7 is shows the gradient waveforms from Fig.3-2. Only the spiral imaging gradients need to be rotated for a 2-shot spiral acquisition. Performing the rotation matrix multiplication to the entire gradient waveforms including the multiple quantum selection gradients was the cause of the echo mismatch shown in Fig.3-6.

The two mismatched echoes in Fig. 3-6 demonstrate the undesirable effects of rotating the coherence selection gradients before acquiring an echo. In a two shot spiral acquisition, the θ in Eq. 3-1 is 180° so that

$$\begin{pmatrix} G_x \\ G_y \\ G_z \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} G_x' \\ G_y' \\ G_z' \end{pmatrix}.$$

Equation 3-2

The rotation matrix operation becomes inversing the gradient waveforms of G_x' and G_y'. Therefore, the rotation matrix operation for all gradient pulses was disabled in the two interleaved spiral sequence and the only waveforms of the spiral imaging gradients were inversed 180° to achieve the same goal. The 2-shot sequence was executed without imaging gradients again to show two overlapping echoes in Fig. 3-6(b). As imaging gradients were enabled images were acquired as shown in Figure 3-8.

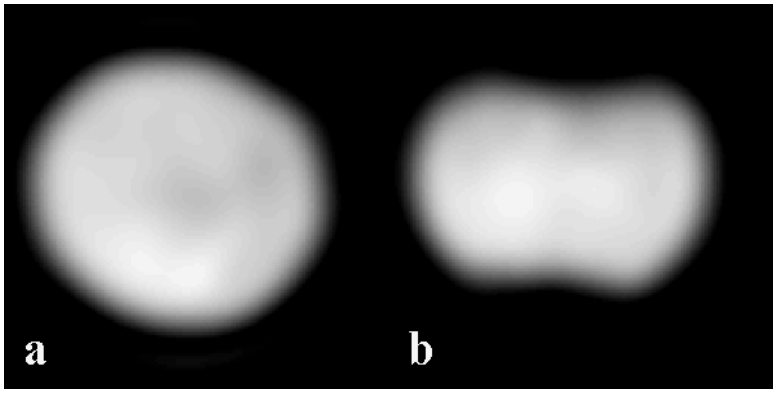

Figure 3-8 Oil Phantom Images using Modified Two Shot Spiral Sel-MQC Acquisition

A plastic bottle of oil was imaged in coronal (a) and axial (b) planes. The slice thickness was 1cm. 2-shot spiral acquisition without using rotation matrix produced images with correct contrast. (Matlab codes for this figure are included in Appendix. C-13.)

3.3 RESULTS

3.3.1 Phantom Studies

A three compartment phantom was constructed for imaging experiments using soybean oil, canola oil and olive oil to produce three levels of PUFA concentrations. The ratio of polyunsaturated fat in soybean, canola and olive oils was approximately 4.5:2:1 as shown in the labels. Two tubes in Fig.3-9, each 3cm in diameter, were filled with soybean oil and olive oil, respectively. They were placed in a beaker of canola oil so that the background had medium concentration of PUFA while one tube had PUFA concentration higher than the background and the other tube had concentration lower than the background. Figure 3-9 is a photo of the phantom for imaging studies and the labels of 3 types of oil.

Figure 3-9 Imaging Phantom Constructed with 3 Types of Oil and Their Package Labels

An imaging phantom was constructed using 3 types of oil with different PUFA concentrations. Soybean oil and olive oil with the highest and lowest PUFA concentration, respectively. They are in a bath of canola oil with medium PUFA concentration.

A coronal image of 1cm slice thickness was acquired by Spiral-SelMQC sequence (Figure 3-10a) with a fast gradient echo image (Figure 3-10b), and with a fast spin echo image (Figure 3-10c), acquired as references. The spin echo image shows identical intensity levels in different compartments and the gradient echo image shows very small difference in image contrast between conpartments. However, the Spiral-SelMQC image shows distinctive PUFA contrast in the phantom. The tube filled with soybean oil stands out with the highest signal intensity while the tube filled with olive oil shows low intensity. The background filled with canola oil shows intermediate intensity that distinguishes its boundaries to two tubes inside. This experiment clearly shows the image contrast reflects the spatial distribution of PUFA in the phantom.

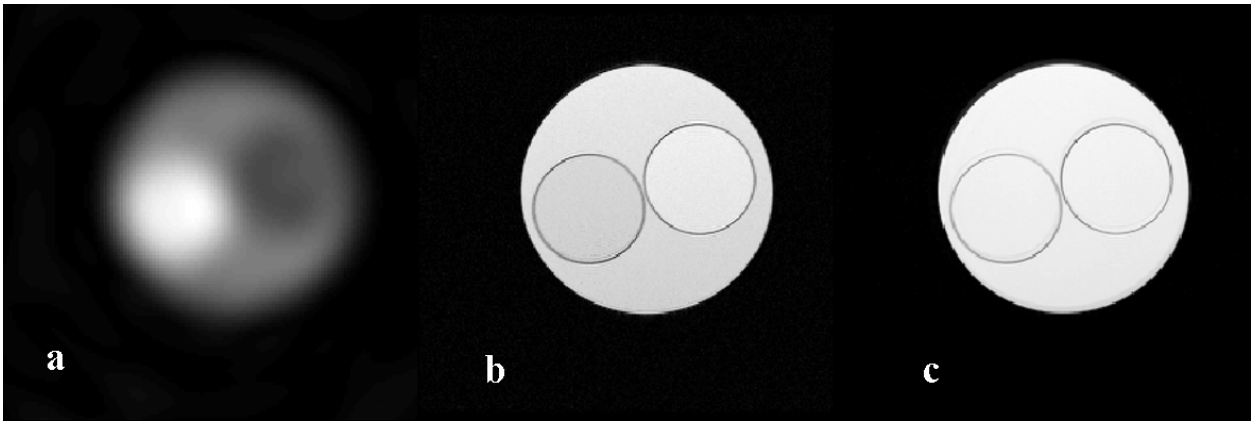

Figure 3-10 PUFA Distribution in an Oil Phantom

(a) Spiral Sel-MQC image shows image intensity representing the concentrations of different oils in 3 compartments. (b) Gradient echo image shows minimum image contrast. (c) Spin echo image shows no image contrast. (Matlab codes for this figure are included in Appendix. C-14.)

3.3.2 *In Vivo* Subject Studies

Human subject studies were performed on 6 healthy volunteers with local (Univ. of Pittsburgh) IRB approval. In volunteer studies fast gradient echo images using GE's 2Dfast sequence were first acquired to obtain anatomical structures (Fig. 3-11, background images in grayscale). TR/TE was 20ms/6ms. Spiral-SelMQC was then applied to acquire two PUFA images of 1 cm thickness (Fig. 3-11, foreground images in red). All images were acquired in the sagittal plane. Fig.3-11 (a and b) shows a center slice and an off-center slice of a 43 year old subject. Fig.3-11 (c and d) shows off center slices from two other subjects of 40 years old and 31 years old. Fig. 3-11 (e) shows a center slice of a 25 years old with a silicone implant. Fig.3-11 (f) shows an off-center slice of a 74 year old subject. The resolution of the gradient echo images was 256×256. The resolution of the spiral-SelMQC images were 16×16 or 20×20. FOV = 14-16cm. The images show interesting correlations between anatomical structures and PUFA distribution in the breast from subject to subject. PUFA originated predominantly from fatty tissues (Fig. 3-11 a-e). It appears that sometimes it was detected in the other regions of the breast (Fig3-11 f).

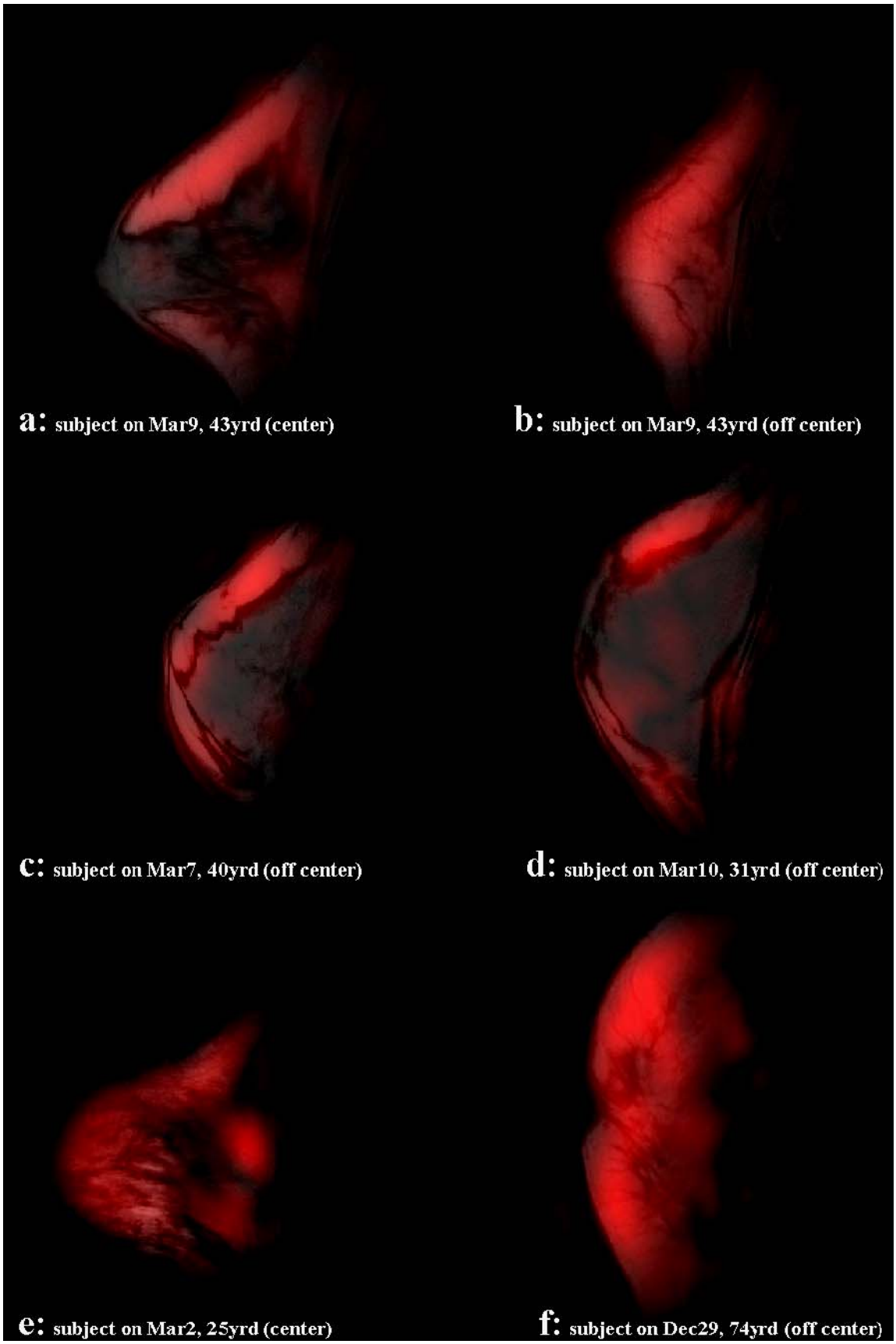
a: subject on Mar9, 43yrd (center)
b: subject on Mar9, 43yrd (off center)
c: subject on Mar7, 40yrd (off center)
d: subject on Mar10, 31yrd (off center)
e: subject on Mar2, 25yrd (center)
f: subject on Dec29, 74yrd (off center)

Figure 3-11 *In Vivo* Images of Spiral Sel-MQC

(a)&(b) show a center slice and an off-center slice of a 43 year old subject. (c) Off center slices from subject of 40 years old (d) Off center slice from subject of 31 years old. (e) Center slice of a 25 years old with a silicone implant (f) Off center slice of a 74 year old. (Matlab codes for this figure are included in Appendix. C-15.)

The individual variation indicated that the PUFA spatial distribution were important tissue properties of an individual breast. The results show that the Spiral Sel-MQC technique could detect possible local alteration of PUFA concentration, and therefore could provide information on the change of breast tissue associated with a developing breast tumor (55).

3.4 DISCUSSION

The performance of the coherence selection was proved by the dominant cross peak of PUFA in Sel-MQC 2D spectra of both phantoms and *in vivo*. Figure 2-13 in the previous chapter shows 2D spectra of the PUFA selection in 2 *in vivo* cases. In all cases, the signal contamination of the nearby MUFA peak at 2.1ppm was suppressed to a minimum. Since both PUFA and MUFA magnetizations were transferred back to 5.4ppm for signal acquisition, the effect of the MUFA contamination would be a negligible increase of signals at 5.4ppm. A more significant source of signal contamination in *in vivo* spectra came from the lipid peak at 1.3ppm, if coherence selection gradient is not strong. This peak, unlike the MUFA peak at 2.1ppm, appeared as an off-resonance signal source in the final signal readout. The frequency offset was 510Hz at 3T. This off-resonance signal can be eliminated entirely by larger gradient amplitudes for coherence selection but this approach would result in a decreased PUFA signal. In the Spiral Sel-MQC, a similar trade off between sensitivity and selectivity has to be made as well. Figure 2-14 in the previous chapter shows two *in vivo* cases of Sel-MQC 1D spectra where the lipid suppression performance is different. In order to better estimate the contribution of the residual signal at 1.3ppm to the final PUFA image using the 5.4ppm signal, we simulated the effect of the off-resonance peak on the PUFA image quality.

In Cartesian imaging, an off-resonance signal source results in a spatially shifted image. The direction of the spatial shift is determined by the direction of the readout in k-space, i.e. the

direction of the linear frequency encodings. In spiral imaging, the same off-resonance source produces a different kind of artifacts. The process of analyzing these artifacts consists of two parts: an analytical generation of the k-space sample values along two spiral trajectories, and an image reconstruction process including re-gridding and FFT. Simulated objects to be imaged are circular discs in the FOV that correspond to analytical functions in the k-space in the form of a Bessel function of the first kind

$$F_{2D}[circ(r)] = \frac{J_1(2\pi\rho)}{\rho} \equiv jinc(\rho), \quad \rho = \sqrt{k_x^{\,2} + k_y^{\,2}} \ .$$

Equation 3-3

Here, *circ(r)* refers to a function in image space where the image intensity is 1 within a radius *r* and 0 outside *r*. The 2D Fourier Transformation of the function *circ(r)* in k-space is defined as a *jinc(ρ)* where $\rho = \sqrt{k_x^{\,2} + k_y^{\,2}}$ is the radius in k-space. $J_1(2\pi\rho)$ is the Bessel function of the first kind. The location, size and intensity of the circular disc *circ(r)* in the image space corresponds to the appropriate phase shift, scaling and intensity of a *jinc(ρ)* function in k-space. Any image space pattern consisting of several circular discs of different sizes and intensities at different locations can be analytically expressed in k-space as a summation of the corresponding *jinc* functions. Eq. 3-3 generates k-space sample values analytically with any pair of k-space coordinates k_x, k_y. When these k-space samples are acquired on resonance, the result of the analytical formula simulates the k-space value correctly if the noise and the relaxation are neglected. If the data acquisition is off-resonance, the signal value is modulated by a phase factor $e^{-i2\pi\Delta ft}$, that is proportional to the size of the off resonance Δf. t is the time when that particular k-space point is sampled. In spiral imaging, t is proportional to the length of the spiral path traveled from the k-space center along the trajectory. During the image reconstruction process, each data point along a spiral trajectory is re-gridded onto a Cartesian matrix and FFT is applied to the matrix to form an image in Cartesian coordinates as shown in the diagram in Figure 3-13.

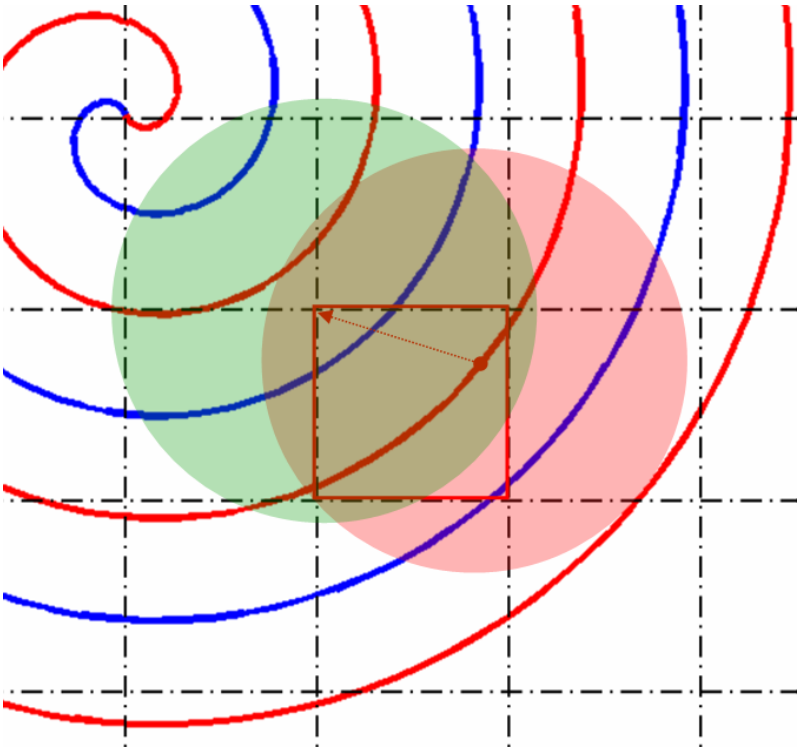

Figure 3-12 A Diagram demonstrating the re-gridding process in spiral reconstruction. The value at each trajectory point is weighted down radially and transferred to the neighbouring points on the Cartesian grid (Red region). The value at each Cartesian grid point is accumulated from neighbouring spiral segments (Green region). The sampling points in the spiral segments are modulated by the off-resonance *Δf*. The accumulated value is cancelled to a larger extent at high spatial frequency.

When an image is generated as an off-resonance source in image space, its k-space samples are modulated by the phase factor $e^{-i2\pi\Delta ft}$ along the trajectory. The phase factor is then carried over to the Cartesian locations through the re-gridding process. The final value at any Cartesian point combines the contributions from several nearby spiral segments but the phase factors from those spiral segments are *not* constructive. Consequently, when an off-resonance source is included in the imaging and reconstruction process, the image intensities are greatly reduced. Figure 3-14 shows the same disc reconstructed on and off-resonance by 530Hz and 530×2=1060Hz.

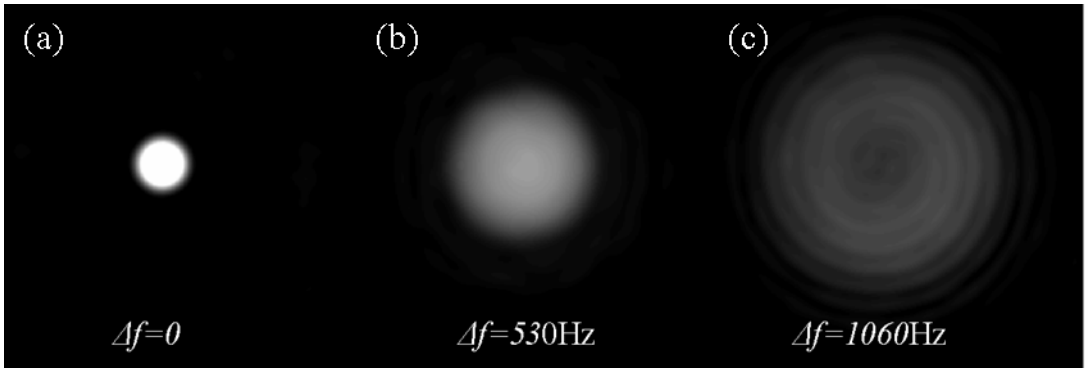

Figure 3-13 Simulations of Off-resonance effects in Spiral Imaging I

A single circular disc reconstructed with 3 different off-resonance values. In (a), the circular disc is reconstructed on-resonance so it is displayed correctly. When the disc is reconstructed off-resonance by 530Hz in (b), the intensity of the disc is smeared out over a bigger region. When it is reconstructed off-resonace by 1060Hz, it contains ringing artifacts typical in spiral imaging. (Matlab codes for this figure are included in Appendix. C-16.)

The on-resonance image (a) correctly shows the shape of the disc while the off-resonance images (b) and (c) show the disc "smeared" over a larger region to lower intensity as the off-resonance value increases. The off-resonance disc is a source of signal contamination when it is acquired with on resonance objects in the image. The simulated images in Figure 3-14 demonstrate the contamination effects of off-resonance signals in spiral imaging.

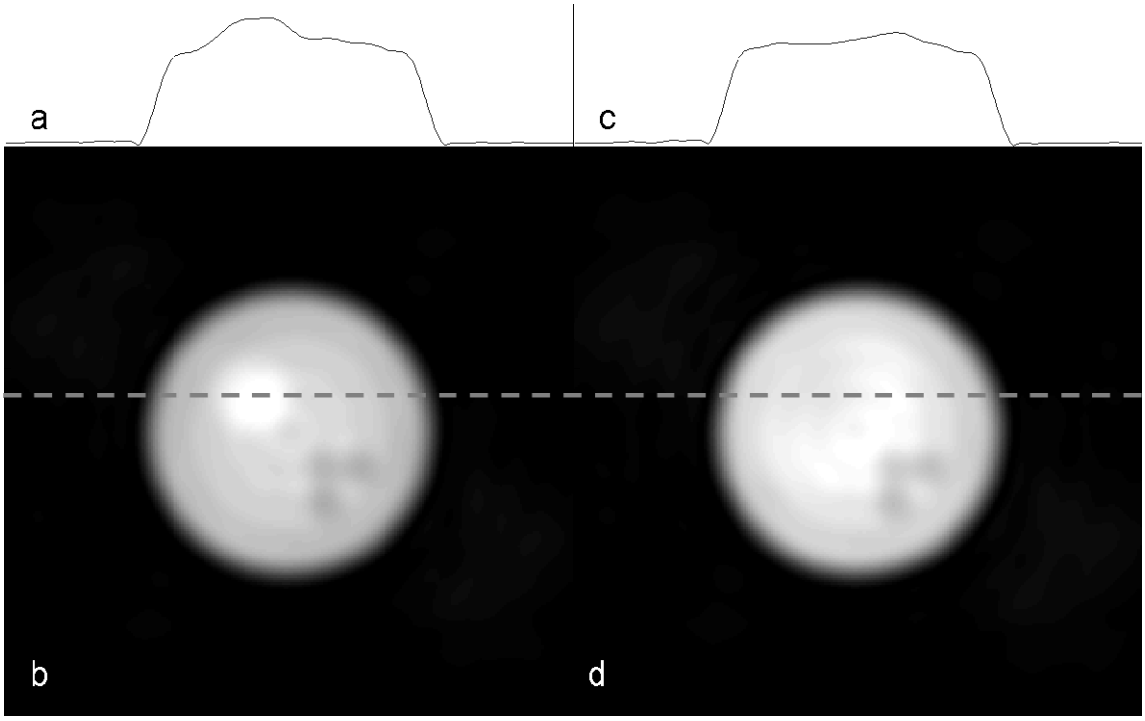

Figure 3-14 Simulations of Off-resonance effects in Spiral Imaging II

(a). Image intensity profile along dashed line in (b). (b) Simulated image with an object with positive intensity *on* resonance. (c) Image intensity profile along dashed line in (d). (d) Simulated image with an object with positive intensity *off* resonance by 530 Hz. (Matlab codes for this figure are included in Appendix. C-17.)

The large circular disc has a diameter of 8 cm in a 16 cm field-of-view. In Fig.3-14, the intensity of the large disc is 1.0. In the lower right corner of the big disc, there are three small discs with a diameter of 1 cm and intensity of -0.5. For the purpose of the simulation, the big disc with three small holes is the spatial distribution of PUFA signal on resonance. The resonance frequency of this compound is the center frequency of the receiver.

Assuming that the off-resonance signal source shown in Figure 3-14 is inserted at the upper left corner of the big disc as shown in Fig. 3-14. If the signal source of this disc is at the center frequency and its intensity is 0.2, the acquired image will show a "bump" at its location (Fig.3-14b). A one dimensional profile through the object will correctly reflect the increased intensity at ~20%, as shown in (Fig.3-14a). If, on the other hand, the signal source of this disc is +530Hz away from the center frequency, the resulting image will show the desired distribution of the on-resonance signal, and the off-resonance signal will be "smeared" out in the image. As the same line profile shows in (Fig.3-14c), the intensity of the off-resonance signal is greatly reduced. The effect of this reduction is proportional to the absolute value of the off resonance,

i.e. the further away the contamination signal is from the center frequency in either direction, the greater the "smearing" effect. (If the off-resonance amount is -530 Hz, the results will be identical.) In this simulation, the off-resonance amount is equal to the actual spectral difference between the lipid peaks at 1.3 ppm and 5.4 ppm on 3T. This suggests that when 5.4 ppm is set as the center frequency for spiral imaging, the contamination of the residual signal at 1.3 ppm is negligible, especially when the signal is as small as shown in Fig.2-14 in the previous chapter. The sensitivity of the current technique to resolve the small structures in Fig.3-14d directly determines the feasibility of applying this technique to detect other *J*-coupled compounds in human tissues.

4.0 TIME-RESOLVED 3D SPIRAL MRI

Time-Resolved MRI is often used to study blood vessels in the form of angiography. High resolution Magnetic Resonance Angiography (MRA) is best conducted with Contrast Enhancement (CE). The standard choice of contrast in MR is Gd compounds. When a Gd-base contrast is injected into a person's blood stream it increases the intensity of the signals produced by water protons so that the arteries are better depicted in the acquired images.

Contrast Enhanced Magnetic Resonance Angiography (CEMRA) has become the method of choice for performing MRA on most organs in humans (65). A typical CEMRA examination completes in 20-30 seconds and requires real-time monitoring methods or a test injection to predict contrast arrival (66,67). These timing techniques complicate the procedures of CEMRA and introduce additional sources of errors. Time resolved data acquisition methods that capture the passing of the contrast agent through the arteries, organ and veins of interest have the advantage of eliminating the need for contrast timing while simultaneously mapping blood flow dynamics. However, typical 3D k-space sampling for high resolution imaging takes too long (10-20 sec) to provide acceptable temporal resolution. By discarding the sampling of depth resolution, 2D projection MRA can be completed in one second (68-70), thereby providing adequate temporal resolution, with high spatial resolution similar to conventional projection x-ray digital subtraction angiography. However, the lack of depth resolution limits the clinical utility of the 2D projection MRA. Combining 3D acquisition and time-resolved acquisition is therefore desirable in Time Resolved MRI. The key to acquiring 3D data fast enough for time resolved imaging is to sample k-space with the most efficient method. In this chapter, we present a sequence for Time Resolved MRI using spiral data acquisition.

Spiral sampling allows azimuthal undersampling and sliding window reconstruction (16,71-73). Spiral sampling also enables long signal readout in one TR that is more efficient than radial sampling. 3D spiral CEMRA was reported recently to generate high spatial resolution 3D

CEMRA in a few seconds (22,73). In this chapter, we incorporate sliding window reconstruction with complex subtraction of pre-contrast mask data and improved off-resonance correction for time resolved 3D spiral MRA. This approach marks the first application of 3D stack-of-spiral data acquisition in time resolved CEMRA. The method and clinical results of this technique was pulished as a communication in *Magnetic Resonance in Medicine* in 2004 (59).

4.1 METHOD

4.1.1 3D Stack-of-Spirals

A 3D stack-of-spiral sequence (Fig. 4-1) was developed based on a 2D spiral sequence originally designed for *f*MRI (74). A slab selection pulse and its companion gradient excite a slab defined along *z* axis. A gradient pulse immediately following the excitation performs both refocusing and phase encoding. A simultaneous pair of gradients performs the spatial encoding in the k_x-k_y plane along a spiral trajectory. The Free Induction Decay (FID) recorded by a RF receiver is a series of k-space samples along that spiral trajectory. The remaining magnetization is destroyed by spoiler gradients along three axis. Delays between the pulses are minimized to save scan time. GE's standard gradient and RF pulses are used for all but the spiral readout gradients. The spiral gradient waveforms are generated by a custom program and loaded into the sequence as scan parameters are prescribed in real time. (The spiral trajectory generation program is included in Appendix A.)

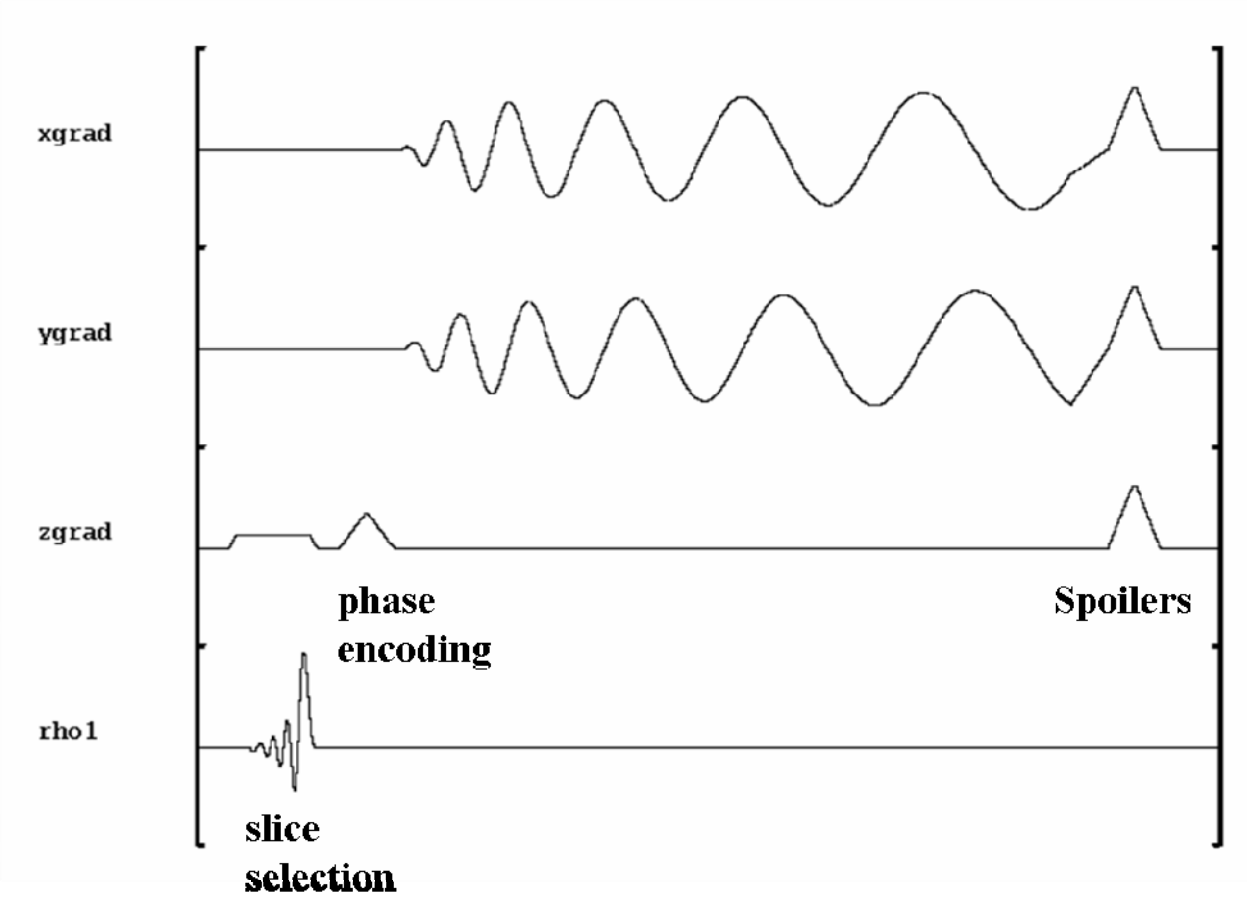

Figure 4-1 Pulse Sequence of 3D Stack of Spirals

A slab is selected by a RF pulse and a selection gradient. Time varying spiral gradients in x and y directions follow a phase encoding in z direction. One spiral interleave in k-space is sampled in each execution of the sequence.

Fig. 4-2 shows the coverage in k_x-k_y plane (left) and the whole volume in k-space (right). Variable density spiral interleaves are prescribed with a sampling density of 1.3×FOV at the center of k-space and 0.7×FOV at the edge of k-space. The connection between the constant density sampling at the center of k-space and the reducing density sampling is carried out by analytical matching boundary conditions. The details of trajectory generation will be discussed in Chapter 5.

The 3D coverage of k-space is carried out by rotating a spiral leaf in k_x-k_y plane in 24 steps and moving it in k_z direction in N_z steps, where N_z is the number of phase encodings. The in-plane rotation is achieved by multiplying G_x and G_y imaging gradients with a rotation matrix for each leaf. The stack in k_z direction is formed as the phase encoding gradient takes N_z

incrementing values. In addition, the entire stack of spiral acquisition is repeated several times to form a temporal series of 3D k-space data.

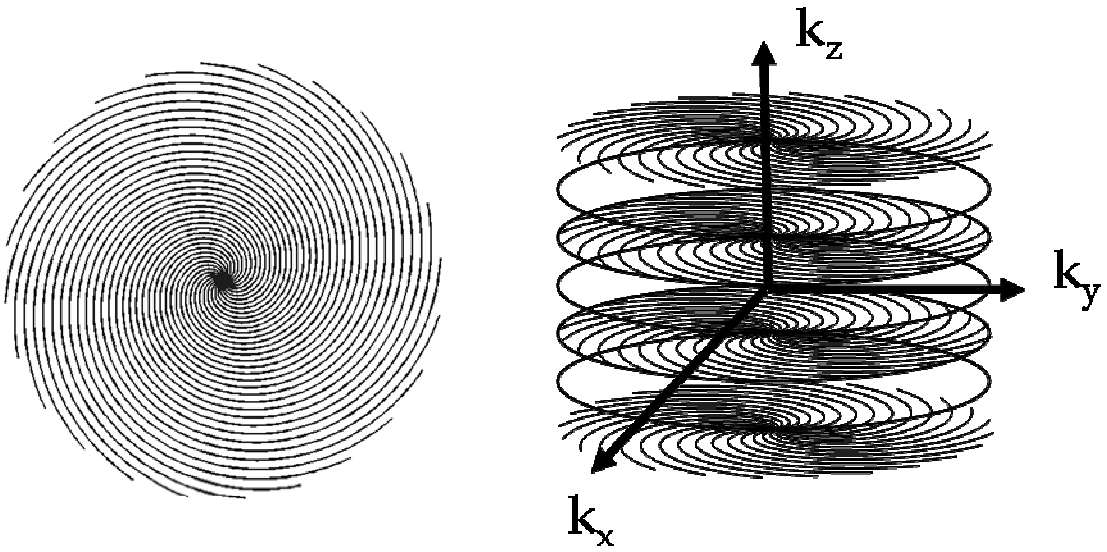

Figure 4-2 K-space Coverage in 3 Dimensions: Stack-of-Spirals

K_x-k_y plane is covered by 24 interleaves. 32 planes in k_z direction form a 3D stack in k-space. Constant density spiral [$k_c(\tau)=k_{max}\ \tau\ e^{i\omega\tau}$] is used at center of the k-space and variable density spiral [$k_v(\tau)=k_{max}\ (a\ \tau^2+b\ \tau+c\ )e^{i\omega\tau}$] is used for the rest of the trajectory.

4.1.2 Data Acquisition & View Ordering

A single channel fast receiver is used for all current applications of time resolved MRI. The fast receiver bandwidth is ±200kHz which translates to one data point every 2.5us. In each execution of the sequence the receiver records and filters one FID and stores it in Transceiver Processing and Storage (TPS, i.e. the memory of the scanner). A complete stack of data set acquired for one temporal phase consists of $24\times N_z$ interleaved acquisitions. As 4-6 temporal phases are prescribed, a block of TPS is allocated, segmented and indexed according to three indices: number of leaf, number of phase encoding and number of temporal phase.

View ordering is the order in which those interleaves are acquired. It is independent from the way they are stored in the memory. View sharing is a technique that allows data to be shared by several temporal frames in reconstruction. The current view ordering is designed for time resolved application to allow view sharing and sliding window reconstruction (Figs. 4-3, 4-4). 24 interleaves of all k_z phase encodings are divided into *n* groups. Leaf numbers ranging from 20 to

40 have been used in similar spiral applications. 24 is selected in our study because it can be divided by integers 2, 3, 4, 6 and 8 so there are several convenient ways of grouping all interleaves. Every n^{th} interleaf of all 24 interleaves is updated as illustrated in Fig. 4-3. For example, when n=4, interleaves [0,4,8,12,16,20] in the first phase encodings are updated then the same leaves in the second phase encoding are updated. After the same 6 leaves in last phase encoding are updated, exactly 1/4 of the k-space for one temporal phase is sampled. Then data sampling continues with the second group of leaves [1,5,9,13,17,21] from the first phase encoding through the last (each bracket represents a group in Fig. 4-4). A complete 3D data frame is reconstructed with four consecutive groups. A new frame is reconstructed for each additional group of interleaves acquired. Thus, the frame time is 1/4 of the scan time for a full k-space data set. n = 4 & 6 was used in our *in vivo* studies. A complete set of k-space data prior to contrast arrival is acquired as mask for complex background subtraction in k-space.

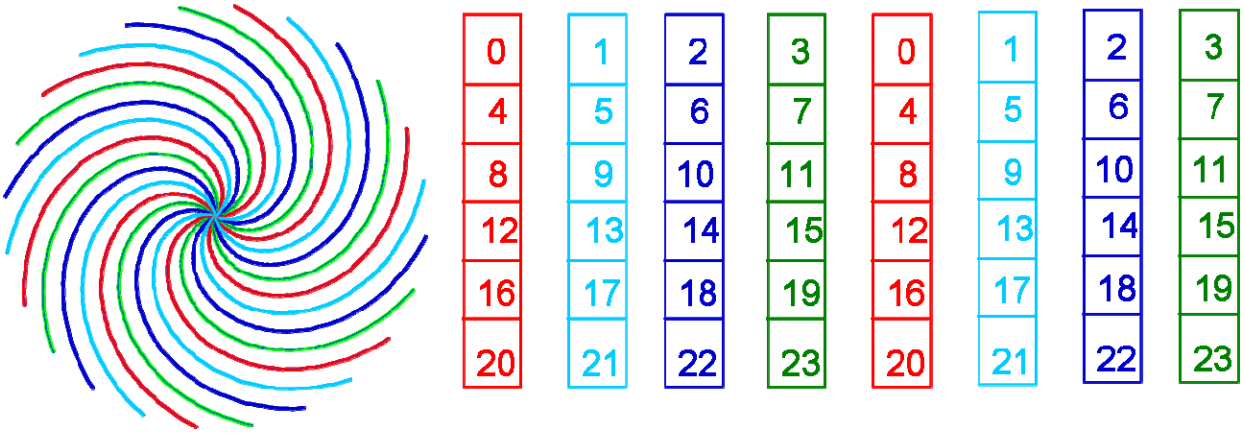

Figure 4-3 View Ordering of Multishot Spiral Trajectories

24 interleaves are divided in 4 groups for all phase encodings. K-space is updated one group at a time with a rotational symmetry.

The advantage of applying the view sharing and view ordering scheme with a multishot 3D spiral acquisition is that a k-space is segmented in an interlaced fashion such that each subset of k-space, a quarter in the example above, is rotationally symmetric with other subsets. This kind of segmentation allows a more regular update of k-space as compared to TRICKS (30) schemes based on 3D Cartesian imaging. As a result, interpolation in data construction is no longer necessary because the interpolating process for a particular time frame as show in Fig. 1-11 is replaced by a simple choice of window position in a temporal series of subsets, or "quarters", of k-space as illustrated below.

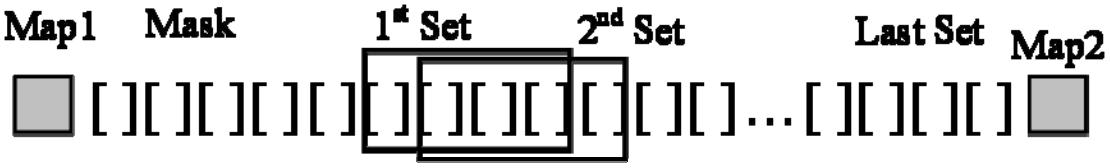

Figure 4-4 Sliding Window Reconstruction For Time-Resolved MRA

K-space is divided into 4 groups. Every time one group is updated a set of 3D images are reconstructed. Two sets of phase maps are acquired before and after the image acquisition. First half of image data is corrected with the first set of phase maps. Second half of image data is corrected with the second set of maps.

4.1.3 Off-Resonance Correction

The effective magnetic field strength that a nuclear spin experiences inside a magnet differs spatially because of main field inhomogeneity, susceptibility variation and chemical shift differences. Consequently if local precession frequency differs from the expected value by $\Delta\omega_0(r)$ then the signal equation becomes

$$s(t) = \int m(r) e^{-ik(t)r - i\Delta\omega_0(r)t} d^2r$$

Equation 4-1

In spiral imaging, sampling lines are longer than those in Cartesian or Projection imaging so that time t in the signal equation above reaches values that are not negligible. This leads to a significant off-resonance term $\Delta\omega_0(r)$. Image blurring occurs where spins precess off-resonance so it has to be corrected in post-processing.

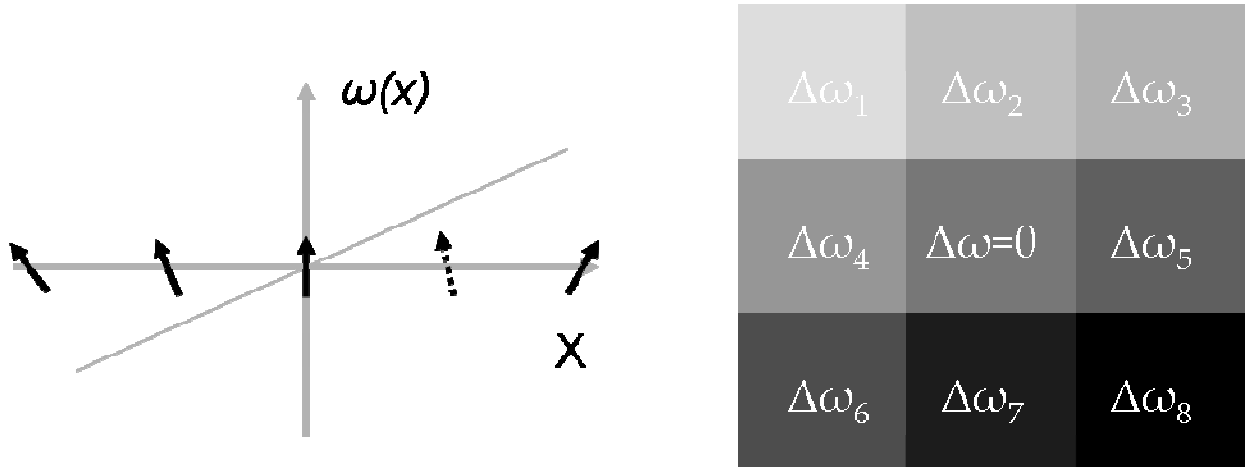

Figure 4-5 Demonstration of Frequency Segmented Correction

(Left): The inhomogeneity of the B_0 field causes spins to precess away from frequencies determined by gradient fields. (Right): An image is reconstructed with a range of off-resonances. The areas with different off-resonance frequencies are corrected and pieced together for a final image.

With the assumption that $\Delta\omega_0(r)$ varies slowly in space, frequency segmentation method (Fig.4-5, right) is often implemented to perform the correction (75,76). First, the range of the possible off-resonance (~±100Hz) is divided into several discrete frequency bins, for example $\Delta\omega_i$, $i=0\text{-}8$ (Fig.4-5, right). Suppose the off-resonance across the whole image is one of the $\Delta\omega_i$ values then the spatial function $\Delta\omega_0(r)$ in Eq. 4-1 becomes a constant $\Delta\omega_i$. The effect of that constant in Eq.4-1 can be removed easily because the time of acquisition t for every k-space point is known from the trajectory. As a result, one base image is generated with regard to an offset value $\Delta\omega_i$. When all base images are generated for $i=0\text{-}8$, each base image contains some portions of the corrected image. A final image can be constructed by selecting corrected portions and combining them into one image. There are two methods of performing the selection process. In one approach the portion in focus in a base image is mathematically selected using a focusing criterion

$$\int |\, Im\{I(r;\Delta\omega_0)\}\,|^{\alpha}\; d^2r = 0, \alpha = 1$$

Equation 4-2

The method above selects the base image with the least imaginary part, assuming that an on-resonance image point is mainly real. That assumption is good when inhomogeneity is small.

Another approach measures the spatial function $\Delta\omega_0(r)$ experimentally and uses it as an inhomogeneity field map. The selection to form a final image among the base images is made based upon the acquired maps. Each set of field maps is generated by two sets of low resolution

images (32×32) acquired separately with echo times separated by 2ms. Precessing spins go out of phase during the interval between signal excitation and data acquisition because of the field inhomogeneity. If the same image is acquired twice with 2 different echo times, two images are $M_1 = m(r)e^{i\varphi_1(r)}$ and $M_2 = m(r)e^{i\varphi_2(r)}$. The field map is

$$\Delta\omega(r) = [\varphi_2(r) - \omega_1(r)]/\Delta t$$

Equation 4-3

Frequency segmented off-resonance correction with field maps was implemented at 10 offset frequencies ranging 100Hz. For this preliminary study, an initial first field map was acquired at the beginning of the scan and was used to correct the first half of temporal frames. A second field map was acquired at the end of the scan for correcting the remaining frames. The scheduling of two field map acquisitions were shown in Figure 4-4. For comparison, the previous off-resonance correction based on minimizing the imaginary part of the image signal was also implemented (22). (The spiral reconstruction program is included in Appendix C.)

4.2 RESULTS

In vivo studies of the Time Resolved MRI sequences were performed on 5 healthy subjects and 12 patients with IRB approved written consent. 5 to 8 ml of Gd based contrast was injected at 1-2 cc/sec in all studies. Arteries in the calves and the head were imaged with a standard head coil on a 1.5T GE MR scanner with software version CNV4. Imaging parameters were: 24 spiral interleaves in k_x-k_y plane, 2640 points per interleaf, 32 k_z slice encoding in anterior-posterior direction, 30-36cm FOV, 1-3 mm slice thickness, 256x256 matrix recon matrix, 1.1/11.7 msec TE/TR, 60° flip angle, and the receiver bandwidth was ±200 kHz. The scan time was 8.9 seconds for a full volume and 5.6 seconds when only 20 of the 32 (62.5% partial sampling in k_z direction) phase encodings were acquired with zero-filling. Field map acquisitions were prescribed with 4 interleaves and 2256 points per interleaf. Two TE times were 1.1 & 3.1msec that resulted in two TRs of 15 & 17msec respectively. Parameters of phase encoding, slice thickness and FOV were the same as image prescription such that field maps overlapped exactly with images for accurate correction. Each acquisition of field maps was 3 seconds. A complete execution of the sequence

was 40-60 seconds that was comparable to the time of contrast passage through a subject's system. A complete data reconstruction included off-resonance correction and sliding window reconstruction on 4 full sets of data and 2 sets of phase maps as described in Figure 4-4. We achieved high quality 3D MRA consistently at a fast frame rate (1 sec per frame) using only 5 to 8 cc Gadolinium contrast injection.

The off-resonance correction method using acquired field maps produced successful results. The figure 4-6 below shows a typical process. Figure 4-6 (a) is a raw image reconstructed using one full data set which contains spiral aliasing and off resonance blurring. Figure 4-6 (b) shows the image reconstructed after the mask data was subtracted in the k-space from the data. The aliasing artifacts are removed. Figure 4-6(c) is the image after correction using the acquired phase map shown in (d). The final image is free of any blurring inside FOV and details of the arteries are shown with sub-millimeter resolution.

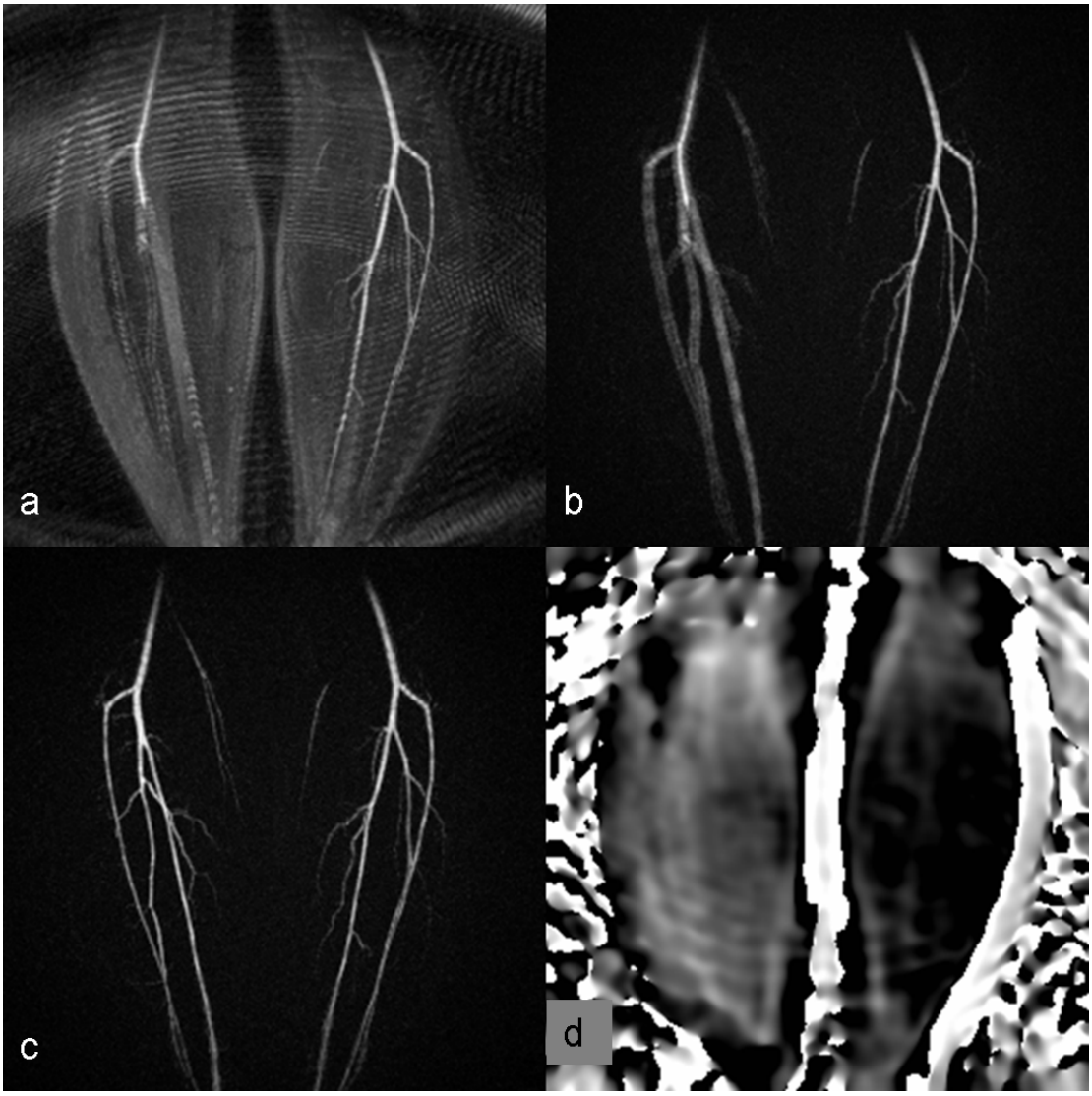

Figure 4-6 Procedure and Result of Off-Resonance Correction

(a) A spiral image with aliasing and off-resonance blurring. (b) Aliasing is removed after mask subtraction. (c) Blurring is removed after off-resonance correction. (d) The phase map used to perform off-resonance correction. (The source code of the spiral reconstruction is included in Appendix B.1. The instruction for this figure is included in Appendix C-18.)

The off-resonance correction method without using an acquired field map was implemented for comparison (Fig. 4-7). The images demonstrate that the field map based off-resonance correction reveals images below with better details than the imaginary minimization based correction. The reason for the inaccuracy of the imaginary minimization based off-resonance correction is that small vascular signals are buried in the background noise. In order to identify the imaginary part of the signal in a voxel, an average over a 16x16 voxels was used. A voxel containing a small vessel contributed only a very small portion to this average.

Consequently, small vessels may not be properly identified using this algorithm. This problem does not occur in the field map based correction, because the inhomogeneity field varies smoothly even around small vessels and thus, can be estimated correctly with a low resolution map.

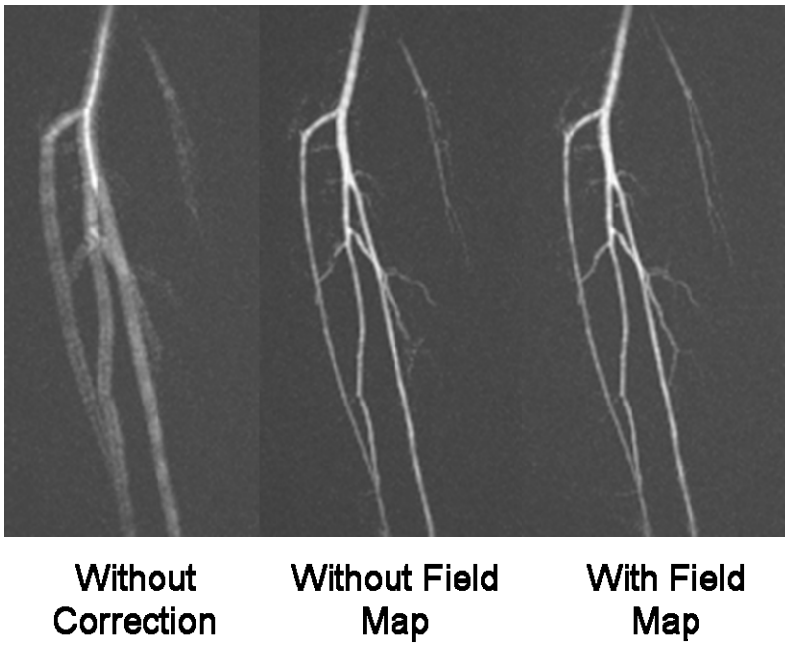

Figure 4-7 Comparison of Two Off-Resonance Correction Methods

The volunteer images below Fig. 4-8 demonstrates the superb resolution achieved by 3D spiral imaging in 0.5mm. The instruction for this figure is included in Appendix C-19.

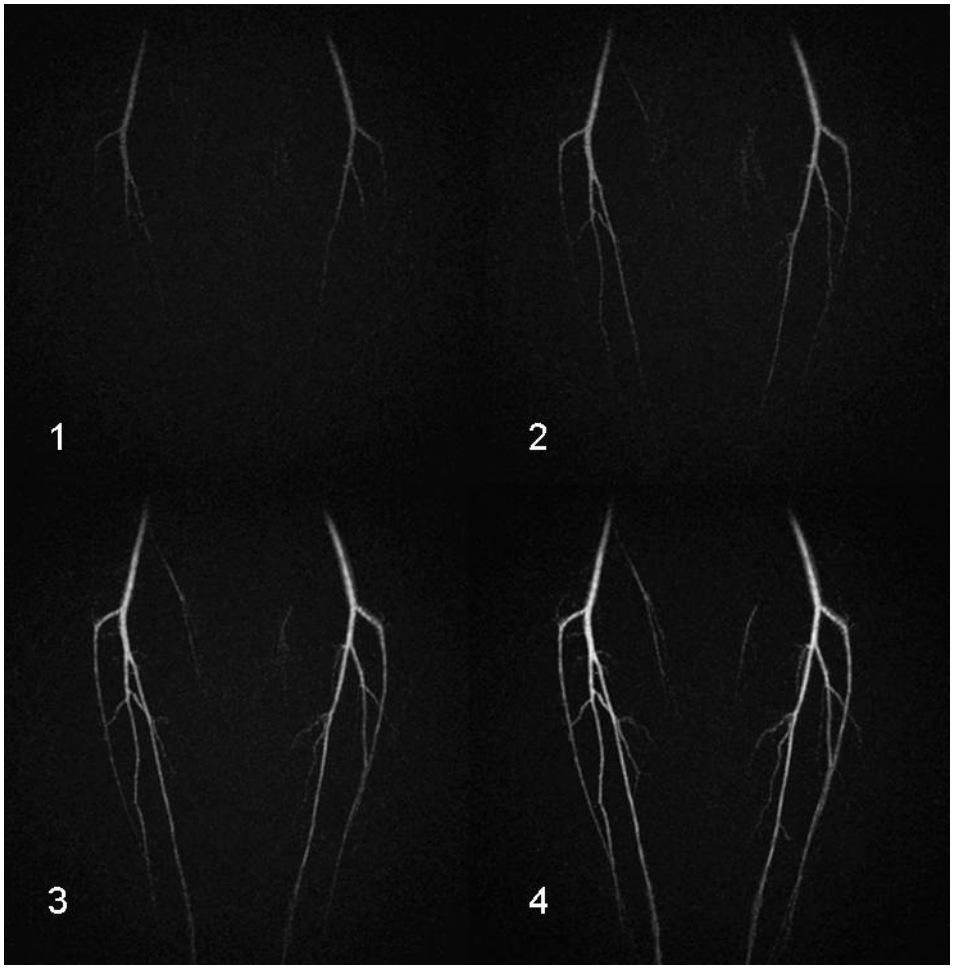

Figure 4-8 4 Frames of Time-Resolved MRA in a healthy volunteer

4 consecutive time frames of the 3D images projected to 2D show the passage of blood through arteries in the calves. The instruction for this figure is included in Appendix C-19.

The patient images in Fig. 4-9 demonstrate the wide range of application of our technique. The top row images were acquired from a patient whose blood arrived in one leg 2 seconds earlier than in the other. This case of differential flow can only be captured with time resolved acquisition and it was captured in 3D with our technique. An application of this technique in the head shows arteries in a volume without artifacts.

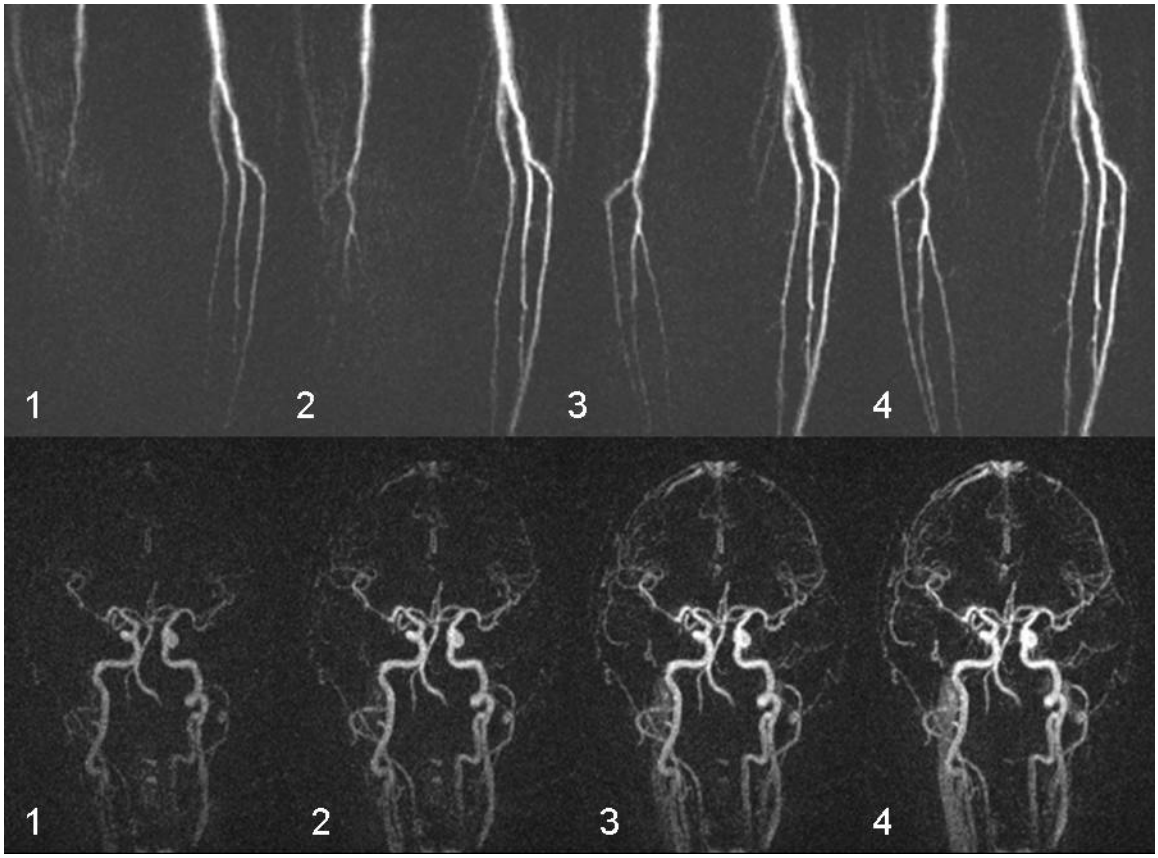

Figure 4-9 Results of Time-Resolved MRA in Human Studies

(Top) 4 consecutive time frames of 3D images projected to 2D show arterial blood arrive in one leg earlier than in the other. This case of differential flow can be captured by time resolved imaging. (Bottom) 4 consecutive time frames of 3D images projected to 2D show passage of blood through arteries in the head. The instruction for this figure is included in Appendix C-19.

The 3D capability of this technique is best demonstrated in (Fig. 4-10) with two images in one temporal frame but projected to 2 different planes. A lesion shown in Fig. 4-10 (b) is not displayed adequately in the standard coronal plane shown so the constructed 3D data set is rotated for a better projection shown in Fig. 4-10 (a).

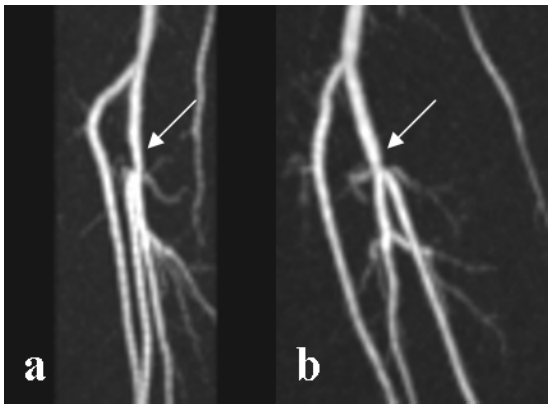

Figure 4-10 Angiograms of Two Projections enabled by 3D Imaging

One 3D image projected in 2D along two different directions shows details of the arteries in (b) that cannot be resolved by 2D imaging. The image on the left was rotated from the standard coronal image on the right to better illustrate a lesion. The instruction for this figure is included in Appendix C-19.

4.3 CONCLUSION

In summary, a Time Resolved 3D spiral magnetic resonance angiography technique was developed that generated high quality 3D angiograms at fast frame rate using a small gadolinium dose. The time resolved 3D spiral MRA technique is feasible in clinical practice and may provide substantial simplification and increased reliability of the 3D MRA examination by eliminating the need for contrast timing. The success of the Time Resolved 3D acquisition technique relies on the inherent higher efficiency of spiral imaging and the optimum parameters chosen for this application. A multishot spiral pattern is suitable for high resolution imaging. Off-resonance correction is necessary to show details of arteries across a FOV of 30cm to 40cm. When spiral imaging is applied to other applications the entire protocol will have to be redesigned to meet different requirements imposed by signal source, desired resolution and dimensional parameters.

5.0 QUANTITATIVE ANALYSIS OF SPIRAL IMAGING

The uniqueness of spiral sampling has been demonstrated in two applications, MRSI and Time Resolved 3D MRI. The advantage of spiral data acquisition is clear but questions are also raised by its difference from Cartesian sampling. An MRI sampling pattern using a Cartesian grid is determined by image parameters in a straightforward fashion. Specifically, the length of each sampling line is determined by image resolution because k-space is sampled from -(1/FOV)×N/2 to (1/FOV)×N/2 in a straight line. The number of lines is the prescribed resolution N. Using spiral sampling, on the other hand, one can choose the length and number of sampling lines for specified resolution and FOV. Furthermore, the density distribution of k-space coverage can be controlled by the design of trajectories for under/over sampling considerations. This task starts by refining the algorithms of spiral trajectory generation.

5.1 SPIRAL TRAJECTORY GENERATION[4]

There have been several publications discussing the design of spiral trajectories. In Glover's early work (77), he described a simple analytical algorithm to generate an Archimedean spiral trajectory. The mathematical description of an Archimedean spiral in k-space is

$$k_c(\tau) = k_{max}\tau e^{i\omega_k\tau},\ \tau = [0,1]$$

Equation 5-1

$k_c(\tau)$ is the desired trajectory in the k-space with a constant sampling density. k_{max} is the radius of the k-space coverage. The angular coefficient ω_k in the exponential term $e^{i\omega_k\tau}$ determines the

[4] The source code of the trajectory generation is included in Appendix A.

length of each trajectory. $|k_c(0)|=0$ at the center of the k-space when $\tau=0$ and $|k_c(1)|=k_{max}$ when $\tau=1$. τ can be solved as a function of time (77) so that

$$\tau(t)=\frac{\frac{1}{2}\beta t^2}{\frac{1}{q}+\frac{\beta t^{4/3}}{2\alpha}},\ \alpha=\left(\frac{9}{4}\frac{S_m\gamma}{k_{max}\omega_k{}^2}\right)^{1/3},\ \beta=\frac{S_m\gamma}{k_{max}}$$

Equation 5-2

S_m is the maximum slew rate (the maximum rate at which the gradient amplitude can be ramped). q controls the behavior of $\tau(t)$ when t is small, and the typical value of q is 0.2. The two functions (Eqs. 5-1 and 5-2) together provide a function *k(t)*. The gradient waveform *g(t)* as a function of time is given by a derivative of *k(t)*

$$g(t)=\frac{k'(t)}{\gamma}=\frac{k'(\tau)\cdot\tau'(t)}{\gamma}$$

Equation 5-3

As the formula of $k_c(\tau)$ implies, the radial distance $|k_c(\tau)|= k_{max}\tau$ is proportional to the angular distance $\theta=\omega\tau$, so the distance between neighboring spiral lines is constant. Therefore, this algorithm generates a constant density trajectory. The highlight of this algorithm is that the solution of τ as a function of time $\tau(t)$ produces a finite slew rate when t is small, as oppose to earlier work by Duyn and Yang (78) in which slew rate approaches to infinity at the beginning of the trajectory.

Since low spatial frequency components at the center of k-space always contain more image information than high spatial frequency components, it is more efficient to oversample the center of k-space and undersample the outer k-space region. This can be achieved by a variable density spiral design. In Dong-hyun Kim et al's paper (79), the authors developed a simple analytical algorithm to generate spiral trajectories that provide a gradually reducing sampling density from the center to the edge of the k-space. The corresponding density function reaches its maximum at the center and goes down in a polynomial fashion. The mathematical formula of such trajectories is

$$k_v(\tau)=k_{max}\tau^n e^{i\omega\tau},\quad \tau=[0,1]$$

Equation 5-4

$k_v(\tau)$ is desired spiral trajectory with a variable density. The additional parameter n is an oversampling factor so that the larger the n is, the more k-space center is oversampled. Kim's

paper provides an example of *n=4* where the sampling density varies greatly in the k-space. This trajectory spends too much time in the center so that the resulting trajectory is too long. In many MRI applications where scan time is limited, the oversampling factor of the center has to be only slightly larger than 1. For example, when spiral trajectories are used in Time Resolved Contrast Enhanced MRI, oversampling of the center of k-space is used to minimize the ringing artifacts (80) that are commonly associated with spiral trajectories (Fig. 5-1). Ringing artifacts are caused by signals from outside the FOV (~30cm) but inside the coverage of a receiver coil (~40cm). So in this case the oversampling factor at the center should be about 1.3 (~40/30). The length of spiral trajectories is proportional to the square of the oversampling factor. An oversampling factor too large (~4) would lead to much longer scan time, resulting in poorer temporal resolution. Unfortunately, Kim's analytical form breaks down when *n* is smaller than 2 because it produces an infinite slew rate at the center of k-space. Combining constant density and variable density can be used to resolve the issue.

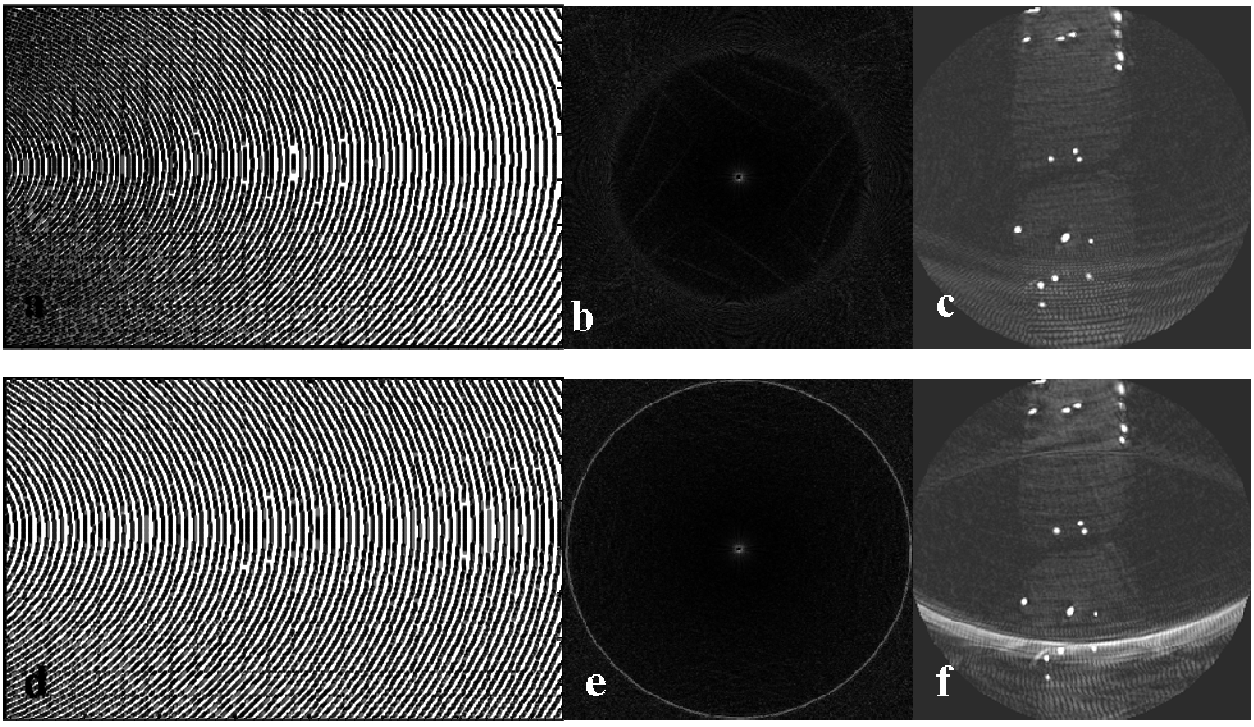

Figure 5-1 Trajectories, PSFs and Images of Variable and Constant Density Spiral

Variable density spiral (a) trajectories produce an image with mild aliasing spread out in a range while constant density spiral (d) produce an image with strong aliasing focused around a ring. (a) and (d) show a comparison of two trajectories. (b) and (e) show a comparison of two Point Spread Functions with equal FOV. (c) and (f) show a comparison of the respective phantom images. (f) shows strong aliasing artifacts due to constant density spiral trajectory while (c) shows mild aliasing due to variable density

trajectory. The phantom consisted of Gd-DTPA doped water sealed in plastic tubing submerged in 2 bottles of water. The inner diameter of the plastic tubing was 5mm and the concentration to water was about 1:10 to produce strong signals compared to water in the background. The bottles were Coke bottles. Two dimensional images of the phatom show cross sections of the tubing as high intensity dots while background water intensity is low. Constant density spiral [$k_c(\tau)=k_{max}\ \tau\ e^{i\omega\tau}$] is used at center of the k-space and variable density spiral [$k_v(\tau)=k_{max}\ (a\ \tau^2+b\ \tau+c\ )e^{i\omega\tau}$] is used for the rest of the trajectory. The source code of generating the variable density spiral is included in Appendix C-20.

In our work, a constant density spiral trajectory is used to sample the center of the k-space. The density is set as 1.3 times the nominal density *(Δk'=Δk/1.3, Δk=1/*FOV*)*. As the trajectories reach a certain point in k-space *(~25%* of k_{max}*)*, the algorithm switches to a variable density formula and the sampling density starts to decrease. When the trajectories reach the end, the density is about 0.7 times the nominal density, which is proportional to FOV . This more sophisticated trajectory design provides detailed control over the sampling pattern. The oversampling at the center is constant. The size of the center region is clearly defined by the boundary between constant density region and variable density region. The undersampling of the outer region is controlled by the parameters of variable density trajectory formula. The C code and instructions for the variable density spiral trajectory is included in Appendix A.

The challenge of our approach is to solve the boundary conditions that connect two analytic formulas. The formula for constant density $k_c(\tau)=k_{max}\ \tau\ e^{i\omega\tau}$ can be imported unchanged to generate the inner part of the trajectories. The variable density formula has to be modified from $k_v(\tau)=k_{max}\ \tau^n\ e^{i\omega\tau}$. Here, τ^n will be replaced by a general polynomial function $f(\tau)$ such that $k_v(\tau)=k_{max}\ f(\tau)e^{i\omega\tau}$. This polynomial function $f(\tau)$ needs to satisfy three conditions outlined below and has at least three parameters. The simplest solution is a parabolic function $f(\tau)=a\ \tau^2+b\ \tau+c$, which is a special case of τ^n when $n=2$. The three conditions can be determined by closely examining the requirements at the point where the formula switches from constent density ($k_c(\tau)$) to variable density ($k_v(\tau)$) at the switching point when $\tau=\tau_s$.

First, $k_c(\tau) = k_v(\tau)$, i.e. a continuous trajectory. Since the angular terms are identical, this leads to the first condition $f(\tau_s)= a\ \tau_s^2+b\ \tau_s+c = \tau_s$.

$$a\tau_s^{\,2} + b\tau_s + c = \tau_s$$

Equation 5-5

Second, $k_c'(\tau_s) = k_v'(\tau_s)$ because the slope of a trajectory is proportional to the gradient applied and the gradient has to be continuous.

$$\left(a\tau^2 + b\tau + c\right)'_{\tau=\tau_s} = 2a\tau_s + b = 1$$

Equation 5-6

Lastly, at the end of a trajectory, $f(\tau) = 1$ for a given $\tau = \tau_e$. This condition guarantees when $\tau = \tau_e$, $f(\tau) = 1$ and $k = k_{max}$.

$$a\tau_e^2 + b\tau_e + c = 1$$

Equation 5-7

Here, τ_e controls the total length of a spiral line and implicitly controls the desired amount of undersampling at the outer region. Three parameters a, b and c can be solved by the three conditions above with two given input values τ_s and τ_e.

It is worth mentioning that $f(\tau)$ equals 1 at the end of a trajectory while τ does not. When τ ends at a number τ_e smaller than 1, the total angle traveled in k-space $\theta = \omega\tau$ is reduced, or there are less turns in the k-space for a spiral.

The gradient wave form is given by $g(t) = k'(t)/\gamma = (1/\gamma)\ \tau'(t)k'(\tau)$ that implies gradient generation depends on two functions, the shape of the trajectory $k(\tau)$ and how it is traced out in time $\tau(t)$. Since $g(t)$ has to be continuous at the connection point, $\tau'(t)$ and $k'(\tau)$ are both continuous. The expression of $\tau(t)$ can be solved using boundary conditions similarly to the procedure of determining $k(\tau)$ above.

$\tau(t)$ and $\tau'(t)$ both should be continuous to satisfy two conditions $\tau_c(t) = \tau_v(t)$ and $\tau_c'(t) = \tau_v'(t)$. The values at the end of the constant density spiral can be calculated from the analytical formula of $\tau_c(t)$ given above (Eq. 5-1). An analytical solution of $\tau_v(t)$ for variable density spiral where $(n=2)$ is chosen to be a simple polynomial function in the form of $\tau_v(t) = (\mu t)^{1/2}$ where μ is a linear parameter (79) because this solution is not exclusive. (A simple analytical form for $\tau(t)$ such as $\tau_v(t) = (\mu t)^{1/2}$ will enable real time generation of the trajectories based on prescribed image parameters.) Both μ and t are unknown and only the value of the power, ½, is pre-determined by the power of $f(\tau) = a\tau^2 + b\tau + c$,2. The parameters μ and t can be solved with the two conditions:

$$\tau_v(t) = (\mu t)^{\frac{1}{2}} = \tau_s$$

Equation 5-8

$$\tau_v'(t) = \frac{1}{2}(\mu t)^{-1/2}\mu = \tau_c'(t)_{\tau=\tau_s}$$

Equation 5-9

Note that *t* here is a parameter and may not necessarily be continuous as a function of time.

In conclusion, a trajectory starts with a given constant density and covers the k-space center (*25%* in radius). In this constant density region, the trajectories follow the formulas of $k_c(\tau)$ and $\tau_c(t)$ as given in Glover's paper (77). When τ reaches τ_s, we solve the three linear Eqs [5-5],[5-6] and [5-7] to obtain the desired $k_v(\tau)=k_{max}(a\tau^2+b\tau+c)e^{i\omega\tau}$ for the variable density region. The order of the function $k_v(\tau)$ leads to the order of $\tau_v(t)$ to be $\tau_v(t) =(\mu t)^{1/2}$. Then, we calculate the derivative of $\tau_c(t)$ with the constant density formula. With the derivative of $\tau_c(t)$ and the transition value τ_s, μ and *t* in $\tau_v(t) =(\mu t)^{1/2}$ can be solved. Then the trajectory continues on with formulae $k_v(t)$ and $\tau_v(t)$. This spiral trajectory algorithm is sophisticated so that there are three parameters to control the k-space coverage. The oversampling and undersampling of the k-space can be customized for any specific purpose. Optimal parameters can be found for various spiral applications. With FOV and resolution determined by a scan prescription, the only arbitrary parameter for a trajectory generation algorithms is the number of interleaves, *nl*. The parameter *nl* is important because it directly leads to the length of each acquisition and the number of excitations needed for an image. The following section attempts to select the choice of *nl* by quantitizing the quality of an image.

5.2 ANALYSIS OF SPIRAL IMAGING PROCESS

A spiral generation algorithm can mathematically satisfy the same sampling requirements with any given number of spirals, assuming the source of the signal is ideal. The criteria for choosing spiral parameters according to individual application was done experimentally in our studies. It will be beneficial to develop a quantitative method to evaluate and compare the performance of a given set of spiral parameters. Such method may provide a guideline for choosing optimal spiral parameters such as the appropriate number of spiral interleaves and leaf lengths to achieve good spatial and temporal resolutions.

5.2.1 Theory

A general method of analyzing MR imaging process was developed by Donald Twieg (81). The method can be carried out to evaluate spiral sampling. The method starts by inspecting a simplified signal equation in one dimension

$$s(t) = \int e^{-t/T_2^*} f(x) e^{-i2\pi k(t)\cdot x} dx$$

Equation 5-10

f(x) stands for a one dimentional image in the image space. The first term in the integral describes T_2^* relaxation. The resulting signal equation plotted versus time is FID. The influence of B_0 inhomogeneity is neglected. If the Fourier Transformation of *f(x)* is $F[k(t)]$, then the signal in the k-space becomes

$$s(t) = e^{-t/T_2^*} F[k(t)]$$

Equation 5-11

First, the final MR image is a linear estimation of $F[k(t)]$. The exponential decay term is the first source of estimation error introduced by the imaging procedure. The T_2^* decay on every sampled point in k-space is determined when that point is sampled. The distribution of T_2^* decay in k-space is, therefore, determined by the sampling pattern. A spiral sampling pattern is represented as a group of FID's, as $s_i(t)$, $(i=1,\ldots,nl)$, where *nl* is the number of spiral interleaves. The j^{th} sampling point along spiral number *i* can be denoted as s_{ij}, which comes from the signal $s_i(t)$ recorded during intervals $[t_{ij}, t_{ij}+dt]$. Each sampling pattern determines a matrix

$$s_{ij} = dt \cdot e^{-t_{ij}/T_2^*} F\left[k\left(t_{ij}\right)\right] \text{ or } s_{nl} = dt \cdot e^{-t_{nl}/T_2^*} F\left[k_{nl}\right]$$

Equation 5-12

where *nl* here can serve as an index of an entire pattern because *nl* is the only free input parameter to a trajectory. A collection of trajectories are labeled with one index *nl*, and all other parameters of a spiral pattern including resolution, FOV is constant for comparison. The function $F[k_{nl}]$ represents a particular sampling of the k-space. The ideal k-space data is denoted as F_d while the sampled version is $F_d(nl)$. The observed MR data of this particular pattern $F_b(nl)$ with *nl* interleaves is

$$F_b(nl) = H(nl) F_d(nl) + V(nl), \quad \text{and } H(nl) = A \cdot dt \cdot e^{-t_{nl}/T_2^*},$$

Equation 5-13

where $F_b(nl)$ is the observed data, $F_d(nl)$ is the discrete data, $H(nl)$ is a decay function and $V(nl)$ is a noise function. If the image reconstruction procedure including gridding and filtering is denoted as H_g, the reconstructed image F_e as an estimate of the source will be

$$F_e(nl) = H_g F_b(nl) = H_g\left[H(nl)F_d(nl) + V(nl)\right].$$

Equation 5-14

The estimation error of this particular sampling pattern in k-space with *nl* spirals is therefore

$$F_{err}(nl) = F_d - F_e(nl).$$

Equation 5-15

In the image space, where Inverse Fourer Transform (iFT) is performed so that $f=iFT[F]$. The estimation error is

$$f_{err}(nl) = f - f_e(nl)$$

Equation 5-16

The quadratic mean (root-mean-square) of $f_{err}(nl)$, $\langle f_{err}(nl)^2 \rangle^{1/2}$ can be used as a scalar indicator of this particular imaging pattern.

5.2.2 Results

The above method is applied numerically to a simulated image to evaluate how small structures are resolved.

Figure 5-2 A Target Image for Simulation of the Spiral Imaging Process

This image consists a collection of holes in a disc. It tests the ability of a spiral imaging technique to resolve small structures on the order of 1 centimeter. (The matlab source code to generate this reference image is included in Appendix C-21).

The FOV of the image is 16cm. The diameters of the disc and the small holes are 8cm and 1 cm, respectively. The spacing among the small holes is 1cm. The primary purpose of using this pattern above is to see how an object smaller than or equal to 1cm is resolved by a low resolution MRSI image. One advantage of this pattern is that each circular pattern, a disc or a hole, corresponds to a 2D analytical function in k-space instead of a digitized 2D matrix in k-space. The analytical k-space function is

$$F_{2D}\left[circ(r)\right] = \frac{J_1(2\pi\rho)}{\rho} \equiv jinc(\rho), \quad \rho = \sqrt{k_x{}^2 + k_y{}^2}, r = \sqrt{x^2 + y^2} \ .$$

Equation 5-17

where $circ(r)=1$ when $(x^2+y^2)^{1/2}\leq 1$ and $circ(r)=0$ otherwise. k_x and k_y are coordinates in the k-space and ρ is the radius in k-space. J_1 is a Bessel function of the first kind, where $J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\alpha+1)}\left(\frac{x}{2}\right)^{2m+\alpha}$, where $\alpha = 1$. When a trajectory is chosen for evaluation, the coordinates of its sampling points will be used as values of k_x and k_y to evaluate the ideal k-space samples at that location. The advantage of using an analytical pattern in k-space is to avoid unwanted error in interpolating a 2D matrix.

The next step is to generated a set of k-space trajectories to be evaluated. These trajectories share the following imaging parameters: FOV=16, resolution=32 and oversampling factor is 1.2 at the center of k-space. The trajectories are only distinguished by the number of interleaves used to meet these imaging parameters. The following table shows the prescribed number of leaves and the resulting lengths of the trajectories.

Table 5-1 Leaf Lengths of All Spiral Trajectories for Comparison

# of leaves	1	2	4	6	8	12
Sample Pts	3230	1610	842	612	506	404
# of leaves	16	20	24	32	48	64
Sample Pts	356	336	312	288	264	252

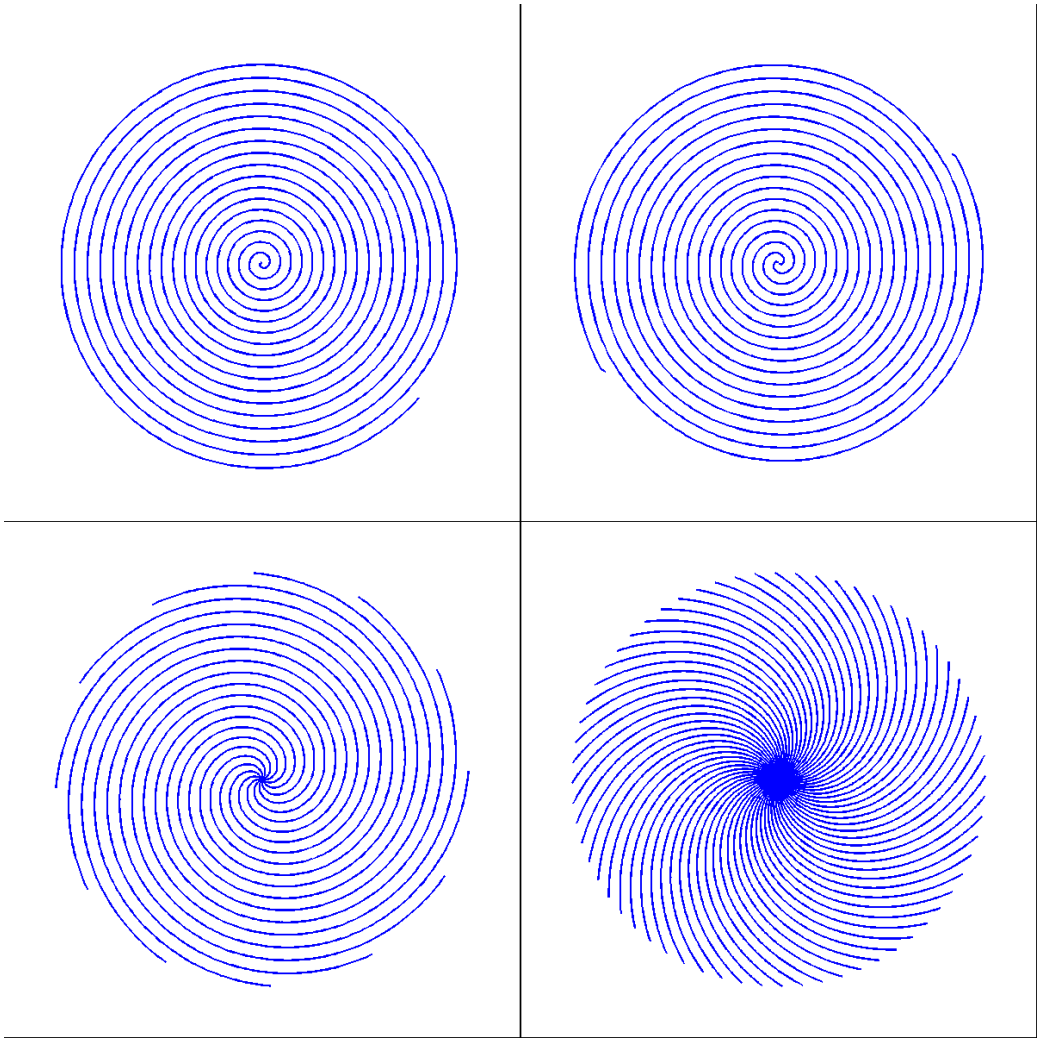

Figure 5-3 Four Examples of Spiral Trajectories

Four spiral trajectories were generated to meet the identical resolution and FOV. The numbers of spiral interleaves are 1, 2, 12 and 64. (Matlab codes for this figure are included in Appendix. C-22.)

Four trajectories are selected to illustrate the difference in k-space coverage. Mathematically each trajectory is *nl* arrays of k-space coordinates (k_x, k_y). Each pair of coordinates determines t_{nl} in the function below for the observed signal at that location

$$F_b(nl) = A \cdot dt \cdot \exp(-t_{nl} / T_2^*) F_d(nl) + V(nl)$$

Equation 5-18

Introducing T_2^* decay into Eq. 5-18 as a source of estimation error is straightforward because the k-space points in a trajectory are generated linearly in time with a rate of 4μs/pts, which is the resolution of the digital waveform. The value of T_2^* is estimated from *in vivo* data.

The determination of the amplitude of noise as a function of *nl* is based on the assumption that the distribution of the noise is Gaussian and it is scaled with the squared root of the number of repetition (NEX). This assumption is adequate in a standard NMR experiment. So if a one-shot trajectory is repeated 64 times then the acquired SNR is 8 times the SNR of a single FID repeated once. It is denoted SNR_u or normalized SNR. SNR_u is determined using *in vivo* data, as well.

A single shot acquisition will have high SNR due to a large number of averaging but its excessive length will result in severe T_2^* decay of the signal. A multishot trajectory, on the other hand, suffers less signal loss due to T_2^* decay but more due to noise. A simple strategy is that if multishot does not offer significant advantage in the final image, less interleaves is advantageous because of the freedom of choosing NEX to adjust scan time.

The final formulation of estimating k-space data is

$$s_{nl}(i) = \sum_p \mathbf{jinc}_p\left(k_x(i), k_y(i)\right) \cdot \mathbf{exp}\left(\frac{-i\delta t}{T_2^*}\right) + \mathbf{awgn}\left(SNR_u \cdot \sqrt{NEX_{nl}}\right)$$

Equation 5-19

The summation of *jinc* functions represents the target image shown in Fig. 5-2. Each disc or circular hole in the image is represented by a *jinc* function distinguished by an index *p* in Eq. 5-19. Each *jinc* function is weighted, scaled and phase shifted in k-space to represent different amplitudes, sizes and locations in the image space. The "**awgn**" (Add White Gaussian Noise, a syntax in Matlab) represents a Gaussian noise source whose amplitude is controlled by an SNR parameter that increases with NEX. NEX, on the other hand, is determined by *nl*. For example, when *nl*=1 and NEX is chosen to be 64 so that the total number of acquisition is 64. When *nl*=2 and the total number of acquisition is fixed at 64, NEX will be 32. When *nl*=32 each leaf is only repeated twice.

The last two parameters in Eq.[5-19], T_2^* and SNR_u are determined by inspecting experimental data. Below are five plots of PUFA echoes acquired from 5 different subjects. The PUFA echoes also provided 1D PUFA spectra displayed in Fig. 2-14. These experiments were performed with Sel-MQC 2D experiments discussed in Chapter 2, and Spiral Sel-MQC experiments discussed in Chapter 3 with the same experimental settings. Slice thickness was 1cm. NEX was 8, 16 or 64.

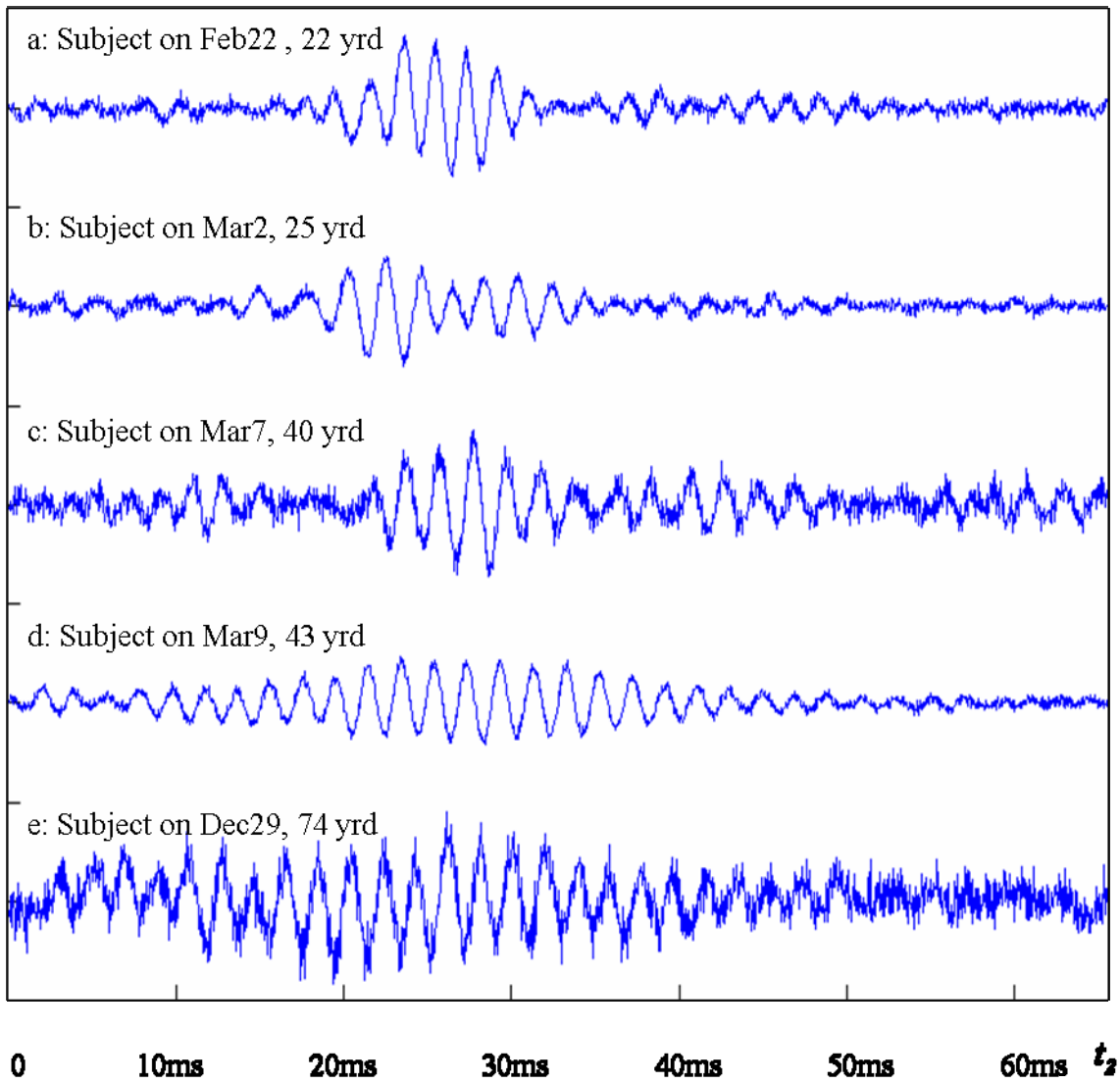

Figure 5-4 Five *In Vivo* PUFA Echoes for SNR Estimation

Five plots of *in vivo* PUFA echoes is the basis for an estimation of the signal-to-noise ratio (SNR). The real parts of the five echoes are plotted above. The SNRs of the five plots are listed in Table 5-2. (Matlab codes for this figure are included in Appendix. C-23.)

SNR_u depends on parameters including the mean of the signal $\bar{s} = \langle s(t) \rangle$, the standard deviation (STD) of the noise $\sqrt{\langle s(t) - \bar{s} \rangle^2}$ and NEX so that

$$SNR_u = \frac{\langle s(t) \rangle}{\sqrt{\langle s(t) - \bar{s} \rangle^2} \cdot \sqrt{NEX}}$$

Equation 5-20

The following table shows the results for the 5 measurements plotted above (a to e)

Table 5-2 Five Estimates of In Vivo PUFA Echoes

Matlab codes for this table are included in Appendix. C-23.

	a	b	c	d	e
Signal mean	300	109	117	193	28.2
Noise STD	30.7	41.0	27.9	25.5	9.18
SNR	9.77	2.66	4.19	7.57	3.07
NEX	64	8	8	64	8
SNR_u	1.22	0.94	1.48	0.94	1.09

The value of SNR_u is 1.13, which is the mean of the experimental values of the 5 experiments. Matlab codes for the calculations in the table are included in Appendix. C-23.

The T_2^* constant will be estimated by fitting the decay curve of the forth echo (Fig.5-4d) because of its highest SNR and ideal echo shape. The form of the fitting function is chosen to be a one-term exponential function.

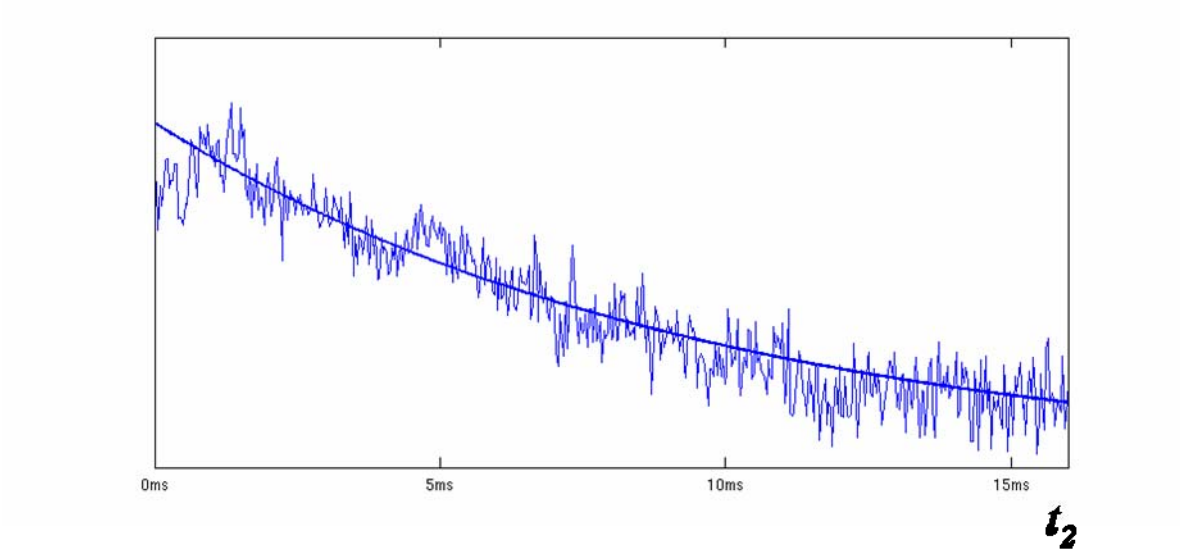

Figure 5-5 An Exponential Fitting of a partial Echo of PUFA

The PUFA partial echo plotted Fig. 5-4 (d) is fitted for an estimation of its T_2*. The magnitude of the data's decay curve is used for the exponential fitting. Matlab codes for the fitting prodedure are included in Appendix. C-23.

The result of the exponential fitting is *s(t)~exp(-t/8ms)* so the value of T_2^* will be 8 ms.

As SNR_u and T_2* are estimated, Eq. 5-19 can be evaluated with 12 sets of spiral trajectories to simulate 12 sets of k-space data of the same target image. Each set of k-space data

consisting of *nl* interleaves is passed through spiral reconstruction program to produce an image. The following are four examples corresponding to the four trajectories plotted in Fig. 5-3.

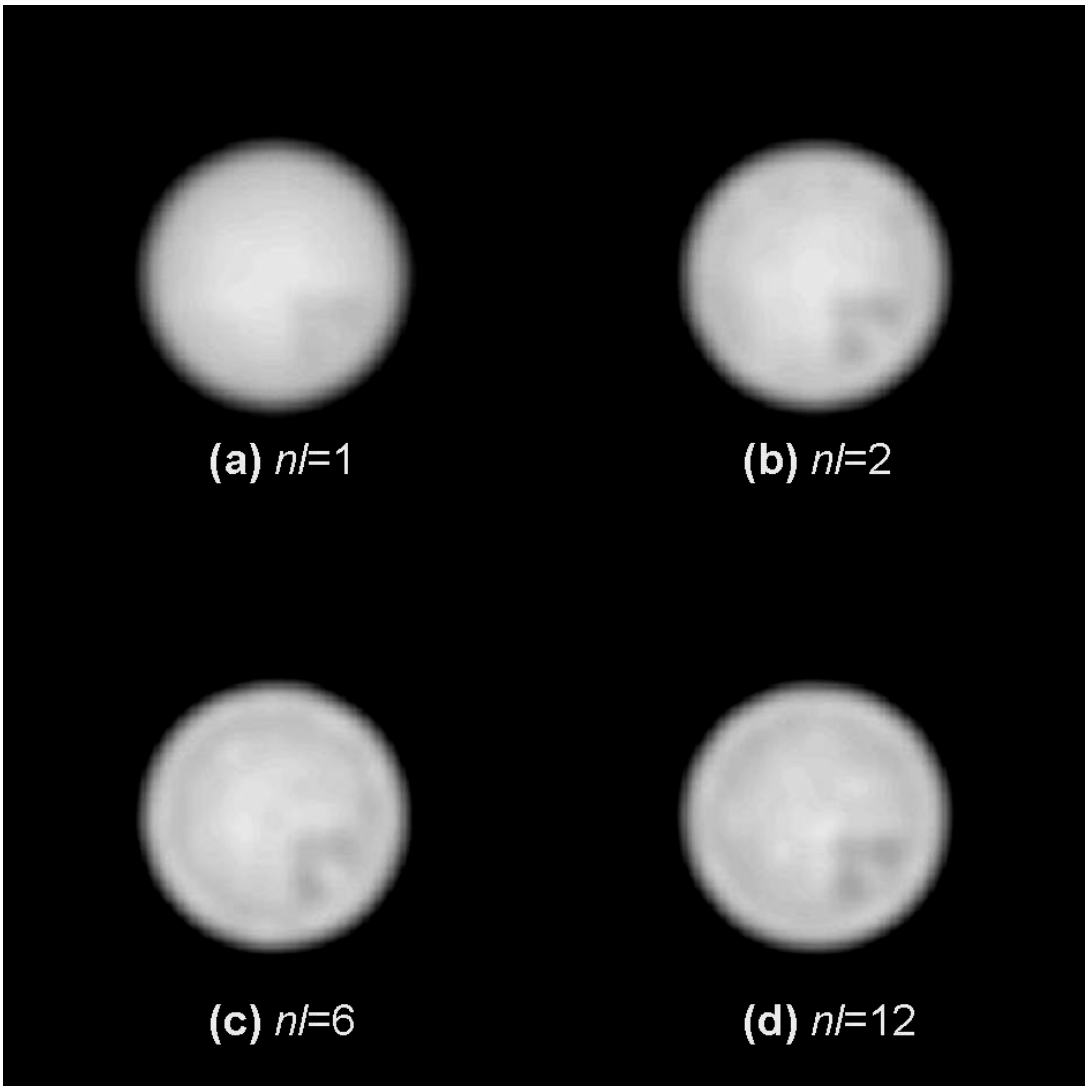

Figure 5-6 Four Examples of Simulated Spiral Images

The single shot image loses too much high resolution information. The images acquired using 2, 12 and 64 interleaves (b-d) show similar resolution. 2 shot spiral is the best choice. There is mild gibbs ringing artifacts, .i.e. the small edge around the circle. It is visible because the circular pattern has a sharp edge and the image resolution (32×32) is too low to resolve it. A stronger fermi filter can remove it. *In vivo* anatomical patterns don't have sharp edges so low resolution images are fine. (Matlab codes for this figure are included in Appendix. C-24.)

The one-shot image loses too much high resolution information due to the long spiral readout. The improvement from *nl=1* to *nl=2* is significant because the readout length is cut in half.

Three dots of 1cm in diameter and 1cm spacing are resolved in Fig.5-6 (b, c & d). The advantage of the two images of *nl=12* and *64* over *nl=2* is not significant.

Before calculating the estimation errors of each image with respect to the reference image (Fig. 5-2), it is important to normalized the intensities. This procedure scales the average intensity of 2nd quadrant of the image to 1.0. The ROI (Region of Interest) used to calculate the quadratic mean of the estimation error is a square located in the 4th quadrant.

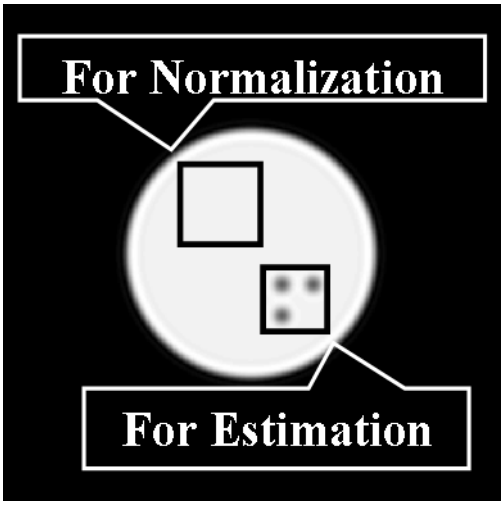

Figure 5-7 Estimating Image Error After Normalization

The average pixel intensity in the box in the 2nd quadrant was used to normalize both the reference image and the simulated image. The quadratic mean of the estimation error in the box in the 4th quadrant is used as an indicator of the image technique. The matlab code for the normalization of image intensity is included in Appendix. C-21.

The reference image shown in Fig.5-7 is denoted by *f* in Eq. 5-16. Each simulated spiral image and the reference is passed through Eq. 5-16 that results in a number that is related to the performance of the imaging method for that image. Each number is determined by a trajectory and a pair of parameters, T_2^* and SNR_u. So one pair of (T_2^*, SNR_u) together with 12 predetermined sets of trajectories generate a plot of 12 points. T_2^* and SNR_u values estimated from experimental data are chosen to produce the plot (Fig.5-8) of estimation errors for 12 sets of trajectories. The plot shows a curve similar to an exponential decay. The estimation error drops quickly from *nl*=1 to *nl*=2 and starts to stabilize. This behavior suggests single shot spiral is the worst configuration when T_2* decay is significant and all the other choices of *nl* produce similar images.

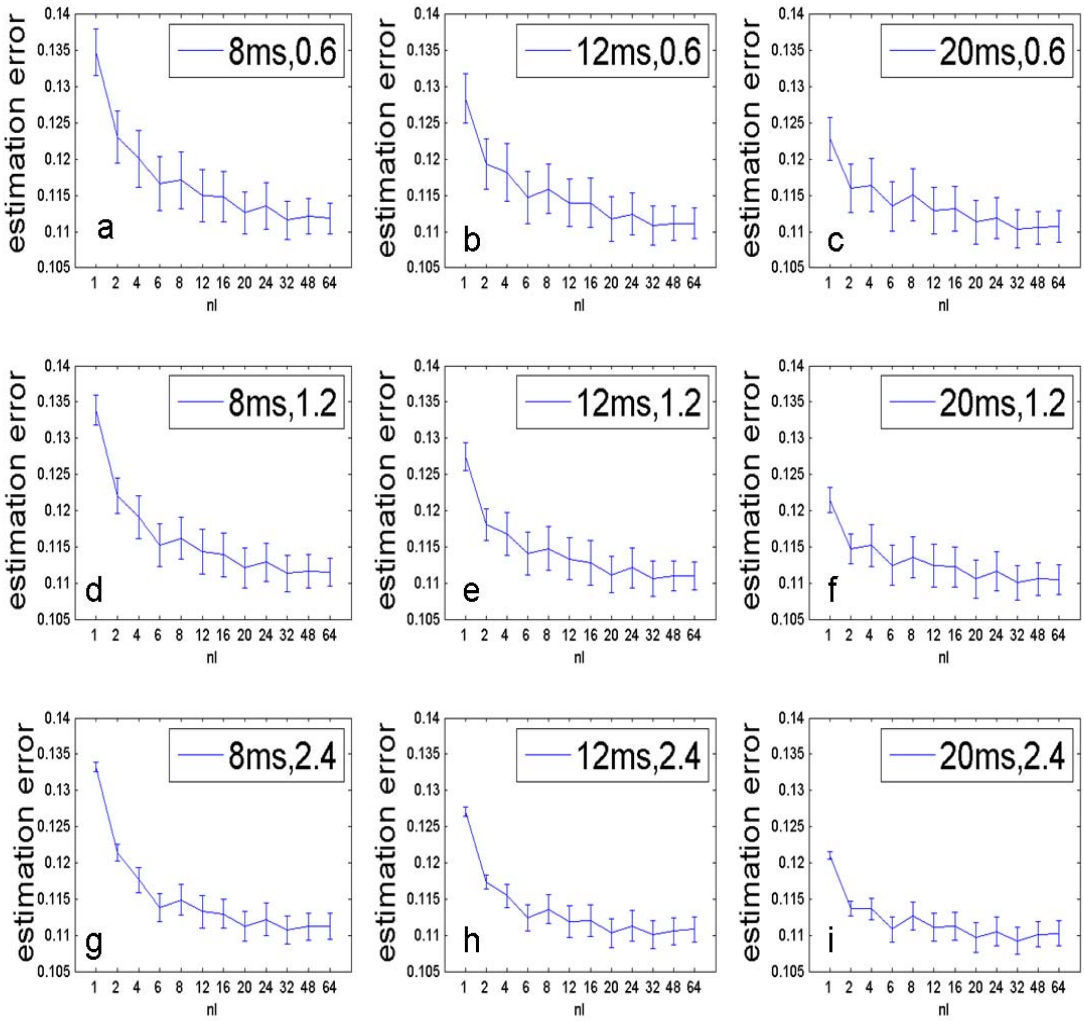

Figure 5-8 Nine Plots of Estimation Errors of 12 Sets of Spiral Trajectories

Estimation errors are plotted versus the number of spiral interleaves (*nl*) in 9 plots (a-i). The 12 values of *nl* are 1, 2, 4, 6, 8, 12, 16, 20, 24, 32, 48 and 64. Each plot represents one simulation of spiral imaging with 12 sets of spiral trajectories. Each simulation (400 iterations) was carried out with a particular relaxation rate (T_2*) and normalized signal-to-noise ratio (SNR_u) shown in the box at the upper right corner. The 12 values of *nl* are along the x axis of each plot. The three columns of curves from left to right show a decreasing slope because as T_2* increases from 8ms to 20ms, long spiral leaves (*nl*=1,2,4) suffer less due to relaxation. The three rows from top to bottom show decreasing error bars because SNR_u increases from 0.6 to 2.4. (Matlab codes for this figure are included in Appendix. C-25.)

The 9 sets of (T_2^*, SNR_u) values determine the slopes of the decay curves and the sizes of the error bars. The three columns of curves from left to right show a decreasing slope because as T_2^* increases from 8ms to 20ms, long spiral leaves (*nl*=1,2,4) suffer less due to relaxation. The three rows from top to bottom show decreasing error bars because SNR_u increases from 0.6 to 2.4.

The method described above treats NMR imaging as a form of linear estimation. The estimation error is defined as the difference of mean image intensity between the reference image and the result image summed across a region of interest (ROI). The error is normalized between 0 and 1 and the errors in the above plot are all within [0.105, 0.14], which is a small range. Furthermore, the plot above suggest that the only poor choice of *nl* is *nl*=1 which can be seen from the four sample images. This has confirmed that the empirically decided value for *nl*=2 is a good choice as the estimation error drops 10% from 0.135 to 0.122 when *nl* changes from 1 to 2. Along all curves in Fig.5.8, the drop of estimation error from *nl*=1 to *nl*=2 is the most significant. The empirical reason was based on the comparison of the measured echo width and readout length. All trajectories used in this method were generated to have an FOV of 16cm and resolution of 32×32. This relatively low resolution requirement is directly responsible for the fact that the choice of *nl* is not critical as long as the exetreme ones (*nl*=1 or 64) are avoided. Other applications of the spiral imaging like dynamic cardiac imaging or *f*MRI will lead to different parameters, trajectories and most importantly different (T_2^*, SNR_u) pairs. For example, in contrast enhanced MRI, T_2^* of water signal (~1s) will be longer than that of PUFA coherence transfer echo and SNR of water protons enhanced by Gd will be hundreds times higher than that of PUFA, too. That application will give the advantage to multishot spiral trajectories.

5.2.3 Discussion

As we compare the time resolved 3D spiral imaging and the Spiral Sel-MQC for spectroscopic imaging discussed in this work, we notice that Spiral Sel-MQC sequence features a more sophisticated signal excitation section. The application in time resolved angiography includes a simple one pulse excitation while the spectroscopic application includes a pulse train of 4 spin-editing pulses. The Spiral Sel-MQC application represents an approach that filters the signal source to optimize the acquired image. The filtration could be used for the purpose of

suppressing water/lipid signals or selecting a particular resonance peak. This approach opens the door to many novel sequence concepts in the future in magnetic resonance as advanced signal excitation (82-87) and acquisition techniques (27,56,57,88) are combined, as shown in our work.

As shown in all our sequence diagrams, spiral data acquisition relies on varying gradient waveforms during data acquisition. The performance of the gradient coils of the scanner in terms of maximum amplitude and slew rate are important parameters in spiral sequence design. The homogeneity of B_0 field also affects the spiral acquisition more than it affects Cartesian based acquisition because of long signal acquisition periods. Therefore, the development and application of spiral MRI/MRSI methods depend greatly on the improvement of scanner's hardware performance. As the manufacturers of MR scanners constantly improve the quality of their products, spiral may soon become a standard method of choice for all time critical applications.

The Signal Absortion Rates (SAR) of both spiral sequences did not limit their *in vivo* applications. The spiral Sel-MQC sequence does not include any spatial saturation pulses or CHESS pulses for water suppression. The RF absorption is resulted from the 4 RF pulses in Sel-MQC. The TR of the sequence is fixed at 2 seconds while the active time of the sequence is only 200ms. Therefore, the amount of RF energy transmitted per unit time during the scan is minimal. In the time resolved spiral angiography application, the sequence only consists of one short RF pulse, the slice-selection pulse at the beginning of the sequence. Therefore, the SAR of this sequence is also insignificant.

CONCLUSION

In vivo spiral Sel-MQC imaging was first demonstrated successfully to map the distribution of Polyunsaturated Fatty Acids in breast tissues as a possible index for cancer development. The Spiral Sel-MQC sequence acquired an image of PUFA with a sub-centimeter resolution in the breast in 2 minutes. The performance of the Sel-MQC spin editing was proven to be optimal by separate 2D *in vivo* spectra. The combination of spin editing and spiral acquisition represented a new approach to *in vivo* MRSI. In time resolved 3D MRI, our stack-of-spiral sequence acquired 3D images of sub-millimeter resolution with a temporal resolution of 2 seconds per frame. High spatial and temporal resolutions achieved together provide a powerful tool to diagnose vascular diseases.

In applications where high resolution is needed to resolve fine object details like arteries, a multi-shot spiral configuration is appropriate. In applications where only low resolution images are desired, the number of leaves can be greatly reduced as long as the limit of T_2* relaxation is taken into account.

Spiral data acquisition has been applied to two distinct areas in magnetic resonance in the scope of this thesis. The speed-up factor compared to Cartesian data acquisition is up to an order of 10. As MR scan times are significantly shortened, advanced MR techniques such as spin editing or time resolved acquisition can be included in routine MR exams in a clinical setting. Compared to other fast imaging techniques like EPI, spiral data acquisition has the advantage of starting the acquisition from the center of the k-space, where most of the signal energy is. The rotationally symmetric pattern of spiral acquisition in the k-space also enables more convenient view ordering and view sharing functions than that provided by Cartesian sampling such as EPI.

Appendix A

MR EXPERIMENTS ON GE SCANNERS

The major challenge of the body of work discussed in this thesis is to implement the advanced MR experiments on clinical whole body scanners (Signa LX 1.5T and 3.0T) manufactured by General Electric. The sequences provided by the manufacturer are a collection of standard imaging and spectroscopy sequences including Spin Echo, 2D and 3D Gradient Echo, Echo Planar Imaging (EPI), Stimulated Echo Spectroscopy (STEAM), Point Resolved Spectroscopy (PRESS) etc. Users have access to the entire developing platform and the source codes of all the standard sequences as building blocks or references. Advanced sequences are developed using most of the building blocks provided by the GE developing platform.

A GE pulse sequence includes three major sections: evaluating parameters, pulse definition and real time execution. A pulse definition section defines the waveforms for gradients in X, Y and Z directions, Radio Frequency (RF) transmition and receiver for data acquisition. The waveforms can be plotted by a simulation software exactly as they are executed in an actual instrument, or Integrated Pulse Generator (IPG). Each pulse is defined by a series of parameters. For example, a trapezoid gradient pulse is defined by its location in a pulse sequence, width, amplitude, durations of the ramp up/down phases (denoted “attack” or “decay”). An RF excitation pulse is usually defined based on a digitally generated waveform plot and its scaling parameters like resolution, pulse width, bandwidth, amplitude, phase etc. A data acquisition pulse has 2 major functions, setting parameters for the receiver filter and allocating memory for storage. The parameters of the pulses used in the definition section are initiated and evaluated using prescriptions entered from a console. For example, a prescribed slice thickness determines the width and amplitude of the slice selection gradient. A prescribed image resolution determines

the duration of data acquired. Some functions of a sequence have to be updated in real time for each excitation. For example, the value of a phase encoding gradient needs to be incremented throughout the execution of the sequence. That task is carried out by a real time execution loop in the sequence called Realtime Sequence Preparation (RSP).

The Time Resolved MRI sequence discussed in Chapter 4 was developed from a custom 2D spiral imaging sequence with excitation and phase encoding features imported from GE's 3D Gradient Echo sequence. Spiral gradient waveforms were generated externally and imported into the sequence in real time. The execution loops were custom designed to perform a temporal series of scans. The 2D Sel-MQC sequences discussed in Chapters 2 was developed from GE's standard STEAMCSI and PRESSCSI sequences with custom designed excitation and time evolution loops. The spiral Sel-MQC imaging sequence discussed in Chapter 2 was developed from GE's STEAMCSI sequence that combined some features from the two previously mentioned sequences.

The source code of these MR pulse sequences developed in our lab are available upon mutual research agreement between GE Healthcare and MR research center at Univ. of Pittsburgh Medical Center. The interested users should contact the principle investigator Dr. Qiuhong He at Univ. of Pittsburgh.

A.1 C CODE OF VARIABLE DENSITY SPIRAL GENERATION

The source code for generating variable density spiral trajectories used in spiral Sel-MQC and Time-Resolved MRA was developed as a standard ANSI C program based on its original version developed by Dr. Douglas Noll at Univ. of Michigan. This C program is included in GE's pulse sequences to generate custom trajectories in real time on the scanner. The algorithms was discussed in Chapter 5.1. The source code is included below as "genspiral2.e". The function genspiral2() defined in this programs takes input parameters including Field-of-View, resolution, oversampling and undersampling parameters. The output of the program is the gradient waveform arrays "gx[]" and "gy[]" along x and y directions for the gradient coils. The identical calculation is also a part of the spiral reconstruction program to generate the same trajectory arrays to process the acquired data.

In addition, the spiral trajectories generated can be saved in binary files “kxout” and “kyout” during the execution of the code in the pulse sequence or in the reconstruction. All Spiral Sel-MQC studies use the same 2-shot spiral trajectories and Time-Resolved Angiography studies use the same 24-shot spiral trajectories. The simulated in Chapter 5.2 uses spiral trajectories with interleaves of 1, 2, 4, 6, 8, 12, 16, 20, 24, … 64. These trajectories were generated and stored as binary files “kxout01” and “kyout01” for 1-shot; “kxout02” and “kyout02” for 2-shot, …, “kxout64” and “kyout64” for 64-shot spiral acquistions. The files are located in “/matlab/work/trajs/lo/” directory. The simulation program does not execute the spiral generation code for every iteration since the number of iterations can be up to 400. It reads the saved trajectories to save execution time. The sampling density distribution functions are also stored in the same folder with file name “densout01”, “densout02”, “densout04”,…, “densout64”.

```
/*************************************************************/
/* June, 2007 update, He Zhu   */
/* new trajectory generation with constant density spiral at the center of k-
space and variable density spiral for the rest*/
/*************************************************************/

/*   multi- shot spiral design
*    uses Duyn's approximate slewrate limited design
*    augmented with archimedian gmax limit
*    inputs (args)
*        D = FOV, cm
*        N = matrix size
*        Tmax = longest acquisition allowed, s
*        dts = output sample spacing, s
*        gtype = trajectory type
*    inputs (CVs)
*        nl = number of interleaves
*        gamp = design grad max, G/cm
*        gslew = design slew rate, mT/m/ms
*        nramp = number of rampdown points
*    outputs
*        Gx, Gy
*        grev
*    time is in sec
     filed is in Gauss
*
*               rev 0 12/26/98  original
*               rev 1 4/15/99   little better calc of ts
*/
float determinant(m11,m12,m13,m21,m22,m23,m31,m32,m33)
   float m11,m12,m13,m21,m22,m23,m31,m32,m33;
{
               float det;
```

```
        det = m11*(m22*m33-m23*m32)-m12*(m21*m33-m23*m31)+m13*(m21*m32-
m22*m31);
        return det;
}


int genspiral2(float opfov, int opxres, float Tmax, float dts, int gtype)
{
  float gamma = 4.257e3;   /* radians/s/Gauss */
  float lamda; /* lamda is actually kmax */
  float omega; /* w=2*n*pi */
  float tau, detau, tauc, taue, prevt=0.0, prevk=0.0; /* tau is a function of
t */
  float c, s, prevc=0.0, prevs=0.0;
  int break_flg=0, first_flg=1;
  float k=0.0;
/*  float delta=1.3;*/
  float aconst;
  float vdconst,vdconst1;
  float denom,numer;
  float aa,bb,cc,coeff;
  int vdflg=0;
  int Gxf[GRESMAX], Gyf[GRESMAX], Gxr[GRESMAX], Gyr[GRESMAX];    /*  for
spiral waves  */
  float dt;
  float Sm;
  int i, j, n;
  float Ts, T, ts, t, theta, dthdt, thetas, x, y;
  float a1, a2, beta, Gmax, bx, by;
  float gx[2*GRESMAX], gy[2*GRESMAX], kxout[2*GRESMAX], kyout[2*GRESMAX];
  float gabs, gmax;
  short gxi, gyi;
  float q=0.2;
#define LIMIT(x, xmin, xmax)   ( (x<xmax)? ((x>xmin)? x:xmin):xmax )
  int ndepnts;
  float agxde, agyde;
  char outfname[50];    /* PDT presently disabled */
  FILE *ofile;                  /* PDT presently disabled */

  Sm = gslew*100;       /* convert slew rate to Gauss/cm/s */
  dt = dts*.5;
  lamda = (float)opxres/(2.0*(float)opfov);
  omega = delta*(float)opxres*M_PI/nl;
  aconst = Sm*gamma/lamda;
  tauc = 0.25;
  taue=taulength/delta; /* this is an important parameter.*/
                /* it (taue) should never go to 1.0 as it may seem*/
                /* because omega contains the oversampling factor delta and
it is kept in the whole traj,*/
                /* taue (end of tau) should be shortened by a factor of delta
to compensate for omega */

  printf("Entering genspiral:  \ntaulength = %f, delta =%f, opfov =  %.2fcm,
opxres = %d,  nl = %d \n", taulength, delta, opfov, opxres, nl);

  printf("\ndt = %6.2fus, dts = %6.2fus, gslew = %6.2f, gamp = %6.2f, Tmax =
```

```
%6.2fms, gtype = %d",dt*1e6, dts*1e6, gslew, gamp,
Tmax*1e3, gtype);
  printf("\nomega=%f, \n",omega);
        /*  slew-rate limited approximation  */
                                /* Define constants and variables as in
Spielman's paper in MRM 50:214*/

  /*printf("\n\nBegin slew rate limited stage:");*/
/*  printf("\nTime for slew rate limited: Ts = %6.2fms",Ts*1e3);

  if(Ts > Tmax)  {
        printf("slew limited readout too long: Ts = %.2fms, Tmax = %.2fms\n",
Ts*1e3, Tmax*1e3);
        epic_error(use_ermes,"slew limited readout too long",
                        EM_PSD_SUPPORT_FAILURE,1,STRING_ARG,"slew limited
readout too long.");

  } */

  gmax = 0;
  i = 0;
                                /* For each tick of the clock from 0 to Ts */
  for (t = 0; k<=lamda; t+=dt)  {
        if (vdflg==0)
                {
                a2 = aconst/(2.0*pow(9.0/4.0*aconst/(omega*omega),1.0/3.0) );
                numer = 0.5*aconst*t*t;
                denom = ( 1.0/q+ a2*pow(t,4.0/3.0));
                tau = numer/denom;
                }
        else    tau = sqrt(vdconst*t);
        if (tau<=tauc&&vdflg==0){
                theta = omega*tau;
                c = cos(theta);
                s = sin(theta);
                k = lamda*tau;
                kxout[i]= k*c;
                kyout[i]= k*s;
        /*      if (i<4) printf("i=%d,tau=%.6f, tn=%.6f, con=%.6f \n", i,
tau, tn, con);*/
                gx[i]=(1.0/gamma)*(k*c-prevk*prevc)/dt;
                gy[i]=(1.0/gamma)*(k*s-prevk*prevs)/dt;
                gabs = hypot(gx[i], gy[i]);
                prevk=k;
                prevc=c;
                prevs=s;
                ts = t;
                thetas = theta;
                if(fabs(gx[i])>=gamp||fabs(gy[i])>=gamp)
                        {
                        printf("\n gamp max out in const when i=%d!\n",i);
                        break_flg=1;
                        break;
                        }
                i++;
                }
        else {
```

```
                if (vdflg==0){
                        printf("\ntransition oldt=%f, aconst=%f, a2=%f,
i=%d\n",t, aconst, a2,i);
                        vdflg=1;
                        detau = (aconst*t*denom-
numer*a2*4.0/3.0*pow(t,.333333))/(denom*denom);
                        vdconst = 2.0*detau*tau;
                        t = tau*tau/vdconst;

coeff=determinant(taue*taue,taue,1.0,tauc*tauc,tauc,1.0,2.0*tauc,1.0,0.0);

aa=determinant(1.0,taue,1.0,tauc,tauc,1.0,1.0,1.0,0.0)/coeff;

bb=determinant(taue*taue,1.0,1.0,tauc*tauc,tauc,1.0,2.0*tauc,1.0,0.0)/coeff;

cc=determinant(taue*taue,taue,1.0,tauc*tauc,tauc,tauc,2.0*tauc,1.0,1.0)/coeff
;
                        printf("\ntransition newt=%f, tau=%f, vdconst=%f,
k=%f, detau=%f\n",t, tau, vdconst,k,detau);
                        printf("Coefficients: coeff=%f, a=%f, b=%f,
c=%f\n",coeff, aa,bb,cc);
                }
                k = (aa*tau*tau+bb*tau+cc)*lamda;
                theta = omega*tau;
                c = cos(theta);
                s = sin(theta);
                kxout[i]= k*c;
                kyout[i]= k*s;
                gx[i]=(1.0/gamma)*(k*c-prevk*prevc)/dt;
                gy[i]=(1.0/gamma)*(k*s-prevk*prevs)/dt;
                gabs = hypot(gx[i], gy[i]);
                prevk=k;
                prevc=c;
                prevs=s;
                ts = t;
                thetas = theta;
                if(fabs(gx[i])>=gamp||fabs(gy[i])>=gamp)
                        {
                        printf("\n gamp max out in variable when i=%d! gx=%f,
gy=%f\n",i,gx[i], gy[i]);
                        break_flg=1;
                        break;
                        }
                i++;
        }
  }

 /*printf("\n\nActual slew limited time: %6.2fms",ts*1e3);*/

        /*  gmax limited approximation  */

  if(break_flg)  {

        printf("\nBegin gradient amplitude limited stage:");
        detau = 0.5*pow(vdconst*t,-0.5)*vdconst;
        vdconst1 = tau*tau*detau*3.0;
        t = pow(tau, 3.0)/vdconst1;
```

```
        printf("t=%f, vdconst1=%f, tau=%f, k=%f\n", t, vdconst1, tau, k);
        for (t=t+dt; k<=lamda; t+=dt)  {
                tau = pow(vdconst1*t, .3333333);
                k = (aa*tau*tau+bb*tau+cc)*lamda;
                if (first_flg){
                        first_flg=0;
                        printf("tau=%f, k=%f\n", tau, k);
                }
                theta = omega*tau;
                c = cos(theta);
                s = sin(theta);
                kxout[i]= k*c;
                kyout[i]= k*s;
                gx[i]=(1.0/gamma)*(k*c-prevk*prevc)/dt;
                gy[i]=(1.0/gamma)*(k*s-prevk*prevs)/dt;
                gabs = hypot(gx[i], gy[i]);
                prevk=k;
                prevc=c;
                prevs=s;
                ts = t;
                thetas = theta;
                i++;
        }
  }
  else {
        T = ts;
        printf("\n No constant gradient required:");
        printf(" Gmax = %.2f\n",gabs);
  }

        sprintf(outfname,"kxout");
        ofile=fopen(outfname,"w");
        fwrite(kxout,i,sizeof(*kxout),ofile);
        fclose(ofile);

        sprintf(outfname,"kyout");
        ofile=fopen(outfname,"w");
        fwrite(kyout,i,sizeof(*kyout),ofile);
        fclose(ofile);
        printf("\nEnd k=%f, tau=%f, i=%d, \n", k, tau, i);
  printf("lamda=%.3f, gamma=%.3f, omega=%.3f, dt=%.3f, Sm=%.3f\n", lamda,
gamma, omega, dt, Sm);
  printf("\ngx[0]= %2.5f, gx[1]= %2.5f, gx[2]= %2.5f, gy[0]= %2.5f,
gy[1]=%2.5f, gy[2]=%2.5f\n", gx[0], gx[1], gx[2], gy[0], gy[1], gy[2]);

        printf("\n\nSlew time = %6.2fms \nGrad time = %6.2fms \nTotal time =
%6.2fms",ts*1e3,(t-ts)*1e3,T*(1e3));

/*  decimate by 2 to get back to 4us sampling */

  n = 0;

  for (j=0; j<i; j+=2)  {
        x = LIMIT(gx[j], -gamp, gamp);
        gxi = x*MAX_PG_WAMP/gamp;
        Gxf[n] = 2*(gxi/2);
        y = LIMIT(gy[j], -gamp, gamp);
```

```
        gyi = y*MAX_PG_WAMP/gamp;
        Gyf[n++] = 2*(gyi/2);
}

bx = Gxf[n-1];
by = Gyf[n-1];

for (j=0; j<nramp; j++)  {
        c = 1 - j/(float)nramp;
        Gxf[n] = 2*((int)(bx*c)/2);
        Gyf[n++] = 2*((int)(by*c)/2);
}

if(gtype) {     /*reverse spirals*/
        for (i=0; i<n; i++) {
                Gxr[i]=-Gxf[n-i];
                Gyr[i]=-Gyf[n-i];
        }

                /*build dephaser*/
        ndepnts=200;
        agxde=0.;
        agyde=0.;

        for (i=0; i<n; i++) {
                agxde+=(float)Gxr[i]/ndepnts;
                agyde+=(float)Gyr[i]/ndepnts;
        }

        for (i=0; i<ndepnts; i++) {
                Gx[i]=-2*((int)(agxde*(float)(i)/ndepnts)/2);
                Gy[i]=-2*((int)(agyde*(float)(i)/ndepnts)/2);
        }

        for (i=0; i<ndepnts; i++) {
                Gx[i+ndepnts]=-2*((int)(agxde*(float)(ndepnts-i)/ndepnts)/2);
                Gy[i+ndepnts]=-2*((int)(agyde*(float)(ndepnts-i)/ndepnts)/2);
        }

        for (i=0; i<n; i++) {
                Gx[i+2*ndepnts]=Gxr[i];
                Gy[i+2*ndepnts]=Gyr[i];
        }

        gres=n+2*ndepnts;


}

else {       /*forward spirals*/

        for (i=0; i<n; i++) {
                Gx[i]=Gxf[i];
                Gy[i]=Gyf[i];
        }

        gres=n-1;
```

```
        printf("\n\nTotal gradient points: gres =  %d", gres);
  }
  printf("\nGx[0]= %d, Gx[1]= %d, Gy[0]= %d, Gy[1]=%d\n", Gx[0], Gx[1],
Gy[0], Gy[1]);
  printf("\nTslew = %.2fms , Tgrad = %.2fms, Total time = %.2fms\n",
ts*1000.0, (T-ts)*1000.0, T*1000.0);
  printf("genspiral SUCCESS\n");
/*
                sprintf(outfname,"Gx");
                    ofile=fopen(outfname,"w");
                     fwrite(Gx,(n+2*ndepnts),sizeof(*Gx),ofile);
                    fclose(ofile);

                    sprintf(outfname,"Gy");
                    ofile=fopen(outfname,"w");
                    fwrite(Gy,(n+2*ndepnts),sizeof(*Gy),ofile);
                    fclose(ofile);*/
  return SUCCESS;

}
```

A.2 TRAJECOTRY OUTPUT PROGRAM

A separated program (gspout) is written using the trajectory generation code used in the spiral pulse sequence and reconstruction program. The purpose of this program is to quickly generate trajecories in binary files for simulation purposes. The algorithm used is identical. The source code is gspout.c. It is compiled on smash and runs on smash.

"gspout" needs a dummy P-files for its execution. On smash, run "gspout –v mar9/P09216.7". At prompt:

```
User Inputs:
opfov:16
opxres:32
nl:1
delta:1.3
taulength:1.1
```

"gspout" outputs three files associated with the calculated trajectory: "kxout01", "kyou01" and "densout01". For trajectories with *nl*=2, 4,…, change the input for *nl* and keep all other

parameters constant. The 12 sets of spiral trajectories for the simulation in Chapter 5 are produced by this program. They are stored in folder “/matlab/work/trajs/lo/”.

Appendix B

SPIRAL RECONSTRUCTION PROGRAMS

The spiral image reconstruction of all *in vivo* studies were performed using ANSI C programs and the reconstructed images were imported into Matlab for further processing and display. The two C programs are "gsp3dvd2" for Time-Resolved spiral MRI and "gsp3dspec2" for spiral Sel-MQC MRSI.

The C source code "gsp3dvd2.c" was developed based on similar programs from Drs. Douglas Noll and Andrew Stenger. We are very gradeful that Dr. Noll agree to let us include it in this appendix. The code "gsp3dspec2.c" was based on "gsp3dvd2.c" with minor modification. Both codes are included in the "recon/" folder with needed include files. "gsp3dvd2.c" is printed below.

The code "gsp3dvd2.c" is compiled using the GCC compiler version 3.3 to generated an executable gsp3dvd2 with the following command of GCC:

```
gcc -o gsp3dvd2 gsp3dvd2.c -lm
```

The varies functions of this program are specified using command line parameters:

- gsp3dvd2 –v [P-file] to perform image reconstruction without inhomogeneity; correction. "–v" to output reconstruction information during execution;
- gsp3dvd2 –v –P [P-file] to perform phase map reconstruction;
- gsp3dvd2 –v –h [P-file] to perform image reconstruction with inhomogeneity correction using previously generated phase maps;
- gsp3dvd2 –v –k [-h] [P-file] to perform background substraction with or without inhomogeneity correction;

- gsp3dvd2 –v –s2 [-h] [-P] [-k] [P-file] to perform reconstruction of the temporal phase specified by the number after "-s". It can be combined with –P, -h, or –k.

See Appendix C18 and Fig. 4-6 for an example of the complete prodecure of spiral reconstruction and inhomogeneity correction. For reconstruction of Spiral Sel-MQC images, the program gsp3dspec2 is used with the identical command line parameters. Because the Spiral Sel-MQC sequence does not need features such as inhomogeneity correction, background substract or multiple temporal phase, the only necessary parameter is "-v" for reconstruction information. See Appendix C13, C14 and C15 for examples.

B.1 C SOURCE CODE OF SPIRAL RECONSTRUCTION

```
/*
 *   Gridding Image Reconstruction for Spiral k-space Acquisitions
 *                          (c) 1999
 *       Copyright by Douglas C. Noll and
 *         the Regents of the University of Michigan
 *                          (c) 1992,1993,1994,1995
 *       Copyright by Douglas C. Noll and the University of Pittsburgh
 *         Algorithm is based on O'Sullivan, IEEE Trans. Med. Imag.,
 *         4(4):200-207, 1985 as refined by Jackson, et al., Trans. Med.
Imag.,
 *         10(2):473-478, 1991.  Sample density correction as described by
 *         Meyer, et al., Magn. Reson. in Med., 28:202-213, 1992. Time-
 *         segmented inhomogeneity correction as described by
 *         Noll, et al., IEEE Trans. Med. Imag., 10(4):629-637, 1991.
 *
 *  John Pauly of Stanford University wrote code that served as the
 *   basis for this program.  Many changes occurred since then.
 *  Several utility routines came from Numerical Recipes in C,
 *   namely bessi0 and four1.
 *  Versions:
 *     gpr2   - converted to spin-echo proj.
 *     gpr3   - window = 2, added apodization correction
 *     gpr6   - added other convolution windows as well
 *              option for sample density correction
 *     gpr8   - added k-b window
 *     gsp1   - 1st run at spiral recon
 *     gsp2   - fixed set up for multi-slices
 *     gsp3   - added multi-phase, reordered input parm list
 *     gsp4   - added over flag (=2 for 2x over, =1 for no over)
 *            - works, but not as well as Craig's version
 *            - fixed delay term on readout
 *     gsp4hc - 1st run at homogeneity correction (with field map)
```

```
 *     gsp4hc2 - uses a previously generated field map
 *     gsp4hc3 - added fixviews subroutine for phase corrections
 *     gsp4hc4 - added section to read in list of files and uncompress
 *     gsp7  - put shifting, rotating, etc. into gridding subroutine
 *           - brought map making into same program (-m) option
 *           - defaut is no correction, (-h) option will do correction
 *           - compressed input is made an option (-c)
 *     gsp9  - 5.x version
 *     gsp10 - added parameters to rhuser variables in header
 *     gsp11 - made subsequent times through faster than first
 *     gsp12 - added phase multiplier
 *           - turned off most output unless using the (-v) option
 *           - added image registration (-r)
 *     gsp14 - added multi-coil recon option
 *           - added (-2) option to double translations with (-r)
 *           - added (-o) option to specify starting image number
 *           - added (-m2,-h2) options to do linear term inhomogeneity corr.
 *           - added (-m3,-h3) options for both linear and time-seg corrs.
 *           - added (-l) option for shift correction in obliques
 *     gsp15 - realtime grad generation hooks from craig
 *           - fixed bug in (-l and -r) options
 *     gsp16 - LX (8.x) version
 *           - modified parameter list for glover pulse sequence
 *           - added hooks to getrttrajghg - Glover's grad code
 *     gsp16a - handling split data acq, fixed (-l) option for 8.x
 *     gsp16b - fixed field mapping, inhomogeneity corrections
 *     gsp3d1 - read in 3D data
 */
/*********************************************************
       gsp3d3 - JQL modify the code from gsp3d2
    ********************************************
       08/27/01 Modify MXNPR from 16 to 64
                Modify MXNIM from 256 to 512
       09/08/01 Add code to save the  information about P-files and
                output-images to file "imgfiles.dat",
                so the "saveup" can use these information.
                why? I failed  to incooperate the function of "saveup"
                with gsp3d3 because I CANNOT add any other structure,
                such as GELXMRHRD lxhdr.
       09/08/01 Write "saveup".
       11/27/01 Add chop mode
       11/28/01 Let chop mode corresponding the loop order

 *********************************************************/

/*********************************************************
       gsp3d4 - JQL modify the codes from gsp3d3
    ************************************************
       12/21/01 Make it compatible with phase-array coil
                (1)define the constant OFFSET_PA_COIL=56
                   (I don't know why the address of data has an
                    offset(56) to the previous coil)
                (2)re-calculate the coiloff
                    (original method is not fit for our P-file)
                (3)multiply the imt with (znom/4)
                    (This seems the bug)
                (4)I didn't test it for multi-slice and multi-phase?
```

```
*************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include "getrttrajghg.c"
#include "getvdtraj.c"
/*HZ 02/05/03*/
#include "lxhdr.h"

#define RECON_FLAG
#define IMAGEDB_H_INCL

#include "rdbm2.h" /* HZ 2-5-2003 */

/* 5x image header stuff */
#define IM_CTR_R 130            /*CENTER R COORD OF PLANE IMAGE*/
#define IM_CTR_A 134            /*CENTER A COORD OF PLANE IMAGE*/
#define IM_CTR_S 138            /*CENTER S COORD OF PLANE IMAGE*/
#define IM_NORM_R 142           /*NORMAL R COORD*/
#define IM_NORM_A 146           /*NORMAL A COORD*/
#define IM_NORM_S 150           /*NORMAL S COORD*/
#define IM_TLHC_R 154           /*R COORD OF TOP LEFT HAND CORNER*/
#define IM_TLHC_A 158           /*A COORD OF TOP LEFT HAND CORNER*/
#define IM_TLHC_S 162           /*S COORD OF TOP LEFT HAND CORNER*/
#define IM_TRHC_R 166           /*R COORD OF TOP RIGHT HAND CORNER*/
#define IM_TRHC_A 170           /*A COORD OF TOP RIGHT HAND CORNER*/
#define IM_TRHC_S 174           /*S COORD OF TOP RIGHT HAND CORNER*/
#define IM_BRHC_R 178           /*R COORD OF BOTTOM RIGHT HAND CORNER*/
#define IM_BRHC_A 182           /*A COORD OF BOTTOM RIGHT HAND CORNER*/
#define IM_BRHC_S 186           /*S COORD OF BOTTOM RIGHT HAND CORNER*/
/* my defines */
#define MXNPR 64   /*08/27/01 JQL change from 16*/
#define MXNIM 512   /*08/27/01 JQL change from 256*/
#define MXNDAT 16384
#define GRESMAX 21000
#define NWEIGHTS 1024
#define SEGSIZE 1000 /* us */
#define SCALE 1.0       /*ZZH, 12/18/02*/
#define OVER 2
#define GRIDL 1.5
#define GRIDB 13.9086
#define PI    3.14159265358979323846
#define MIN(a,b) (((a) < (b)) ? (a) : (b))
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
#define MXNZPE 64       /*ZZH, changed from 24, on 05/06/03*/
#define GAM 4257.0
#define LSCOMMAND "cp -rf lxhdr.dat lx.dat"

#define SUCCESS 1

/*ZZH, 04/08/2003, changed from 56 to 20*/
#define OFFSET_PA_COIL 0                /*if tsp=2.5us, fast_rec==1*/
#define OFFSET_PA_COIL1 20              /*if tsp=4us, correspondingly
bandwidth=125*/
#define OFFSET_PA_COIL2 8               /*if tsp=8us, correspondingly
```

```
bandwidth=62.5*/

#define NCORRSAMP 100 /* ALN number of samples to be averaged for baseline
correction */

  float mean[2];

/*double rint();*/
POOL_HEADER header;
RDB_HEADER_REC *h_rec = &header.rdb_hdr_rec;

/* HZ */
EXAMDATATYPE *e_rec = &header.rdb_hdr_exam;
SERIESDATATYPE *s_rec = &header.rdb_hdr_series;
MRIMAGEDATATYPE *i_rec = &header.rdb_hdr_image;

/*EXAMDATATYPE *e_rec = (EXAMDATATYPE *)header.rdb_hdr_exam;
SERIESDATATYPE *s_rec = (SERIESDATATYPE *)header.rdb_hdr_series;
MRIMAGEDATATYPE *i_rec = (MRIMAGEDATATYPE *)header.rdb_hdr_image;*/

RDB_SLICE_INFO_ENTRY *a_rec = (RDB_SLICE_INFO_ENTRY
*)header.rdb_hdr_data_acq_tab;

/* HZ's code for variable density */
double delta, taulength;

int plane, zftoutput;
int istation = 0;
int nsub=0; /* view sharing parameter */
int fold=4;
int station_offs,station_size;/* HZ */
int ib1[MXNDAT][2];
short int mip[MXNIM][MXNIM];
short ib1short[MXNDAT][2],ib1short2[MXNDAT][2],ib1mask[MXNDAT][2];
float w1[2][MXNIM*OVER];
float w4[2][MXNZPE];
float w3[2][MXNZPE];
float p3d0[2][MXNZPE];
float p3d[2][MXNZPE];
float prd[2][MXNPR][MXNDAT];
float im[2][MXNIM*OVER][MXNIM*OVER];
float nav[2][MXNPR][MXNZPE];
float ****im3d;
float ****prd3d;
float sampim[MXNIM*OVER][MXNIM*OVER];
float grim[2][MXNIM*OVER][MXNIM*OVER];
float weight[NWEIGHTS+1];
float refim[MXNIM][MXNIM][MXNZPE];
float refimmag[MXNIM][MXNIM][MXNZPE];
float finalim[2][MXNIM][MXNIM];
float *t2k[2][MXNPR];

/*ZZH, 04/22/03*/
float *ktemp[2][MXNPR];

float ws[MXNIM*OVER][MXNIM*OVER];
float kdens[MXNDAT];
```

```
float m[2][MXNZPE][MXNZPE];
float regxs[40][2000],regys[40][2000],regrot[40][2000];
float refl[3];
char kroot[256];
char refroot[256];
char uncfile[256];
char regfile[256];
char com[256];
FILE *fk, *fi, *fi2, *fo1, *fo2, *fr,*ff;
int npr, nim, ndat, ndatfr, samp1, isl,chop, samp_cor, file, nex;
int imnpr; /* HZ, this variable has the # of image interleaves permanently.
npr will be changed into pmnl if (pmdomap) */
int chop_kz; /*11/28/01 JQL*/
int nsamp, nslices, nsegs,sliceorder;
int slnum,phnum,nphases,fsize,nph1,nphmult,rawnum,zpenum,nzpe;
int fliptb, fliplr, numrot, lrshift,tbshift;
int coilnum,ncoils,mcskip,imoffset=0;
int fs_flg=0, ts_flg=0; /*ZZH, 12/03/2002*/
int map_flg=0,hc_flg=0,com_flg=0,first_time=1,phs_flg=0,out_flg=0,loc_flg=0;
int
allcoils_flg=0,reg_flg=0,reg2x_flg=0,lincor_flg=0,linmap_flg=0,svraw_flg=0,ou
tputdata_flg=0,pm_flg=0;
int rev_flg = 0, noz_flg=0, test_flg=0, twod_flg=0, nav_flg=0,
revsp_flg=0,complex_flg=0, gradcomp_flg=0;
int linear_flg=0;
int display_flg = 0;
int vd_flg=0;   /*use variable density traj, ZZH, 05/27/03*/
int mask_flg=0; /* back ground subtraction */
int base_flg=0; /*ZZH, 06/16/03*/
int field2d_flg = 0; /*ZZH, 08/12/03*/
int zip_flg; /* HZ */
int kzzip_flg=0;

/*ZZH, 4/22/03*/
int extract_flg=0;
int extract_size=0;
int ndatph, ndatfrph, opxresph, concatph, nprph, pmimg_offsph;

float pt,gridl,gridb,maxk,magfact,phasefact,delta_pt,gradstep;
float factxx,factxy,factyx,factyy,zoomer;
float pix_shifth,pix_shiftv;
double ts,gts,fsgcm,opfov;
int risetime, densamp;
int gtype, opxres, ngap, concat;
int
pmdomap=0,pmnl,pmopxres,pmnframes,pmdoconcat,pmrhfrsize,pmmap_only=0,pmgres,p
mcrt_NSegments;
int pmopte, pmoptr,pmimg_offs, im_baseline, imrhfrsize, frsize, row_offset;
int opptsize;
/* HZ */
int plane, zftoutput;
double slewrate, fast_rec_lpf, mapdel, pmapdel, samptime;
float fudge=0.0;
float correction_data[MXNZPE][MXNPR][NCORRSAMP][2];

/*ZZH, 12/04/2002, parmameters used for frequency-segmented correction*/
#define NFSEGMENTS 8
```

```
int nfs = NFSEGMENTS;
double wmax[MXNZPE], wmin[MXNZPE];      /*maximum of off-resonance*/
double delta_fs;        /*frequency segment*/
/*END, ZZH*/

#include "gsp3d9.h"

int     main(argc,argv)
int argc;
char **argv;
{
  int stslice,enslice,segnum,midsamp,sampwind;
  int tmpslice,pp,ss;
  float jj;
  char *fnb;
  int i,j, temp_offset;/*ZZH, 04/08/03*/

/*Get -flags from command line*/

if (argc< 2) {
        fprintf(stderr,"Usage: %s [-m or -h] [-c] [-p] [-v] [-r regfile] [-a]
[-2] [-o offset] rawfile1 rawfile2...\n",argv[0]);
        return(0);
}

/*
The commands we use are:
STEP 1: Generate the field map
gsp3d9a -P2 Pfile < parm2
STEP 2: Do inhomogeneity correction
gsp3d9a -h3 Pfile < parm2

In case there is no phase map correction required, we use command:
gsp3d9a Pfile < parm2

*/

file = 1;
while (argv[file][0] == '-') {
        switch (argv[file++][1]) {
                case 'h':
/* inhomogeneity correction */
                switch (argv[file-1][2]) {
                        case '2': lincor_flg = 1; break;
/* linear field map correction only */
                        /*ZZH, 12/03/2002*/
                        case '3': lincor_flg = 1; hc_flg = 1; ts_flg = 1;
break;                 /* linear and time-segmented correction */
                        case '4': lincor_flg = 1; hc_flg = 1; fs_flg =1;
break;        /* linear and frequency-segmented correction */
                        case '5': hc_flg = 1; ts_flg =1;  break;        /*
linear and frequency-segmented correction */
                        /*end ZZH*/
                        /*end ZZH*/
                        default: hc_flg = 1; fs_flg = 1; break;
/* freq-segmented correction only */
                } break;
```

```
                case 'm':
/* make phase maps for hom. corr. */
                switch (argv[file-1][2]) {
                        case '2': linmap_flg = 1; break;
/* make only linear phase map */
                        case '3': linmap_flg = 1; map_flg = 1; break;
/* make linear and complete phase map */
                        default: map_flg = 1; break;
/* make phase map only */
                } break;
                case 'z': zip_flg = 1; break;
/* zero-padding HZ */
                case 'y': kzzip_flg =1; break;
/* zero-padding in kx direction */
                case 's':
                switch (argv[file-1][2]) {
                        case '1': istation = 0; break;
                        case '2': istation = 1; break;
                        case '3': istation = 2; break;
                        case '4': istation = 3; break;
                        case '5': istation = 4; break;
                        case '6': istation = 5; break;
                        case '7': istation = 6; break;
                        case '8': istation = 7; break;
                        default: istation =0; break;
                } break;/* multi-station HZ */;
                case 'V':
                switch (argv[file-1][2]) {
                        case '1': nsub = 1; break;
                        case '2': nsub = 2; break;
                        case '3': nsub = 3; break;
                        case '4': nsub = 4; break;
                        default: nsub =0; break;
                } break;/* view sharing HZ */;
                case 'P':
/* simul inhomogeneity correction */
                switch (argv[file-1][2]) {
                        case '2': linmap_flg = 1; map_flg = 1; pm_flg = 1;
zip_flg = 0; break;  /* create simul phase map */
                        case '4': pm_flg = 1; zip_flg = 0; break;
/* create images from simul phase map data */
                        default: map_flg=1; pm_flg =1; zip_flg = 0; break;
/* create simul phase map without linear phase map*//*ZZH, 12/03/02*/
                } break;
                case 'c': com_flg = 1; break;                           /*
compressed rawfile input */
                case 'p': phs_flg = 1; break;                           /*
output phase images */
                case 'v': out_flg = 1; break;                           /*
verbose - output status info */
                case 'a': allcoils_flg = 1; break;                      /*
output images for all coils */
                case 'r': reg_flg = 1; strcpy(regfile,argv[file++]); break;
/* image registration */
                case '2': reg2x_flg = 1; break;                         /*
doubles translations with reg */
                case 'o': imoffset = atoi(argv[file++]); break;          /*
```

```
offset in output image numbers */
                /*case 's': svraw_flg = 1; break;  */                    /*
save mag raw data into images */
                case 'l': loc_flg = 1; break;                           /*
ture locations, shift images */
                case 'q': rev_flg = 1; break;                           /*
reverse spiral acquisition */
                case 't': test_flg = 1; break;                          /*
recon only first image */
/*              case 'z': noz_flg = 1; break;    */                     /* no
z ft */
/*              case 'd': twod_flg = 1; break;     */                   /* 2d
acquisition */
                case 'C': complex_flg = 1; break;                       /*
output complex images */
                case 'R': revsp_flg = 1; break;                          /*
recon only first image */
                case 'g': gradcomp_flg = 1; break;
                case 'D': outputdata_flg = (argv[file-1][2] -
48);break;/*output data at each step in reconstruction*/
                case 'L': linear_flg = 1; break;
/*ZZH, 03/06/03, chose linear field map estimation method*/
                case 'e': extract_flg =1;break;                 /*ZZH,
04/22/03, use reduced kspace data*/
/*              case 'V': vd_flg = 1; break;*/  /*use variable density traj*/
                case 'd': display_flg =1; break;        /* simple and useless
output info for dummies */
                case 'b': base_flg = 1; break; /*ZZH, 06/16/03*/
                case 'f': field2d_flg = 1; break; /*ZZH, 08/12/03*/
                case 'k': mask_flg = 1; break; /* HZ back ground subtraction
*/
                default: fprintf(stderr,"%s: Did not recognize
%s",argv[0],argv[file-1]);
        }
}

if (!(ff=fopen("imgfiles.dat","w+"))) {
        fprintf(stderr,"Can't open %s!\n","imagefiles.dat"); exit(0);
} /*09/08/01 JQL */

for (rawnum = 0; rawnum < (argc - file); rawnum++){                     /*
for each raw Pfile */
        fnb = argv[rawnum+file];
        fprintf(ff,"%s\n",fnb);

        if (com_flg) {
/* If Pfile is compressed, get system to uncompress */
                sprintf(uncfile,"%s.uncp", fnb);
                if (out_flg) printf("Processing raw file %s\n", fnb);
                sprintf(com,"uncompress -c %s > %s", fnb, uncfile);
                system(com);
        }
        else    sprintf(uncfile,"%s", fnb);

  /* get some parameters from header of first file */
        if (rawnum == 0) {                                              /*
For the first raw pfile */
```

```
                get_hdr_info();
/* Get parameters from pfile header */
                read_inputs();
/* Get user input parameters */
                cal_map();
/* Get trajectory for spiral, make density weighting map */
        if (display_flg) {

                printf("(Above displayed information is internal parameters.
Please Ignore.)\n");

printf("\n*************************************************************
*************\n");
                if( map_flg ||pm_flg) printf("Constructing phase maps for %2d
images....\n", nzpe);
                else if (hc_flg) printf("Constructing %2d images with phase
map correction.... \n", nzpe);
                else printf("Constructing %2d images... \n", nzpe);
        }

                if (map_flg) magfact = phasefact = 0.0;
                if (reg_flg) {                                          /* If
registration flag */
                        if (!(fr=fopen(regfile,"r"))) {
                                fprintf(stderr,"Can't open %s!\n",regfile);
exit(0);
                        }
                        while (fscanf(fr,"%d %d",&pp,&ss)==2) {
                                fscanf(fr,"%f %f %f
%f",&regxs[ss][pp],&regys[ss][pp],&regrot[ss][pp],&jj);
                                /* printf("%d %d %f %f %f
%f\n",pp,ss,regxs[ss][pp],regys[ss][pp],regrot[ss][pp],jj); */
                                if (reg2x_flg) {
                                        regxs[ss][pp] *= 2.0;
                                        regys[ss][pp] *= 2.0;
                                }
                        }
                        fclose(fr);
                }
        }

        for (coilnum = 0; coilnum < ncoils; coilnum ++){                /*
For each coil */

                if (isl == 0) {stslice = 0; enslice = nslices;}
                else {
                        if (sliceorder) tmpslice = isl;
                        else            tmpslice = (isl%2 == 1) ? ((isl+1)/2)
: ((isl+nslices+1)/2);
                        stslice = tmpslice-1;
                        enslice = tmpslice;
                }
                for (slnum = stslice; slnum < enslice; slnum++) {
/* For each slab: */

                        if(hc_flg){
/* If homog corr. load phase maps*/
```

```
                                if (kzzip_flg) nzpe *=2;/* HZ */
                                for(zpenum=0; zpenum<nzpe; zpenum++) {
                                        load_ref_hc(slnum,zpenum);
/* load ref phase maps */
                                        if(revsp_flg)
rev_ref_hc(slnum,zpenum);
                                }
                                if (kzzip_flg) nzpe /=2;/* HZ */
                        }

/* allocate memory for 3d data arrays */
                        /*allocate_mem();*/     /*ZZH, 05/18/03*/

                        for(phnum = 0; phnum < nphases; phnum++) {
/* For each temporal phase: */
                                if (out_flg) {
                                        printf("Co %d, Ph %2d, Sl %2d:
Ld.\n",coilnum+1,phnum+1,slnum+1);
                                        fflush(stdout);
                                }
                                if (display_flg&&map_flg) printf("Step %d of
2...\n", phnum+1);
                                /*ZZH, 05/18/03*/
                                if (extract_flg&&phnum){
                                        /*restore parameter of phase map*/
                                        ndatfr=ndatfrph;
                                        ndat=ndatph;
                                        npr=nprph;
                                        opxres=opxresph;
                                        concat=concatph;
                                        pmimg_offs=pmimg_offsph;
                                        /*recalculate the trajectory and
weighting function based on parameters of phase map*/
                                        cal_map();
                                }

                                allocate_mem();


                                /*ZZH, 08/12/03*/
                                if(field2d_flg) {
                                        load_projs();
                                        refocus();
                                        fixviews();
                                        shiftview();
                                        load_ft1();                     /*
construct Bessel window, do convolution gridding*/
                                        if (out_flg) { printf("Fl.");
fflush(stdout); }
                                        fermi_filt1();                  /*
create smoothing fermi filter */
                                        fermi_filt2();                  /*
multiply data by fermi filter */
                                                                        /*
plain old recon */
                                        copy_dat();
                                        ft_image();             /* Do FT
```

```
without phase correction */
                                        if (out_flg) { printf("ZPe %2d:
Wr.",zpenum+1); fflush(stdout);}

                                        if (phs_flg){

write_image_phs(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);

write_imagemc_3d(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);
                                        }
                                        if (map_flg || linmap_flg) {
                                                if (phnum == 0) {

make_phase1_filt(zpenum);
                                                }
                                                else {

make_phase2_filt(zpenum); /*smooth phase and calculate phase diff */
                                                        if (ncoils > 1) {
/* If more than one coil, store phase data */

write_ref_mc(slnum,zpenum,coilnum);
                                                        }
                                                        else
write_ref(slnum,zpenum); /* output phase maps to file */
                                                }
                                        }
                                }
                                else{

                                        if (!noz_flg && !twod_flg) {
                                                for(zpenum = 0; zpenum <
nzpe; zpenum++) {      /* For each kz phase encode, load data */
                                                        if (out_flg) {
printf("ZPe %2d: Ld.\n",zpenum+1);fflush(stdout); }
                                                        if(!load_projs()) {
printf ("error in %d\n",zpenum); break;}    /* reads in spiral data */
                                                        if(revsp_flg)

reverse_order();

                                                        refocus_z(zpenum);
/*Do nothing with input parm2*/
                                                        if (base_flg)

baseline_corr();        /*ZZH, 06/16/03*/

copy_prd_to_prd3d(zpenum);
                                                }
                                                if (kzzip_flg==1) nzpe *=2;
/* kz zero padding starts from here */
                                                if (out_flg) {

printf("\nZFT.\n"); fflush(stdout); }


                                                /*zfilter_prd3d_hanning();*/
                                                zft_prd3d_discrete(-1,1);
/* Perform discrete FT in z direction */
```

```
                                        }
                                        for(zpenum = 0; zpenum < nzpe;
zpenum++) {      /* For each kz phase encode: */

                                                if(lincor_flg) {
/* If linear phase correction */

load_ref_lincor(slnum,zpenum);  /* load three paramters for linear correction
*/
                                                        if(revsp_flg)
rev_ref_lc();
                                                }
                                                if(noz_flg || twod_flg) {
                                                        if(!load_projs())
break;
                                                        if(revsp_flg)
reverse_order();
                                                        refocus();
/* demodulation */
                                                        fixviews();
/* remove mag phase variations */
                                                }
                                                else {

copy_prd3d_to_prd(zpenum);
                                                        refocus();
/* demodulation  at center frequency f0*/
                                                        fixviews();
/* remove mag phase variations */
                                                        shiftview();
/* shift fov     HZ*/
                                                }

                                                /*ZZH, 04/08/03*/
                                                if(outputdata_flg){
                                                        write_kspace(0);
                                                }
                                                else{
                                                        temp_offset=0;
                                                       if(ts==8.E-
6&&!(int)fast_rec_lpf)
temp_offset=OFFSET_PA_COIL2;
                                                       if(ts==4.E-
6&&!(int)fast_rec_lpf)
temp_offset=OFFSET_PA_COIL1;
                                                       if(ts==2.5E-6)

temp_offset=OFFSET_PA_COIL;
                                                        for (j=0; j<npr;
j++){
                                                                for (i=0;
i<ndat; i++){
if(i>ndat-temp_offset/2){
```

```
prd[0][j][i]=0;

prd[1][j][i]=0;
                                                                    }
                                                                }
                                                            }
                                                        }


                                                        if(nav_flg)
make_nav(zpenum,phnum);      /* navigator correction */

                                                        if (first_time || reg_flg ||
lincor_flg) {      /* If first regridding: */
                                                                load_ft1();
/* construct Bessel window, do convolution gridding*/
                                                                if (out_flg) {
printf("Fl."); fflush(stdout); }
                                                                if
(outputdata_flg==zpenum+1) write_kspace(1);
                                                                fermi_filt1();
/* create smoothing fermi filter */
                                                                fermi_filt2();
/* multiply data by fermi filter */
                                                                if
(outputdata_flg==zpenum+1) write_kspace(2);
                                                                first_time = 0;
                                                        }
                                                        else {
                                                                if(extract_flg)
load_ft1();                        /* construct Bessel window, do convolution
gridding*/
                                                                load_ft2();
/* For every other time: */
                                                                if (out_flg) {
printf("Fl."); fflush(stdout); } /*regridding*/
                                                                if
(outputdata_flg==zpenum+1) write_kspace(1);
                                                                if(extract_flg)
fermi_filt1();                    /* create smoothing fermi filter */
                                                                fermi_filt2();
/* multiply data by fermi filter */
                                                                if
(outputdata_flg==zpenum+1) write_kspace(2);
                                                        }
                                                        if(revsp_flg) flip_grim();
                                                        if (svraw_flg) write_raw();
                                                        if (out_flg) { printf("FT.");
fflush(stdout); }
                                                        /*ZZH, 12/03/2002*/
                                                        if (hc_flg) {
/* If homog corr requested do segemented correction: */
                                                                if (ts_flg) {
                                                                        nsegs =
floor(SCALE*ndat*samptime/SEGSIZE);     /* calculate no. of time segments */
                                                                        sampwind =
SEGSIZE/(SCALE*samptime);            /* no. samples per half segment (tau)*/
```

```
                                                                for (segnum =
0; segnum <= nsegs; segnum++) {   /* For each time segment: */
                                                                        if
(out_flg) { printf("%d.",segnum+1); fflush(stdout); }

midsamp = segnum*sampwind;      /* middle of current segment */

wind_dat(midsamp,sampwind);     /* window the data */

ft_image();                     /* FT the data in the window */

make_final(segnum,zpenum);
                                                                        /*add
contributions from all segs, the way it does is to multiply each segment data
with exp(i*w*tj), where tj=nT/N, the center of each segment*/
                                                                    }
                                                                }
                                                                else if(fs_flg) {

delta_fs = (wmax[zpenum]-wmin[zpenum])/nfs ;

copy_dat();

ft_image();

make_final_fs(-1, zpenum);
                                                                        for
(segnum = 0; segnum <= nfs; segnum++) {

if (out_flg) { printf("%d.",segnum+1); fflush(stdout); }

conjugated(segnum, zpenum);

ft_image();

make_final_fs(segnum, zpenum);
                                                                        }
                                                                    }
                                                            }
                                                            /*end, ZZH*/
                                                            else {

/* plain old recon */
                                                                    copy_dat();
                                                                    ft_image();
/* Do FT without phase correction */
                                                            }
                                                            if (display_flg&&!map_flg)
printf("Constructing Image No. %2d:",zpenum+1);         fflush(stdout);

                                                            if (outputdata_flg==zpenum+1)

write_kspace(5);
                                                            if (out_flg) { printf("ZPe
%2d: Wr.",zpenum+1); fflush(stdout);}

                        /* write images */
                                                            if (phs_flg){
```

```
write_image_phs(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);
                                                        /*ZZH, 11/14/02,
write simultaneous phase map images*/

write_imagemc_3d(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);
                                                        /*end modify by ZZH*/
                                                }
                                                if (map_flg || linmap_flg) {
                                                        if (phnum == 0) {

make_phase1_filt(zpenum);       /*get the smoothed phase data with a n-points
smooting average filter*/
                                                        }
                                                        else {

make_phase2_filt(zpenum); /*smooth phase and calculate phase diff */
                                                                if (ncoils >
1) {       /* If more than one coil, store phase data */

write_ref_mc(slnum,zpenum,coilnum);
                                                                        /*if
(coilnum == (ncoils-1)) {

write_ref_mcf(slnum,zpenum);

write_ref(slnum,zpenum);
                                                                        }*/
                                                                }
                                                                else
write_ref(slnum,zpenum); /* output phase maps to file */
                                                        }
                                                }
                                                else {  /*output images
according to users specified format, divide out convolution function */
                                                        if (allcoils_flg){

write_image(coilnum+1,slnum+1,phnum+1,rawnum, zpenum+1);

write_imagemc_3d(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);
                                                        }
                                                        else if (twod_flg)
write_imagemc(coilnum+1,slnum+1,phnum+1,rawnum);
                                                        else if
(gradcomp_flg) write_imagemc_gc(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);
                                                        else if (complex_flg)
write_imagemc_complex(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);
                                                        else
write_imagemc_3d(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);
                                                        if (phs_flg)
write_image_phs(coilnum+1,slnum+1,phnum+1,rawnum,zpenum+1);
                                                }
                                                if (out_flg) { printf("\n");
}


                                        } /* z phase encode loop */
```

```
                        }/* else? */
                    free_mem(); /* free memory for 3d data arrays */
                    if (kzzip_flg==1) nzpe /=2; /* HZ kz zero padding */
                    zftoutput=0;/* HZ */
                    } /* phases loop */
                } /* slices loop */
        } /* coils loop */
        if (com_flg) {
                sprintf(com,"rm -f %s", uncfile);
                system(com);
        }
} /* raw file loop */

if (out_flg) { printf("Done!!\n"); }
return SUCCESS;

} /* main */


/*************************************************************************
*************************************************************/
/*                                      FUNCTION DEFINITIONS
*/
/*************************************************************************
*************************************************************/

int     allocate_mem() {

  int i,j,k,l;

/*      im3d=(float****) malloc(2*sizeof(float ***));
        for(i=0;i<2;i++){
                im3d[i]=(float***) malloc(nim*OVER*sizeof(float **));
                for(j=0;j<nim*OVER;j++){
                        im3d[i][j]=(float**) malloc(nim*OVER*sizeof(float
*));
                        for(k=0;k<nim*OVER;k++){
                                im3d[i][j][k]=(float*)
malloc(nzpe*sizeof(float));
                        }
                }
        }*/
        if (kzzip_flg==1) nzpe *=2;/* kz zero padding */

        prd3d=(float****) malloc(2*sizeof(float ***));
        for(i=0;i<2;i++){
                prd3d[i]=(float***) malloc(npr*sizeof(float **));
                for(j=0;j<npr;j++){
                        prd3d[i][j]=(float**) malloc(ndat*sizeof(float *));
                        for(k=0;k<ndat;k++){
                                prd3d[i][j][k]=(float*)
malloc(nzpe*sizeof(float));
                        }
                }
        }
        for (i=0;i<2;i++)
                for (j=0;j<npr;j++)
```

```
                        for (k=0;k<ndat;k++)
                                for (l=0;l<nzpe;l++) prd3d[i][j][k][l]=0.0;

        if (kzzip_flg==1) nzpe /=2;/* kz zero padding */
        printf("nzpe in allocate_mem is %d\n",nzpe);
        return SUCCESS;
}


float   bessi0(x)                       /* See Numerical recipes in C 6.6
Modified Bessel functions of integral order */
float x;                                /* Returns the modified Bessel
Function I0(x) for any real x */
{
        float ax,ans;
        double y;

        if ((ax=fabs(x)) < 3.75) {
                y=x/3.75;
                y*=y;
                ans=1.0+y*(3.5156229+y*(3.0899424+y*(1.2067492
                        +y*(0.2659732+y*(0.360768e-1+y*0.45813e-2)))));
        }
        else {
                y=3.75/ax;
                ans=(exp(ax)/sqrt(ax))*(0.39894228+y*(0.1328592e-1
                        +y*(0.225319e-2+y*(-0.157565e-2+y*(0.916281e-2
                        +y*(-0.2057706e-1+y*(0.2635537e-1+y*(-0.1647633e-1
                        +y*0.392377e-2)))))))));
        }
        return ans;
}

int getmaxtraj(int msize, float* kx, float* ky, int xres)
{
        /*
        input:
        msize -- the reduced matrix size;
        kx,ky --        the coordinate of k-space
        xres      --    the number of points of the original leaf
        output:
        res     --      the number of points per interleaf for the reduced
maxtrix size
        */
        int res;
        int i, j;
        float absk;

        for (i=0; i<xres; i++){
                absk=hypot(kx[i], ky[i]);
                if(absk>msize/2)
                        break;
        }
        res = i-1;
        return res;
}
```

```
int     cal_map()                               /* recreates kspace
trajectory, and makes density map */
{
  int i;
  int res, res1;
  int getrttrajghg();
  int getvdtraj();
  double kx,ky,kxo,kyo,ggx,ggy,ggm,kksr,kksi,Tmax;

  /* kaiser-bessel functions - initialize terms */
        gridl = GRIDL; gridb = GRIDB;/*PI*GRIDL*/              /* PDT change
gridb according to Meyer IEEE Vol 10No.3 Sep 1991 p473*/

        refl[0] = refl[1] = refl[2] = 0.0;

        Tmax = MIN(gts*GRESMAX,ts*MXNDAT);
        if (map_flg)
                res = getvdtraj(opxres, npr, ts, gts, fsgcm, opfov, slewrate,
Tmax, delta, 1.1, gtype, t2k[0], t2k[1]);
        else    res = getvdtraj(opxres, npr, ts, gts, fsgcm, opfov, slewrate,
Tmax, delta, taulength, gtype, t2k[0], t2k[1]);
        if(extract_flg){               /*use the center part of k-space of
the high resolution dataset*/
                res1=getmaxtraj(extract_size, t2k[0][1], t2k[1][1], res);
/*calculate the number of pts to cover the center k-space*/
                ndatfr=res1;
                ndat=ndatfr*(concat+1);
                row_offset=frsize-res1;
        }
        nsamp = ndat;

        if (out_flg) {
                printf("row_offset=%d,  ndat = %d, npr = %d, ts = %.2fus, gts
= %.2fus, fsgcm = %.2f, opfov = %.2fcm, risetime = %d\n", row_offset, ndat,
npr, ts*1e6, gts*1e6, fsgcm, opfov, risetime);
                printf("acq. matrix = %d, nom. resolution = %.2f
mm\n",opxres,10*opfov/opxres);
                printf("k-space points = %d\n",res);
                printf("nsamp=%d\n", nsamp);
        }

                                                                /* sample
density correction */
  /* Using Craig's formula from MRM, 28, 202-213 (1992) */
        if (samp_cor == 1) {
                kdens[0] = 0.0;
                for (i=1; i < nsamp; i++){
                                                              /* might wish
to add correction for delay */
                        kx = t2k[0][0][i];
                        ky = t2k[1][0][i];
                        kxo = t2k[0][0][i-1];
                        kyo = t2k[1][0][i-1];
                        kksr = kx*kxo + ky*kyo;
                        kksi = ky*kxo - kx*kyo;
                                                        /* printf("%d: %f %f
%f %f %f %f\n",i,kx,ky,kxo,kyo,kksr,kksi); */
```

```
                        ggx = kx - kxo;
                        ggy = ky - kyo;
                        if (((ggm = hypot(ggx,ggy)) > 0.) &&
(hypot(kksr,kksi) > 0.) )
                                kdens[i] = ggm * fabs(sin(atan2(ggy,ggx) -
atan2(ky,kx)) * (hypot(kx,ky) -
hypot(kxo,kyo))/atan2(kksi,kksr));
                        else    kdens[i] = 0.;
    /*  printf("%f %f %f\n", kx, ky,kdens[i]);  */
    /* components to density correction:
      ggm = gradient strength; linear velocity
      sin(atan2(ggy,ggx) - atan2(ky,kx)) =  outward component of velocity
             These first two came from the Meyer paper.
      (hypot(kx,ky) - hypot(kxo,kyo))/atan2(kksi,kksr) = radial density
             (distance outward)/(anglar distance along arc)
             This term is different from the previous term in that this
             measure true line-line distances in the radial direction.
             It was added for variable density trajectories (except for the
             the first 2 or 3 points, it is constant for uniform density
             spiral trajectories.
    */
                }
                for (i=nsamp; i < ndat; i++) kdens[i] = 0.0;
        }
        else for (i=0; i < ndat; i++) kdens[i] = 1.0;
        return SUCCESS;
}

/* HZ 2-5-2003*/
/* shift fov with spiral data */
int shiftview()
{
  int i,j;
  float xshift,yshift,cs,si,phi,temp;

        if (plane==2) {
                xshift=-(*i_rec).ctr_R/(opfov*10.0);
                yshift=-(*i_rec).ctr_A/(opfov*10.0);
                        }
        if (plane==4) {
                xshift=-(*i_rec).ctr_A/(opfov*10.0);
                yshift=0.0;
                        }
        if (plane==8) {
                xshift=-(*i_rec).ctr_R/(opfov*10.0);
                yshift=0.0;
                        }
        if(plane<=8){
                for (i=0;i<npr;i++){
                        for (j=0;j<(ndat);j++){

                                phi=t2k[0][i][j]*xshift+t2k[1][i][j]*yshift;
                                cs=cos(2*PI*phi);
                                si=sin(2*PI*phi);
                                temp=prd[0][i][j]*cs-prd[1][i][j]*si;
                                prd[1][i][j]=prd[1][i][j]*cs+prd[0][i][j]*si;
                                prd[0][i][j]=temp;
```

```
                        }
                }
        }
        return SUCCESS;
}


int     copy_dat()
{
  int i,j;

        if (zip_flg) {
                for(i=0; i<2*nim*OVER; i++){
                        for (j=0; j<2*nim*OVER; j++){
                                im[0][i][j]=0.0;
                                im[1][i][j]=0.0;
                        }
                }

                for(i=0; i<nim*OVER; i++) {
                        for (j=0; j<nim*OVER; j++) {
                                im[0][i+256][j+256] = grim[0][i][j];
                                im[1][i+256][j+256] = grim[1][i][j];
                        }
                }
                nim *=2;
        } /* HZ if zip_flg*/

        else {
                for(i=0; i<nim*OVER; i++) {
                        for (j=0; j<nim*OVER; j++) {
                                im[0][i][j] = grim[0][i][j];
                                im[1][i][j] = grim[1][i][j];
                        }
                }
        }
                return SUCCESS;
}


int     copy_prd_to_prd3d(int npe)
{
  int j,k;

        if (kzzip_flg==1) {

                for(j=0; j<npr; j++) {
                        for(k=0; k<ndat; k++) {
                                prd3d[0][j][k][npe+nzpe/2] = prd[0][j][k];
                                prd3d[1][j][k][npe+nzpe/2] = prd[1][j][k];
                        }
                }
        }

        else {
                for(j=0; j<npr; j++) {
```

```
                        for(k=0; k<ndat; k++) {
                                prd3d[0][j][k][npe] = prd[0][j][k];
                                prd3d[1][j][k][npe] = prd[1][j][k];
                        }
                }
        }

        return SUCCESS;
}

int     copy_prd3d_to_prd(int npe)
{
  int j,k;

        for(j=0; j<npr; j++){
                for(k=0; k<ndat; k++) {
                        prd[0][j][k] = prd3d[0][j][k][npe];
                        prd[1][j][k] = prd3d[1][j][k][npe];
                }
        }
        return SUCCESS;
}



int     fermi_filt1()                   /* Create fermi filter */
{
  float ww,rad;
  int i,j,offs;
  rad = nim/2*OVER;
        if ( maxk*OVER/zoomer < rad )  rad = maxk*OVER/zoomer;
                                /* printf("rad = %f ",rad); */
        offs = nim/2*OVER;
        for(i=0; i<nim*OVER; i++) {
                for (j=0; j<nim*OVER; j++) {
                        ww = 1. / (1. + exp((hypot((float)(offs-
i),(float)(offs-j)) - rad) *6.4 /rad));
                                        /*      ww = 1.0; */
                        ws[i][j] = ww/(0.001+ws[i][j]);
                }
        }
        return SUCCESS;
}

int     fermi_filt2()                   /* Multiply data by fermi filter */
{
  int i,j;
        for(i=0; i<nim*OVER; i++) {
                for (j=0; j<nim*OVER; j++) {
                        grim[0][i][j] *= ws[i][j];
                        grim[1][i][j] *= ws[i][j];
                }
        }
        return SUCCESS;
}

void fft(fwd,n,xr, xi)
```

```
int *fwd, *n;
float xr[],xi[];
{
  void four1();

        four1(xr,xi, *n, *fwd);
}


int     fixviews(){                             /* remove mag and phase
variations (e.g. Hu's method) */
  int i, j;
  float p0r, p0i,prr, pri, cc, dd, tmp;

        p0r = p0i = 0.;
        for (i=0; i<npr; i++) {
                for (j=0; j<= samp1; j++) {
                        p0r += prd[0][i][j];
                        p0i += prd[1][i][j];
                }
        }
        p0r /= (1.0 * npr);
        p0i /= (1.0 * npr);

  /* printf("Correction angle = "); */
        for (i=0; i<npr; i++) {
                prr = pri = 0.;
                for (j=0; j<= samp1; j++) {
                        prr += prd[0][i][j];
                        pri += prd[1][i][j];
                }
    /*
    printf("%f %f\n",hypot(pri,prr),atan2(pri,prr));   */
    /* phase correction angle */
                tmp = atan2(p0i,p0r) - atan2(pri,prr);
                if (tmp >  PI) tmp -= 2.0*PI;
                if (tmp < -PI) tmp += 2.0*PI;
                cc = cos(tmp*phasefact);
                dd = sin(tmp*phasefact);
    /* magnitude correction term   */
                tmp = hypot(p0r,p0i) / hypot(prr,pri);
                cc *= tmp*magfact + (1.0 - magfact);
                dd *= tmp*magfact + (1.0 - magfact);
        /* printf("%4.2f, ",atan2(dd,cc)*180.0/PI); */
                for (j=0; j< ndat; j++) {
                        prr = prd[0][i][j];
                        pri = prd[1][i][j];
                        tmp = cc*prr - dd*pri;
                        prd[1][i][j] = dd*prr + cc*pri;
                        prd[0][i][j] = tmp;
                }
        }
  /* printf("\n"); */
        return SUCCESS;
}

int     flip_grim() {
```

```
    int i, j;
    float tempr[OVER*MXNIM][OVER*MXNIM], tempi[OVER*MXNIM][OVER*MXNIM];

        for (i=0; i<nim*OVER; i++) {
                for (j=0; j<nim*OVER; j++) {
                        tempr[i][j]=grim[0][nim*OVER-i-1][nim*OVER-j-1];
                        tempi[i][j]=grim[1][nim*OVER-i-1][nim*OVER-j-1];
                }
        }

        for (i=0; i<nim*OVER; i++) {
                for (j=0; j<nim*OVER; j++) {
                        grim[0][i][j]=tempr[i][j];
                        grim[1][i][j]=tempi[i][j];
                }
        }
        return SUCCESS;
}



#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(rdat,idat,nn,isign)
float rdat[],idat[];
int nn,isign;
{
        int n,mmax,m,j,j1,istep,i,mmaxby2;
        double wtemp,wr,wpr,wpi,wi,theta;
        float tempr,tempi;

        n=nn << 1;
        j=1;
        for (i=0;i<nn;i++) {
                j1 = (j-1)/2;
                if (j1 > i) {
                        SWAP(rdat[j1],rdat[i]);
                        SWAP(idat[j1],idat[i]);
                }
                m=nn;
                while (m >= 2 && j > m) {
                        j -= m;
                        m >>= 1;
                }
                j += m;
        }
        mmax=2;
        while (n > mmax) {
                istep=2*mmax;
                mmaxby2 = mmax/2;
                theta=6.28318530717959/(isign*mmax);
                wtemp=sin(0.5*theta);
                wpr = -2.0*wtemp*wtemp;
                wpi=sin(theta);
                wr=1.0;
```

```
                wi=0.0;
                for (m=0;m<mmaxby2;m++) {
                        for (i=m;i<nn;i+=mmax) {
                                j=i+mmaxby2;
                                tempr=wr*rdat[j]-wi*idat[j];
                                tempi=wr*idat[j]+wi*rdat[j];
                                rdat[j]=rdat[i]-tempr;
                                idat[j]=idat[i]-tempi;
                                rdat[i] += tempr;
                                idat[i] += tempi;
                        }
                        wr=(wtemp=wr)*wpr-wi*wpi+wr;
                        wi=wi*wpr+wtemp*wpi+wi;
                }
                mmax=istep;
        }
}

#undef SWAP


int     free_mem() {

 int i,j,k;

/*      for(i=0;i<2; i++) {
                for(j=0;j<nim*OVER; j++) {
                        for(k=0;k<nim*OVER; k++) {
                                free(im3d[i][j][k]);
                        }
                        free(im3d[i][j]);
                }
                free(im3d[i]);
        }
        free(im3d);*/
        for(i=0;i<2; i++) {
                for(j=0;j<npr; j++) {
                        for(k=0;k<ndat; k++) {
                                free(prd3d[i][j][k]);
                        }
                        free(prd3d[i][j]);
                }
                free(prd3d[i]);
        }
        free(prd3d);

        return SUCCESS;
}


int     ft_image()                      /* Do 2 one dimensional Fourier
transforms */
                        /* The second one dimensional transform in y is
performed only from
                                kx = (OVER-1)*nim/2 to (OVER+1)*nim/2 to
reduce processing time.
                                The extra data outside this region does not
```

```
contain any useful information
                                and will be cropped later anyway */
{
  int i,j, fftd, n,offs,offs2;

        n = nim*OVER; fftd = 1;
        offs = n/2;
        offs2 = (OVER-1)*nim/2;
        for(i=0; i<nim*OVER; i++) {
                for (j=0; j<offs; j++) {
                        w1[0][j] = im[0][i][j+offs];
                        w1[1][j] = im[1][i][j+offs];
                        w1[0][j+offs] = im[0][i][j];
                        w1[1][j+offs] = im[1][i][j];
                }
                fft(&fftd, &n, w1[0], w1[1]);
                for (j=0; j<offs; j++) {
                        im[0][i][j+offs] = w1[0][j];
                        im[1][i][j+offs] = w1[1][j];
                        im[0][i][j] = w1[0][j+offs];
                        im[1][i][j] = w1[1][j+offs];
                }
        }
        for(j=0; j<nim; j++) {
                for (i=0; i<offs; i++) {
                        w1[0][i] = im[0][i+offs][j+offs2];
                        w1[1][i] = im[1][i+offs][j+offs2];
                        w1[0][i+offs] = im[0][i][j+offs2];
                        w1[1][i+offs] = im[1][i][j+offs2];
                }
                fft(&fftd, &n, w1[0], w1[1]);
                for (i=0; i<offs; i++) {
                        im[0][i+offs][j+offs2] = w1[0][i];
                        im[1][i+offs][j+offs2] = w1[1][i];
                        im[0][i][j+offs2] = w1[0][i+offs];
                        im[1][i][j+offs2] = w1[1][i+offs];
                }
        }
        return SUCCESS;
}


int     get_hdr_info()                                  /* Get header info
from first file on list */
{


    float headflt();


        if (!(fi=fopen(uncfile,"r"))) {
                fprintf(stderr,"Can't open %s!\n",uncfile);
                return(0);
        }

    /* header */
        fread(&header,sizeof(header),1,fi);
```

```
        opptsize =  (int) rint((*h_rec).rdb_hdr_point_size);
        nph1 = 1;
        nzpe = (int) rint((*h_rec).rdb_hdr_user16);
        if(nzpe==1) twod_flg = 1;
        nphmult = (int) rint((*h_rec).rdb_hdr_user1);
        if (nphmult < 1) nphmult = 1;
        if ((map_flg || linmap_flg)  && ( nph1*nphmult == 1) && !pm_flg) {
                fprintf(stderr,"Illegal mapping file %s!\n",uncfile);
                exit(0);
        }
        if ((map_flg || linmap_flg) && !pm_flg) nphases=2;
        else nphases = nph1*nphmult;
        if (test_flg) nphases=1;
        nslices = (*h_rec).rdb_hdr_nslices/nph1;
        ndatfr = (*h_rec).rdb_hdr_frame_size;
        npr = (int) rint((*h_rec).rdb_hdr_user4);
        imnpr = npr; /* HZ */
                /*11/30/        01 JQL */
        chop = 1;
        chop_kz = 1;    /*ZZH, 02/05/03*/
        if( rint((*h_rec).rdb_hdr_user18) == 1)chop_kz=1; /*chop and
kz_inside*/
        if( rint((*h_rec).rdb_hdr_user18) == 2)chop=-1;/*chop and iv_inside*/
        printf("chop(iv)=%d, chop_kz(kz)=%d\n",chop,chop_kz);

        nex = 1;
        densamp = 5;
        sliceorder = (int) rint((*h_rec).rdb_hdr_user17);
        sliceorder =1;
                                                /* 0=interleaved,
1=sequential */
                                                /* glover cv's */
        gtype = (int) rint((*h_rec).rdb_hdr_user5);
        if (gtype==1) revsp_flg=1;
        opxres = (int) rint((*h_rec).rdb_hdr_user3);
        slewrate = (*h_rec).rdb_hdr_user7;
        ngap = (int) rint((*h_rec).rdb_hdr_user10); /* for concat readout */
        concat = (int) rint((*h_rec).rdb_hdr_user11); /* for concat readout
*/
        ndat = ndatfr*(concat+1);
        fast_rec_lpf = (*h_rec).rdb_hdr_user12; /* fast rec center frequency
in kHz */
        mapdel = (double)(*h_rec).rdb_hdr_user15;/* field mapping offset
(us)*/
        samptime = (*h_rec).rdb_hdr_user13; /* in (us) */
        fsgcm = (*h_rec).rdb_hdr_user6;
        risetime = (int) rint(fsgcm*10000.0/slewrate);
        opfov = (*h_rec).rdb_hdr_user0;
        ts = ((*h_rec).rdb_hdr_user13)*1e-6;  /* in sec */
        gts = 4*1e-6;  /* 4 us gradient spacing in sec */
        ncoils = (*h_rec).rdb_hdr_dab[0].stop_rcv-
(*h_rec).rdb_hdr_dab[0].start_rcv+1;
        mcskip = (*h_rec).rdb_hdr_raw_pass_size/ncoils;
        /*mcskip -= (nslices*nph1*nzpe*(nphmult*npr+1)*ndat*2*opptsize);*/
        plane= (*i_rec).plane; /* HZ */
```

```
        pmnl = (*h_rec).rdb_hdr_user19;
        pmdomap = (*h_rec).rdb_hdr_user20;
        pmopxres = (*h_rec).rdb_hdr_user21;
        pmcrt_NSegments = (*h_rec).rdb_hdr_user22;
        pmnframes = (*h_rec).rdb_hdr_user23;
        pmdoconcat = (*h_rec).rdb_hdr_user24;
        pmrhfrsize = (*h_rec).rdb_hdr_user25;
        pmapdel = (double)(*h_rec).rdb_hdr_user26;
        pmmap_only = (*h_rec).rdb_hdr_user27;
        pmgres = (*h_rec).rdb_hdr_user28;
        pmoptr = (*h_rec).rdb_hdr_user29;
        pmopte = (*h_rec).rdb_hdr_user30;
        imrhfrsize = (*h_rec).rdb_hdr_user31;
/* HZ */
        delta = (*h_rec).rdb_hdr_user32;
        taulength = (*h_rec).rdb_hdr_user33;
        fold = (*h_rec).rdb_hdr_user34;



        if (out_flg) {
                printf("nphases = %d (%d*%d)\n",nphases,nph1,nphmult);
                printf("slices = %d\n",nslices);
                printf("samples in each acq frame = %d\n",ndatfr);
                printf("samples in one spiral readout = %d\n",ndat);
                printf("field of view = %.1fmm\n",opfov);
                printf("interleaves = %d\n",npr);
                printf("time of scan = %s\n",(*h_rec).rdb_hdr_scan_time);
                printf("date of scan = %s\n",(*h_rec).rdb_hdr_scan_date);
                printf("header size = %d\n", sizeof(header));
                printf("nphmult = %d\n",nphmult);
                printf("nph1 = %d\n",nph1);
                printf("nzpe = %d\n",nzpe);
                printf("revspiral = %d\n",revsp_flg);
                printf("fsgcm = %.1fGauss/cm, slewrate = %.1fT/m/s, risetime
= %d\n", fsgcm, slewrate, risetime);
                printf("concat = %d, ngap = %d\n",concat, ngap);
                printf("opxres = %d\n",opxres);
                printf("fast_rec_lpf = %.2fkHz\n",fast_rec_lpf);
                printf("mapdel from header = %.2fus\n",mapdel);
                printf("samptime = %.2fus\n",samptime);
                printf("ts = %.2fus, gts = %.2fus\n",ts*1e6,gts*1e6);
                printf("rotation = %d\n",(*h_rec).rdb_hdr_rotation);
                printf("transpose = %d\n",(*h_rec).rdb_hdr_transpose);
                printf("aps tg = %ld\n",(*h_rec).rdb_hdr_ps_aps_tg);
                printf("mps tg = %ld\n",(*h_rec).rdb_hdr_ps_mps_tg);
                printf("aps frequency = %ld\n",(*h_rec).rdb_hdr_ps_aps_freq);
                printf("xshift = %.2f\n",(*h_rec).rdb_hdr_xoff);
                printf("yshift = %.2f\n",(*h_rec).rdb_hdr_yoff);
                printf("coils = %d\n\n",ncoils);
                printf("pmdomap = %d\n",pmdomap);
                printf("pm_flg = %d\n",pm_flg);
                printf("opptsize = %d\n",opptsize);
                        /* HZ  2-4-2003 */
                printf("Image Plane Type: %d\n",plane);
                printf("View sharing fold: %d\n",fold);
```

```
        }

        pmimg_offs = 0;


        frsize = ndatfr;
        im_baseline = 2*opptsize*frsize;
        printf("im_baseline = %d\n",im_baseline);
        if(pmdomap){
                ndatfr = imrhfrsize;
                ndat = imrhfrsize*(concat+1);
                row_offset = frsize - imrhfrsize;
                if(out_flg){
                        printf("pmmap_only = %d\n",pmmap_only);
                        printf("pmnframes = %d\n",pmnframes);
                        printf("pmcrt_NSegments = %d\n",pmcrt_NSegments);
                        printf("pmnl = %d\n",pmnl);
                        printf("pmopxres = %d\n",pmopxres);
                        printf("pmdoconcat = %d\n",pmdoconcat);
                        printf("pmapdel = %.2fus\n",pmapdel);
                        printf("pmgres = %d\n",pmgres);
                        printf("pmoptr = %dus\n",pmoptr);
                        printf("pmopte = %dus\n",pmopte);
                        printf("frsize = %d\n",frsize);
                        printf("pmrhfrsize = %d\n",pmrhfrsize);
                        printf("imrhfrsize = %d\n",imrhfrsize);
                        printf("row_offset = %d\n\n",row_offset);
                }
        }

        if(pm_flg){
                /*ZZH, 05/18/03*/
                if(!extract_flg){
                        if(pmmap_only==0)
                                pmimg_offs =
nphases*nzpe*npr*(concat+1)*frsize*2*opptsize;
                        ndatfr = pmrhfrsize;
                        ndat = ndatfr*(pmdoconcat+1);
                        row_offset = (frsize-ndatfr);

                        nphmult = pmnframes;
                        nphases = nphmult*nph1;
                        npr = pmnl;
                        opxres = pmopxres;
                        mapdel = pmapdel;
                        concat = pmdoconcat;
                }
                else{
                        /*store value*/
                        ndatfrph=pmrhfrsize;
                        ndatph=ndatfrph*(pmdoconcat+1);
                        nprph=pmnl;
                        opxresph=pmopxres;
                        concatph=pmdoconcat;
                        nphases=2;
                        if(pmmap_only==0)
```

```
                                pmimg_offsph =
nzpe*npr*(concat+1)*frsize*2*opptsize;
                }
        }



        fclose(fi);


        return SUCCESS;
}


/* routine to get floats out of GE header - needed because
   header is not word aligned */
float headflt(addr)
char *addr;
{
   float fltval;

        memcpy((char*)&fltval,addr,4);
        return(fltval);
}

int headint(addr)
char *addr;
{
   int intval;

        memcpy((char*)&intval,addr,4);
        return(intval);
}

short headshrt(addr)
char *addr;
{
   short shrtval;

        memcpy((char*)&shrtval,addr,4);
        return(shrtval);
}


float kaiser(l, b, u)           /* Create Kaiser-Bessel window for
convolution */
float l, b, u;
{
    float f;

        if (fabs(u) <= 0.5*fabs(l)){
                f = bessi0(b*sqrt(1.0-(2.0*u/l)*(2.0*u/l)));
                return f;
        }
        else    return 0.0;
}
```

```
int     load_ft1(void)                          /* create convolution
function and perform convolution */
{
  int i, j, lx, ly;
  float kx,ky,w,pr,pi,mkr,dkx,dky,dwin,w2,wx;
  float tmpd,rot1,rot2,rotfact;
  int imnum,tmpslice,ie;

        if (reg_flg) {
                ie = slnum+1;
                imnum = imoffset + rawnum*nphases + phnum;       /* +1 if not
Mark's numbering */
                if (sliceorder)tmpslice = ie;
                else     tmpslice = (ie <= (nslices+1)/2) ? (ie*2 - 1) :
((ie-(nslices+1)/2)*2);
                tmpslice -= 1;                                  /* for Mark's
numbering scheme */
                if (rev_flg) tmpslice = nslices - tmpslice - 1;
    /* printf("%d %d %f %f
%f\n",imnum,tmpslice,regxs[tmpslice][imnum],regys[tmpslice][imnum],regrot[tmp
slice][imnum]); */
        }

  /* convolution function ---- kaiser-bessel */
        /* This little for loop generates a lovely kaiser bessel gaussian-
shaped hump that is
                NWEIGHTS points wide with the peak at NWEIGHTS/2 */

        for (i=0; i < NWEIGHTS+1; i++){
                weight[i] = kaiser((float)NWEIGHTS,gridb,(float)(i-
NWEIGHTS/2));
        }
        for (i=0; i<nim*OVER; i++){                     /* initialize all
storage points to zero */
                for (j=0; j<nim*OVER; j++) {
                        sampim[i][j] = 0.0;
                        grim[0][i][j] = 0.0;    /* gridded image real */
                        grim[1][i][j] = 0.0;    /* gridded image complex */
                        ws[i][j] = 0.0;
                }
        }

        maxk = 0.;
        dwin = gridl;                                                   /*
data window size */
        rotfact = 2.0*PI/nim/OVER;
        for (i=0; i<npr; i++) {
                for (j=0; j< (ndat-samp1-1); j++) {
                        w2 = kdens[j];                                  /*
density weighting term */
                        kx = t2k[0][i][j];                              /*
get kx co-ordinate of data point*/
                        ky = t2k[1][i][j];                              /*
get ky co-ordinate of data point*/
                        if ( (mkr = hypot(kx,ky)) > maxk) maxk = mkr;   /*
```

```
keep track of max length of k vector */
                        pr = prd[0][i][j+samp1];                        /*
real part of data point */
                        pi = prd[1][i][j+samp1];                        /*
imaginary part of data point */
                        dkx = (factxx*kx + factxy*ky)*OVER;             /* kx
stretch and rotation */
                        dky = (factyx*kx + factyy*ky)*OVER;             /* ky
stretch and rotation */
                                        /* At this point dkx and dky are the
cartesian co-ordinates in strecthed and rotated kspace.
                                        dkx and dky both range from -
nim/2*OVER to nim/2*OVER */

                        if (reg_flg) {
                                rot1 = cos(rotfact*dkx*(pix_shifth*nim -
regxs[tmpslice][imnum] + lrshift));
                                rot2 = -sin(rotfact*dkx*(pix_shifth*nim -
regxs[tmpslice][imnum] + lrshift));
                                tmpd = pr*rot1 + pi*rot2;
                                pi = pi*rot1 - pr*rot2;
                                pr = tmpd;
                                rot1 = cos(rotfact*dky*(pix_shiftv*nim -
regys[tmpslice][imnum] + tbshift));
                                rot2 = -sin(rotfact*dky*(pix_shiftv*nim -
regys[tmpslice][imnum] + tbshift));
                                tmpd = pr*rot1 + pi*rot2;
                                pi = pi*rot1 - pr*rot2;
                                pr = tmpd;
                                rot1 = cos(-
PI/180.0*regrot[tmpslice][imnum]);
                                rot2 = -sin(-
PI/180.0*regrot[tmpslice][imnum]);
                                tmpd = dkx*rot1 + dky*rot2;
                                dky = dky*rot1 - dkx*rot2;
                                dkx = tmpd;
                        }
                        else if (lrshift || tbshift || loc_flg) {
                                rot1 = cos(rotfact*dkx*(pix_shifth*nim +
lrshift));
                                rot2 = -sin(rotfact*dkx*(pix_shifth*nim +
lrshift));
                                tmpd = pr*rot1 + pi*rot2;
                                pi = pi*rot1 - pr*rot2;
                                pr = tmpd;
                                rot1 = cos(rotfact*dky*(pix_shiftv*nim +
tbshift));
                                rot2 = -sin(rotfact*dky*(pix_shiftv*nim +
tbshift));
                                tmpd = pr*rot1 + pi*rot2;
                                pi = pi*rot1 - pr*rot2;
                                pr = tmpd;
                        }
                        dkx += (nim/2 - refl[2]*j)*OVER;                /*
translate dkx to start from zero to nim*OVER, and do linear correction*/
                        dky += (nim/2 - refl[1]*j)*OVER;                /*
translate dky to start from zero to nim*OVER, and do linear correction*/
```

```
                        for (lx = ceil(dkx-dwin); lx<=floor(dkx+dwin); lx++)
{
                                /* select a 2*dwin sized data window around
dkx and run the for loop
                                        for each integer value of dkx within
the window */


                                if ((lx<0) || (lx>=OVER*nim)) continue;


                                wx = weight[ (int) rint((((dkx-
lx)/dwin)*NWEIGHTS/2 )) + NWEIGHTS/2] *w2;
                                /* scale the length of the kaiser-bessel
window to the size of the data window
                                        around kx and find the appropriate
weight.  Also, multiply by the
                                        data density weighting factor w2
which comes from kdens[i] */

                                /* Now do the the exact same windowing and
weighting process in the y direction */
                                for (ly = ceil(dky-dwin);
ly<=floor(dky+dwin); ly++) {
                                        if ((ly<0) || (ly>=OVER*nim))
continue;
                                        w = wx*weight[ (int) rint((((dky-
ly)/dwin)*NWEIGHTS/2 )) + NWEIGHTS/2];
                                        sampim[lx][ly] += w * j;        /*
keeps track of time associated with each k-space location for hc-corr*/
                                        grim[0][lx][ly] += pr * w;      /*
sum contributions to real data */
                                        grim[1][lx][ly] += pi * w;      /*
sum contributions to imaginary data */
                                        ws[lx][ly] += w;                /*
sum all weights applied to a particular location in k-space*/
                                }
                        }
                }
        }

        for (i=0; i<nim*OVER; i++)
                for (j=0; j<nim*OVER; j++) {
                        sampim[i][j] /= (0.001+ws[i][j]);               /*
calculates the weighted average of acquisition time of each k-space point */
                }
        return SUCCESS;
}


int     load_ft2()                                      /* Do convolution for
regridding */
{
  int i, j, lx, ly;
  float kx,ky,w,pr,pi,mkr,dkx,dky,dwin,w2,wx;
```

```
  float tmpd,rot1,rot2,rotfact;


        for (i=0; i<nim*OVER; i++){
                for (j=0; j<nim*OVER; j++) {
                        grim[0][i][j] = 0.0;
                        grim[1][i][j] = 0.0;
                }
        }

        maxk = 0.;
        dwin = gridl;
        rotfact = 2.0*PI/nim/OVER;
        for (i=0; i<npr; i++) {
                for (j=0; j< (ndat-samp1-1); j++) {
                        w2 = kdens[j];
                        kx = t2k[0][i][j];
                        ky = t2k[1][i][j];
                        if ( (mkr = hypot(kx,ky)) > maxk) maxk = mkr;
                        pr = prd[0][i][j+samp1];
                        pi = prd[1][i][j+samp1];
                        dkx = (factxx*kx + factxy*ky)*OVER;
                        dky = (factyx*kx + factyy*ky)*OVER;
                        if (lrshift || tbshift || loc_flg) {
                                rot1 = cos(rotfact*dkx*(pix_shifth*nim +
lrshift));
                                rot2 = -sin(rotfact*dkx*(pix_shifth*nim +
lrshift));
                                tmpd = pr*rot1 + pi*rot2;
                                pi = pi*rot1 - pr*rot2;
                                pr = tmpd;
                                rot1 = cos(rotfact*dky*(pix_shiftv*nim +
tbshift));
                                rot2 = -sin(rotfact*dky*(pix_shiftv*nim +
tbshift));
                                tmpd = pr*rot1 + pi*rot2;
                                pi = pi*rot1 - pr*rot2;
                                pr = tmpd;
                        }
                        dkx += nim/2*OVER;
                        dky += nim/2*OVER;
                        for (lx = ceil(dkx-dwin); lx<=floor(dkx+dwin); lx++)
{
                                if ((lx<0) || (lx>=OVER*nim)) continue;
                                wx = weight[ (int) rint((((dkx-
lx)/dwin)*NWEIGHTS/2 )) + NWEIGHTS/2] *w2;
                                for (ly = ceil(dky-dwin);
ly<=floor(dky+dwin); ly++) {
                                        if ((ly<0) || (ly>=OVER*nim))
continue;
                                        w = wx*weight[ (int) rint((((dky-
ly)/dwin)*NWEIGHTS/2 )) + NWEIGHTS/2];
                                        grim[0][lx][ly] += pr * w;
                                        grim[1][lx][ly] += pi * w;
                                }
                        }
                }
```

```
        }
        return SUCCESS;
}


int     load_projs()                                    /* read in spiral
data */
{
  int i, j,k, n, chopper=1;
  int nodd;
  int coiloff, nproj, offs, maskoffs;
  int phindex;
  float cs,sn,tr,ti;
  static int first_run = 1;
  int temp_offset;
/*  int fold = 4;*/

        printf("nzpe in load_proj is %d\n", nzpe);
        nodd = (nphmult*npr*nzpe*(concat+1))%2;

        /*ZZH, 08/12/03*/
        if (field2d_flg)
                zpenum=0;

        phindex=phnum*nzpe+zpenum;

        if(extract_flg&&phnum)
                phindex=(phnum-1)*nzpe+zpenum;

                        /* openfile and read header */
        if ((phindex == 0) && ((slnum ==0) || (isl != 0)) && (coilnum == 0)){
                if (!(fi=fopen(uncfile,"r"))) {
                        fprintf(stderr,"Can't open %s!\n",uncfile);
                        return(0);
                }

/*                      if (!(fi2=fopen(uncfile,"r"))) {
                        fprintf(stderr,"Can't open %s!\n",uncfile);
                        return(0);
                }*/
                                        /* header */
                fread(&header,sizeof(header),1,fi);
        }

        nproj = slnum*((concat+1)*nphmult*npr*nzpe+1+nodd) +
(concat+1)*phindex*npr;

/*      if (pmdomap)
                station_offs = istation * (1+imnpr*nzpe +
2*pmnl*nzpe)*frsize*2*opptsize;
        else    station_offs = istation * (1+imnpr*nzpe
)*frsize*2*opptsize;*/

        if (pmdomap)
                station_size = (1+imnpr*nzpe +
2*pmnl*nzpe)*frsize*2*opptsize;
        else    station_size = (1+imnpr*nzpe )*frsize*2*opptsize;
```

```
        station_offs = istation * station_size;

        if (out_flg)    printf("Multi-Station Offset = %d\n", station_offs);


        /*ZZH, 04/08/2003, changed */
        temp_offset=0;
       if(ts==8.E-6&&!(int)fast_rec_lpf)
              temp_offset=OFFSET_PA_COIL2;
       if(ts==4.E-6&&!(int)fast_rec_lpf)
              temp_offset=OFFSET_PA_COIL1;
       if(ts==0.0000025)
              temp_offset=OFFSET_PA_COIL;
       if(out_flg) printf("\nts=%.8f, temp_offset=%d\n", ts, temp_offset);
        coiloff = mcskip*coilnum+temp_offset;

        if (mask_flg) {
                maskoffs = sizeof(header) + im_baseline +
nproj*frsize*2*opptsize + coiloff;/* HZ */
                if (out_flg) printf("maskoffs = %d\n", maskoffs);
        }
        offs = sizeof(header) + im_baseline + pmimg_offs +
nproj*frsize*2*opptsize + coiloff + station_offs;
        if(out_flg)     printf("\nslnum = %d, phindex = %d, npr = %d,frsize =
%d, phnum = %d, coilnum = %d, nproj = %d,ndatfr = %d, row_offset = %d,offs =
%d, pmimg_offs = %d,
coiloff=%d",slnum,phindex,npr,frsize,phnum,coilnum,nproj,ndatfr,row_offset,of
fs,pmimg_offs,coiloff); /*12/21/01 JQL turn on*/

        if(!base_flg){          /*ZZH, 06/16/03*/
        if (first_run) {
          mean[0] = 0.0;
          mean[1] = 0.0;
          fseek(fi, sizeof(header), 0);
          if ((i=fread(ib1short,2*opptsize,ndatfr,fi))!=ndatfr) return(0);
          for(k=0;k<ndatfr;k++){
                mean[0] += ib1short[k][0];
                mean[1] += ib1short[k][1];
          }
          mean[0] /= ndatfr;
          mean[1] /= ndatfr;
          if(out_flg) printf("Mean value is %f + %f i\n",mean[0],mean[1]);
          first_run = 0;
        }
        }
        else{
          mean[0] = 0.0;
          mean[1] = 0.0;
        }

        fseek(fi,offs,0);
/*      fseek(fi2,offs+station_size,0);*/
        for (n=0; n<npr; n++) {
                if(opptsize==2){
                        if ((i=fread(ib1short,2*opptsize,ndatfr,fi))!=ndatfr)
return(0);
```

```
                        if (mask_flg){
                                fseek(fi,-(offs-maskoffs)-
2*opptsize*ndatfr,1);
                                if
((i=fread(ib1mask,2*opptsize,ndatfr,fi))!=ndatfr) return(0);
                                fseek(fi,(offs-maskoffs),1);
                        }
                        if (nsub>0){
                                fseek(fi,station_size-2*opptsize*ndatfr,1);
                                if
((i=fread(ib1short2,2*opptsize,ndatfr,fi))!=ndatfr) return(0);
                                fseek(fi,-station_size,1);
                        }
                        if (n%fold<nsub)
                        {
                                for(k=0;k<ndatfr;k++){
                                        ib1[k][0] = (int)ib1short2[k][0];
                                        ib1[k][1] = (int)ib1short2[k][1];
                                }
                        }
                        else
                        {
                                for(k=0;k<ndatfr;k++){
                                        ib1[k][0] = (int)ib1short[k][0];
                                        ib1[k][1] = (int)ib1short[k][1];
                                }
                        }

                        if (mask_flg){

                                for(k=0;k<ndatfr;k++){
                                        ib1[k][0] -= (int)ib1mask[k][0];
                                        ib1[k][1] -= (int)ib1mask[k][1];
                                }
                        }
                }
                else
                        if ((i=fread(ib1,2*opptsize,ndatfr,fi))!=ndatfr)
return(0);
                fseek(fi,row_offset*2*opptsize,1);
/*              fseek(fi2,row_offset*2*opptsize,1);*/
                for (j=0; j<i; j++) {
                        prd[0][n][j] = (float)ib1[j][0] - mean[0];
                        prd[1][n][j] = (float)ib1[j][1] - mean[1];
/*PDT add constant to raw data test */

                        prd[0][n][j] *= chopper;
                        prd[1][n][j] *= chopper;
                }
                if (concat) {
                        if(opptsize==2){
                                if
((i=fread(ib1short,2*opptsize,ndatfr,fi))!=ndatfr) return(0);
                                for(k=0;k<ndatfr;k++){
                                        ib1[k][0] = (int)ib1short[k][0];
                                        ib1[k][1] = (int)ib1short[k][1];
                                }
```

```
                        }
                        else
                                if
((i=fread(ib1,2*opptsize,ndatfr,fi))!=ndatfr) return(0);
                        fseek(fi,row_offset*2*opptsize,1);
                        for (j=0; j<i; j++) {
                                prd[0][n][j+ndatfr+ngap] =
(float)ib1[j][0]*chopper;
                                prd[1][n][j+ndatfr+ngap] =
(float)ib1[j][1]*chopper;
                        }
                }
                chopper *= chop;
        }

        if ((phindex == (nphases*nzpe - 1)) && ((slnum == (nslices - 1)) ||
(isl != 0)) && coilnum == (ncoils - 1)){
                fclose(fi);
        /*      fclose(fi2);*/
        }

        return(1);
        return SUCCESS;
}



int     load_ref_hc(int ie, int pe) {

  int i, j;
  float bm[MXNIM];
  char fn[80];

        if (pe>99) sprintf(fn,"ref.%d.co%1d.sl%1d.pe%d",nim,coilnum,ie,pe);
        else if(pe>9)
sprintf(fn,"ref.%d.co%1d.sl%1d.pe0%d",nim,coilnum,ie,pe);
        else sprintf(fn,"ref.%d.co%1d.sl%1d.pe00%d",nim,coilnum,ie,pe);

        if (!(fo1=fopen(fn,"r"))) {fprintf(stderr,"Can't open reference file
%s!\n",fn); return(0); }

        for (i=0; i<nim; i++) {
                fread(bm,nim,sizeof(*bm),fo1);
                for (j=0; j<nim; j++) {
                        refim[i][j][pe] = bm[j];
                }
        }

        /*ZZH, 12/05/2002, to find wmax*/
        wmax[pe] = refim[0][0][pe];
        wmin[pe] =  refim[0][0][pe];
        for (i = 0; i<nim; i++) {
                for (j = 0; j<nim; j++) {
                        if (refim[i][j][pe]>wmax[pe]) {
                                wmax[pe] = refim[i][j][pe];
                        }
                        if (refim[i][j][pe]<wmin[pe]) {
```

```
                                wmin[pe] = refim[i][j][pe];
                        }
                }
        }
        wmax[pe]/=mapdel;
        wmin[pe]/=mapdel;

        if(out_flg){
                printf("wmax=%.7f\t", wmax[pe]);
                printf("wmin=%.7f\t", wmin[pe]);
        }
        /*end ZZH*/

        fclose(fo1);
        return SUCCESS;
}

int     load_ref_lincor(int ie, int npe) {      /* Get three terms to define
plane for linear field map correction */


  char fn[80];

        if(npe>99) sprintf(fn,"refl.co%1d.sl%1d.pe%d",coilnum,ie,npe);
        else if(npe>9) sprintf(fn,"refl.co%1d.sl%1d.pe0%d",coilnum,ie,npe);
        else sprintf(fn,"refl.co%1d.sl%1d.pe00%d",coilnum,ie,npe);
        if (!(fo1=fopen(fn,"r"))) {fprintf(stderr,"Can't open reference file
%s!\n",fn); return(0); }
        fread(refl,3,sizeof(*refl),fo1);
        fclose(fo1);
    /*printf("slice = %d, pe = %d, constant = %f, x-grad = %f, y-grad =
%f\n",ie,npe,refl[0],refl[1],refl[2]);*/
        return SUCCESS;
}


#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
void loc_calc(p1,p2,p3,rot,trans)
float *p1,*p2,*p3;
int rot,trans;
{
  /*
    top left     - p1
    bottom left  - p2
    top right    - p3
    bottom right - p4
  */
  float p4[3],tempr;
  int i,n;

  /* calc missing corner (bottom right) */
        for (i=0; i<3; i++)     p4[i] = p3[i] + (p2[i] - p1[i]);

        if (trans) {
                for (i=0; i<3; i++) {
                        SWAP(p2[i],p3[i]);
                }
```

```
        }

        for (n=0; n<rot; n++) {
                for (i=0; i<3; i++) {
                        SWAP(p1[i],p3[i]);
                        SWAP(p3[i],p4[i]);
                        SWAP(p4[i],p2[i]);
                }
        }
  /* reformat to match image header */
        for (i=0; i<3; i++) {
                SWAP(p2[i],p3[i]);
                SWAP(p3[i],p4[i]);
        }
}
#undef SWAP



int     make_final(int segnum,int npe) {
/*add contributions from all segs, the way it does is to multiply each
segment data with exp(i*w*tj), where tj=nT/N, the center of each segment*/

  int i, j,offs;
  float phfact;

        phfact = -segnum;
        offs = (OVER-1)*nim/2;
        if (segnum == 0) {
                for (i=0; i<nim; i++) {
                        for (j=0; j<nim; j++) {
                                finalim[0][i][j] = im[0][i+offs][j+offs];
                                finalim[1][i][j] = im[1][i+offs][j+offs];
                        }
                }
        }
        else for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++){
                        finalim[0][i][j] +=

im[0][i+offs][j+offs]*cos(phfact*refim[i][j][npe]*SEGSIZE/SCALE/mapdel) -

im[1][i+offs][j+offs]*sin(phfact*refim[i][j][npe]*SEGSIZE/SCALE/mapdel);
                        finalim[1][i][j] +=

im[0][i+offs][j+offs]*sin(phfact*refim[i][j][npe]*SEGSIZE/SCALE/mapdel) +

im[1][i+offs][j+offs]*cos(phfact*refim[i][j][npe]*SEGSIZE/SCALE/mapdel);
                }
        }
        if (segnum == nsegs){
                for (i=0; i<nim; i++){
                        for (j=0; j<nim; j++){
                                im[0][i+offs][j+offs] = finalim[0][i][j];
                                im[1][i+offs][j+offs] = finalim[1][i][j];
                        }
                }
```

```
        }
        return SUCCESS;
}

int     make_phase1_filt(int npe)  /* smoothe phase data by summing over a
filter window */
{
          int i, j, k, offs;
          int hfsize;
          float fbuf[2][OVER*MXNIM];
          float sumr,sumi;
          float bm[MXNIM];
          char  fn[80];

          hfsize = (fsize-1)/2;
          offs = (OVER-1)*nim/2;

        for (i=0; i<nim; i++) {
                for (j= MAX(0,(offs-hfsize)); j<=
MIN((nim+offs+hfsize),OVER*MXNIM-1); j++){
                        fbuf[0][j] = 0.;
                        fbuf[1][j] = 0.;
                        for (k= MAX(0,(i+offs-hfsize)); k<=
MIN((i+offs+hfsize),OVER*MXNIM-1); k++){
                                fbuf[0][j] += im[0][k][j];
                                fbuf[1][j] += im[1][k][j];
                        }
                }
                for (j=0; j<nim; j++) {
                        sumr = sumi = 0.;
                        for (k= MAX(0,(j+offs-hfsize)); k<=
MIN((j+offs+hfsize),OVER*MXNIM-1); k++){
                                sumr += fbuf[0][k];
                                sumi += fbuf[1][k];
                        }
                        refim[i][j][npe] = atan2(sumi,sumr);
                }
        }
        /*ZZH, 12/03/2002*/
        if (phs_flg) {
                sprintf(fn,"refa.%d.pe%1d",nim,npe);
                if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0); }
                for (i=0; i<nim; i++) {
                        for (j=0; j<nim; j++) {
                                        bm[j] = refim[i][j][npe];
                        }
                        fwrite(bm,nim,sizeof(*bm),fo1);                 /*
write all the phase data to file */
                }
                fclose(fo1);
        }
        /*end ZZH*/
        return SUCCESS;
}

int     make_phase2_filt(int npe)  /* smoothe phase data and calculate phase
```

```
difference for phase map */
{
          int i, j, k, offs;
          int hfsize;
          float fbuf[2][OVER*MXNIM];
          float sumr,sumi,phaseoff;
          float tmp[MXNIM][MXNIM];
          float bm[MXNIM];
          char  fn[80];


          /* phaseoff is value to offset freq map if phase twist (pt) != 0 */
        phaseoff = -2.0*PI*(mapdel/samptime)*pt/ndat;
        hfsize = (fsize-1)/2;
        offs = (OVER-1)*nim/2;

        for (i=0; i<nim; i++){
                for (j= MAX(0,(offs-hfsize)); j<=
MIN((nim+offs+hfsize),OVER*MXNIM-1); j++){
                        fbuf[0][j] = 0.;
                        fbuf[1][j] = 0.;
                        for (k= MAX(0,(i+offs-hfsize)); k<=
MIN((i+offs+hfsize),OVER*MXNIM-1); k++){
                                fbuf[0][j] += im[0][k][j];
                                fbuf[1][j] += im[1][k][j];
                        }
                }
                for (j=0; j<nim; j++) {
                        sumr = sumi = 0.;
                        for (k= MAX(0,(j+offs-hfsize)); k<=
MIN((j+offs+hfsize),OVER*MXNIM-1); k++){
                                sumr += fbuf[0][k];
                                sumi += fbuf[1][k];
                        }
                        refimmag[i][j][npe] = hypot(sumi,sumr);
                        tmp[i][j]=atan2(sumi,sumr);
                        refim[i][j][npe] -= (atan2(sumi,sumr) - phaseoff);
                        /*refim[i][j][npe] *= -1;                         ZZH,
12/27/02, make field map to be phi2-phi1*/
                }
        }
        /*ZZH, 12/03/2002*/
        if (phs_flg) {
                sprintf(fn,"refb.%d.pe%1d",nim,npe);
                if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0); }
                for (i=0; i<nim; i++) {
                        for (j=0; j<nim; j++) {
                                bm[j] = tmp[i][j];
                        }
                        fwrite(bm,nim,sizeof(*bm),fo1);                 /*
write all the phase data to file */
                }
                fclose(fo1);
        }
        /*end ZZH*/
        return SUCCESS;
```

```
}

int     make_nav(int npe, int phn) {

  int i, j;
  float prr, pri, por, poi, cc, aa, bb;

  /* get zeros from first image */
        if(phn==0) {
                p3d0[0][npe] = p3d0[1][npe] = 0.;
                for (i=0; i<npr; i++){
                        for (j=0; j<= samp1; j++) {
                                p3d0[0][npe] += prd[0][i][j];
                                p3d0[1][npe] += prd[1][i][j];
                        }
                        p3d0[0][npe] /= (1.0 * npr);
                        p3d0[1][npe] /= (1.0 * npr);
                }
        }

  /* get zeros from image */
        por=p3d0[0][npe];
        poi=p3d0[1][npe];
        for (i=0; i<npr; i++) {
                prr = pri = 0.;
                for (j=0; j<= samp1; j++) {
                        prr += prd[0][i][j];
                        pri += prd[1][i][j];
                }
                cc=hypot(prr,pri)*hypot(por,poi);
                aa=(prr*por+pri*poi)/cc;
                bb=(prr*poi-pri*por)/cc;
    /*phase_diff= atan2(prr,pri)-atan2(p3d0[0][npe],p3d0[1][npe]);
    if (phase_diff >  PI) phase_diff -= 2.0*PI;
    if (phase_diff < -PI) phase_diff += 2.0*PI;
    co=cos(phase_diff);
    si=sin(phase_diff);
    printf("aa=%f bb=%f\n",aa,bb);
    printf("co=%f si=%f\n",co,si);*/
                for (j=0; j< ndat; j++) {
                        prr = prd[0][i][j];
                        pri = prd[1][i][j];
                        prd[0][i][j] = aa*prr - bb*pri;
                        prd[1][i][j] = aa*pri + bb*prr;
                }
        }
        return SUCCESS;
}


int     read_inputs()           /* gets user defined parameters for
reconstruction */
{
  int   k;
  float tmp1,tmp2;

        printf("\nUser Inputs:\n");
```

```
        if (out_flg) printf("Image Size: ");
        /*scanf("%d",&nim);*/           /*ZZH, 04/22/03*/
        nim=256;
        if (out_flg) printf("%d\n",nim);

        if (out_flg) printf("Slice No (0->all): ");
        /*scanf("%d", &isl);*/          /*ZZH, 04/22/03*/
        isl=0;
        if (out_flg) printf("%d\n",isl);

        if (zip_flg) fprintf(ff,"%d\n",nim*2);
                else fprintf(ff,"%d\n",nim); /*09/18/01 JQL*/

        if (out_flg) printf("Input Delay (# of samples): ");
        /*scanf("%d",&samp1);*/         /*ZZH, 04/22/03*/
        samp1=0;
        if (out_flg) printf("%d\n",samp1);

        if (out_flg) printf("Phase Twist: ");
        /*scanf("%f",&pt);*/            /*ZZH, 04/22/03*/
        pt=0;
        if (out_flg) printf("%.2f\n",pt);


        samp_cor = 1;

        /*
          printf("Sample Density Correction (0,1):\n");
          scanf("%d", &samp_cor);
          printf("Flip tb (0,1), flip lr (0,1), rotate (0-3):\n");
          scanf("%d %d %d", &fliptb, &fliplr, &numrot);
          */

        if (out_flg) printf("lr shift: ");
        /*scanf("%d",&lrshift);*/               /*ZZH, 04/22/03*/
        lrshift=0;
        if (out_flg) printf("%d\n",lrshift);

        if (out_flg) printf("tb shift: ");
        /*scanf("%d",&tbshift);*/               /*ZZH, 04/22/03*/
        tbshift=0;
        if (out_flg) printf("%d\n",tbshift);

        if (out_flg) printf("zoom: ");
        /*scanf("%f",&zoomer);*/                /*ZZH, 04/22/03*/
        zoomer=1;
        if (out_flg) printf("%.2f\n",zoomer);

        factxx = -1.0/zoomer; factxy = 0.0; factyx = 0.0; factyy = -
1.0/zoomer;
          /*
          if (fliptb) { factyx *= -1.0; factyy *= -1.0; }
          if (fliplr) { factxx *= -1.0; factxy *= -1.0; }
          for (k = 0; k < numrot; k++) {
            tmp1 = factyx; tmp2 = factyy;
            factyx = factxx; factyy = factxy;
```

```
            factxx = -tmp1; factxy = -tmp2;
          }
          */
        if (loc_flg) {
                if(revsp_flg) {
                                                                /*pix_shifth
= -(*h_rec).rdb_hdr_yoff/(*h_rec).rdb_hdr_im_size;

pix_shiftv = *h_rec).rdb_hdr_xoff/(*h_rec).rdb_hdr_im_size;*/
                        pix_shifth = (*h_rec).rdb_hdr_yoff/128;
                        pix_shiftv = -(*h_rec).rdb_hdr_xoff/128;
                }
                else  {
                        pix_shifth =
(*h_rec).rdb_hdr_yoff/(*h_rec).rdb_hdr_im_size;
                        pix_shiftv = -
(*h_rec).rdb_hdr_xoff/(*h_rec).rdb_hdr_im_size;
                }
                printf("  shifth = %f, shiftv = %f (in
mm)\n",pix_shifth*opfov*10,pix_shiftv*opfov*10);
        }
        else    pix_shifth = pix_shifth = 0.0;

          /* uses parameters from header file */
        factxx = -1.0/zoomer; factxy = 0.0; factyx = 0.0; factyy = -
1.0/zoomer;
        if ((*h_rec).rdb_hdr_transpose) {
                tmp1 = factyx; tmp2 = factyy;
                factyx = factxx; factyy = factxy;
                factxx = tmp1; factxy = tmp2;
        }
        for (k = 0; k < (*h_rec).rdb_hdr_rotation; k++) {
                tmp1 = factyx; tmp2 = factyy;
                factyx = -factxx; factyy = -factxy;
                factxx = tmp1; factxy = tmp2;
        }

        if (out_flg) printf("Mag Cor Factor: ");
        /*scanf("%f",&magfact);*/               /*ZZH, 04/22/03*/
        magfact=0;
        if (out_flg) printf("%.2f\n",magfact);

        if (out_flg) printf("Phase Cor Factor: ");
        /*scanf("%f",&phasefact);*/             /*ZZH, 04/22/03*/
        phasefact=0;
        if (out_flg) printf("%.2f\n",phasefact);

        if (out_flg) printf("Filter Size (odd): ");
        /*scanf("%d",&fsize);*/         /*ZZH, 04/22/03*/
        fsize=7;
        if (out_flg) printf("%d\n",fsize);

        if (out_flg) printf("Mapdel from inputs: ");
        /*scanf("%lf",&mapdel);*/               /*ZZH, 04/22/03*/
        mapdel=2000;
        if (out_flg) printf("%.2fus\n",mapdel);
```

```
        if (out_flg) printf("delta_pt: ");
        /*scanf("%f",&delta_pt);*/              /*ZZH, 04/22/03*/
        delta_pt=0;
        if (out_flg) printf("%.2f\n",delta_pt);

        /*ZZH, 04/22/03*/
        if(extract_flg){
                if (out_flg) printf("Extract matrix Size: ");
                scanf("%d", &extract_size);
                /*extract_size=32;*/
                if (out_flg) printf("%d\n", extract_size);
        }

        return SUCCESS;
}

int     refocus(){
  int i, j, j2;
  float tr, ti,cs,sn,pt2;

        pt2 = (fast_rec_lpf*1e3*ts);            /* 100kHz * 2.5E-6 seconds =
0.25 cycles */
        for (j=0; j<ndatfr; j++) {
                cs = cos(-2.0*PI*(j*pt2 + j*pt/ndat) - refl[0]*j);
                sn = sin(-2.0*PI*(j*pt2 + j*pt/ndat) - refl[0]*j);
                for (i=0; i<npr; i++) {
                        tr = prd[0][i][j]; ti = prd[1][i][j];
                        prd[0][i][j] = tr*cs - ti*sn;
                        prd[1][i][j] = tr*sn + ti*cs;
                                                        /* printf("%f %f\n",
prd[0][i][j],prd[1][i][j]); */
                }
        }
        if (concat) {
                for (j=0; j<ndatfr; j++) {
                        j2 = j+ndatfr+ngap;
                        cs = cos(-2.0*PI*(j2*pt2 + j2*pt/ndat) - refl[0]*j2);
                        sn = sin(-2.0*PI*(j2*pt2 + j2*pt/ndat) - refl[0]*j2);
                        for (i=0; i<npr; i++) {
                                tr = prd[0][i][j2]; ti = prd[1][i][j2];
                                prd[0][i][j2] = tr*cs - ti*sn;
                                prd[1][i][j2] = tr*sn + ti*cs;
                                                        /* printf("%f %f\n",
prd[0][i][j2],prd[1][i][j2]); */
                        }
                }
                                                        /* fill in missing
points in gap between views for concat readouts */
                for (i=0; i<npr; i++) {
                        for (j=0; j<ngap; j++) {
                                prd[0][i][ndatfr+j] = 0.5*(prd[0][i][ndatfr-
1] + prd[0][i][ndatfr+ngap]);
                                prd[1][i][ndatfr+j] = 0.5*(prd[1][i][ndatfr-
1] + prd[1][i][ndatfr+ngap]);
                        }
                }
                                        /* for (j=ndatfr-5; j<ndatfr+5; j++)
```

```
                                                printf("%f
%f\n",prd[0][0][j],prd[1][0][j]); */

        }
        return SUCCESS;
}

int     refocus_z(int npe)
{
  int i,j;
  float tr,ti,cs,sn,zphase;

        zphase=2.0*PI*delta_pt*npe/(ndat*(nzpe-1));

        for (j=0; j<ndat; j++) {
                cs = cos(-2.0*PI*j*pt/ndat - refl[0]*j - zphase*j);
                sn = sin(-2.0*PI*j*pt/ndat - refl[0]*j - zphase*j);
                for (i=0; i<npr; i++) {
                        tr = prd[0][i][j];
                        ti = prd[1][i][j];
                        prd[0][i][j] = tr*cs - ti*sn;
                        prd[1][i][j] = tr*sn + ti*cs;
                }
        }
        return SUCCESS;
}

/*refocus_z2(int npe, int ie)
{
  int i,j,tmpslice;
  float tr,ti,cs,sn,zphase,slphase,cphase;

  cphase=2.0*PI*pt/ndat;
  zphase=2.0*PI*delta_pt*npe/(ndat*(nzpe-1));
  tmpslice = (ie <= (nslices+1)/2) ? (ie*2 - 1) : ((ie-(nslices+1)/2)*2);
  slphase=2.0*PI*GAM*gradstep*sliceloc[tmpslice-1]*(npe-
nzpe/2)*(.000008)/ndat;

  printf("tmpslice=%d cphase=%f slphase=%f\n",tmpslice,cphase,slphase*1000);

  for (j=0; j<ndat; j++) {
    cs = cos(-cphase*j - refl[0]*j - zphase*j - slphase*j);
    sn = sin(-cphase*j - refl[0]*j - zphase*j - slphase*j);
    for (i=0; i<npr; i++) {
      tr = prd[0][i][j];
      ti = prd[1][i][j];
      prd[0][i][j] = tr*cs - ti*sn;
      prd[1][i][j] = tr*sn + ti*cs;
    }
  }
}*/


int     reverse_order() {                       /* reverse order of data for
reverse spiral acquisition */

  int i, j;
```

```
  float tempr[MXNDAT], tempi[MXNDAT];

        for (i=0; i<npr; i++) {
                for (j=0; j<ndat; j++) {
                        tempr[j] = prd[0][i][ndat-j-1];
                        tempi[j] = prd[1][i][ndat-j-1];
                }
                for (j=0; j<ndat; j++) {
                        prd[1][i][j] = tempr[j];
                        prd[0][i][j] = tempi[j];
                }
        }
        return SUCCESS;
}


int     rev_ref_hc(int ie, int pe) {

  int i, j;
  float temp;

        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        temp = -refim[i][j][pe];
                        refim[i][j][pe] = temp;
                }
        }
        return SUCCESS;
}


int     rev_ref_lc() {

        refl[0]=-refl[0];
        refl[1]=-refl[1];
        refl[2]=-refl[2];

        return SUCCESS;
}

int     wind_dat(midsamp,sampwind)                      /* Window data for
time segmented correction */
int midsamp,sampwind;
{
  float ww,rad;
  float stndat, enndat;
  int i,j;

        stndat = midsamp - sampwind;
        enndat = midsamp + sampwind;

        for(i=0; i<nim*OVER; i++) {
                for (j=0; j<nim*OVER; j++) {
                        rad = sampim[i][j];
                        if ((rad > stndat) && (rad < enndat)){
                                ww = .5*(1.0 + cos(PI*(rad-
midsamp)/sampwind));         /* this is a nice windowing weighting function
```

```
that

ensures that the sum of all overlapping segments is unity */
                        }
                        else ww = 0.;
                        im[0][i][j] = ww*grim[0][i][j];                 /*
puts a little window over only the data we are interested in for this segment
*/
                        im[1][i][j] = ww*grim[1][i][j];
                }
        }
        return SUCCESS;
}

int     write_image(int nc,int ie,int nph,int nraw, int npe) {

  int i, j, mx,offs,tmpslice,imnum;
  float imx, imi, imr, imm, wctmp, wcmax;
  float wc[MXNIM];
  short int bm[MXNIM];
  char fn[80];
  int znorm;

        if(noz_flg)
                znorm=1;
        else
                znorm=nzpe;


  /* kaiser-bessel correction */
        wcmax = sinh(sqrt(gridb*gridb))/sqrt(gridb*gridb);

        for (i=0; i<nim; i++) {
                wctmp = PI*PI*gridl*gridl*(i-nim/2)*(i-
nim/2)/(nim*nim*OVER*OVER/4) - gridb*gridb;
                if(wctmp == 0.)         wc[i] = 1.0*wcmax;
                else if(wctmp < 0.)      wc[i] = sqrt(-wctmp)/sinh(sqrt(-
wctmp))*wcmax;
                else                    wc[i] =
sqrt(wctmp)/sin(sqrt(wctmp))*wcmax;
        }

        imnum = imoffset + nraw*nphases + (nph-1)*nzpe + npe;
        if (sliceorder) tmpslice = ie;
        else    tmpslice = (ie <= (nslices+1)/2) ? (ie*2 - 1) : ((ie-
(nslices+1)/2)*2);

        if (rev_flg) tmpslice = nslices - tmpslice + 1;

        if (imnum > 99)         sprintf(fn,"sl%1d.%1d.%d",tmpslice,nc,imnum);
        else if (imnum >9)
sprintf(fn,"sl%1d.%1d.0%d",tmpslice,nc,imnum);
        else
sprintf(fn,"sl%1d.%1d.00%d",tmpslice,nc,imnum);

        if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open %s!\n",fn);
return(0); }
```

```
        imx = 0.0;
        offs = (OVER-1)*nim/2;
        for (i=0; i<nim; i++) {         /* grab only the center circle of the
image of radius nim and multiply by weights*/
                for (j=0; j<nim; j++) {
                        if (hypot((float)(i-nim/2),(float)(j-nim/2)) <
.51*nim){
                                imr = im[0][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];
                                imi = im[1][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];
                                imm = sqrt(imi*imi+imr*imr);            /*
keep track of largest pixel magnitude */
                                if (imm > imx) imx = imm;
                                bm[j] =
512*imm/(nim*nim*OVER*OVER)/(znorm/4);
                        }
                        else bm[j] = 0;
                }
                fwrite(bm,nim,2,fo1);
        }
        mx = imx*512/(nim*nim*OVER*OVER)/(znorm);
        if (out_flg) printf("max=%3d",mx);


        fclose(fo1);

        return SUCCESS;
}

int     write_imagemc(nc,ie,nph,nraw)
int nc,ie,nph,nraw;
{
  int i, j, mx,offs,tmpslice,imnum;
  float imx, imi, imr, imm, imt, wctmp, wcmax;
  float wc[MXNIM];
  short imtmp[MXNIM*MXNIM];
  short int bm[MXNIM];
  char fn[80];


        imnum = imoffset + nraw*nphases + nph;
        if (sliceorder) tmpslice = ie;
        else            tmpslice = (ie <= (nslices+1)/2) ? (ie*2 - 1) : ((ie-
(nslices+1)/2)*2);
        if (rev_flg) tmpslice = nslices - tmpslice + 1;

        if (imnum > 99)         sprintf(fn,"sl%1d.%d",tmpslice,imnum);
        else if (imnum >9)      sprintf(fn,"sl%1d.0%d",tmpslice,imnum);
        else                    sprintf(fn,"sl%1d.00%d",tmpslice,imnum);

        if (nc != 1) {
                if (!(fo1=fopen(fn,"r"))) fprintf(stderr,"Can't open
%s!\n",fn); return(0);
                if (fread(imtmp,sizeof(short),nim*nim,fo1) != nim*nim ) {
                        fprintf(stderr,"Error reading %s!\n",fn); return(0);
```

```
                }
                fclose(fo1);
        }
        else {
                for (i=0; i<nim*nim; i++)       imtmp[i] = 0;
        }

        if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open %s!\n",fn);
return(0); }

  /* kaiser-bessel correction */
        wcmax = sinh(sqrt(gridb*gridb))/sqrt(gridb*gridb);

        for (i=0; i<nim; i++) {
                wctmp = PI*PI*gridl*gridl*(i-nim/2)*(i-
nim/2)/(nim*nim*OVER*OVER/4) - gridb*gridb;
                if(wctmp == 0.)         wc[i] = 1.0*wcmax;
                else if(wctmp < 0.)     wc[i] = sqrt(-wctmp)/sinh(sqrt(-
wctmp))*wcmax;
                else                    wc[i] =
sqrt(wctmp)/sin(sqrt(wctmp))*wcmax;
        }

        imx = 0.0;
        offs = (OVER-1)*nim/2;
        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        if (hypot((float)(i-nim/2),(float)(j-nim/2)) <
.51*nim){
                                imt = imtmp[i*nim + j];
                                imt *= (nim*nim*OVER*OVER)/512.0;
                                imr = im[0][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];
                                imi = im[1][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];
                                imm = sqrt(imi*imi+imr*imr + imt*imt);
                                if (imm > imx) imx = imm;
                                bm[j] = 512*imm/(nim*nim*OVER*OVER);
                        }
                        else    bm[j] = 0;
                }
                fwrite(bm,nim,2,fo1);
        }
        mx = imx*512/(nim*nim*OVER*OVER);
        if (out_flg) printf("max=%3d",mx);
        fclose(fo1);
        return SUCCESS;
}


int     write_imagemc_complex(nc,ie,nph,nraw,npe)
int nc,ie,nph,nraw;
{
  int i, j,mx,offs,tmpslice, tmp_ph;
  float imi, imr, immr, immi, imt, wctmp, wcmax;
  float wc[MXNIM];
  short imtmp[2*MXNIM*MXNIM];
```

```
  short int bm[2*MXNIM];
  short int bm2[MXNIM];
  char fn[80];
  int znorm;

        if(noz_flg)
                znorm = 1;
        else    znorm=nzpe;

        tmpslice = ie;
        tmp_ph=(map_flg || linmap_flg)?(nph+1):nph;

        if (npe > 99) {
                sprintf(fn,"sl%1d.pe%d.complex.00%d",tmpslice,npe,tmp_ph);
        }
        else if (npe >9) {
                sprintf(fn,"sl%1d.pe0%d.complex.00%d",tmpslice,npe,tmp_ph);
        }
        else {
                sprintf(fn,"sl%1d.pe00%d.complex.00%d",tmpslice,npe,tmp_ph);
        }

        if (nc != 1) {
                if (!(fo1=fopen(fn,"r"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0);}
                if (fread(imtmp,sizeof(short),2*nim*nim,fo1) != 2*nim*nim ) {
                        fprintf(stderr,"Error reading %s!\n",fn); return(0);
                }
                fclose(fo1);
        }
        else {
                for (i=0; i<nim*nim; i++)       imtmp[i] = 0;
        }

        if (!(fo1=fopen(fn,"w"))) {
                fprintf(stderr,"Can't open %s!\n",fn); return(0);
        }

  /* kaiser-bessel correction */
        wcmax = sinh(sqrt(gridb*gridb))/sqrt(gridb*gridb);

        for (i=0; i<nim; i++) {
                wctmp = PI*PI*gridl*gridl*(i-nim/2)*(i-
nim/2)/(nim*nim*OVER*OVER/4) - gridb*gridb;
                if(wctmp == 0.)         wc[i] = 1.0*wcmax;
                else if(wctmp < 0.)      wc[i] = sqrt(-wctmp)/sinh(sqrt(-
wctmp))*wcmax;
                else                    wc[i] =
sqrt(wctmp)/sin(sqrt(wctmp))*wcmax;
        }

        offs = (OVER-1)*nim/2;
        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        if (hypot((float)(i-nim/2),(float)(j-nim/2)) <
.51*nim){
                                imr = im[0][-j+nim+offs-1][-i+nim+offs-
```

```
1]*wc[i]*wc[j];
                                imi = im[1][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];

                                imt = imtmp[2*i*nim + 2*j];
                                imt *= (znorm/4)*(nim*nim*OVER*OVER)/512.0;
                                immr = sqrt(imr*imr + imt*imt);

                                imt = imtmp[2*i*nim + 2*j + 1];
                                imt *= (znorm/4)*(nim*nim*OVER*OVER)/512.0;
                                immi = sqrt(imi*imi + imt*imt);

                                bm[2*j] =
512*immr/(nim*nim*OVER*OVER)/(znorm/4);
                                bm[2*j+1] =
512*immi/(nim*nim*OVER*OVER)/(znorm/4);
                        }
                        else {
                                bm[2*j] = 0;
                                bm[2*j+1] = 0;
                        }
                }
                fwrite(bm,2*nim,2,fo1);
        }
        fclose(fo1);
        return SUCCESS;
}

int     write_imagemc_gc(int nc,int ie,int nph,int nraw,int npe) {

  int i, j, mx,offs,tmpslice;
  float imx, imi, imr, imm, imt, wctmp, wcmax;
  float wc[MXNIM];
  short imtmp[MXNIM*MXNIM];
  short int bm[MXNIM];
  char fn[80];
  short int tmpim[MXNIM][MXNIM];
  int znorm;

        if(noz_flg)     znorm=1;
        else            znorm=nzpe;

        if (sliceorder) tmpslice = ie;
        else            tmpslice = (ie <= (nslices+1)/2) ? (ie*2 - 1) : ((ie-
(nslices+1)/2)*2);

        if (rev_flg) tmpslice = nslices - tmpslice + 1;

  /* do for the first phase encode only */
        if(npe==1) {
                if (nph > 99)           sprintf(fn,"sl%1d.%d",tmpslice,nph);
                else if (nph >9)        sprintf(fn,"sl%1d.0%d",tmpslice,nph);
                else
sprintf(fn,"sl%1d.00%d",tmpslice,nph);
        }

        if (nc != 1) {
```

```
                if (!(fo2=fopen(fn,"r"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0); }
                if (fread(imtmp,sizeof(short),nim*nim,fo2) != nim*nim ) {
                        fprintf(stderr,"Error reading %s!\n",fn); return(0);
                }
                fclose(fo2);
        }
        else {
                for (i=0; i<nim*nim; i++)       imtmp[i] = 0;
                if (!(fo2=fopen(fn,"w"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0); }
        }

        fprintf(ff,"%s\n",fn); /*09/08/01 JQL*/

  /* kaiser-bessel correction */
        wcmax = sinh(sqrt(gridb*gridb))/sqrt(gridb*gridb);

        for (i=0; i<nim; i++) {
                wctmp = PI*PI*gridl*gridl*(i-nim/2)*(i-
nim/2)/(nim*nim*OVER*OVER/4) - gridb*gridb;
                if(wctmp == 0.)         wc[i] = 1.0*wcmax;
                else if(wctmp < 0.)      wc[i] = sqrt(-wctmp)/sinh(sqrt(-
wctmp))*wcmax;
                else                    wc[i] =
sqrt(wctmp)/sin(sqrt(wctmp))*wcmax;
        }

        imx = 0.0;
        offs = (OVER-1)*nim/2;
        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        if (npe==1)     tmpim[i][j] = 0;
                        if (hypot((float)(i-nim/2),(float)(j-nim/2)) <
.51*nim){
                                imt = imtmp[i*nim + j];
                                imt *= (nim*nim*OVER*OVER)/512.0;
                                imr = im[0][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];
                                imi = im[1][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];
                                imm = sqrt(imi*imi+imr*imr + imt*imt);
                                if (imm > imx) imx = imm;
                                tmpim[i][j] +=
512*imm/(nim*nim*OVER*OVER*znorm);
                        }
                        else    tmpim[i][j] += 0;
                }
        }

        mx = imx*512/(nim*nim*OVER*OVER*znorm);
        if (out_flg) printf("max=%3d",mx);

        if(npe==nzpe) {
                for (i=0; i<nim; i++) {
                        for (j=0; j<nim; j++) {
                                bm[j] = tmpim[i][j];
```

```
                        }
                        fwrite(bm,nim,2,fo2);
                }
                fclose(fo2);
        }
        return SUCCESS;
}


int     write_imagemc_3d(int nc,int ie,int nph,int nraw,int npe) {

  int i, j, k, mx,offs,tmpslice,imnum;
  float imx, imi, imr, imm, imt, wctmp, wcmax;
  float wc[MXNIM];
  short imtmp[MXNIM*MXNIM];
  short int bm[MXNIM];
  char fn[80];
  int znorm;
  FILE *fout;
  int tmp_ph;

        if(noz_flg)     znorm=1;
        else            znorm=nzpe;

        imnum = imoffset + nraw*nphases + (nph-1)*nzpe + npe;
        if (sliceorder) tmpslice = ie;
        else tmpslice = (ie <= (nslices+1)/2) ? (ie*2 - 1) : ((ie-
(nslices+1)/2)*2);

        if (rev_flg) tmpslice = nslices - tmpslice + 1;


        /*ZZH, 11/15/2002, to make the names of phase imgs the same as those
of the magnitude imgs*/
        tmp_ph=(map_flg || linmap_flg)?(nph+1):nph;
        /*ZZH, 03/04/03*/
        if (npe > 99)
sprintf(fn,"sl%d.pe%d.00%d",tmpslice,npe,tmp_ph);
        else if (npe >9)
sprintf(fn,"sl%d.pe0%d.00%d",tmpslice,npe,tmp_ph);
        else
sprintf(fn,"sl%d.pe00%d.00%d",tmpslice,npe,tmp_ph);
        /*ZZH, end modify*/


        if (nc != 1) {                                                  /* If
more than one coil, get images from other coils */
                if (!(fo1=fopen(fn,"r"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0); }
                if (fread(imtmp,sizeof(short),nim*nim,fo1) != nim*nim ) {
                        fprintf(stderr,"Error reading %s!\n",fn); return(0);
                }
                fclose(fo1);
        }
        else    for (i=0; i<nim*nim; i++) imtmp[i] = 0;         /* initialize
temp image storage to zero */
```

```
        if (!(fo1=fopen(fn,"w"))) {
                fprintf(stderr,"Can't open %s!\n",fn); return(0);
        }
                                                                /* kaiser-
bessel correction */
        wcmax = sin(sqrt(gridb*gridb))/sqrt(gridb*gridb);

        for (i=0; i<nim; i++) {
                wctmp = PI*PI*gridl*gridl*(i-nim/2)*(i-
nim/2)/(nim*nim*OVER*OVER/4) - gridb*gridb;
                if(wctmp == 0.)         wc[i] = 1.0*wcmax;
                else if (wctmp < 0.)    wc[i] = sqrt(-wctmp)/sin(sqrt(-
wctmp))*wcmax;
                else                    wc[i] =
sqrt(wctmp)/sin(sqrt(wctmp))*wcmax;     /* 1 over inverse transform of kaiser
bessel*/
        }

        imx = 0.0;
        offs = (OVER-1)*nim/2;
        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        if (npe==1) mip[i][j]=0;
                        /* HZ */
                /*      if (hypot((float)(i-nim/2),(float)(j-nim/2)) <
.51*nim){*/      /* grab only center of image space */
                        if (1){
                                imt = imtmp[i*nim + j];                 /*
get image pixel from other coils if necessary */
                                imt *=(znorm/4)*(nim*nim*OVER*OVER)/512.0;
/* unnormalze the pixel from other coils */

/*12/21/01 JQL  multiply with(znorm/4)*/
                                imr = im[0][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];  /* divide kernel in from real part */
                                imi = im[1][-j+nim+offs-1][-i+nim+offs-
1]*wc[i]*wc[j];  /* divide kernel from imag part */
                                imm = sqrt(imi*imi+imr*imr + imt*imt);
/* calculate magnitude of pixel intensity */
                                if (imm > imx) imx = imm;               /*
keep track of largest pixel intensity */
                                bm[j] = 4*512*imm/(nim*nim*OVER*OVER*znorm);
/* normalize pixel intensity */
                                if (mask_flg) bm[j] *=10;/* HZ */
                                if (bm[j]>=mip[i][j]) mip[i][j]=bm[j];
                        }
                        else    bm[j] = 0;
                }
                fwrite(bm,nim,2,fo1);                   /* write on row of
the image to image slice file */
        }
        mx = imx*512/(nim*nim*OVER*OVER*znorm);
        if (out_flg) printf("max=%3d",mx);              /* print out max
pixel intensity */
        fclose(fo1);
```

```
        if(nc==1){
                fprintf(ff,"%s\n",fn);
        }

        sprintf(fn,"correction_data");
        if(!(fout=fopen(fn,"w"))) {
          fprintf(stderr,"Can't open %s!\n",fn);
          return(0);
        }
        for(i=0; i < nzpe; i++) {
          for(j=0; j < npr; j++) {
            for(k=0; k < NCORRSAMP; k++) {

fprintf(fout,"%f,%f\n",correction_data[i][j][k][0],correction_data[i][j][k][1
]);
            }
          }
        }
        fclose(fout);

        if (npe==nzpe){
                if (istation*4+nsub+1>9)
sprintf(fn,"mip.phase%d",istation*4+nsub+1);
                        else sprintf(fn,"mip.phase0%d",istation*4+nsub+1);

                if (!(fo1=fopen(fn,"w"))) {
                        fprintf(stderr,"Can't open %s!\n",fn); return(0);
                }

                for (i=0; i<nim; i++) {
                        for (j=0; j<nim; j++)   bm[j]=mip[i][j];
                        fwrite(bm,nim,2,fo1);
                }
                 fclose(fo1);
        }
        if (zip_flg) nim = (int)(nim/2);  /* HZ */
        if (display_flg) printf ("Done\n");
        return SUCCESS;
}



int     write_image_phs(nc,ie,nph,nraw,npe)
int ie,nph,nraw,npe;
{
          int i,j,offs,tmpslice,imnum;
          float imi, imr;
          short int bm[MXNIM];
          char fn[80];
          int tmp_ph;

        imnum = imoffset + nraw*nphases + (nph-1)*nzpe + npe;

        if (sliceorder)         tmpslice = ie;
        else                    tmpslice = (ie <= (nslices+1)/2) ? (ie*2 - 1)
: ((ie-(nslices+1)/2)*2);
```

```
        if (rev_flg) tmpslice = nslices - tmpslice + 1;

        /*ZZH, 11/15/2002, to make the names of phase imgs the same as those
of the magnitude imgs*/
        tmp_ph=(map_flg || linmap_flg)?(nph+1):nph;
        if (npe > 99)
sprintf(fn,"sl%1d.pe%d.phs.00%d",tmpslice,npe,tmp_ph);
        else if (npe >9)
sprintf(fn,"sl%1d.pe0%d.phs.00%d",tmpslice,npe,tmp_ph);
        else
sprintf(fn,"sl%1d.pe00%d.phs.00%d",tmpslice,npe,tmp_ph);
        /*ZZH, end modify*/

        if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open %s!\n",fn);
return(0); }

        offs = (OVER-1)*nim/2;
        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        if (hypot((float)(i-nim/2),(float)(j-nim/2)) <
.51*nim){
                                imr = im[0][-j+nim+offs-1][-i+nim+offs-1];
                                imi = im[1][-j+nim+offs-1][-i+nim+offs-1];
                                bm[j] = 1000 * atan2(imi,imr);
                        }
                        else    bm[j] = 0;
                }
                fwrite(bm,nim,2,fo1);
        }
        fclose(fo1);
        return SUCCESS;
}

int     write_kspace(int step)

{
  int i, j;
  float ks[nim*OVER][nim*OVER][2];
  char fn[80];
  FILE *fout;
  int temp_offset;

        if(step){
                sprintf(fn,"data%d.co%d.pe%d",step,coilnum+1, zpenum+1);

                if (!(fout=fopen(fn,"w"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0); }

                printf("\nK-space %d",step);

                if(step >= 5) {
                        for (i=0; i<nim*OVER; i++) {
                                for (j=0; j<nim*OVER; j++) {
                                        ks[i][j][0] = im[0][i][j];
                                        ks[i][j][1] = im[1][i][j];
                                        if(i>254 && i<258 && j>254 && j<258)
```

```
printf("\n%f",hypotf(ks[i][j][0],ks[i][j][1]));
                                }
                        }
                }
                else if (step < 5) {
                        for (i=0; i<nim*OVER; i++) {
                                for (j=0; j<nim*OVER; j++) {
                                        ks[i][j][0] = grim[0][i][j];
                                        ks[i][j][1] = grim[1][i][j];
                                        if(i>254 && i<258 && j>254 && j<258)

printf("\n%f",hypotf(ks[i][j][0],ks[i][j][1]));
                                }
                        }
                }
                fwrite(ks,sizeof(float),2*nim*OVER*nim*OVER,fout);
                fclose(fout);
        }
        else {
                /*write raw data*/
                sprintf(fn, "zftdat.co%d.pe%d",coilnum+1, zpenum+1);
                if(zpenum==0){
                        if (!(fo1=fopen(fn,"w+")))
                                {fprintf(stderr,"Can't open file %s!\n",fn);
return(0);}
                }
                else if (!(fo1=fopen(fn,"a+b")))
                        {fprintf(stderr,"Can't open file %s!\n",fn);
return(0);}
                temp_offset=0;
                if(ts==8.E-6&&!(int)fast_rec_lpf)
                        temp_offset=OFFSET_PA_COIL2;
                if(ts==4.E-6&&!(int)fast_rec_lpf)
                        temp_offset=OFFSET_PA_COIL1;
                if(ts==2.5E-6)
                        temp_offset=OFFSET_PA_COIL;
                for (j=0; j<npr; j++){
                        for (i=0; i<ndat; i++){
                                fwrite(&prd[0][j][i], 4, 1, fo1);
                                fwrite(&prd[1][j][i], 4, 1, fo1);
                                if(ncoils>1){
                                 /*for cardiac coil, we have to set the last
few points to zeros to avoid the artifacts*/
                                        if(i>ndat-temp_offset/2){
                                                prd[0][j][i]=0;
                                                prd[1][j][i]=0;
                                        }
                                }
                        }
                }
        }

        return SUCCESS;
}
```

```
int     write_proj() {                  /* write raw spiral data, disabled at
present */

  int i;
  float temp[MXNDAT];
  char outfname[50];
  FILE *ofile;

        for(i=0;i<ndat;i++)     temp[i]=prd[1][0][i];

        sprintf(outfname,"prd.ph%d.sl%d.pe%d",phnum+1,slnum+1,zpenum+1);
        ofile=fopen(outfname,"w");
        fwrite(temp,ndat,sizeof(*temp),ofile);
        fclose(ofile);

        return SUCCESS;
}


int     write_raw()
{
  int i, j, mx;
  float imx, imi, imr, imm;
  short int bm[OVER*MXNIM];
  char fn[80];

        sprintf(fn,"raw.%d.%d.i",slnum+1,phnum+1);

        if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open %s!\n",fn);
return(0); }

        imx = 0.0;
        for (i=0; i<OVER*nim; i++) {
                for (j=0; j<OVER*nim; j++) {
                        imr = grim[0][j][i];
                        imi = grim[1][j][i];
                        imm = sqrt(imi*imi+imr*imr);
                        if (imm > imx) imx = imm;
                        bm[j] = imm;
                }
                fwrite(bm,OVER*nim,2,fo1);
        }
        mx = imx;
        if (out_flg) printf("max=%3d",mx);
        fclose(fo1);
        return SUCCESS;
}

double determinant(m11,m12,m13,m21,m22,m23,m31,m32,m33)
   double m11,m12,m13,m21,m22,m23,m31,m32,m33;
{
                double det;

        det = m11*(m22*m33-m23*m32)-m12*(m21*m33-m23*m31)+m13*(m21*m32-
m22*m31);
        return det;
}
```

```
int     write_ref(int sl, int ie)
{
  int i, j;
  float bm[MXNIM];
  char fn[80];
  float max;
  double tmpxr, tmpyr, tmpxi, tmpyi, tmpc, tmpcr, tmpci;
  float out[3];
  double x, y, S, Sx, Sy, Sxx, Sxy, Syy, Sf, Sxf, Syf, del, delx, dely, delf,
sigma2[MXNIM][MXNIM];

        if (linmap_flg) {
                for (i=0; i<nim; i++) {
                        for (j=0; j<nim; j++) {
                                if (refim[i][j][ie] > PI) refim[i][j][ie] -=
2*PI;              /* wrap the phase between -2Pi and 2Pi */
                                if (refim[i][j][ie] < -PI) refim[i][j][ie] +=
2*PI;
                        }
                }

                /*PDT wrote the following code to implement Meyers linear
phase map correction
                 according to paper in MRM 35:278-282(1996) instead of the
current method used below this code*/

                if (linear_flg){
                S=0;Sx=0;Sy=0;Sf=0;Sxx=0;Syy=0;Sxy=0;Sxf=0;Syf=0;

                for (i=0; i<nim; i++) {
                        x = (double)(i-nim/2);
                        for (j=0; j<nim; j++) {
                                y = (double)(j-nim/2);
                                sigma2[i][j] =
1.0/(refimmag[i][j][ie]*refimmag[i][j][ie]+0.0001);
                                S += 1.0/sigma2[i][j];
                                Sx += x/sigma2[i][j];
                                Sy += y/sigma2[i][j];
                                Sf += refim[i][j][ie]/sigma2[i][j];
                                Sxx += (x*x)/sigma2[i][j];
                                Syy += (y*y)/sigma2[i][j];
                                Sxy += (x*y)/sigma2[i][j];
                                Sxf += x*refim[i][j][ie]/sigma2[i][j];
                                Syf += y*refim[i][j][ie]/sigma2[i][j];
                        }
                }



                del = determinant(S,Sx,Sy,Sx,Sxx,Sxy,Sy,Sxy,Syy);
                delf = determinant(Sf,Sx,Sy,Sxf,Sxx,Sxy,Syf,Sxy,Syy);
                delx = determinant(S,Sf,Sy,Sx,Sxf,Sxy,Sy,Syf,Syy);
                dely = determinant(S,Sx,Sf,Sx,Sxx,Sxf,Sy,Sxy,Syf);


                tmpc = delf/del;
```

```
                tmpxr = delx/del;
                tmpyr = dely/del;
                /*ZZH,02/06*/
                if (tmpc > PI) tmpc -= 2*PI;            /* wrap the phase
between -2Pi and 2Pi */
                if (tmpc < -PI) tmpc += 2*PI;
                if ((tmpc > 0.5*PI)||(tmpc < -0.5*PI)){
                        tmpc = 0;               /* tan function is only
defined from -Pi/2 to Pi/2 */
                        tmpxr = 0;
                        tmpyr = 0;
                }
                }/*END PDT code*/
                else{
                max = 0.0;
                for (i=0; i<nim; i++) {
                        for (j=0; j<nim; j++) {
                                if (hypot((float)(i-nim/2),(float)(j-nim/2))
< nim/2) {
                                        if (refimmag[i][j][ie] > max) max =
refimmag[i][j][ie];
                                }
                        }
                }
                max /= 4.0;

                tmpxr = tmpyr = tmpxi = tmpyi = tmpcr = tmpci = 0.0;

                for (i=0; i<nim-1; i++) {
                        for (j=0; j<nim-1; j++) {
                                if (refimmag[i][j][ie] > max) {
                                        tmpxr +=
refimmag[i][j][ie]*refimmag[i][j+1][ie]
                                                *cos(refim[i][j][ie]-
refim[i][j+1][ie]);
                                        tmpxi +=
refimmag[i][j][ie]*refimmag[i][j+1][ie]
                                                *sin(refim[i][j][ie]-
refim[i][j+1][ie]);
                                        tmpyr +=
refimmag[i][j][ie]*refimmag[i+1][j][ie]
                                                *cos(refim[i][j][ie]-
refim[i+1][j][ie]);
                                        tmpyi +=
refimmag[i][j][ie]*refimmag[i+1][j][ie]
                                                *sin(refim[i][j][ie]-
refim[i+1][j][ie]);
                                }
                        }
                }

                tmpxr = -atan2(tmpxi,tmpxr);
                tmpyr = -atan2(tmpyi,tmpyr);
                for (i=0; i<nim-1; i++) {
                        for (j=0; j<nim-1; j++) {
                                if (refimmag[i][j][ie] > max) {
                                        tmpcr += refimmag[i][j][ie]
```

```
                                                *cos(refim[i][j][ie] -
tmpxr*(j-nim/2) - tmpyr*(i-nim/2));
                                        tmpci += refimmag[i][j][ie]
                                                *sin(refim[i][j][ie] -
tmpxr*(j-nim/2) - tmpyr*(i-nim/2));
                                }
                        }
                }

                tmpc = atan2(tmpci,tmpcr);
                }

                out[0] = tmpc*samptime/mapdel;                          /*
radians of phase per sampling period constant offset from zero*/
                out[1] = tmpxr*nim*samptime/mapdel/2.0/PI;              /*
cycles of phase per sampling period accumulated in x*/
                out[2] = tmpyr*nim*samptime/mapdel/2.0/PI;              /*
cycles of phase per sampling period accumulated in y*/
                if(out_flg)
                        printf("image = %d, linear_flg = %d, constant = %.7f,
x-grad = %.7f, y-grad = %.7f\n",ie,linear_flg, out[0],out[1],out[2]);

                /* write constant, x-slope, and y-slope to file */
                /*if (ie>99)    sprintf(fn,"refl.sl%1d.pe%d",sl,ie);
                else if(ie>9) sprintf(fn,"refl.sl%1d.pe0%d",sl,ie);
                else sprintf(fn,"refl.sl%1d.pe00%d",sl,ie);*/

                if (ie>99)
sprintf(fn,"refl.co%1d.sl%1d.pe%d",coilnum,sl,ie);
                else if(ie>9)
sprintf(fn,"refl.co%1d.sl%1d.pe0%d",coilnum,sl,ie);
                else sprintf(fn,"refl.co%1d.sl%1d.pe00%d",coilnum,sl,ie);
                if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0); }
                fwrite(out,3,sizeof(*out),fo1);
                fclose(fo1);

                for (i=0; i<nim-1; i++) {
                        for (j=0; j<nim-1; j++) {
                                tmpcr = refimmag[i][j][ie]
                                        *cos(refim[i][j][ie] - tmpxr*(j-
nim/2) - tmpyr*(i-nim/2) - tmpc);
                                tmpci = refimmag[i][j][ie]
                                        *sin(refim[i][j][ie] - tmpxr*(j-
nim/2) - tmpyr*(i-nim/2) - tmpc);
                                refim[i][j][ie]= atan2(tmpci,tmpcr);
                        }
                }
        }

        if (map_flg) {
                /*if (ie>99) sprintf(fn,"ref.%d.sl%1d.pe%d",nim,sl,ie);
                else if(ie>9) sprintf(fn,"ref.%d.sl%1d.pe0%d",nim,sl,ie);
                else sprintf(fn,"ref.%d.sl%1d.pe00%d",nim,sl,ie);*/

                if (ie>99)
sprintf(fn,"ref.%d.co%1d.sl%1d.pe%d",nim,coilnum,sl,ie);
```

```
                else if(ie>9)
sprintf(fn,"ref.%d.co%1d.sl%1d.pe0%d",nim,coilnum,sl,ie);
                else sprintf(fn,"ref.%d.co%1d.sl%1d.pe00%d",nim,coilnum,
sl,ie);
                if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open
%s!\n",fn); return(0); }
                for (i=0; i<nim; i++) {
                        for (j=0; j<nim; j++) {
                                if (refim[i][j][ie] > PI) refim[i][j][ie] -=
2*PI;              /* wrap the phase between -2Pi and 2Pi */
                                if (refim[i][j][ie] < -PI) refim[i][j][ie] +=
2*PI;
                                if (refim[i][j][ie] > 0.5*PI) refim[i][j][ie]
= 0.5*PI;         /* tan function is only defined from -Pi/2 to Pi/2 */
                                if (refim[i][j][ie] < -0.5*PI)
refim[i][j][ie] = -0.5*PI;
                                        bm[j] = refim[i][j][ie];
                        }
                        fwrite(bm,nim,sizeof(*bm),fo1);                 /*
write all the phase data to file */
                }
                fclose(fo1);
        }
        return SUCCESS;
}

int     write_ref_mc(sl,ie,nc)
int sl, ie,nc;
{
         int i, j;
  float bm[MXNIM];
  char fn[80];

        /*sprintf(fn,"ref.%d.%d.sl%1d.pe%1d",nim,nc,sl,ie);
        if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open %s!\n",fn);
return(0); }

        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        bm[j] = refim[i][j][ie];
                }
                fwrite(bm,nim,sizeof(*bm),fo1);
        }
        fclose(fo1);

        sprintf(fn,"refm.%d.%d.sl%1d.pe%1d",nim,nc,sl,ie);

        if (!(fo1=fopen(fn,"w"))) {fprintf(stderr,"Can't open %s!\n",fn);
return(0); }

        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        bm[j] = refimmag[i][j][ie];
                }
                fwrite(bm,nim,sizeof(*bm),fo1);
        }
        fclose(fo1);*/
```

```
        /*ZZH, 05/07/03*/
        write_ref(sl, ie);

        return SUCCESS;
}

int     write_ref_mcf(int sl, int ie) {

  int i, j, k;
  float bm[MXNIM];
  char fn[80];
  float maptmp[MXNIM][MXNIM];

        for (i=0; i<nim; i++) {
                for (j=0; j<nim; j++) {
                        refim[i][j][ie] = 0;
                        refimmag[i][j][ie] = 0;
                }
        }

        for (k=0; k<ncoils; k++) {
                sprintf(fn,"ref.%d.%d.sl%1d.pe%1d",nim,k,sl,ie);
                if (!(fo1=fopen(fn,"r"))) {fprintf(stderr,"Can't open
reference file %s!\n",fn); return(0); }
                for (i=0; i<nim; i++) {
                        fread(bm,nim,sizeof(*bm),fo1);
                        for (j=0; j<nim; j++) { maptmp[i][j] = bm[j];}
                }
                fclose(fo1);
                unlink(fn);
                sprintf(fn,"refm.%d.%d.sl%1d.pe%1d",nim,k,sl,ie);
                if (!(fo1=fopen(fn,"r"))) {fprintf(stderr,"Can't open
reference file %s!\n",fn); return(0); }
                for (i=0; i<nim; i++) {
                        fread(bm,nim,sizeof(*bm),fo1);
                        for (j=0; j<nim; j++) {
                                refim[i][j][ie] += maptmp[i][j]*bm[j];
                                refimmag[i][j][ie] += bm[j];
                        }
                }
                fclose(fo1);
                unlink(fn);
        }
        for (i=0; i<nim; i++){
                for (j=0; j<nim; j++){
                        refim[i][j][ie] /= (refimmag[i][j][ie] + 0.001);
                }
        }
        return SUCCESS;
}



int     zfilter_prd3d_hanning() {

  int i,j,k;
```

```
  float tempr,tempi,weight;


        for(k=0; k<nzpe; k++) {
                weight=.54+.46*sin((float)k*PI/((float)nzpe-1.0));
                for(i=0; i<npr; i++) {
                        for(j=0; j<ndat; j++){
                                tempr=prd3d[0][i][j][k];
                                tempi=prd3d[1][i][j][k];
                                prd3d[0][i][j][k]=weight*tempr;
                                prd3d[1][i][j][k]=weight*tempi;
                        }
                }
        }
        return SUCCESS;
}



int     zft_prd3d_discrete(int sign1, int sign2)                        /*
discrete FT in z direction */
{
  int i,j,k,l,p,offs,n;

  /*11/28/01 JQL*/
  float w4_temp0,w4_temp1;
  int chopper_kz;
/*HZ*/
  int s;


        n = nzpe;
        offs = nzpe/2-1;

        for(i=0; i<nzpe; i++) {
                for(j=0; j<nzpe; j++) {
                        m[0][i][j]=cos((2*PI)*(((float)(i*j))/n));
                        m[1][i][j]=sin((2*PI)*(((float)(i*j))/n));
                }
        }

        for(i=0; i<npr; i++) {
                for(j=0; j<ndat; j++) {
                        /*HZ*/
                        if (zftoutput==0){
                                        zftoutput=1;
                                        if (out_flg){
                                                printf("Original:\n");
                                                for (s=0;s<nzpe;s++)
                                                        printf("%.3f,%.3f
\n",prd3d[0][i][j][s],prd3d[1][i][j][s]);
                                        }

                                }

                /*      for(k=0; k<offs; k++) {
```

```
                                w4[0][k] = prd3d[0][i][j][k+offs];
                                w4[1][k] = prd3d[1][i][j][k+offs];
                                w4[0][k+offs] = prd3d[0][i][j][k];
                                w4[1][k+offs] = prd3d[1][i][j][k];
                        } */

                        for(k=0; k<nzpe-offs; k++) {
                                w4[0][k] = prd3d[0][i][j][k+offs];
                                w4[1][k] = prd3d[1][i][j][k+offs];
                                }
                        for (k=nzpe-offs; k<nzpe; k++) {
                                w4[0][k] = prd3d[0][i][j][k-nzpe+offs];
                                w4[1][k] = prd3d[1][i][j][k-nzpe+offs];
                        }
                        /*HZ*/

                        for(l=0; l<nzpe; l++) {
                                w3[0][l]=0;
                                w3[1][l]=0;
                                chopper_kz=1;/*11/28/01 JQL*/
                                for(p=0; p<nzpe; p++) {
          /*
          w3[0][l]+=m[0][l][p]*w4[0][p]+sign1*m[1][l][p]*w4[1][p];
          w3[1][l]+=m[0][l][p]*w4[1][p]+sign2*m[1][l][p]*w4[0][p];
         *//*11/28/01 JQL  These codes will be replaced by the following
codes*/

          /*11/28/01 JQL  New codes start here*/
                                        w4_temp0=chopper_kz*w4[0][p];
                                        w4_temp1=chopper_kz*w4[1][p];
                                        chopper_kz*=chop_kz;

w3[0][l]+=m[0][l][p]*w4_temp0+sign1*m[1][l][p]*w4_temp1;

w3[1][l]+=m[0][l][p]*w4_temp1+sign2*m[1][l][p]*w4_temp0;
          /*11/28/01 JQL  New codes end here*/

                                }
                        }
                        /*HZ*/
                        /*for(k=0; k<offs; k++) {
                                prd3d[0][i][j][k+offs] = w3[0][k];
                                prd3d[1][i][j][k+offs] = w3[1][k];
                                prd3d[0][i][j][k] = w3[0][k+offs];
                                prd3d[1][i][j][k] = w3[1][k+offs];
                        }*/
                        for(k=0; k<nzpe; k++) {
                                prd3d[0][i][j][k] = w3[0][(k+1)%nzpe];
                                prd3d[1][i][j][k] = w3[1][(k+1)%nzpe];
                        }
                /*HZ*/
                }
        }
        return SUCCESS;
}

int     zft_prd3d_discrete2(int sign1, int sign2)
```

```
{
  int i,j,k,l,p,n;

        n = nzpe;

        for(i=0; i<nzpe; i++) {
                for(j=0; j<nzpe; j++) {
                        m[0][i][j]=cos((2*PI)*(((float)(i*j))/n));
                        m[1][i][j]=sin((2*PI)*(((float)(i*j))/n));
                }
        }

        for(i=0; i<npr; i++) {
                for(j=0; j<ndat; j++) {
                        for(k=0; k<nzpe; k++) {
                                w4[0][k] = prd3d[0][i][j][k];
                                w4[1][k] = prd3d[1][i][j][k];
                        }
                        for(l=0; l<nzpe; l++) {
                                w3[0][l]=0;
                                w3[1][l]=0;
                                for(p=0; p<nzpe; p++) {

w3[0][l]+=m[0][l][p]*w4[0][p]+sign1*m[1][l][p]*w4[1][p];

w3[1][l]+=m[0][l][p]*w4[1][p]+sign2*m[1][l][p]*w4[0][p];
                                }
                        }
                        for(k=0; k<nzpe; k++) {
                                prd3d[0][i][j][k] = w3[0][k]/nzpe;
                                prd3d[1][i][j][k] = w3[1][k]/nzpe;
                        }
                }
        }
        return SUCCESS;
}

int baseline_corr() {
  int datapt,leaf,i;

  int navg = NCORRSAMP;

  float corr_fact[2];


   for(leaf = 0; leaf < npr; leaf++) {
      corr_fact[0] = 0.0;
      corr_fact[1] = 0.0;
      for(datapt = (ndat - navg); datapt < ndat; datapt++) {
        correction_data[zpenum][leaf][datapt-(ndat-navg)][0] =
prd[0][leaf][datapt];
        correction_data[zpenum][leaf][datapt-(ndat-navg)][1] =
prd[1][leaf][datapt];
        corr_fact[0] += prd[0][leaf][datapt];
        corr_fact[1] += prd[1][leaf][datapt];
      }
      corr_fact[0] /= navg;
```

```
      corr_fact[1] /= navg;
      printf("leaf %i: correction factor is %f +
%fi\n",leaf,corr_fact[0],corr_fact[1]);
      for(datapt = 0; datapt < ndat; datapt++) {
        prd[0][leaf][datapt] -= corr_fact[0];
        prd[1][leaf][datapt] -= corr_fact[1];
      }
    }
  return(1);
}


int conjugated(int segnum, int npe)
{

        int i, j;
        double alpha;

        alpha =  wmax[npe]-(segnum+1/2)*delta_fs;

        if (zip_flg) {

                for(i=0; i<nim*OVER; i++){
                        for (j=0; j<nim*OVER; j++){
                                im[0][i][j]=0.0;
                                im[1][i][j]=0.0;
                        }
                }

                for(i=0; i<nim; i++) {
                        for (j=0; j<nim; j++) {         /*multiply by
exp(iwt)*/

im[0][i+256][j+256]=grim[0][i][j]*cos(alpha*sampim[i][j]*samptime)-
grim[1][i][j]*sin(alpha*sampim[i][j]*samptime);

im[1][i+256][j+256]=grim[0][i][j]*sin(alpha*sampim[i][j]*samptime)+grim[1][i]
[j]*cos(alpha*sampim[i][j]*samptime);
                        }
                }

        } /* HZ zero-padding */

        else {
                for(i=0; i<nim*OVER; i++) {
                        for (j=0; j<nim*OVER; j++) {            /*multiply by
exp(iwt)*/

im[0][i][j]=grim[0][i][j]*cos(alpha*sampim[i][j]*samptime)-
grim[1][i][j]*sin(alpha*sampim[i][j]*samptime);

im[1][i][j]=grim[0][i][j]*sin(alpha*sampim[i][j]*samptime)+grim[1][i][j]*cos(
alpha*sampim[i][j]*samptime);
                        }
                }
        }
        return SUCCESS;
```

```
}

int make_final_fs(int segnum, int npe)
{
        int i, j, offs, iseg;
        double temp;
        static int t_flg;
        int factor_zip; /* HZ for zero-padding: phase maps are always
generated w.o. zip, so they need to be stretched to match zipped images*/

        offs = (OVER-1)*nim/2;
        if (zip_flg) factor_zip = 2;
                else factor_zip = 1;

        if (segnum==-1) {
                t_flg = 0;
                for (i =0; i<nim; i++) {
                        for (j=0; j<nim; j++) {
                                finalim[0][i][j]=im[0][i+offs][j+offs];
                                finalim[1][i][j]=im[1][i+offs][j+offs];
                        }
                }
        }
        else for (i = 0; i<nim; i++) {
                for (j = 0; j<nim; j++) {
                                temp =
refim[i/factor_zip][j/factor_zip][npe]/mapdel-
(wmin[npe]+(segnum+0.5)*delta_fs); /* HZ i,j divided by 2 for zero-padding*/
                                if ((temp>=-delta_fs/2)&&(temp<=delta_fs/2))
{
                                        t_flg++;

finalim[0][i][j]=im[0][i+offs][j+offs];

finalim[1][i][j]=im[1][i+offs][j+offs];
                        }
                }
        }
        if (segnum == nfs){
                printf("t_flg=%d\t", t_flg);
                for (i=0; i<nim; i++){
                        for (j=0; j<nim; j++){
                                im[0][i+offs][j+offs] = finalim[0][i][j];
                                im[1][i+offs][j+offs] = finalim[1][i][j];
                        }
                }
        }

        return SUCCESS;
}
```

Appendix C

INSTRUCTIONS AND MATLAB CODES FOR SPECTRUM/IMAGE PROCESSING

Each figure or sub-figure is processed and generated in matlab. These matlab codes consists of built-in functions of Matlab and self-developed functions stored in the 'matlab/work' directory. They are included in Appendix C. A list of all matlab instructions and functions is included. All functions are defined in Matlab M-files (.m) located in the "matlab/work"[5] folder.

Table 5-3 Appendix C Subsections and Figure Indices

Appendix	Figure	Contents
C1	2-3	L-COSY and DQF-COSY Comparison
C2		**readp**.m
C3	2-9	L-COSY 2D Spectrum
C4	2-10	Unoptimized Sel-MQC 2D Spectrum
C5	2-11	Six 2D Spectra for Optimizing Sel-MQC Selection
C6	2-12	Raw Data Plot of 2D Sel-MQC of PUFA Signal
C7	2-13	In Vivo 2D Spectra of Sel-MQC
C8	2-14	In Vivo 1D Spectra of Sel-MQC
C9	3-1	Sel-MQC Signal for Spiral Image Acquisition
C10	3-3, 3-4	Single Shot Spiral Image & Trajectory & Matlab functions **readim**(), **readtraj**()
C11	3-5	2-Shot Spiral Imaging Artifacts

[5] The paths specifying all files in the Appendix C are printed in Windows format with Backslashes, i.e. "\". To run these codes and functions in Matlab running on Linux or Unix, replace backslashes ("\") with slashes ("/"). The hard drive label "C:\" in windows needs to be replaced with the appropriate paths, such as "/home/users/".

C12	3-6	2-Shot Echo Mismatch
C13	3-8	COR & SAG Images of Spiral Sel-MQC Images in Phantum Study
C14	3-10	Sel-MQC Spiral Images of PUFA Distribution in Phantom Study
C15	3-11	In Vivo Images of Spiral Sel-MQC & Matlab function **twolayers**()
C16	3-13	Off-Resonance Simulation I & Matlab functions **gridspoff**() and **genspoff**()
C17	3-14	Off-Resonance Simulation II
C18	4-6, 4-7	Procedure and Comparison of Off-Resonance Correction in Spiral MRA
C19	4-8 to 4-10	Spiral MRA Results
C20	5-1	Constant Density Spiral vs. Variable Density Spiral & Matlab function **psffull**()
C21	5-2	Reference Image for Spiral Simulation & Matlab functions **face**.m, **junc**.m, **fermi**.m and **scaleim**.m
C22	5-3	Four Trejctory Examples of Spiral Simulation
C23	5-4, 5-5	Five *In Vivo* Echoes of PUFA Sel-MQC & Procedures to Estimate SNR and T_2* Decay Constant From *In Vivo* Data
C24	5-6	Four Examples of Simulated Spiral Images & Matlab functions **spiralmain**(), **gridspiral**(), **imerr**() and **genspdata**().
C25	5-8	Nine Plots of Estimation Errors of Spiral Simulations

C.1 COSY COMPARISON (FIGURE 2-3)

Figure 2-3 contains 1D spectra and a 2D spectrum showing the advantage of DQF-COSY over L-COSY.

(a) “aug29/P09216.7” is L-COSY data on phantom.

(b) “sep19/P43520.7” is DQF-COSY data on phantom.

(c) “oct6/P04096.7” is DQF-COSY data on fresh pork.

L-COSY was not performed on pork.

```
%fig2-3a:
fid=readp('c:\vdrecon\aug29\P09216.7',2048,1,64);
s=fftshift(fft2(fftshift(fid)));
plot(abs(sum(s(:,901:1300))));

%fig2-3b:
fid=readp('c:\vdrecon\sep19\P43520.7',2048,1,64);
s=fftshift(fft2(fftshift(fid)));
plot(abs(sum(s(:,901:1300))));

%fig2-3c:
fid=readp('c:\vdrecon\oct6\P04096.7',2048,1,64);
s=fftshift(fft2(fftshift(fid)));
contour(abs(s(1:50,901:1300)),30);
```

C.2 READP.M

The readp() function called above is defined in readp.m.

```
function a=readp(f,leafln, leafnum, nzpe);
% a=readp(f,leafln,leafnum,nzpe);
% f: file path i.e. 'P01024.7'
% leafln: leaf length
% leafnum: # of leaves
% nzpe: # of phase encodings
fi=fopen(f,'r','b');
fseek(fi,60464+leafln*4,'bof');
%fseek(fi,60464,'bof');
a=complex(zeros(nzpe,leafln, leafnum));
temp=fread(fi,[2,inf],'int16');
fclose(fi);
for ii=1:nzpe
    head=1+(ii-1)*leafln*leafnum;
    tail=head+leafln*leafnum-1;
    a(ii,:,:)=reshape(temp(1,head:tail)+i*temp(2,head:tail),leafln,leafnum);
end
```

C.3 L-COSY 2D SPECTRUM (FIG. 2-9)

“jan31/P47616.7” is the 2D data acquired by L-COSY on soybean oil. L-cosy 2D spectrum "s" is folded after fft2. So, “s” to “s2” unfolds the spectrum to show correct 2D peak pattern.

```
fid=readp('c:\vdrecon\jan31\P47616.7',2048,1,128);
s=fftshift(fft2(fftshift(fid)));
s2(1:78,:)=s(51:128,:);
s2(79:128,:)=s(1:50,:);
contour(abs(s2(:,701:1100))',60);
```

C.4 UNOPTIMIZED SEL-MQC 2D SPECTRUM (FIG. 2-10)

“feb15/P15872.7” is the 2D data acquired by unoptimized Sel-MQC sequence.

```
%Sel-MQC 2D spectrum without selection enhancement:
%strong leakage peaks are shown.

fid=readp('c:\vdrecon\feb15\P15872.7',2048,1,64);
s=fftshift(fft2(fftshift(fid)));
contour(abs(s(:,1100:-1:701))',20);
```

C.5 OPTIMIZED SEL-MQC 2D SPECTRUM (FIG. 2-11)

The following 6 files in the “jun25” folder are 2D phantom data for optimizing 2D Sel-MQC sequence. Subplot (f) is the setting for PUFA selection. It is also the setting for the raw data plot in Fig2-12.

```
%fig2.11(a)

fida=readp('c:\vdrecon\jun25\P09728.7',2048,1,64);
sa=fftshift(fft2(fftshift(fida)));
contour(abs(sa(:,701:1100))',20);

%fig2.11(b)
```

```
fidb=readp('c:\vdrecon\jun25\P10752.7',2048,1,64);
sb=fftshift(fft2(fftshift(fidb)));
contour(abs(sb(:,701:1100))',20);

%fig2.11(c)

fidc=readp('c:\vdrecon\jun25\P05632.7',2048,1,64);
sc=fftshift(fft2(fftshift(fidc)));
contour(abs(sc(:,701:1100))',20);

%fig2.11(d)
fidd=readp('c:\vdrecon\jun25\P07680.7',2048,1,64);
sd=fftshift(fft2(fftshift(fidd)));
contour(abs(sd(:,701:1100))',20);

%fig2.11(e)
fide=readp('c:\vdrecon\jun25\P06656.7',2048,1,64);
se=fftshift(fft2(fftshift(fide)));
contour(abs(se(:,701:1100))',20);

%fig2.11(f)
fidf=readp('c:\vdrecon\jun25\P02560.7',2048,1,64);
sf=fftshift(fft2(fftshift(fidf)));
contour(abs(s(:,701:1100))',20);

subplot(3,2,1);
contour(abs(sa(:,1100:-1:701))',20);
subplot(3,2,2);
contour(abs(sd(:,1100:-1:701))',20);
subplot(3,2,3);
contour(abs(sb(:,1100:-1:701))',20);
subplot(3,2,4);
contour(abs(se(:,1100:-1:701))',20);
subplot(3,2,5);
contour(abs(sc(:,1100:-1:701))',20);
subplot(3,2,6);
contour(abs(sf(:,1100:-1:701))',20);
```

C.6 RAW DATA PLOT OF PUFA SEL-MQC SPECTRUM (FIG. 2-12)

"jun25/P02560.7" is the raw data of optimized 2D Sel-MQC sequence. It is plotted in 3D here without Fourier Transformation. The 3D data plot needs to be manually rotated for the a better view.

```
fid=readp('c:\vdrecon\jun25\P02560.7',2048,1,64);
surf(abs(fid(:,1:500)));
```

C.7 IN VIVO 2D SPECTRA OF PUFA SEL-MQC (FIG. 2-13)

These files are *in vivo* 2D Sel-MQC data from 2 subjects on Mar. 9 and 7 of 2006. A Hann's window was used to process the Sel-MQC echo in 2D spectroscopy. The position of the echo was moved accordingly with the "for" loop for each time incrementation. All 2D spectroscopic experiments were performed with 180° pulse at 6.3ms so the suppression of the peak at 1.3ppm was not optimal. In the following 1D and Spiral Sel-MQC experiments where 2D incrementation is unnecessary, the 180° pulse width was 11.7ms.

```
%left column
fid=readp('c:\vdrecon\mar9\P10240.7',2048,1,64);
fil=zeros(64,2448);
for i=1:64
fil(i,int16(401-2.5*i):int16(800-2.5*i))=hann(400);
end;
imagesc(abs(fil));
fil2=fil(:,401:2448);
imagesc(abs(fil2));
nfid=fid.*fil2;
s=fftshift(fft2(fftshift(nfid)));
subplot(3,1,1);
plot(abs(sum(s(1:40,810:840)')));
subplot(3,1,2);
contour(abs(s(1:40,701:1100)'));
subplot(3,1,3);
fid=readp('c:\vdrecon\mar9\P10752.7',2048,1,64);
nfid=fid.*fil2;
s2=fftshift(fft2(fftshift(nfid)));
contour(abs(s2(1:40,701:1100)'));

%right column
fid=readp('c:\vdrecon\mar7\P11264.7',2048,1,64);
nfid=fid.*fil2;
s=fftshift(fft2(fftshift(nfid)));
subplot(3,1,1);
plot(abs(sum(s(1:40,810:840)')));
subplot(3,1,2);
contour(abs(s(1:40,701:1100))');
subplot(3,1,3);
fid=readp('c:\vdrecon\mar7\P11776.7',2048,1,64);
nfid=fid.*fil2;
s2=fftshift(fft2(fftshift(nfid)));
contour(abs(s2(1:40,701:1100))',2);
```

C.8 1D IN VIVO SEL-MQC SPECTRUM (FIG.2.14)

These two files are 1D Sel-MQC spectroscopic data of *in vivo* exams on the same subject studies shown in Fig.2-13. They were conducted on Feb. 22 and Mar. 9 of 2006. They are plotted as five PUFA echoes. Because the 180° pulse was 11.7 with a narrower bandwidth, the peak at 1.3ppm was suppressed to background level in (a). A Hann window function was used for all cases to cover the echo signals. Because the *in vivo* echoes were short, the resulting spectral peaks were broad (as broad as 1ppm).

```
l(1:2048)=0;
l(301:1300)=hann(1000);
fid1=readp('c:\vdrecon\mar9\P08192.7',2048,1,1);
s1=fftshift(fft(fid1.*l));
subplot(1,2,1);
plot(abs(s1(976:1030)));
fid2=readp('c:\vdrecon\mar7\P08704.7',2048,1,1);
s2=fftshift(fft(fid2.*l));
subplot(1,2,2);
plot(abs(s2(976:1030)));
```

C.9 COHERENCE ECHO AND 1D SPECTRUM OF PUFA SEL-MQC (FIG.3-1)

"mar31/P44032.7" is the data of 1D Sel-MQC sequence on an oil phantom with 180 degree pulse width at 11.7ms. In this phantom study, shimming and PUFA selection were optimal. The echo was long and clean. No window function was used. That resulted the sharp 1D spectral peak.

```
fid=readp('c:\vdrecon\mar31\P44032.7',2048,1,1);
s=fftshift(fft(fid));
subplot(3,1,1);
plot(real(fid));
subplot(3,1,2);
plot(abs(s(951:1050)));
subplot(3,1,3);
plot(abs(fid));
```

C.10 SINGLE SHOT SPIRAL IMAGING (FIG.3-3, 3-4)

The single shot Spiral Sel-MQC image was constructed with “gsp3dspec2” program on smash.

```
% first run c recon program on smash
% gsp3dspec2 -v jul4/P12800.7
% the recon program outputs "sl1.001" for image and "kxout01",.. for traj

im=readim('c:\matlab\fig3.3\sl1.001',256);
xout=readtraj('c:\matlab\fig3.3\kxout01');
yout=readtraj('c:\matlab\fig3.3\kyout01');
subplot('position',[0 0 0.5 1]);
imagesc(abs(im));colormap(gray);
subplot('position',[0.5 0 0.5 1]);
fulltraj(xout,yout,1);



%fid for fig3-4
fid=readp('c:\vdrecon\jul4\P02048.7',2048,1,1);
plot(abs(fid(1,:)));
```

The readim() function called in the above codes is defined in readim.m

```
function a=readim(f,res);
%     a=readim(f); f is the file path; a is the output array(256*256);
fi=fopen(f,'r','b');
fseek(fi,0,'bof');
a=fread(fi,[res,res],'int16')';
fclose(fi);
```

Functions readtraj() is defined in readtraj.m

```
function a=readtraj(f);
fi=fopen(f,'r','b');
fseek(fi,0,'bof');
a=fread(fi,'float32');
fclose(fi);
```

C.11 TWO-SHOT SPIRAL IMAGING ARTIFACTS (FIG.3-5)

Two shot Spiral Sel-MQC image was constructed with “gsp3dspec2” program on smash.

```
% first run c recon program on smash
```

```
% gsp3dspec2 -v oct11/P10240.7
% the recon program outputs "sl1.001" for image and "kxout01",.. for traj

im=readim('c:\matlab\fig3.5\sl1.001',256);
xout=readtraj('c:\matlab\fig3.5\kxout02');
yout=readtraj('c:\matlab\fig3.5\kyout02');
subplot('position',[0 0 0.5 1]);
imagesc(abs(im));colormap(gray);
subplot('position',[0.5 0 0.5 1]);
fulltraj(xout,yout,2);
```

C.12 ECHO MISMATCH (FIG. 3-6)

These files are Spiral Sel-MQC data in "oct11"

```
% echo comparison for two shot spiral image acquisition
fid=readp('c:\vdrecon\oct11\P04096.7',2048,1,2);
fid2=readp('c:\vdrecon\oct11\P11264.7',2048,1,2);
subplot('position',[0 0.5 1 0.5]);
plot(abs(fid(1,:)));
hold on;
plot(abs(fid(2,:)));
subplot('position',[0 0 1 0.5]);
plot(abs(fid2(1,:)));
hold on;
plot(abs(fid2(2,:)));
```

C.13 CORONAL AND SAGITTAL TWO SHOT SPIRAL IMAGES ON PHANTOM

The coronal image in Fig3-8 (a) is reconstructed from the pfile "oct11/P13824.7" and the axial image in Fig.3-8(b) is from "oct11/P16384.7". The reconstructed image files are in "Fig3.8/a" and "fig3.8/b".

```
%run these commands on smash and get respective output files 'sl1.001'
%(a)
% gsp3dspec2 -v oct11/P13824.7
%(b)
% gsp3dspec2 -v oct11/P16384.7
%(a)
im=readim('c:\matlab\fig3.8\a\sl1.001',256);
imagesc(abs(im));
%(b)
im=readim('c:\matlab\fig3.8\b\sl1.001',256);
imagesc(abs(im));
```

C.14 PHANTOM IMAGES OF PUFA DISTRIBUTION (FIG.3-10)

The Spiral Sel-MQC image is “apr3/P49604.7”. It is reconstructed with “gsp3dspec2” on smash. The output file is “sl1.001”. Fast GRE file is “apr3/P50688.7”. Fast Spin Echo file is “apr3/P53248.7”. Both files are reconstructed in matlab. The Fast GRE file needed chopping for correct display.

```
subplot('position',[0 0 0.333 1]);
im=readim('c:\matlab\fig3.10\sl1.001',256);
imagesc(abs(im));colormap(gray);
subplot('position',[0.333 0 0.333 1]);
fid=readp('c:\vdrecon\apr3\P50688.7',256,1,256);
im=fftshift(fft2(fftshift(fid)));
im2=zeros(256);
im2(1:128,:)=im(129:256,:);
im2(129:256,:)=im(1:128,:);
im2=im2(256:-1:1,:);
imagesc(abs(im2'));colormap(gray);
subplot('position',[0.667 0 0.333 1]);
fid=readp('c:\vdrecon\apr3\P53248.7',256,1,256);
im=fftshift(fft2(fftshift(fid)));
im2=im(256:-1:1,:);
imagesc(abs(im2'));colormap(gray);
```

C.15 IN VIVO IMAGES OF SEL-MQC (FIG. 3-11)

These files are in vivo studies conducted on “dec29”, “mar2”, “mar7”, “mar9” and “mar10”. The subject on Dec. 29, 2005 was the first human study. It was conducted before the 2D Sel-MQC sequence was developed. Two slices of the mar9 study were included as the best PUFA selection *in vivo*.

All Spiral Sel-MQC image are reconstructed with “gsp3dspec2” program in “vdrecon/”. The program “gsp3dspec2” was compiled on smash and runs on smash. It outputs a binary file “sl1.001” as the low resolution Spiral Sel-MQC image. The high resolution images are

reconstructed in matlab directly. A matlab function “twolayers.m” (included below) was written to overlay low resolution Spiral Sel-MQC images on high resolution Fast GRE images. The code of “twolayers.m” is included in this subsection.

(a)

```
fid=readp('c:\vdrecon\mar9\P01536.7',256,1,256);
im=fftshift(fft2(fftshift(fid)));
im2(1:128,:)=im(129:256,:);
im2(129:256,:)=im(1:128,:);
im2=im2(256:-1:1,256:-1:1);
im2=im2';

%run 'gsp3dspec2 -v mar9/P04096.7' on smash
% get image file sl1.001
im=readim('c:\matlab\fig3.11\a\sl1.001',256);
twolayers(im,im2,0.8);
```

(b)

```
fid=readp('c:\vdrecon\mar9\P06656.7',256,1,256);
im=fftshift(fft2(fftshift(fid)));
im2(1:128,:)=im(129:256,:);
im2(129:256,:)=im(1:128,:);
im2=im2(256:-1:1,256:-1:1);
im2=im2';
im=readim('c:\matlab\fig3.11\b\sl1.001',256);
twolayers(im,im2,0.9);
```

(c)

```
fid=readp('c:\vdrecon\mar7\P14848.7',256,1,256);
im=fftshift(fft2(fftshift(fid)));
im2(1:128,:)=im(129:256,:);
im2(129:256,:)=im(1:128,:);
im2=im2(256:-1:1,256:-1:1);
im2=im2';
%run 'gsp3dspec2 -v mar7/P14336.7' on smash
% get image file sl1.001
im=readim('c:\matlab\fig3.11\c\sl1.001',256);
twolayers(im,im2,0.8);
```

(d)

```
%run 'gsp3dspec2 -v mar10/P01536.7' on smash
% get image file sl1.001

fid=readp('c:\vdrecon\mar10\P01536.7',256,1,256);
im=fftshift(fft2(fftshift(fid)));
im2(1:128,:)=im(129:256,:);
im2(129:256,:)=im(1:128,:);
im2=im2(256:-1:1,256:-1:1);
im2=im2';
%run 'gsp3dspec2 -v mar10/P04096.7' on smash
% get image file sl1.001
```

```
im=readim('c:\matlab\fig3.11\d\sl1.001',256);
twolayers(im,im2,0.8);
```

(e)

```
fid=readp('c:\vdrecon\mar2\P10752.7',256,1,256);
im=fftshift(fft2(fftshift(fid)));
im2(1:128,:)=im(129:256,:);
im2(129:256,:)=im(1:128,:);
im2=im2(256:-1:1,256:-1:1);
im2=im2';
%run 'gsp3dspec2 -v mar2/P09728.7' on smash
% get image file sl1.001
im=readim('c:\matlab\fig3.11\e\sl1.001',256);
twolayers(im,im2,0.9);
```

(f)

```
fid=readp('c:\vdrecon\dec29\P24064.7',256,1,160);
fid2=zeros(256);
fid2(49:208,:)=fid;
im=fftshift(fft2(fftshift(fid2)));
im2=im(:,256:-1:1)';

%run 'gsp3dspec2 -v dec29/P23552.7' on smash
% get image file sl1.001
im=readim('c:\matlab\fig3.11\f\sl1.001',256);
im=im(:,256:-1:1);
twolayers(im,im2,0.7);
```

The twolayers() function is defined in twolayers.m. “fore” is a 2D array containing the foreground image. “back” is a 2D array containing the background image. “alpha” is the ratio between 0 and 1 to adjust their relative intensity.

```
function over=twolayers(fore,back,alpha);
% function over=twolayers(fore,back);
% fore: file of front layer i.e. lo-res, spiral
% back: file of background layer i.e. hi-res, gradient echo
fore=abs(fore)/max(max(abs(fore)));
back=abs(back)/max(max(abs(back)));
over=zeros(256,256,3);
over(:,:,1)=fore+back*alpha;
over(:,:,2)=back*alpha;
over(:,:,3)=back*alpha;
figure;
subplot('Position',[0 0 1 1]);
imagesc(abs(over/(1+alpha)));
```

C.16 OFF-RESONANCE SIMULATION I (FIG. 3-13)

The matlab program "gridspoff.m" takes k-space data on spiral trajectories and grids it to a 2D Cartesian array. The files "kxout02" and "kyout02" contain the coordinates of the data points on the trajectories. "densout02" contains the density weighting of each coordinate. The k-space data on each coordinate is generated by a matlab program "genspoff.m". The value of the off-resonance dot is "dw". It is equall to 0, 530 and 1060 for subplots a, b and c. The "flag" input determines if the on-resonance patter is included or not. The on-resonance pattern is a big disc with three small dots. They are not shown in Fig3-13 but are shown in fig3-14. "gridspoff.m" also uses FOV and # of leaves as parameters.

The separated Matlab codes to generate Fig.3-13 a, b and c are included below.

```
% gridspoff.m calls genspoff.m
% gridspoff.m
% pathkx: path of kx trajectory
% pathky: path of ky trajectory
% pathdens: path of sampling density for weighting
% opfov: field of view ( = 16cm)
% nl: number of leaves (= 2 for this simulation)
% nzpe: number of slices/phase encodings (=1)
% dw: off-resonance value of the signal source in the 2nd quadrant
% flag: 1 to show 3 dots in the 4th quadrant; 0 not to show them

%genspoff.m
% pathkx: path of kx trajectory
% pathky: path of ky trajectory
% opfov: field of view ( usually 16)
% nl: number of leaves
% dw: off-resonance value of the signal source in the 2nd quadrant
% flag: 1 to show 3 dots in the 4th quadrant; 0 not to show them

pathkx='c:\matlab\work\trajs\lo\kxout02';
pathky='c:\matlab\work\trajs\lo\kyout02';
pathdens='c:\matlab\work\trajs\lo\densout02';

%(a)
[kim,kl]=gridspoff(pathkx,pathky,pathdens,16,2,1,0,0);
im=fftshift(fft2(fftshift(kim)));
imagesc(abs(im(129:384,129:384)));
colormap(gray);

%(b)
[kim,kl]=gridspoff(pathkx,pathky,pathdens,16,2,1,530,0);
im=fftshift(fft2(fftshift(kim)));
imagesc(abs(im(129:384,129:384)));
colormap(gray);
```

```
%(c)
[kim,kl]=gridspoff(pathkx,pathky,pathdens,16,2,1,1030,0);
im=fftshift(fft2(fftshift(kim)));
imagesc(abs(im(129:384,129:384)));
colormap(gray);
```

The function genspoff() is defined in “genspoff.m”.

```
function raw=genspoff(kx,ky,nl,fov,dw,flag);
% pathkx: path of kx trajectory
% pathky: path of ky trajectory
% opfov: field of view ( usually 16)
% nl: number of leaves
% dw: off-resonance value of the signal source in the 2nd quadrant
% flag: 1 to show 3 dots in the 4th quadrant; 0 not to show them
imag=sqrt(-1);
l=length(kx);
raw=zeros(l,nl);
tsp=4;
t=[0:tsp:tsp*(l-1)]';
off=dw*0.000002*pi*t;
for i=0:nl-1
    c=cos(2*pi*i/nl);
    s=sin(2*pi*i/nl);
    x=(kx*c-ky*s)/fov;
    y=(kx*s+ky*c)/fov;
    x(2)=eps;x(1)=eps;y(2)=eps;y(1)=eps;
%     raw(:,i+1)=jinc(3.6,4.4,0,0,x,y);
%     raw(:,i+1)=(jinc(3.5,4.5,0,0,x,y)-
0.7*jinc(1.5,1.5,1.75,0,x,y)+0.7*jinc(1.5,1.5,-1.75,0,x,y));
    f=jinc(4,4,0,0,x,y);
%    f=sinc(8*x).*sinc(8*y)*16-sinc(4*x).*sinc(4*y)*4;
%     f=f+jinc(1.5,1.5,1.75,0,x,y)+jinc(1.0,1.0,0,2.25,x,y);
%     f=f+jinc(0.5,0.5,-1.75,0,x,y)+jinc(0.5,0.5,-1.75,1.25,x,y);
%     f=f+jinc(0.5,0.5,-1.75,-1.25,x,y);
%     f=f+jinc(0.25,0.25,0,-2.5,x,y)+jinc(0.25,0.25,1,-2.5,x,y);
%     f=f+jinc(0.25,0.25,-1,-2.5,x,y)+jinc(0.25,0.25,-2,-2.5,x,y);
%     f=f+jinc(0.25,0.25,2,-2.5,x,y);
%     f=f+jinc(0.1,0.1,-0.5,0,x,y)+jinc(0.1,0.1,-1,0.5,x,y);
%     f=f+jinc(0.1,0.1,-0.75,0.25,x,y)+jinc(0.1,0.1,-0.25,-0.25,x,y);
%     f=f+jinc(0.1,0.1,0,-0.5,x,y)+jinc(0.1,0.1,0.25,-0.75,x,y);
for ii=1:7
    for j=1:7
        r=abs(ii-4)+abs(j-4);
%         if r<5 && (ii<=4 || j<=4)
          if r<4 && ii<4 && j<4
           f=f-0.4*jinc(0.25,0.25,(ii-4),(j-4),x,y);
        end
    end
end
foff=0.2*jinc(1,1,1,1,x,y).*exp(-imag*off);
if flag==1
    raw(:,i+1)=f;
```

```
end
%    relaxT2=exp(-t/T2);
%    fil=fermi1(l,l*0.9,l*0.05);
    f=foff;
%    temp=f.*relaxT2;
    opnex=64/nl;
%    raw(:,i+1)=awgn(temp,snrunit*sqrt(opnex));
    raw(:,i+1)=raw(:,i+1)+foff;
end
```

The function gridspoff() is defined in "gridspoff.m"

```
function
[kspacem,klength]=gridspoff(pathkx,pathky,pathdens,opfov,nl,nzpe,dw,flag);
% pathkx: path of kx trajectory
% pathky: path of ky trajectory
% pathdens: path of sampling density for weighting
% opfov: field of view ( usually 16)
% nl: number of leaves
% nzpe: number of slices/phase encodings (usually 1)
% dw: off-resonance value of the signal source in the 2nd quadrant
% flag: 1 to show 3 dots in the 4th quadrant; 0 not to show them
nim=512+8;
dwin=1.5;
kspace=zeros(nim,nim);
kspacem=zeros(512,512,nzpe);
ws=zeros(nim,nim);
xout=readtraj(pathkx);
yout=readtraj(pathky);
klength=length(xout);
kx1=xout(1:klength)*2;
ky1=yout(1:klength)*2;
klength=length(kx1);
kx=zeros(nl,klength);
ky=zeros(nl,klength);
for i=0:nl-1
    c=cos(2*pi*i/nl);
    s=sin(2*pi*i/nl);
    tx=kx1*c-ky1*s;
    ty=kx1*s+ky1*c;
    kx(i+1,:)=tx(:)';
    ky(i+1,:)=ty(:)';
end
kdens=zeros(klength);
kdens(1)=eps;
% for i=2:klength
%     kxb=kx(1,i);
%     kyb=ky(1,i);
%     kxa=kx(1,i-1);
%     kya=ky(1,i-1);
%     ksr=kxb*kxa+kyb*kya;
%     ksi=kyb*kxa-kxb*kya;
%     ggx=kxb-kxa;
%     ggy=kyb-kya;
%     ggm=hypot(ggx,ggy);
```

```
%     if ((ggm > 0) && (hypot(ksr,ksi) > 0))
%         kdens(i) = ggm *abs(sin(atan2(ggy,ggx) - atan2(kya,kxa)) *
(hypot(kxb,kyb)- hypot(kxa,kya))/atan2(ksi,ksr));
%         else  kdens(i) = eps;
%     end
% end
kdens=readtraj(pathdens);
k1=kaiser(1024,0.5);
%k1=readtraj('/home/henry/vdspiral/vdrecon/kaisout');
kker=k1*k1';
%read real data from a pfile for 1.5 st spiral
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%fid2=readp(pfile,klength,nl,nzpe);
% refoc=[1:klength]'*pi*0.5;
% fid2=fft(fid2,32);
% fid=squeeze(fid2(16,:,:));
% for leaf=1:nl
%     fid(:,leaf)=squeeze(fid(:,leaf))./(cos(refoc)+i*sin(refoc));
% end
% if ( nl==1)
%     fid=fid';
% end

%read data from 3T Sel-MQC spiral or do simulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if (data=='r')
%     fid=readp(pfile,2048,nl,nzpe);
%     fid=fft(fid);
% %    fid=squeeze(fid2(1,:,:));
%     if ( nl==1)
%         fid=fid';
%     end
% end

    nzpe=1;
    t=genspoff(xout,yout,nl,opfov,dw,flag);
    fid=zeros(1,klength,nl);
    fid(1,:,:)=t;


step=floor(512/dwin);
for zpe=1:nzpe
    zpe
for leaf=1:nl
%     leaf
    for i=1:klength
        ix=floor(kx(leaf,i)-dwin)+1;
        iy=floor(ky(leaf,i)-dwin)+1;
        dkx=floor(step*(ix+dwin-kx(leaf,i)));
        dky=floor(step*(iy+dwin-ky(leaf,i)));
        w=kker(dkx+1:step:dkx+1+step*2,dky+1:step:dky+1+step*2);
        t=w*fid(zpe,i,leaf)*kdens(i);

ws(ix+nim/2:ix+2+nim/2,iy+nim/2:iy+2+nim/2)=ws(ix+nim/2:ix+2+nim/2,iy+nim/2:i
y+2+nim/2)+w;
```

```
kspace(ix+nim/2:ix+2+nim/2,iy+nim/2:iy+2+nim/2)=kspace(ix+nim/2:ix+2+nim/2,iy
+nim/2:iy+2+nim/2)+t;
    end
end
kspacet=kspace(5:516,5:516);
fermirad=24;
kspacet=kspacet.*fermi(512,fermirad,4);
%kspacem(:,:,zpe)=fftshift(fft2(fftshift(kspacet)));
%imagesc(abs(kspacem(:,:,zpe)));
kspacem(:,:,zpe)=kspacet;
end
%imagesc(abs(ws(200:300,200:300)));
```

C.17 OFF-RESONANCE SIMULATION II (FIG.3-14)

Fig.3-14 performs a similar simulation to that in Fig.3-13. The difference is that the "flag" is set to 1 so the on-resonance pattern is shown with the off-resonance pattern. When the off-resonance pattern is off by 530Hz, the simulation results in c & d. When the off-resonance pattern is off by 0 Hz (on-resonance), the simulation results in a & b. The functions called are gridspoff() and genspoff() same as those used for Fig. 3-13. The following are the codes to call those programs and perform the 1D cross section for a & b.

```
%gridspoff.m calls genspoff.m
%gridspoff.m
% pathkx: path of kx trajectory
% pathky: path of ky trajectory
% pathdens: path of sampling density for weighting
% opfov: field of view ( usually 16)
% nl: number of leaves
% nzpe: number of slices/phase encodings (usually 1)
% dw: off-resonance value of the signal source in the 2nd quadrant
% flag: 1 to show 3 dots in the 4th quadrant; 0 not to show them

%genspoff.m
% pathkx: path of kx trajectory
% pathky: path of ky trajectory
% opfov: field of view ( usually 16)
% nl: number of leaves
% dw: off-resonance value of the signal source in the 2nd quadrant
% flag: 1 to show 3 dots in the 4th quadrant; 0 not to show them

pathkx='c:\matlab\work\trajs\lo\kxout02';
pathky='c:\matlab\work\trajs\lo\kyout02';
```

```
pathdens='c:\matlab\work\trajs\lo\densout02';

%(b)
subplot('position',[0 0 0.5 0.8]);
[kim,kl]=gridspoff(pathkx,pathky,pathdens,16,2,1,0,1);
im=fftshift(fft2(fftshift(kim)));
imagesc(abs(im(129:384,129:384)));
colormap(gray);

%(a)
subplot('position',[0 0.8 0.5 0.2]);
plot(abs(im(240,129:384)));

%(d)
subplot('position',[0.5 0 0.5 0.8]);
[kim,kl]=gridspoff(pathkx,pathky,pathdens,16,2,1,530,1);
im=fftshift(fft2(fftshift(kim)));
imagesc(abs(im(129:384,129:384)));
colormap(gray);

%(c)
subplot('position',[0.5 0.8 0.5 0.2]);
plot(abs(im(240,129:384)));
```

C.18 OFF-RESONANCE CORRECTION OF MRA (FIG. 4-6, 4-7)

Fig.4-6 shows the procedure of off-resonance correction for 3D Spiral MRA. All the reconstruction is performed by the program "gsp3dvd2". It is compiled on smash and runs on smash. Its varias functions are specified by command line paramters. The source code of gsp3dvd2.c is included in Appendix B.

Subplot (a) is the raw image without off-resonance correction or background substraction. Parameter "-v" (verbose mode) shows output during execution. "-s" selects the third temporal phase of the time series. The reconstructed 32 images are "sl1.pe001.001" to "sl1.pe032.001" in folder "fig4.6/a/". The Most Intensity Projection (MIP) image was processed using Matlab functions "max" and "squeeze".

```
% run this on smash: no background subtraction or inhomogeneity correction
% gsp3dvd2 -v -s3 P33792.7
data=read32im('c:\matlab\fig4.6\a');
a=max(data);
a=squeeze(a);
imagesc(abs(a));colormap(gray);
```

Subplot (b) adds a parameter "-k" that activates the background (mask) substraction. The substraction is performed in k-space before gridding so it can be performed with or without off-resonance correction. The reconstructed 32 images are "sl1.pe001.001" to "sl1.pe032.001" in folder "fig4.6/b/".

```
%run this on smash: with background subtraction, no inhomogeneity correction
%gsp3dvd2 -v -s3 -k P33792.7
data=read32im('c:\matlab\fig4.6\b');
a=max(data);
a=squeeze(a);
imagesc(abs(a));colormap(gray);
```

Subplot (c) first generates the phase maps with the parameter "-P". It then does the off-resonance correction with parameter "-h". The reconstructed 32 images are "sl1.pe001.001" to "sl1.pe032.001" in folder "fig4.6/c/".

```
%run these on smash: first line generates field maps.
% second line performs background subtraction and inhomogeneity correction.
%gsp3dvd2 -v -s4 -P P33792.7
%gsp3dvd2 -v -s3 -h -k P33792.7
data=read32im('c:\matlab\fig4.6\c');
a=max(data);
a=squeeze(a);
imagesc(abs(a));colormap(gray);
```

Subplot (d) shows the phase map. The reconstructed 32 phase maps are "ref.256.co0.s10.pe000" to "ref.256.co0.s10.pe031" in folder "fig4.6/d/"

```
%step c generates all phase maps.
%(d) is one slice of phase map.
data=read32pm('c:\matlab\fig4.6\d');
a=squeeze(data(23,:,256:-1:1));imagesc(real(a'));colormap(gray);
```

This figure contains magnified portions of the study in Fig.4-6. The image without correction is from Fig.4-6a. The image corrected with field maps is from Fig.4-6c. The image construction without field maps is performed with a separated reconstructed code "gsp3dz" developed based on "gsp3dvd2". Its executable and source code "gsp3dz.c" are included in "vdrecon" folder.

C.19 RESULTS OF SPIRAL MRA (FIG.4-8, 9, 10)

Fig. 4-8 contains the same data reconstructed in Fig.4-6. It is reconstructed for four different temporal phases. Fig.4-9 contains results for two patient studies with identical reconstruction procedures used in Fig.4-6. (These patient studies were performed under the supervision of Dr. Martin Prince and Dr. Yi Wang in Cornell University, where raw data files were archived.) Fig.4-10 contains an image projected in coronal (b) and another rotated image (a). The rotation was performed on GE 1.5T Signa LX scanner with built-in GE software "IVI", (Interactive Vessel Imaging.)

C.20 CONSTANT DENSITY VS. VARIABLE DENSITY (FIG.5-1)

The trajectory files are output in "kxoutv" and "kyoutv" for variable density spiral and "kxoutc" and "kyoutc" for constant density spiral. They were generated by the pulse sequence when the "genspiral2.c" code is executed. The names of the ouput files are changed to keep them different. The constant density code came from the original spiral sequence from Drs. Stenger and Noll.

The function fulltraj() draws the entire trajectories and it can be zoomed in manually to show the details as in Fig.5-1. The function psffull() calculates the Point Spread Function (PSF) of each spiral trajectory.

```
%(a) variable density trajectory
xoutv=readtraj('c:\matlab\fig5.1\kxoutv');
youtv=readtraj('c:\matlab\fig5.1\kyoutv');
fulltraj(xoutv,youtv,24);

%(d) constent density trajectory
xoutc=readtraj('c:\matlab\fig5.1\kxoutc');
youtc=readtraj('c:\matlab\fig5.1\kyoutc');
fulltraj(xoutc,youtc,24);

%(b) variable density point spread function
psfv=psffull(xoutv,youtv,24,0);

%(e) constent density point spread function
psfc=psffull(xoutc,youtc,24,0);

%(c) & (f) phantom study pfiles was not archived.
```

Functions psffull() is defined in psffull.m

```
function psf=psffull(kx,ky,nl,center);
% kx=readtraj('c:\inetpub\ftproot\kxout');
% ky=readtraj('c:\inetpub\ftproot\kyout');
% fulltraj(kx,ky,nl);
% kx-> array of kx
% ky-> array of ky
% nl # of leaves/rotations
% center is a flag for the bright spot at center
% when center=0, the spot is cut out so the ring appears.
psf=zeros(512,512);
len=length(kx);
r=max(abs(sqrt(kx.*kx+ky.*ky)));
for i=0:nl-1
    c=cos(2*pi*i/nl+0.25);
    s=sin(2*pi*i/nl+0.25);
    tx=kx*c-ky*s;
    ty=kx*s+ky*c;
    for j=1:len
        psf(round(256+255*tx(j)/r),round(256+255*ty(j)/r))=1;
    end
end
psf=abs(fftshift(fft2(fftshift(psf))));
if center==0
    psf(254:259,254:259)=0;
end
imagesc(psf);
colormap(gray);
```

C.21 REFERENCE IMAGE FOR SIMULATION (FIG. 5-2)

Fig.5-2 contains a numerically generated pattern as a reference for spiral imaging. The pattern is generated by a function called face(). The parameter of face() is the radius of a Fermi filter in k-space. This filter softens the hard edges of the pattern. The image intensity in the 2nd quadrant is normalized by function scaleim().

```
% generate reference image
ref=face(32);
colormap(gray);
```

Function face() is defined in face.m

```
function f=face(fermirad);
kx=[-8:0.0625:8-0.0625];
ky=-kx;
[x,y]=meshgrid(kx,ky);
x(129,129)=eps;
y(129,129)=eps;
%f=sinc(8*x).*sinc(8*y)*16;
% f=sinc(8*x).*sinc(8*y)*16-sinc(4*x).*sinc(4*y)*4;
%f=jinc(4,4,0,0,x,y)+0.7*jinc(1.5,1.5,1.75,0,x,y)-0.7*jinc(1.5,1.5,-
1.75,0,x,y);
f=jinc(4,4,0,0,x,y);
for i=1:7
    for j=1:7
        r=abs(i-4)+abs(j-4);
%        if r<5 && (i<=4 || j<=4)
            if r<4 && i<4 && j<4
            f=f-0.5*jinc(0.25,0.25,(i-4),(j-4),x,y);
        end
    end
end

%figure;
nim=length(kx);
f=f.*fermi(nim,fermirad,fermirad/6.4);
f=fftshift(ifft2(f));
f=f(:,256:-1:1);
f=scaleim(f,91,115,91,115);
imagesc(abs(f));
```

Function jinc() is defined in jinc.m.

```
function j=jinc(a,b,x,y,kx,ky)
% jinc jinc(kx,ky) with scaling parameters a and b, and shifting parameter x
and y

r=sqrt((a*kx).^2+(b*ky).^2);
j=a*b.*exp(-i*2*pi*x.*kx).*exp(-i*2*pi*y.*ky).*besselj(1,2*pi*r)./r;
```

Function fermi() is defined in fermi.m

```
function ffilter = fermi(xdim, cutoff, trans_width)
% FERMI creates a 2D Fermi filter.
%   FFILTER = FERMI(XDIM,CUTOFF,TRANS_WIDTH) calculates a 2D
%     Fermi filter on a grid with dimensions XDIM * XDIM.
%    The cutoff frequency is defined by CUTOFF and represents
%     the radius (in pixels) of the circular symmetric function
%     at which the amplitude drops below 0.5
%    TRANS_WIDTH defines the width of the transition.
%
%    Author: Wally Block, UW-Madison  02/23/01.
%
%    Call: ffilter = fermi(xdim, cutoff, trans_width);

[X,Y] = meshgrid(-xdim/2:1:(xdim/2 -1),-xdim/2:1:(xdim/2 -1));
radius = [X.^2 + Y.^2].^0.5;
ffilter = 1./(1 + exp((radius -cutoff)/trans_width));
```

Function scaleim() is defined in scaleim.m. This function scales the intensity of the image in (x_1:x_2, y_1:y_2) region to 1.0.

```
function im=scaleim(a,x1,x2,y1,y2);
ref=sum(sum(abs(a(x1:x2,y1:y2))))/((abs(x2-x1)+1)*(abs(y2-y1)+1));
im=a/ref;
```

C.22 FOUR EXAMPLES OF SPIRAL TRAJECTORIES (FIG. 5-3)

Fig.5-3 shows four examples of spiral trajectories. These trajectories are store in four pairs of binary files: (kxout01 & kyout01), (kxout02 & kyout02), (kxout12, kyout12) and (kxout64, kyout64). They consist of 1, 2, 12 and 64 interleaves. These four trajectories are plotted individually. The fulltraj() function is called.

```
% all trajectories are pre-generated for fast execution
% because the algorithms takes long
% only reading from the files in ~/matlab/work/trajs/lo
%for 1 shot spiral, the two files are kxout01, kyout01
%for 2 shots, they are kxout02, kyout02
%to plot a trajectory, use fulltraj(kx,ky,nl)
% fulltraj(kx,ky,nl);
% kx-> array of kx
% ky-> array of ky
% nl # of leaves/rotations

%to plot 1 shot
kx=readtraj('c:\matlab\work\trajs\lo\kxout01');
ky=readtraj('c:\matlab\work\trajs\lo\kyout01');
fulltraj(kx,ky,1);

%to plot 2 shots
kx=readtraj('c:\matlab\work\trajs\lo\kxout02');
ky=readtraj('c:\matlab\work\trajs\lo\kyout02');
fulltraj(kx,ky,2);

%to plot 12 shots
ky=readtraj('c:\matlab\work\trajs\lo\kxout12');
kx=readtraj('c:\matlab\work\trajs\lo\kyout12');
fulltraj(kx,ky,12);

%to plot 64 shots
```

```
ky=readtraj('c:\matlab\work\trajs\lo\kxout64');
kx=readtraj('c:\matlab\work\trajs\lo\kyout64');
fulltraj(kx,ky,64);
```

C.23 FIVE PLOTS OF IN VIVO PUFA ECHOES TO ESTIMATE SNR AND T2* DECAY CONSTANT (FIG. 5-4, TAB. 5-2)

Five *in vivo* echoes are plotted and used ase the basis for estimating SNR and T_2* decay constant.

```
fid1=readp('c:\vdrecon\feb22\P23040.7',2048,1,1);
subplot('position',[0 0.8 1 0.2]);
plot(real(fid1));
fid2=readp('c:\vdrecon\mar2\P09216.7',2048,1,1);
subplot('position',[0 0.6 1 0.2]);
plot(real(fid2));
fid3=readp('c:\vdrecon\mar7\P13824.7',2048,1,1);
subplot('position',[0 0.4 1 0.2]);
plot(real(fid3));
fid4=readp('c:\vdrecon\mar9\P08192.7',2048,1,1);
subplot('position',[0 0.2 1 0.2]);
plot(real(fid4));
fid5=readp('c:\vdrecon\dec29\P20992.7',2048,1,1);
subplot('position',[0 0 1 0.2]);
plot(real(fid5));
```

SNR is defined as the mean of the amplitude at echo peak divided by the standard deviation of the the noise. Either the beginning or the end of the echo is used for noise measurement.

```
s(1)=mean(abs(fid1(800:900)));
n(1)=std(fid1(1900:2000));
s(2)=mean(abs(fid2(800:900)));
n(2)=std(fid2(100:200));
s(3)=mean(abs(fid3(800:900)));
n(3)=std(fid3(1:100));
s(4)=mean(abs(fid4(800:900)));
n(4)=std(fid4(1900:2000));
s(5)=mean(abs(fid5(800:900)));
n(5)=std(fid5(1900:2000));

s =

  300.3789  108.9555  116.9302  193.0982   28.2042

n =

   30.7001   41.0494   27.9416   25.4877   9.1815
```

```
s./n

ans =

    9.7843    2.6543    4.1848    7.5761    3.0718
```

The fitting was performed with 500 points (from point 1001 to 1500 in 2048 points) at the decay curve of the forth echo. Since the natural log of an exponential decay function becomes a linear function, matlab built-in function "**polyfit**" was used. The syntax "`a=polyfit(1:500,log(abs(d)),1);`" fits the log of the signal magnitude "d" to index 1 to 500. The third parameter of "1" specifies that the fitting is for a first order polynomial function (a_1x+a_2). The result is

```
a =

   -0.0038    5.3273
```

-0.0038 is the coefficient of the first order term and 5.3273 is the linear constant. The T2* constant is determined by 32us * (1/ 0.0038)=8.4ms. The data and the fit are plotted together.

```
fid4=readp('c:\vdrecon\mar9\P08192.7',2048,1,1);
d=fid4(1001:1500);
t=1:500;
a=polyfit(1:500,log(abs(d)),1);
plot(abs(d));
hold on;
plot(exp(5.3273-0.0038*t));
```

C.24 FOUR EXAMPLES OF SIMULATED SPIRAL IMAGES (FIG.5-6)

The spiralmain() function simulates 12 sets of trajectories with one pair of values for T_2* and SNR. "ref" is the reference image. The parameter "nex" is the number of repetitions. The result is shown as a plot of estimation errors with error bars. The bigger the value of "nex" is, the smaller the error bars. Four examples of simulated images are displayed. Single shot spiral loses too much image details when T_2* decay is fast. The spiralmain() function calls gridspiral() function, which calls genspdata() function.

```
%spiralmain.m simulates 12 spiral images using 1 shot, 2 shots
%4 shots, 6, 8 12 16 20 24 32 48 64, etc
%the point is to show that 1 shot spiral loses too much signal because of T2*
decay
%it gets better when shots increase, the increase beyond 2 shots is small.
%24, 32, 48 and 64 shots are ridiculous because readout is too short for low
%resolution imaging.
% each image is compared to a reference image "ref". see thesis from fig5-6.
% the result of the comparison is a number.
% 12 trajectories produce 12 error numbers.
% nex repetitions produce nex iterations


%function [errarray,imarray]=spiralmain(T2,snrunit,ref,nex);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [erraray, imarray]=spiralmain (T2, snrunit, ref, nex);
% T2 in micro seconds.
% ref is generated with face.m : ref=face(32);
% nex is the number of repeatition
% errarray is a 12 by nex matrix
% imarray is a 256 by 256 by 12 matrix

ref=face(32);
%for example, T2*=12000us, snr=2, nex=20
[err,imarray]=spiralmain(12000,2,ref,20);
%errorbar.m plots 12 errors with error bars from nex iterations
% the larger the nex, the smaller the error bars, the longer the execution
time
% below is one plot for T2*=12000us=12ms, snr=2,
errorbar(mean(err'),std(err'));
% four image examples are
subplot(2,2,1);
imagesc(abs(imarray(:,:,1)));
subplot(2,2,2);
imagesc(abs(imarray(:,:,2)));
subplot(2,2,3);
imagesc(abs(imarray(:,:,4)));
subplot(2,2,4);
imagesc(abs(imarray(:,:,6)));
colormap(gray);


% there is a little gibbs ringing artifacts.i.e. the small edge around the
circle.
% that's because the pattern has a sharp edge
% and the images have low resolution
% stronger fermi filter can remove it
% real in vivo patterns don't have sharp edges so low resolution images are
fine.
```

The function spiralmain() is defined in spiralmain.m.

```
function [errarray,imarray]=spiralmain(T2,snrunit,ref,nex);
% [erraray, imarray]=spiralmain (T2, snrunit, ref, nex)
% T2 in micro seconds.
% ref is generated with face.m : ref=face(32);
```

```
% nex is the number of repeatition
% errarray is a 12 by nex matrix
% imarray is a 256 by 256 by 12 matrix
T2, snrunit
errarray=zeros(12,nex);
imarray=zeros(256,256,12);
pathkx=int16(ones(12,31));
pathky=int16(ones(12,31));
pathdens=int16(ones(12,33));
    nl=[1 2 4 6 8 12 16 20 24 32 48 64];

    for ii=1:12
        if ii<6
        pathkx(ii,:)=sprintf('c:\\matlab\\work\\trajs\\lo\\kxout0%d',nl(ii));
        pathky(ii,:)=sprintf('c:\\matlab\\work\\trajs\\lo\\kyout0%d',nl(ii));

pathdens(ii,:)=sprintf('c:\\matlab\\work\\trajs\\lo\\densout0%d',nl(ii));
        else
        pathkx(ii,:)=sprintf('c:\\matlab\\work\\trajs\\lo\\kxout%2d',nl(ii));
        pathky(ii,:)=sprintf('c:\\matlab\\work\\trajs\\lo\\kyout%2d',nl(ii));

pathdens(ii,:)=sprintf('c:\\matlab\\work\\trajs\\lo\\densout%2d',nl(ii));
        end
    end
nzpe=1;
opfov=16;
for j=1:nex
%    j
    for leafindex=1:12
        i=leafindex;
%         'number of leaves'
%         nl(i)

[kspace,klength]=gridspiral(char(pathkx(i,:)),char(pathky(i,:)),char(pathdens
(i,:)),opfov,nl(leafindex),nzpe,T2,snrunit,'s');
        klength;
        t=fftshift(fft2(fftshift(kspace)));
        im=t(129:384,129:384);
        tempim=scaleim(im,91,130,91,130);
        imarray(:,:,i)=imarray(:,:,i)+tempim;
        %err=imerr(imarray(:,:,i),ref,141,165,141,165)*sqrt(nl(i)*klength)
        err=imerr(tempim,ref,135,165,135,165);
        errarray(i,j)=err;
    end
 %   plot(errarray);
end
%plot(errarray/(nex));
temp=errarray';
erraray=temp;
%errorbar(mean(temp),std(temp));
imarray=imarray/nex;
```

Function imerr() is defined in imerr.m

```
function err=imerr(im, ref, x1, x2, y1, y2);
errm=(abs(im(x1:x2,y1:y2))-abs(ref(x1:x2,y1:y2))).^2;
```

```
%surf(abs(errm));
err=sqrt(sum(sum(errm))/((abs(x2-x1)+1)*(abs(y2-y1)+1)));
```

Function gridspiral() is defined in gridspiral.m

```
function
[kspacem,klength]=gridspiral(pathkx,pathky,pathdens,opfov,nl,nzpe,T2,snrunit,
data);
nim=512+8;
dwin=1.5;
kspace=zeros(nim,nim);
kspacem=zeros(512,512,nzpe);
ws=zeros(nim,nim);
xout=readtraj(pathkx);
yout=readtraj(pathky);
klength=length(xout);
kx1=xout(1:klength)*2;
ky1=yout(1:klength)*2;
klength=length(kx1);
kx=zeros(nl,klength);
ky=zeros(nl,klength);
for i=0:nl-1
    c=cos(2*pi*i/nl);
    s=sin(2*pi*i/nl);
    tx=kx1*c-ky1*s;
    ty=kx1*s+ky1*c;
    kx(i+1,:)=tx(:)';
    ky(i+1,:)=ty(:)';
end
kdens=zeros(klength);
kdens(1)=eps;

kdens=readtraj(pathdens);
k1=kaiser(1024,0.5);
%k1=readtraj('/home/henry/vdspiral/vdrecon/kaisout');
kker=k1*k1';
%read real data from a pfile for 1.5 st spiral
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%fid2=readp(pfile,klength,nl,nzpe);
% refoc=[1:klength]'*pi*0.5;
% fid2=fft(fid2,32);
% fid=squeeze(fid2(16,:,:));
% for leaf=1:nl
%     fid(:,leaf)=squeeze(fid(:,leaf))./(cos(refoc)+i*sin(refoc));
% end
% if ( nl==1)
%     fid=fid';
% end

%read data from 3T Sel-MQC spiral or do simulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (data=='r')
    fid=readp(pfile,2048,nl,nzpe);
    fid=fft(fid);
```

```
%    fid=squeeze(fid2(1,:,:));
    if ( nl==1)
        fid=fid';
    end
end
if (data=='s')
    nzpe=1;
    t=genspdata(xout,yout,nl,opfov,T2,snrunit);
    fid=zeros(1,klength,nl);
    fid(1,:,:)=t;
end

step=floor(512/dwin);
for zpe=1:nzpe
    zpe
for leaf=1:nl
%     leaf
    for i=1:klength
        ix=floor(kx(leaf,i)-dwin)+1;
        iy=floor(ky(leaf,i)-dwin)+1;
        dkx=floor(step*(ix+dwin-kx(leaf,i)));
        dky=floor(step*(iy+dwin-ky(leaf,i)));
        w=kker(dkx+1:step:dkx+1+step*2,dky+1:step:dky+1+step*2);
        t=w*fid(zpe,i,leaf)*kdens(i);

ws(ix+nim/2:ix+2+nim/2,iy+nim/2:iy+2+nim/2)=ws(ix+nim/2:ix+2+nim/2,iy+nim/2:i
y+2+nim/2)+w;

kspace(ix+nim/2:ix+2+nim/2,iy+nim/2:iy+2+nim/2)=kspace(ix+nim/2:ix+2+nim/2,iy
+nim/2:iy+2+nim/2)+t;
    end
end
kspacet=kspace(5:516,5:516);
fermirad=22;
kspacet=kspacet.*fermi(512,fermirad,4);
%kspacem(:,:,zpe)=fftshift(fft2(fftshift(kspacet)));
%imagesc(abs(kspacem(:,:,zpe)));
kspacem(:,:,zpe)=kspacet;
end
%imagesc(abs(ws(200:300,200:300)));
```

Function genspdata() is defined in genspdata.m

```
function raw=genspdata(kx,ky,nl,fov,T2,snrunit);
imag=sqrt(-1);
l=length(kx);
raw=zeros(l,nl);
tsp=4;
t=[0:tsp:tsp*(l-1)]';
dw=0;
off=dw*0.000002*pi*t;
for i=0:nl-1
    c=cos(2*pi*i/nl);
    s=sin(2*pi*i/nl);
    x=(kx*c-ky*s)/fov;
    y=(kx*s+ky*c)/fov;
```

```
    x(2)=eps;x(1)=eps;y(2)=eps;y(1)=eps;
%     raw(:,i+1)=jinc(3.6,4.4,0,0,x,y);
%     raw(:,i+1)=(jinc(3.5,4.5,0,0,x,y)-
0.7*jinc(1.5,1.5,1.75,0,x,y)+0.7*jinc(1.5,1.5,-1.75,0,x,y));
    f=jinc(4,4,0,0,x,y);
%    f=sinc(8*x).*sinc(8*y)*16-sinc(4*x).*sinc(4*y)*4;
%     f=f+jinc(1.5,1.5,1.75,0,x,y)+jinc(1.0,1.0,0,2.25,x,y);
%     f=f+jinc(0.5,0.5,-1.75,0,x,y)+jinc(0.5,0.5,-1.75,1.25,x,y);
%     f=f+jinc(0.5,0.5,-1.75,-1.25,x,y);
%     f=f+jinc(0.25,0.25,0,-2.5,x,y)+jinc(0.25,0.25,1,-2.5,x,y);
%     f=f+jinc(0.25,0.25,-1,-2.5,x,y)+jinc(0.25,0.25,-2,-2.5,x,y);
%     f=f+jinc(0.25,0.25,2,-2.5,x,y);
%     f=f+jinc(0.1,0.1,-0.5,0,x,y)+jinc(0.1,0.1,-1,0.5,x,y);
%     f=f+jinc(0.1,0.1,-0.75,0.25,x,y)+jinc(0.1,0.1,-0.25,-0.25,x,y);
%     f=f+jinc(0.1,0.1,0,-0.5,x,y)+jinc(0.1,0.1,0.25,-0.75,x,y);
for ii=1:7
    for j=1:7
        r=abs(ii-4)+abs(j-4);
%        if r<5 && (ii<=4 || j<=4)
          if r<4 && ii<4 && j<4
           f=f-0.4*jinc(0.25,0.25,(ii-4),(j-4),x,y);
        end
    end
end
foff=0.2*jinc(1,1,1,1,x,y).*exp(-imag*off);
    raw(:,i+1)=f;
    relaxT2=exp(-t/T2);
%    fil=fermi1(l,l*0.9,l*0.05);
%    f=foff;
    temp=f.*relaxT2;
    opnex=64/nl;
    raw(:,i+1)=awgn(temp,snrunit*sqrt(opnex));
%    raw(:,i+1)=raw(:,i+1)+foff;
end
```

C.25 NINE PLOTS OF ESTIMATION ERRORS OF 12 SETS OF TRAJECTORIES (FIG.5-8)

Nine pairs of values for T2* and SNR are used to run through the simulation of 12 sets of trajectories. The result is nine plots. Each plot consists of 12 points showing the image error of 12 trajectories for that specific pair of (T_2*, SNR). When T_2* constant is short, the curve is steep.

When T_2* constant is long, the curve is flat. The conclusion is that when spiral is used to image a fast decaying signal, i.e. PUFA, stay away from single shot.

```
% nine plots are nine combinations of (T2*, snrunit)
%(a)
[erra,imarraya]=spiralmain(8000,0.6,ref,400);

%(b)
[errb,imarrayb]=spiralmain(12000,0.6,ref,400);

%(c)
[errc,imarrayc]=spiralmain(20000,0.6,ref,400);

%(d)
[errd,imarrayd]=spiralmain(8000,1.2,ref,400);

%(e)
[erre,imarraye]=spiralmain(12000,1.2,ref,400);

%(f)
[errf,imarrayf]=spiralmain(20000,1.2,ref,400);

%(g)
[errg,imarrayg]=spiralmain(8000,2.4,ref,400);

%(h)
[errh,imarrayh]=spiralmain(12000,2.4,ref,400);

%(i)
[erri,imarrayi]=spiralmain(20000,2.4,ref,400);

subplot(3,3,1);
errorbar(mean(erra'),std(erra'));

subplot(3,3,2);
errorbar(mean(errb'),std(errb'));

subplot(3,3,3);
errorbar(mean(errc'),std(errc'));

subplot(3,3,4);
errorbar(mean(errd'),std(errd'));

subplot(3,3,5);
errorbar(mean(erre'),std(erre'));

subplot(3,3,6);
errorbar(mean(errf'),std(errf'));

subplot(3,3,7);
errorbar(mean(errg'),std(errg'));

subplot(3,3,8);
errorbar(mean(errh'),std(errh'));
```

```
subplot(3,3,9);
errorbar(mean(erri'),std(erri'));
```

BIBLIOGRAPHY

1. Hahn EL. Spin Echoes. Physical Review 1950;80(4):580.
2. Bloch F. Nuclear Induction. Phys Rev 1946;70(7-8):460--474.
3. Purcell EMaT, H. C. and Pound, R. V. Resonance Absorption by Nuclear Magnetic Moments in a Solid. Phys Rev 1946;69(1-2):37--38.
4. Lauterbur PC. Image Formation by Induced Local Interactions: Examples Employing Nuclear Magnetic Resonance. Nature 1973;242(5394):190-191.
5. Damadian R. Tumor detection by nuclear magnetic resonance. Science 1971;171(976):1151-1153.
6. Haacke EM, Brown RW, Thompson MR, Venkatesan R. Magnetic Resonance Imaging: Physical Principles and Sequence Design. New York: Wiley-Liss; 1999.
7. Levitt MH. Spin Dynamics: Basics of Nuclear Magnetic Resonance. Chichester: Wiley; 2001.
8. Slichter CP. Principles of Magnetic Resonance. Heidelberg: Springer-Verlag; 1990.
9. Rabi IIaR, N. F. and Schwinger, J. Use of Rotating Coordinates in Magnetic Resonance Problems. Rev Mod Phys 1954;26(2):167--171.
10. Garwood M, DelaBarre L. The return of the frequency sweep: designing adiabatic pulses for contemporary NMR. J Magn Reson 2001;153(2):155-177.
11. Charles H. Cunningham DBVAPCDXREHNSJMP. Design of symmetric-sweep spectral-spatial RF pulses for spectral editing. Magnetic Resonance in Medicine 2004;52(1):147-153.
12. Lei H, Peeling J. Off-resonance effects of the radiofrequency pulses used in spectral editing with double-quantum coherence transfer. J Magn Reson 2000;144(1):89-95.
13. Bloch FaH, W. W. and Packard, Martin. Nuclear Induction. Phys Rev 1946;69(3-4):127.
14. Bloch FaH, W. W. and Packard, M. The Nuclear Induction Experiment. Phys Rev 1946;70(7-8):474--485.
15. Bloembergen NaP, E. M. and Pound, R. V. Relaxation Effects in Nuclear Magnetic Resonance Absorption. Phys Rev 1948;73(7):679--712.
16. Meyer CH, Hu BS, Nishimura DG, Macovski A. Fast spiral coronary artery imaging. Magn Reson Med 1992;28:202-213.
17. Stenger VA, Peltier S, Boada FE, Noll DC. 3D spiral cardiac/respiratory ordered fMRI data acquisition at 3 Tesla. Magn Reson Med 1999;41(5):983-991.
18. Griffin JL, Lehtimaki KK, Valonen PK, Grohn OH, Kettunen MI, Yla-Herttuala S, Pitkanen A, Nicholson JK, Kauppinen RA. Assignment of 1H nuclear magnetic resonance visible polyunsaturated fatty acids in BT4C gliomas undergoing ganciclovir-thymidine kinase gene therapy-induced programmed cell death. Cancer Res 2003;63(12):3195-3201.

19. Hakumaki JM, Poptani H, Sandmair A-M, Yla-Herttuala S, Kauppinen RA. 1H MRS detects polyunsaturated fatty acid accumulation during gene therapy of glioma: Implications for the in vivo detection of apoptosis. Nat Med 1999;5(11):1323-1327.
20. M. Albert Thomas NBKYND. Volume-localized two-dimensional correlated magnetic resonance spectroscopy of human breast cancer. Journal of Magnetic Resonance Imaging 2001;14(2):181-186.
21. Singer S, Sivaraja M, Souza K, Millis K, Corson JM. 1H-NMR Detectable Fatty Acyl Chain Unsaturation in Excised Leiomyosarcoma Correlate with Grade and Mitotic Activity. J Clin Invest 1996;98(2):244-250.
22. Amann M BM, Floemer F, Schoenberg SO, Schad LR. Three-dimensional spiral MR imaging: Application to renal multiphase contrast-enhanced angiography. Magn Reson Med 2002;48:290-296.
23. Brown TR, Kincaid BM, Ugurbil K. NMR Chemical Shift Imaging in Three Dimensions. PNAS 1982;79(11):3523-3526.
24. Mansfield P. Spatial Mapping of the Chemical Shift in NMR. Magn Reson Imaging 1984;1(3):370-386.
25. Posse S, Tedeschi G, Risinger R, Ogg R, Le Bihan D. High speed 1H spectroscopic imaging in human brain by echo planar spatial-spectral encoding. Magn Reson Med 1995;33(1):34-40.
26. Webb P, Spielman D, Macovski A. A fast spectroscopic imaging method using a blipped phase encode gradient. Magn Reson Med 1989;12(3):306-315.
27. Wolfgang Dreher DL. Fast proton spectroscopic imaging with high signal-to-noise ratio: Spectroscopic RARE. Magnetic Resonance in Medicine 2002;47(3):523-528.
28. Haase A, Frahm J, Hanicke W, Matthaei D. 1H NMR chemical shift selective (CHESS) imaging. Phys Med Biol 1985;30(4):341-344.
29. He QH, Shungu DC, Vanzijl PCM, Bhujwalla ZM, Glickson JD. Single-Scan in Vivo Lactate Editing with Complete Lipid and Water Suppression by Selective Multiple-Quantum-Coherence Transfer (Sel-MQC) with Application to Tumors. Journal of Magnetic Resonance, Series B 1995;106(3):203-211.
30. Korosec FR, Frayne R, Grist TM, Mistretta CA. Time-resolved contrast-enhanced 3D MR angiography. Magn Reson Med 1996;36:345-351.
31. Mistretta CA, Grist TM, Korosec FR, Frayne R, Peters DC, Mazaheri Y, Carrol TJ. 3D time-resolved contrast-enhanced MR DSA: advantages and tradeoffs. Magn Reson Med 1998;40(4):571-581.
32. Peters DC, Korosec FR, Grist TM, Block WF, Holden JE, Vigen KK, Mistretta CA. Undersampled projection reconstruction applied to MR angiography. Magn Reson Med 2000;43(91-101).
33. Vigen KK, Peters DC, Grist TM, Block WF, Mistretta CA. Undersampled projection-reconstruction imaging for time-resolved contrast-enhanced imaging. Magn Reson Med 2000;43:170-176.
34. Abragam A. The Principles of Nuclear Magnetism. Oxford: Oxford University Press; 1961.
35. Ernst RR, Bodenhausen G, Wokaun A. Principles of Nuclear Magnetic Resonance in One and Two Dimensions. Oxford: Oxford University Press; 1987.
36. Goldman M. Quantum Description of High-Resolution NMR in Liquids. 1988.
37. Nielsen MA, Chuang IL. Quantum Computation and Quantum Information. 2000.

38. Ryner LN, Sorenson JA, Thomas MA. Localized 2D J-resolved 1H MR spectroscopy: strong coupling effects in vitro and in vivo. Magn Reson Imaging 1995;13(6):853-869.
39. Ryner LN, Sorenson JA, Thomas MA. 3D localized 2D NMR spectroscopy on an MRI scanner. J Magn Reson B 1995;107(2):126-137.
40. Dreher W, Leibfritz D. On the use of two-dimensional-J NMR measurements for in vivo proton MRS: measurement of homonuclear decoupled spectra without the need for short echo times. Magn Reson Med 1995;34(3):331-337.
41. M. Albert Thomas KYNBPDAKBSMFJCRLPMBG. Localized two-dimensional shift correlated MR spectroscopy of human brain. Magnetic Resonance in Medicine 2001;46(1):58-67.
42. Maudsley AA, Wokaun A, Ernst RR. Coherence transfer echoes. Chemical Physics Letters 1978;55(1):9-14.
43. Bax A, De Jong PG, Mehlkopf AF, Smidt J. Separation of the different orders of NMR multiple-quantum transitions by the use of pulsed field gradients. Chemical Physics Letters 1980;69(3):567-570.
44. Sakurai JJ. Modern Quantum Mechanics: Addison Wesley; 1994.
45. McConnell J. Nuclear Magnetic Relaxation in Liquids. Cambridge: Cambridge University Press; 1987.
46. Jackson JD. Classical Electrodynamics. 1998.
47. Kvistad KA, Bakken IJ, Gribbestad IS, Ehrnholm B, Lundgren S, Fjosne HE, Haraldseth O. Characterization of neoplastic and normal human breast tissues with in vivo (1)H MR spectroscopy. J Magn Reson Imaging 1999;10(2):159-164.
48. Jagannathan NR, Singh M, Govindaraju V, Raghunathan P, Coshic O, Julka PK, Rath GK. Volume localized in vivo proton MR spectroscopy of breast carcinoma: variation of water-fat ratio in patients receiving chemotherapy. NMR Biomed 1998;11(8):414-422.
49. Roebuck JR, Cecil KM, Schnall MD, Lenkinski RE. Human breast lesions: characterization with proton MR spectroscopy. Radiology 1998;209(1):269-275.
50. Davis AL, Laue ED, Keeler J, Moskau D, Lohman J. Absorption-mode two-dimensional NMR spectra recorded using pulsed field gradients. Journal of Magnetic Resonance (1969) 1991;94(3):637-644.
51. Hurd RE. Gradient-enhanced spectroscopy. Journal of Magnetic Resonance (1969) 1990;87(2):422-428.
52. Keeler J. Understanding NMR Spectroscopy Cambridge: Wiley; 2005.
53. Tesiram YA, Saunders D, Towner RA. Application of proton NMR spectroscopy in the study of lipid metabolites in a rat hepatocarcinogenesis model. Biochim Biophys Acta 2005;1737(1):61-68.
54. Griffin JL, Blenkiron C, Valonen PK, Caldas C, Kauppinen RA. High-resolution magic angle spinning 1H NMR spectroscopy and reverse transcription-PCR analysis of apoptosis in a rat glioma. Anal Chem 2006;78(5):1546-1552.
55. He Q, Shkarin P, Hooley RJ, Lannin DR, Weinreb JC, Bossuyt VIJ. In vivo MR Spectroscopic Imaging of Polyunsaturated Fatty Acids (PUFA) in Healthy and Cancerous Breast Tissues by Selective Multiple -Quantum Coherence Transfer (Sel-MQC)-A Preliminary Study (Accepted). Magn Reson Med.
56. Wolfgang Dreher DL. A new method for fast proton spectroscopic imaging: Spectroscopic GRASE. Magnetic Resonance in Medicine 2000;44(5):668-672.

57. Dreher W, Geppert C, Althaus M, Leibfritz D. Fast proton spectroscopic imaging using steady-state free precession methods. Magn Reson Med 2003;50(3):453-460.
58. Meyer CH, Hu BS, Nishimura DG, Macovski A. Fast spiral coronary artery imaging. Magn Reson Med 1992;28(2):202-213.
59. Zhu H, Buck DG, Zhang Z, Zhang H, Wang P, Stenger VA, Prince MR, Wang Y. High temporal and spatial resolution 4D MRA using spiral data sampling and sliding window reconstruction. Magn Reson Med 2004;52(1):14-18.
60. Adalsteinsson E, Irarrazabal P, Topp S, Meyer C, Macovski A, Spielman DM. Volumetric spectroscopic imaging with spiral-based k-space trajectories. Magn Reson Med 1998;39(6):889-898.
61. Adalsteinsson E, Irarrazabal P, Spielman DM, Macovski A. Three-dimensional spectroscopic imaging with time-varying gradients. Magn Reson Med 1995;33(4):461-466.
62. Hiba B, Serduc R, Provent P, Farion R, Remy C, Ziegler A. 2D J-resolved spiral spectroscopic imaging at 7 T: application to mobile lipid mapping in a rat glioma. Magn Reson Med 2004;52(3):658-662.
63. Hiba B, Faure B, Lamalle L, Decorps M, Ziegler A. Out-and-in spiral spectroscopic imaging in rat brain at 7 T. Magn Reson Med 2003;50(6):1127-1133.
64. Mayer D, Kim DH, Adalsteinsson E, Spielman DM. Fast CT-PRESS-based spiral chemical shift imaging at 3 Tesla. Magn Reson Med 2006;55(5):974-978.
65. Prince MR, Grist TM, Debatin JF. 3D Contrast MR Angiography. Berlin: Springer; 2003.
66. Prince MR, Chenevert TL, Foo T, Londy FJ, Ward J, Maki J. Contrast-enhanced abdominal MR angiography: optimization of imaging delay time by automating the detection of contrast material arrival in the aorta. Radiology 1997;203(1):109-114.
67. Wilman AH, Riederer SJ, King B, Debbins J, Rossman PJ, Ehman RL. Fluoroscopically trigered contrast-enahnced three-dimensional MR angiography with elliptical centric view order: application to the renal arteries. Radiology 1997;205(1):137-146.
68. Wang Y, Johnston DL, Breen JF, Huston J, 3rd, Jack CR, Julsrud PR, Kiely MJ, King BF, Riederer SL, Ehman RL. Dynamic MR digital subtraction angiography using contrast enhancement, fast data acquisition, and complex subtraction. Magn Reson Med 1996;36(4):551-556.
69. Hennig J, Scheffler K, Laubenberger J, Strecker R. Time-resolved projection angiography after bolus injection of contrast agent. Magn Res Med 1997;37(3):341-145.
70. Winchester PA, Lee HM, Khilnani NM, Wang Y, Trost D, Bush HL. Two-dimensional MR digital subtraction angiography of the lower extremity, comparison with x-ray angiography. JVIR supplement 1998;9:174.
71. Irarrazabal P, Nishimura DG. Fast three dimensional magnetic resonance imaging. Magn Reson Med 1995;33(5):656-662.
72. Nishimura DG, Irarrazabal P, Meyer CH. A velocity k-space analysis of flow effects in echo-planar and spiral imaging. Magn Reson Med 1995;33(4):549-556.
73. Kerr AB, Pauly JM, Hu BS, Li KC, Hardy CJ, Meyer CH, Macovski A, Nishimura DG. Real-time interactive MRI on a conventional scanner. Magn Reson Med 1997;38(3):355-367.
74. Stenger VA, Peltier S, Boada FE, Noll DC. 3D Spiral Cardiac/Respiratory Ordered fMRI Data Acquisition at 3T. Proc ISMRM 1998:297.

75. Noll DC. Reconstruction methods for magnetic resonance imaging: PhD Thesis, Stanford University; 1991.
76. Man LC, Pauly JM, Macovski A. Multifrequency interpolation for fast off-resonance correction. Magn Reson Med 1997;37(5):785-792.
77. Glover GH. Simple analytic spiral K-space algorithm. Magn Reson Med 1999;42(2):412-415.
78. Duyn JH, Yang Y. Fast spiral magnetic resonance imaging with trapezoidal gradients. J Magn Reson 1997;128(2):130-134.
79. Kim DH, Adalsteinsson E, Spielman DM. Simple analytic variable density spiral design. Magn Reson Med 2003;50(1):214-219.
80. Tsai CM, Nishimura DG. Reduced aliasing artifacts using variable-density k-space sampling trajectories. Magn Reson Med 2000;43(3):452-458.
81. Twieg DB. The k-trajectory formulation of the NMR imaging process with applications in analysis and synthesis of imaging methods. Med Phys 1983;10(5):610-621.
82. Choi C, Coupland NJ, Bhardwaj PP, Kalra S, Casault CA, Reid K, Allen PS. T2 measurement and quantification of glutamate in human brain in vivo. Magn Reson Med 2006;56(5):971-977.
83. Choi C, Coupland NJ, Kalra S, Bhardwaj PP, Malykhin N, Allen PS. Proton spectral editing for discrimination of lactate and threonine 1.31 ppm resonances in human brain in vivo. Magn Reson Med 2006;56(3):660-665.
84. Choi C, Coupland NJ, Bhardwaj PP, Malykhin N, Gheorghiu D, Allen PS. Measurement of brain glutamate and glutamine by spectrally-selective refocusing at 3 Tesla. Magn Reson Med 2006;55(5):997-1005.
85. Choi C, Coupland NJ, Hanstock CC, Ogilvie CJ, Higgins AC, Gheorghiu D, Allen PS. Brain gamma-aminobutyric acid measurement by proton double-quantum filtering with selective J rewinding. Magn Reson Med 2005;54(2):272-279.
86. Prescot AP, Frederick BD, Wang L, Brown J, Jensen JE, Kaufman MJ, Renshaw PF. In vivo detection of brain glycine with echo-time-averaged (1)H magnetic resonance spectroscopy at 4.0 T. Magn Reson Med 2006;55(3):681-686.
87. Keltner JR, Wald LL, Frederick BD, Renshaw PF. In vivo detection of GABA in human brain using a localized double-quantum filter technique. Magn Reson Med 1997;37(3):366-371.
88. Wolfgang Dreher DL. New method for the simultaneous detection of metabolites and water in localized in vivo ^{<FONT SIZE='-1'>1</FONT>}H nuclear magnetic resonance spectroscopy. Magnetic Resonance in Medicine 2005;54(1):190-195.