IMPLEMENTATION OF DURGA

IMPLEMENTATION OF DURGA – A PSEUDO-RANDOM, NON-CARTESIAN SAMPLING SCHEME

By
PAUL POLAK, B.Eng Mgmt

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the
Requirements for the Degree
Master of Applied Science

McMaster University

Master of Applied Science (2009)
Biomedical Engineering
McMaster University
Hamilton, Ontario

TITLE:	Implementation of Durga – a pseudo-random, non-cartesian sampling scheme
AUTHOR:	Paul Polak, B.Eng Mgmt (McMaster University)
SUPERVISORS:	Dr. C. Anand & Dr. M. Noseworthy
NUMBER OF PAGES:	xii, 104

Abstract

One important topic in Magnetic Resonance Imaging (MRI) is a desire for faster and more efficient scanning, with an elimination, or at least minimization, of artifacts in the resulting image. Reducing patient discomfort, increasing scanner throughput, difficulties in imaging dynamic elements (cardiovascular system), and minimization of movement artifacts motivate faster scan times. To this end, more complicated and mathematically intense sampling strategies have been developed which either under-sample or ignore portions of k-space. These missing samples manifest themselves as specific artifacts indicative of the sampling strategy and the amount of under-sampling.

Durga uses pseudo-random, volumetric and velocity insensitive k-space trajectories, which are derived from second-order cone optimization problems [2]. Under-sampling a random trajectory results in artifacts which resemble incoherent noise [19] instead of aliased images. Velocity insensitive trajectories do not require rewinders to balance first or higher order moments. Further, volumetric or 3D k-space sampling strategies can choose to ignore slice select rewinders by beginning and terminating sampling readout at the centre of k-space, interspersed with a finite time around the RF pulse. These combine to increase the sampling duty-cycle, resulting in increased efficiency and decreased imaging time.

The initial experiments described in this thesis indicate Durga's potential for use in various ultra-fast MR applications, in particular time-sensitive, high contrast applications, where the presence of random, white-noise does not represent a major detriment to the images. In particular, the efficacy of magnetic resonance angiography (MRA), albeit in simulated, static experiments, is investigated with positive results.

This dissertation is intended to be a summary of the implementation of Durga on a 3.0T GE MRI system, required in the fulfillment of a M.A.Sc in Biomedical Engineering from McMaster University. Included is basic MR

theory, overview of various sampling strategies and techniques, outline of the hardware and software tools, results and discussion.

Acknowledgements

It is my pleasure to thank and acknowledge all those who helped make this thesis possible.

First and foremost to my supervisors Dr. Christopher Anand and Dr. Michael Noseworthy for their invaluable advice, wisdom, and expertise into my research. Their insights from both a theoretical and clinical/practical standpoint have been critical to this work, and have made a challenging project enjoyable. Perhaps even more instrumental has been their faith in me as a student and researcher, especially when it seemed that Durga would not come to a successful conclusion. It is no doubt obvious to say that this thesis would be unwritten without them.

I would like to thank the Department of Biomedical Engineering at McMaster University for their financial and educational support. In particular, Dr. John Brash for his leadership in the direction of the new department, and his able administrators in David Ryan and Laura Kobayashi.

Durga could not have been finished without the guidance and expertise of Jeffrey Stainsby of GE Research. His knowledge of EPIC and his willingness to help made a seemingly impossible problem tractable.

I would like to send my regards and gratitude to the entire lab in the Imaging Research Centre at St. Joseph's Healthcare in Hamilton. I am especially grateful to Norm Konyer and the MRI technicians for their help in my research. For my fellow students Ali, Adrian, Graham, Peter, Sergei and the rest of the lab I thank you for the educational and collegial environment you created – it would not have been the same without you.

Finally my heartfelt gratitude to my family and loved ones – without their support I would not be the person I am today.

To K8

Contents

List of Figures

List of Tables

Listings

Chapter 1

Magnetic Resonance Theory

1.1 Spin Physics

Quantum spin is a fundamental property of matter – every particle (*ie.* electrons, protons, even whole atoms) has a total spin. This number is a non-negative integral multiple of 1/2. Electrons, protons, neutrons all have a spin of 1/2, and these spin numbers are the same for all elementary particles in all atoms. What follows is a terse description of the underlying quantum properties – for a more thorough examination of these and the Stern and Gerlach experiment behind the discovery of quantum spin, consult [9] [18].

Spin is fundamental to describing the magnitude of angular momentum in quantum mechanics, given as[1]

$$\|\boldsymbol{S}\| = \hbar\sqrt{s(s+1)} \tag{1.1}$$

where s is the spin number, and $\hbar$ is the reduced Planck constant ($\hbar = \frac{h}{2\pi}$). The proportionality between the vector angular momentum $\boldsymbol{S}$ and the magnetic moment $\boldsymbol{\mu}$ can be expressed as

$$\boldsymbol{\mu} = \gamma\boldsymbol{S} \tag{1.2}$$

where γ is the gyromagnetic ratio. For the proton, this value is

$$\gamma = 2.675 \times 10^8 \text{ rad} \cdot \text{s}^{-1} \cdot \text{T}^{-1} \tag{1.3}$$

[1] In this text, vector quantities are denoted in boldface ($\boldsymbol{B}$), while scalars are in regular face (B).

Nucleus	Spin	$\gamma\ (=\frac{\gamma}{2\pi})\ \ (\mathrm{MHz}\cdot\mathrm{T}^{-1})$	Human Brain Abundance
^{1}H	1/2	42.58	88 M
^{17}O	5/2	-5.77[2]	16 mM
^{19}F	1/2	40.08	4 μM
^{23}Na	3/2	11.27	80 mM
^{31}P	1/2	17.25	75 mM

Table 1.1: Nuclei with their associated spins (increments of $\hbar$), gyromagnetic ratios, and human brain abundances ($\mathrm{M} = \mathrm{mole}\cdot\mathrm{L}^{-1}$).

For MRI, we are only concerned with nuclei that have a non-zero magnetic moment, and thus by (1.2), a non-zero angular momentum. Nuclei with even/even numbers of protons/neutrons (^{16}O, ^{12}C) have zero total angular momentum, and thus cannot be imaged with MR techniques. However, there are many other nuclei naturally occurring in the body which have angular momentum. Table 1.1 summarizes some of these nuclei, but the most commonly imaged is ^{1}H, because of its relative abundance in the form of water in the human body.

In the absence of an external magnetic field, the direction of angular momentum exists in only one state. However, in a magnetic field $\boldsymbol{B}$ these states become quantized into $2s+1$ possible values for the azimuthal quantum number m_s, which takes on the values:

$$m_s = -s, -s+1, -s+2, ...s-2, s-1, s \tag{1.4}$$

The main magnetic field $\boldsymbol{B}$ is usually defined as being oriented along the z-axis in MRI, and thus

$$\boldsymbol{B} = B_0\hat{z} \tag{1.5}$$

From the results of the Stern-Gerlach experiments, we know that the angular momentum will be quantized into distinct states. Using this, we can relate m_s to the z-component of the angular momentum in the external magnetic field as

$$S_z = m_s\hbar \tag{1.6}$$

[2]Negative gyromagnetic ratios correspond to a magnetic moment that is aligned anti-parallel to the angular momentum vector.

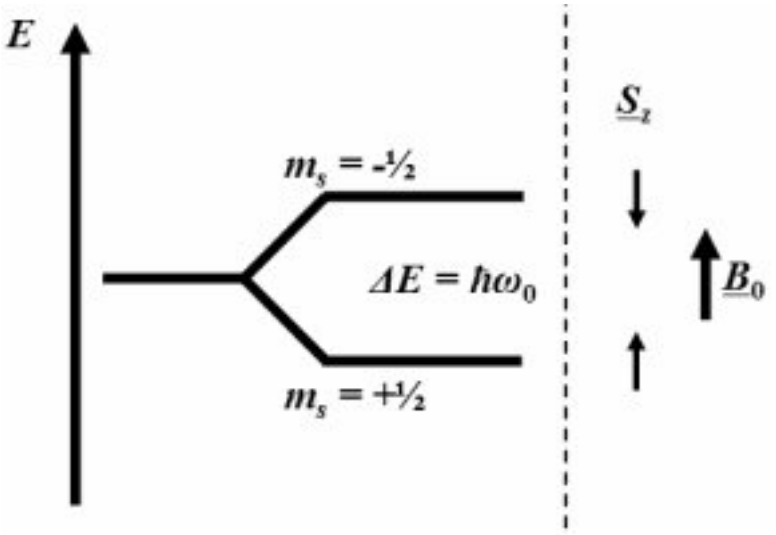

Figure 1.1: Zeeman splitting in main magnetic field B_0, higher energy state at the top.

If we now consider the classical potential energy of a magnetic dipole in an external magnetic field $\boldsymbol{B}$, we have:

$$E = -\boldsymbol{\mu} \cdot \boldsymbol{B} \tag{1.7}$$

From (1.2), (1.6), and (1.5), we find an expression for the energy of a magnetic moment dipole in an MRI experiment:

$$E = -\gamma m_s \hbar B_0 \tag{1.8}$$

If we consider the case for ^{1}H (*ie.* the proton), the values for m_s are $\pm 1/2$. By (1.8), we see that two energy levels are predicted – this is an example of the *Zeeman effect*, or "splittings" in the nuclear energy levels of magnetic moments in an external magnetic field (see Figure 1.1).

The amount of energy released or absorbed by a transition between the energy states is denoted by ΔE, and using (1.8) has the expression:

$$\begin{aligned} \Delta E &= E_{m_s=-1/2} - E_{m_s=1/2} \\ \Delta E &= \frac{1}{2}\gamma\hbar B_0 - (-\frac{1}{2}\gamma\hbar B_0) \\ \Delta E &= \gamma\hbar B_0 \end{aligned} \tag{1.9}$$

From (1.9), we define the *Larmor precession frequency* as the ratio of the change in energy per reduced Planck constant

$$\frac{\Delta E}{\hbar} = \omega_0 = \gamma B_0 \tag{1.10}$$

The Larmor frequency is the frequency of precession of the magnetic moments about the main magnetic field. It is at this frequency that we must trigger an RF pulse in order to "tip" our magnetization vector from the z-axis into the xy-plane. We will return to this idea in a later section.

1.2 Spin Density

There are two energy states for the proton – a lower state and a higher state, corresponding to spins of $1/2$ and $-1/2$. With regards to common nomenclature, the spins parallel to $\boldsymbol{B}$, or "spin-up" are at the lower energy state, while the anti-parallel "spin-down" protons are at the higher energy state. At equilibrium, the number of spins in either state is almost equal, with a slight excess of spins in the lower energy state. This point is critical to MRI experiments – it is this slight excess that wholly provides our signal.

It is important to note that the spin orientations are not rigid, and for the two-spin state the protons are continuously changing from parallel to anti-parallel and back. For a system at equilibrium this implies that for every change from spin-up to spin-down, there is another change from spin-down to spin-up, thereby maintaining our net total spin excess.

If we consider a proton as being connected to a larger lattice of spins at a temperature T, we can quantify the probability of a spin being in a given state using the Boltzmann factor:

$$P(\epsilon) = \frac{e^{-\epsilon/kT}}{\sum_\epsilon e^{-\epsilon/kT}} \tag{1.11}$$

where ϵ is the energy of the small system (*ie.* a proton), $P(\epsilon)$ denotes the probability of the system being in that energy state (*ie.* the probability of a spin being parallel or anti-parallel), and k is Boltzmann's constant. Given our two-state spin system of the proton, and using (1.8), we can express the probabilities of our spin being either spin-up ($m_s = 1/2$) or spin-down

$(m_s = -1/2)$ as

$$P_{m_s=1/2} = P_+ = \frac{e^{\hbar\omega_0/2kT}}{e^{-\hbar\omega_0/2kT} + e^{\hbar\omega_0/2kT}} \tag{1.12}$$

$$P_{m_s=-1/2} = P_- = \frac{e^{-\hbar\omega_0/2kT}}{e^{-\hbar\omega_0/2kT} + e^{\hbar\omega_0/2kT}} \tag{1.13}$$

Let us now consider the spin excess of protons. If the total number of protons in our system is N, we can separate this into the two components in our system – the number of protons in a spin-up orientation, and the number in spin-down, such that $N = N_+ + N_-$. Since we are interested in the excess of protons in the spin-up orientation (*ie.* $N_+ - N_-$) we note that this is equal to the difference in probabilities between the spin-up and spin-down states, multiplied by the number of total spins

$$\begin{aligned} N_+ - N_- &= N(P_+ - P_-) \\ \frac{N_+ - N_-}{N} &= P_+ - P_- \end{aligned} \tag{1.14}$$

Since $P_+ > P_-$[3], the ratio of the number of spin excess protons to the total number will also be positive. For a system at human body temperature of $310K$, and a magnetic field of $1.5T$, (1.14) works out to be about 5 excess protons for every million total. Although this might appear to be too small to derive any useful signal in MRI, consider that in a few grams of tissue, there are Avogadro's constant (6.022×10^{23}) number of molecules[4]. In consideration of this fact, we can see that the practical application of MRI for human body tissue is viable.

1.3 RF Signals and Magnetization Vectors

As previously mentioned, we can apply an RF pulse to our system at equilibrium in order to "tip" our magnetization vector from the z-axis into the xy-plane. The energy required to do this is the energy gap stated by (1.9),

[3]But just barely. In fact, given human body temperature of 310 K, and a B_0 field strength of 1.5 T, $P_+ \approx 0.500002476$, while $P_- \approx 0.499997524$.

[4]In $1mL$ of water, we have $\frac{2}{18} \times 6.022 \times 10^{23}$ number of protons, so that the total spin excess is $\approx 3.3138 \times 10^{17}$

and we require an RF pulse at the frequency of the Larmor precessional frequency (1.10). In this section, a classical approach to RF tipping will be taken – for a more rigorous quantum approach, consult [9].

We should first more formally define what we mean by "magnetization vector". Let us consider a sufficiently small volume V such that the main magnetic field can be assumed to be uniform over the region. We can express our magnetization vector $\boldsymbol{M}$ as the vector sum of the of the individual magnetic moments in that volume:

$$\boldsymbol{M} = \frac{1}{V} \sum_{V-protons} \boldsymbol{\mu}_i \tag{1.15}$$

As discussed in the previous section, the protons at equilibrium are slightly more likely to be aligned parallel to $\boldsymbol{B}$ and thus our magnetization vector is also parallel to $\boldsymbol{B}$. The protons are not static at equilibrium, but rather are precessing at the Larmor frequency, much in the manner of a spinning gyroscope suspended from a string. However, whereas the gyroscope's precession is due to gravity operating perpendicular to its spin axis, in MRI the torque is created by the spins aligning along the magnetic field direction. The interaction of the proton's spin and the magnetic field $\boldsymbol{B_0}$ produces the magnetic moment, which causes the precession of the proton about the z-axis. This precession is not evident in our magnetization vector, because the relative phases of our protons are equally distributed such that the vector sum of any transverse xy-components is zero.

When the spins are excited by an RF pulse at the Larmor frequency, they are "tipped" into the xy-plane. This RF magnetic field is usually denoted as $\boldsymbol{B_1}$, and the angle from the z-axis to the tipped magnetization vector is called the *flip angle*, and can be expressed as

$$\theta = \gamma B_1 \tau \tag{1.16}$$

where θ denotes the flip angle, B_1 is the magnitude of the RF magnetic field, and τ the time interval for which the RF pulse is turned "on". Since the value for γ is constant for ^{1}H protons, for a given flip angle we see that short values of τ require large values for B_1, which increase the specific absorption rate, or SAR values for a human patient. We will discuss SAR limits in a later section.

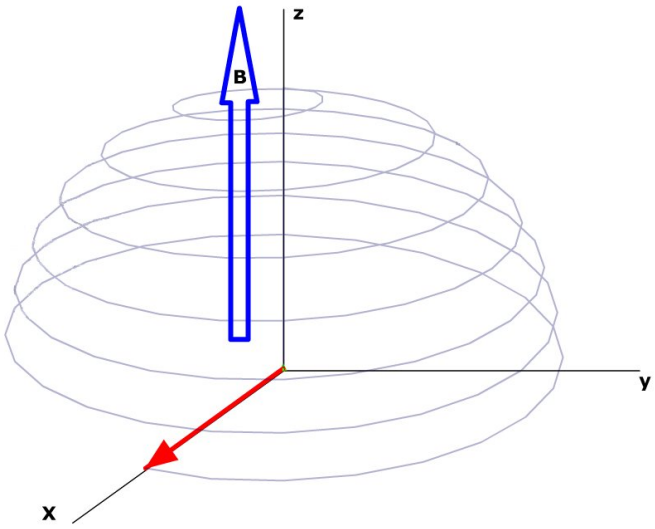

Figure 1.2: "Spiralling" of magnetization vector (red vector) toward the xy-plane in main magnetic field, B (blue).

1.4 T_1, T_2, T_2^* and the Bloch Equations

After the application of the RF pulse, the magnetization vector "spirals" down toward the xy-planefrom its equilibrium positive orientation along the z-axis (see Figure 1.2).

After the pulse is switched off, the magnetization vector relaxes towards its equilibrium aligned along the z-axis; however, this relaxation is actually the result of two separate relaxation components – a longitudinal relaxation along the z-axis, and a transverse relaxation process which occurs in the xy-plane.

Longitudinal relaxation, also called *spin-lattice* interaction, occurs as a result of individual proton spins re-aligning with the main magnetic field after being tipped into the transverse plane. It is shown in [9] that the rate of spin-lattice relaxation for the magnetization vector is proportional to the amount of magnetization that has already recovered – if M_0 is the initial magnitude of the vector before the application of the RF pulse, and M_z is the magnitude at a given time $t > 0$, then

$$\frac{dM_z}{dt} = \frac{M_0 - M_z}{T_1} \tag{1.17}$$

We see here the introduction of the longitudinal relaxation factor T_1, which governs the rate of regrowth of M_z. T_1 is measured in milliseconds, is determined experimentally, and varies from tissue to tissue (very short for bone, and much longer for CSF and water).

Transverse, or *spin-spin* relaxation occurs as a result of the various proton spins interacting with each other. When the magnetization vector is flipped into the transverse plain, the individual spins are all in phase, as they rotate through the xy-plane around the z-axis. Each proton is affected not only by the main magnetic field, but also by their interactions with their neighbours – hence the term spin-spin relaxation. We can model the *effective* magnetic field experienced by a given proton by

$$B_{\mathit{eff}} = B_0(1 - \sigma) \tag{1.18}$$

where σ represents the amount of magnetic shielding a proton experiences due to its neighbours. These interactions cause local field inhomogeneities, such that the protons are not precessing about the z-axis at the Larmor frequency, but rather (potentially) slower or faster. This has the effect of *dephasing* the xy-component of the magnetization vector, causing the vectors to "fan out" over time. This occurs because although all the magnetization vectors are aligned after the RF pulse, their individual deviations from the main magnetic field cause their precession to be either slower or faster than the ideal Larmor frequency. This transverse relaxation is governed the factor T_2, which, like T_1, is measured in milliseconds. T_2 is usually much smaller than T_1 for a given tissue.

In actual MRI experiments, there is a further dephasing factor introduced by B_0 inhomogeneities. We can classify these losses as a separate decay factor T_2', and we relate this factor to T_2 as:

$$\frac{1}{T_2^*} = \frac{1}{T_2} + \frac{1}{T_2'} \tag{1.19}$$

This defines the term T_2^*, which qualitatively is the combination of transverse relaxation due to local, random spin interactions and field inhomogeneities. With some pulse sequences (notably, spin-echo experiments), the T_2' effects can be recovered, leading to a total transverse relaxation factor approaching T_2.

If we define our transverse magnetization vector to be

$$\boldsymbol{M_T} = M_x\hat{x} + M_y\hat{y} \tag{1.20}$$

then from [9] and relating the derivative of angular spin momentum to magnetic moment and external magnetic field[5], we can find a differential equation for the transverse magnetization vector:

$$\frac{d\boldsymbol{M_T}}{dt} = \gamma \boldsymbol{M_T} \times \boldsymbol{B} - \frac{1}{T_2}\boldsymbol{M_T} \tag{1.21}$$

If we separate equation (1.21) into its components, we have, along with equation (1.17) the *Bloch differential equations* used to describe the magnetization vector in a constant external magnetic field:

$$\frac{dM_x}{dt} = \omega_0 M_y - \frac{M_x}{T_2} \tag{1.22}$$

$$\frac{dM_y}{dt} = -\omega_0 M_x - \frac{M_y}{T_2} \tag{1.23}$$

If we consider that the RF pulse is switched off at time $t = 0$, and that

$$\begin{aligned} M_z(\infty) &= M_0 \\ M_x(\infty) &= M_y(\infty) = 0 \end{aligned}$$

then the solution to the Bloch equations (1.17, 1.22, 1.23) becomes

$$M_x(t) = e^{-t/T_2}(M_x(0)\cos(\omega_0 t) + M_y(0)\sin(\omega_0 t)) \tag{1.24}$$

$$M_y(t) = e^{-t/T_2}(M_y(0)\cos(\omega_0 t) - M_x(0)\sin(\omega_0 t)) \tag{1.25}$$

$$M_z(t) = M_z(0)e^{-t/T_1} + M_0(1 - e^{-t/T_1}) \tag{1.26}$$

It is important to note that the above solutions are only applicable to the static field case (*ie.* RF pulse switched off), and do not take T_2^* effects or other field inhomogeneities into consideration.

[5] $\frac{d\boldsymbol{S}}{dt} = \boldsymbol{\mu} \times \boldsymbol{B}$. For a more in-depth discussion of torque in a magnetic field, consult a basic physics textbook.

1.5 Free Induction Decay

After the application of the RF pulse, the magnetization vector decays according to the Bloch equations and is governed by the two decay factors, T_1 and T_2. It is this decay which creates our signal, from which we can generate an image. According to Faraday's law, a voltage is generated in any coil through which a magnetic flux changes. We can describe this as:

$$V(t) = -\frac{d\Phi}{dt} \tag{1.27}$$

where the magnetic flux through the coil is a surface integral over the coil area

$$\Phi = \int_{coil\ area} \boldsymbol{B} \cdot d\boldsymbol{S} \tag{1.28}$$

We can write an expression for the signal in the receiver coil as

$$s(t) = \omega_0 \Lambda \int_V M_{T^*}(\boldsymbol{r}, t) \mathcal{B}_{T^*}(\boldsymbol{r})\ dr^3 \tag{1.29}$$

where the expression is a volume integral, evaluated at all locations where the net magnetization is non-zero. Λ is a constant representing the gain factors from the signal detection electronics, M_{T^*} is the complex representation of the magnetization factor in the transverse plane[6], and $\mathcal{B}_{T^*}$ is the complex "receive" field produced by the detection coil in the transverse plane[7]. For a thorough discussion of the derivation, consult [9].

We can see from (1.29) that our detected signal depends upon the Larmor frequency – this implies that the higher our main magnetic field (B_0), the more signal we can expect from our MRI experiment. We will discuss signal to noise calculations in a later section.

1.6 Gradients

It should be noted that the free induction decay (FID) in the receiver coil is due to the entire object being imaged. In order to spatially localize the FID, we can apply magnetic field *gradients* which perturb the Larmor frequency experienced by the proton spins [13]. In the simplest case, we apply a linear

[6] $M_{T^*} = M_x + iM_y$, and $M_{T^*}(\boldsymbol{r}, t) = e^{-t/T_2(\boldsymbol{r})} e^{-i\omega_0 t} M_{T^*}(\boldsymbol{r}, 0)$

[7] $\mathcal{B}_{T^*} = \mathcal{B}_x + i\mathcal{B}_y$

gradient along the z-axis, and find that the resulting magnetic field in the z-direction is

$$B_z(z,t) = B_0 + zG(t) \tag{1.30}$$

and the precessional frequency experienced by a given proton is

$$\omega(z,t) = \omega_0 + \gamma z G(t) \tag{1.31}$$

Now consider the simple case of trying to obtain a 1D image; what we term an "image" is actually the spin density of protons, which we define as $\rho(z)$ in the z-direction. If time $t = 0$ is when the RF pulse is applied to our sample, and we ignore any transverse relaxation effects, from [16], we can write the FID signal:

$$s(t) = \int \rho(z) e^{-i\gamma z \int_0^t G(t')dt'} dz \tag{1.32}$$

If we define $k(t)$ as $k(t) = \gamma/(2\pi) \int_0^t G(t')dt'$, we can then write (1.32) as

$$s(t) = \int \rho(z) e^{-i2\pi z k} dz \tag{1.33}$$

where $k = k(t)$ is our set of *spatial frequencies*. We can see that (1.33) is simply the Fourier transform of our spin density, $\rho(z)$:

$$s(k) = \mathcal{F}\{\rho(z)\} \tag{1.34}$$

which thus implies that our spin density function $\rho(z)$ (and therefore our resulting image) can be recovered with the *inverse Fourier transform* of our signal

$$\rho(z) = \mathcal{F}^{-1}\{s(k)\} \tag{1.35}$$

Thus $\rho(z)$ and $s(k)$ form a Fourier transform pair, and our sample $\rho(z)$ is said to be *frequency encoded* along the z-direction (although the usual methodology is to frequency encode along the x-axis).

Gradients, because they disturb the precessional frequencies of the spins, actively dephase the FID, not unlike T_2' effects (1.19). Gradient echo sequences seek to minimize the dephasing caused by the gradients themselves, by adding "rephasing" lobes in an effort to recapture lost signal. The signal will also dephase over time, thereby decreasing the resulting signal to noise ratio.

1.7 Sampling and Fourier Transform

The equation (1.35) in the previous section describes the resulting image if and only if the underlying signal $s(k)$ is continuous and complete in *k-space*. Qualitatively, k-space is frequency space, or data space of the image, and results as a consequence of the FID. Ideally, we would collect the data continuously across all of k-space, but this is impractical for two reasons – we do not have an infinite amount of time to collect the data, and because of relaxation effects the FID will decay to zero in a finite period. Thus, we sample the resulting FID with an analog to digital converter, thereby collecting *discrete* data points with which to reconstruct our image. We might imagine that collecting more data samples results in a "better" image – this is correct, but this also increases our scanning time, which (especially in fast scanning) we wish to minimize as much as possible. The optimization problem of acquiring what data, and in which order is referred to as a "k-space trajectory", and is the focus of many papers in the literature ([1] [2] [6] [8] [12] for example).

We would like to minimize the sampling time, but still acquire enough samples in k-space for "sufficient" coverage – that is, be able to reconstruct an image with little or no artifacts with the available k-space data. The process of sampling can be defined as a sum of delta functions in k-space, and we specify this as

$$u(k) = \Delta k \sum_{p=-\infty}^{\infty} \delta(k - p\Delta k) \tag{1.36}$$

where Δk is the difference between successive sampling points in k-space, This operation is also known as a "comb" or "sampling" function in the frequency domain. As might be guessed from the limits of the summation, (1.36) represents an infinite sampling function, and the resulting sampled signal when combined with the actual k-space signal $s(k)$ is

$$\begin{aligned} s_\infty(k) &= s(k)u(k) \\ &= \Delta k \sum_{p=-\infty}^{\infty} s(p\Delta k)\delta(k - p\Delta k) \end{aligned} \tag{1.37}$$

where the sampled signals are given by $s(p\Delta k)$. If we then take the inverse Fourier transform of (1.37) in order to obtain a 1D image, we find

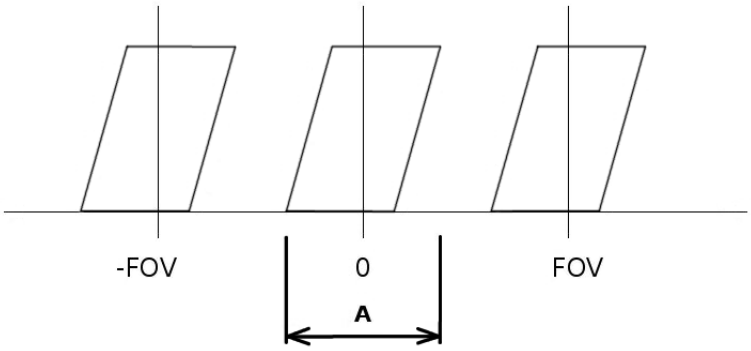

Figure 1.3: 1D object of length A repeated at intervals of $1/\Delta k$ or the FOV.

$$\hat{\rho}(x) = \Delta k \sum_{p=-\infty}^{\infty} s(p\Delta k) e^{i2\pi p \Delta k x} \tag{1.38}$$

where $\hat{\rho}(x)$ represents the image *derived from our process of infinitely sampling the frequency domain*, in order to differentiate it from the actual image, $\rho(x)$. Examining (1.38), we can see that $\hat{p}(x)$ will be periodic where $e^{i2\pi p\Delta k x}$ takes on identical values, which in this case will be for values of $x' = x + 1/\Delta k$, so that

$$\hat{\rho}(x) = \hat{\rho}(x + 1/\Delta k) \tag{1.39}$$

The inverse of the spacing of data samples in k-space, or $1/\Delta k$ represents the *Field of View*, or FOV. We can describe the FOV as the interval over which the reconstructed image $\hat{\rho}(x)$ repeats itself. (see Figure 1.3).

For simplicity, if we consider an object in one dimension (x-direction), whose length along x is given as A, we would like the FOV to be larger, such that

$$FOV > A \quad or \quad \Delta k < 1/A \tag{1.40}$$

This represents the *Nyquist criterion*, and specifies the theoretically maximum sampling spacing, Δk, we can have before we get an artifact from under-sampling, usually called *aliasing*. Aliasing manifests itself as image "wrap-around", as part of an image overlaps with another because the FOV is less than the length in the x-direction. Although undesirable, we tolerate some aliasing in an effort to decrease scanning time – in fact, some imaging techniques (such as SENSE [23]) use aliasing and "un-roll" the complete

image via post imaging processing.

In order to remove the multiple images which are described by (1.39), we can pass our single $s(k)$ through a "windowing" function or filter. We can model this truncation by the use of the *rect* function, and thus equation (1.37) becomes

$$\begin{aligned} s_m(k) &= s(k)\, u(k)\, rect\left(\frac{k+\frac{1}{2}\Delta k}{2n\Delta k}\right) \\ &= \Delta k \sum_{p=-n}^{n-1} s(p\Delta k)\delta(k-p\Delta k) \end{aligned} \tag{1.41}$$

where the total number of sampled points is no longer infinite, but is equal to $2n$. Our reconstructed image then becomes

$$\hat{\rho}_m(x) = \Delta k \sum_{p=-n}^{n-1} s(p\Delta k) e^{i2\pi p\Delta k x} \tag{1.42}$$

and we can see that (1.39) still holds, and so too the Nyquist criterion.

Examining (1.42) we can see that it resembles the form of the *Discrete Fourier Transform*

$$\begin{aligned} g\left(\frac{qL}{2n}\right) \equiv \mathcal{D}^{-1}(G) &= \frac{1}{2n} \sum_{p=-n}^{n-1} G\left(\frac{p}{L}\right) e^{i2\pi pq/2n} \\ &\quad where\ q = -n, -n+1, \ldots, n-2, n-1 \end{aligned} \tag{1.43}$$

If we sample along the x-direction at a discrete number of points (instead of continuously), we see that we can write $x = q\Delta x$, where Δx is the distance between successive points. Using this, the fact that $\hat{\rho}(x)$ and $s(p\Delta k)$ are a Fourier transform pair, and making the substitutions $1/L = \Delta k$ and $L = 2n\Delta x$ in (1.42) we find a form more easily applicable to our MR imaging equation:

$$\hat{\rho}_m(q\Delta x)\Delta x = \frac{1}{2n} \sum_{p=-n}^{n-1} s(p\Delta k) e^{i\pi pq/n} \tag{1.44}$$

where $q = -n, -n+1, \ldots, n-2, n-1$ as before. What this indicates is that the inverse discrete Fourier transform of our sampled k-space data is equal to

our spin density (at discrete points) times the distance between points in the x-direction (although we can easily expand (1.44) into three dimensions, such that the Δx term represents the voxel volume). (1.44) represents our goal in MRI, and describes the general principle behind the imaging technique.

Theoretically, we can choose any number of points N, such that $N = 2n$. However, for practical purposes we would like to be able to use a fast Fourier transform (FFT) algorithm in order to perform the above transformation, and as such we choose N such that $N = 2^m$, where m is some positive integer.

1.8 k-space

If we extend our discussion in the previous section to more than the x-direction, such that we have a vector quantity $\boldsymbol{r} = (x, y)$ for a 2D image, or $\boldsymbol{r} = (x, y, z)$ for three dimensions, we can then write the result of our previous section as $\hat{\rho}(\boldsymbol{r})$. This is a complex quantity which arises out of our imaging because of the global and local phase shifts in the process due to magnet inhomogeneity, gradient non-linearity *etc.*, and there is a local and global phase error inherent in our calculated proton density. Thus we find

$$\hat{\rho}(\boldsymbol{r}) = \rho(\boldsymbol{r})e^{i\phi} \tag{1.45}$$

where ϕ represents our phase error. We can overcome the deviations caused by the phase error in the real part of our acquired image $\hat{\rho}(\boldsymbol{r})$ by taking the magnitude image, or

$$\|\hat{\rho}(\boldsymbol{r})\| = \sqrt{\Re[\hat{\rho}(\boldsymbol{r})]^2 + \Im[\hat{\rho}(\boldsymbol{r})]^2} \tag{1.46}$$

This is the most common technique in clinical use, and is usually the "image" referred in MRI. However, the phase image provides additional information with each pixel proportional to the calculated local phase value, and some imaging methodologies use this information (*ie.* thermometry and susceptibility weighted imaging). We can obtain $\phi(\boldsymbol{r})$ from

$$\phi(\boldsymbol{r}) = \arctan\left(\frac{\Im[\rho(\boldsymbol{r})]}{\Re[\rho(\boldsymbol{r})]}\right) \tag{1.47}$$

Because arctan is periodic over multiples of 2π, the phase image is also periodic over this interval.

As a consequence of $\rho(\boldsymbol{r})$ being real, we can use the fact that the negative portion of k-space is simply the complex conjugate of the positive portion, or

$$\hat{\rho}(-\boldsymbol{r}) = \hat{\rho}^*(\boldsymbol{r}) \tag{1.48}$$

We can use this relation to decrease our acquisition time, since we theoretically only need to acquire half of k-space, and then easily derive the rest. However, this technique results in a loss in signal to noise ratio (see Section 1.12) by a factor of $\sqrt{2}$, because of the noise inherent in the signal. When we use conjugate symmetry, we are including this noise in our derived data point. Of course, there would still be noise if we actually acquire the data – but because noise is random, it is more likely to cancel out and thus impact our signal less.

One useful characteristic of k-space is in regard to low versus high spatial frequencies. The lower spatial frequencies are necessary for slow spatially changing parts of the imaged object, while for finer detail and smaller structures, the higher spatial frequencies are required. Thus the overall contrast and shape of the object is due to the low spatial frequencies, or centre of k-space, while the higher frequencies, or periphery of k-space, provides edge detection of the object. Because of these facts, the centre is k-space is generally more valuable in terms of image reconstruction, and the centre of k-space provides the best value for sampling time. Many k-space trajectories acquire the centre of k-space first, when the signal is strongest, and acquire the periphery of k-space later for the same FID echo.

1.9 Pulse Sequences

Pulse sequences can be separated into two methodologies – 2D and 3D sequences. 2D sequences use a slice selection gradient (usually in the z-direction) at the same time as a selective RF pulse to excite a "slice" of protons. Once excited, the k_y and k_x (often referred to as "phase encode" and "frequency encode" respectively) gradients are used to navigate through k-space and collect the data (see Figure 1.4). In comparison, 3D sequences use a volumetric RF pulse (sometimes with no slice-select – see explanation of Durga below), and then the z gradient acts as a second phase encoding to move through k-space. They require three-dimensional FFT's to obtain an image, and have some advantages over a multi-slice 2D approach:

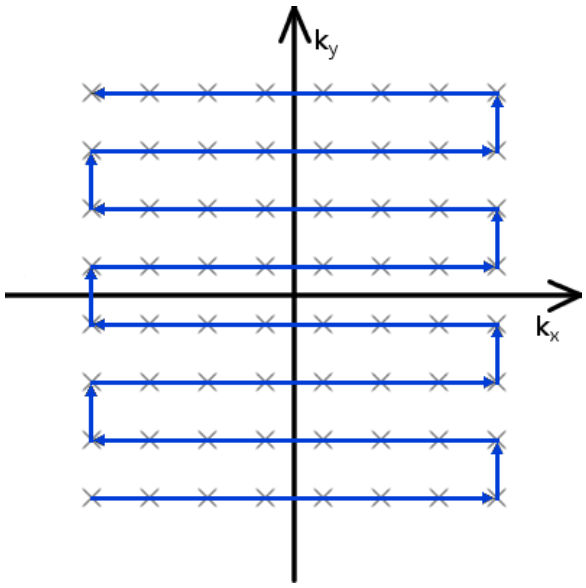

Figure 1.4: k-space trajectory on a Cartesian grid, samples (denoted by 'X') acquired in both k_x and k_y directions.

- ability to change slice thickness with Δz, as opposed to multi-slice 2D, which relies on RF amplitude and duration
- "gaps" between slices are necessary in multi-slice 2D to prevent RF leakage – these are not required in 3D
- reducing RF duration results in a large bandwidth, thereby exciting a large volume
- short TE and potentially fine resolution can be used to reduce T_2^* dephasing
- potential for superior SNR, albeit at increased imaging times

Most pulse sequences collect k-space data in a Cartesian grid. These Cartesian k-space trajectories acquire equidistant points in Fourier space, so that for 2D imaging Δk_x and Δk_y are constant throughout the sequence. 3D imaging on a Cartesian grid has the Δk_z component acting as the second phase encoding gradient.

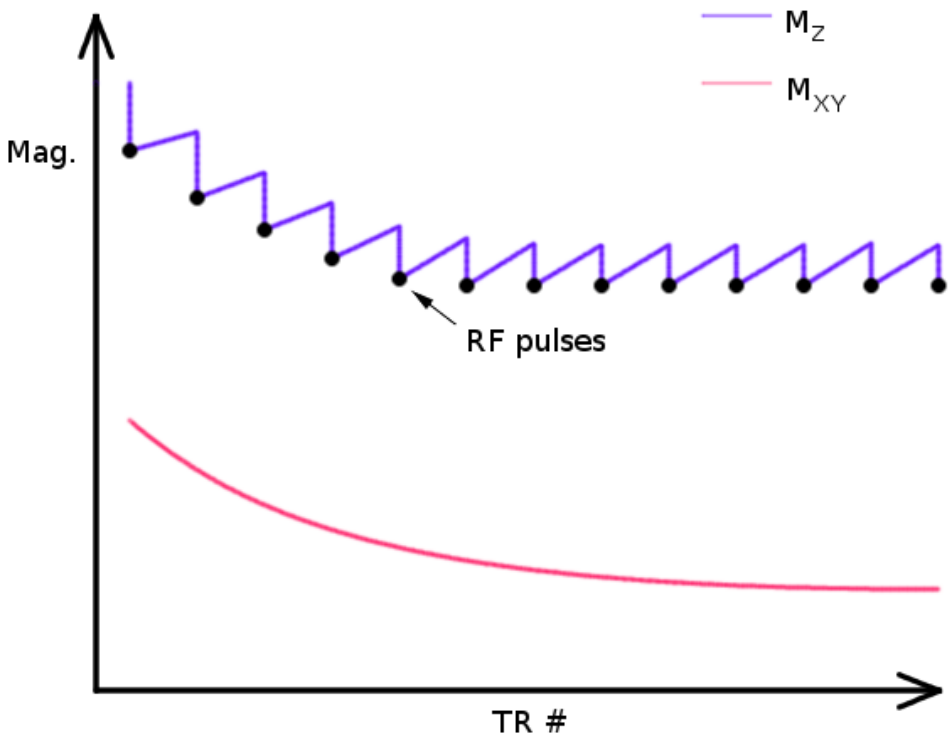

Figure 1.5: Steady-state imaging. Magnetization level on vertical axis, increasing TR count on bottom axis.

1.9.1 Steady-state Imaging

Steady-state imaging is the result of using shorter TRs and partial flip angles to leave the resulting FID in a steady-state condition, such that the amount of longitudinal magnetization recovered by the sequence becomes equal to the amount tipped into the transverse plane. Phase shifts must remain constant, and the resulting image contrast is determined by a complex, non-linear interaction between T_1 and T_2. Spoiled steady-state imaging uses crusher gradients or phase cycled RF pulses to eliminate or reduce the transverse magnetization between pulses, such that the image contrast would essentially be determined solely upon T_1.

The optimal flip angle, or *Ernst angle* is the flip angle which results in the largest FID in steady-state imaging. This is determined by:

$$\cos \alpha_E = e^{-TR/T_1} \tag{1.49}$$

1.10 Gradient Moments

Gradient moments, and specifically gradient moment nulling, are useful tools in analyzing the effects of gradients in different pulse sequences. Gradient moments are defined in MRI in a similar fashion to their use in mechanics or physics, such that

$$m_n(t) = \int_0^t G(u)u^n \, du \tag{1.50}$$

where u is the integration variable in time. We can see from (1.50) that the zeroth, or m_0 moment, is simply the area under the gradient for single point of interest in time. Most pulse sequences have the m_0 moment nulled, since this nulls any phase angles static spins acquire while a gradient is on.

If we consider *moving* spins, such as those in a constant velocity peripheral vein or artery, we can see a moving spin will acquire more phase angle (assuming that the gradient is positive) than a static spin. This occurs because the spins experience a changing magnetic field as they move. We can see the effect of this in Figure 1.6.

It is obvious that in this case simply applying a negative lobe, equal in area to the positive lobe which created the phase angle, will not null the moving spins as they would the static spins. In order to null both spin types, we must null the m_1 and m_0 gradient moments as demonstrated by a simple gradient sequence shown in Figure 1.7.

First order nulling is often called velocity compensation or flow compensation, since it effectively nulls constant velocity spins. Higher order moment nulling (*ie.* acceleration compensation for second order moments) techniques also exist; however, these are rarely implemented because of increasing complexity and increasing the minimum TE.

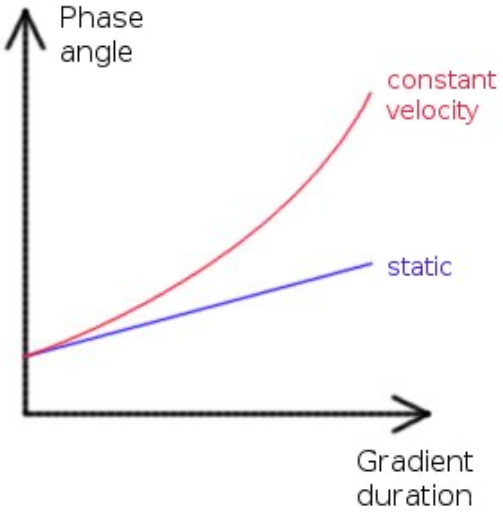

Figure 1.6: Phase angle accumulation comparison of static and moving spins in a constant gradient field.

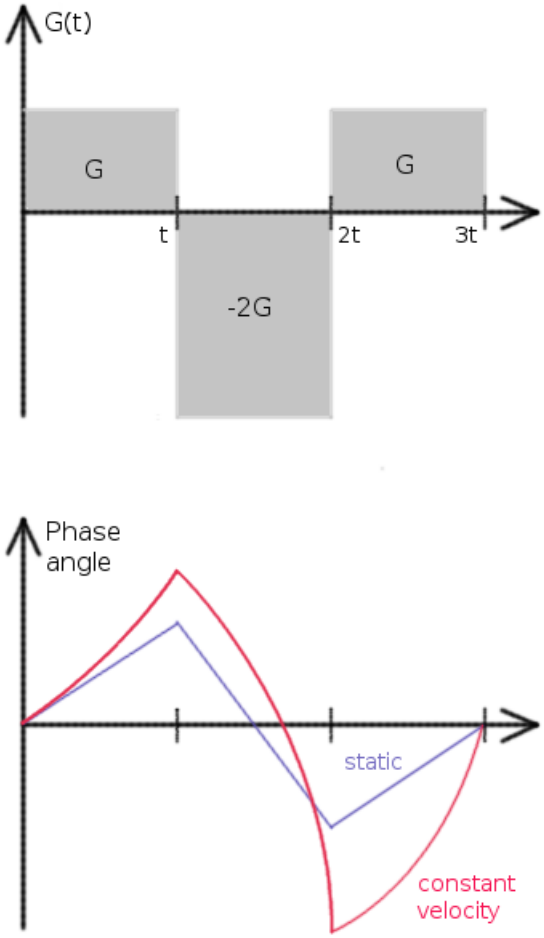

Figure 1.7: First gradient moment nulling pulse sequence and accumulated phase angle.

1.11 non-Cartesian k-space Trajectories

Expanding our arguments to 2D k-space we define a vector $\boldsymbol{k}$ such that

$$\boldsymbol{k} = k_x\hat{x} + k_y\hat{y} \tag{1.51}$$

Although in MR imaging the Cartesian grid is still the most common methodology, non-Cartesian algorithms have been demonstrated to possess several advantages. Ahn, Kim and Cho [1] proposed a spiral trajectory, which begins at the centre of k-space (low frequencies) and samples discrete points along a spiral arm. If the set of discrete points in k-space is expressed in polar coordinates, such that $k_r = \sqrt{k_x^2 + k_y^2}$ and $k_\theta = \arctan[k_y/k_x]$, we can define the spacing between successive arms (Δk_r) to be constant, and each point separated by a constant angle (Δk_θ).

However, spiral and other non-Cartesian strategies suffer from the fact that the sampling points in k-space do not fall on a regular Cartesian grid, so the image cannot be obtained by a direct application of the FFT. Although different methods exist for attaining an image from non-Cartesian k-space data (including a non-uniform FFT [25]), the most common method is the use of a regridding kernel to transform the data to lie on a grid. Then the FFT can be used to resolve an image in the usual way. Because of the non-uniform sampling density for spiral trajectories, we apply a *density compensation function* (DCF) to the k-space data. This has the purpose of minimizing reconstruction errors, particularly in the centre of k-space where the sampling density is high.

We can express the full gridding as follows – if $s(\boldsymbol{k})$ is our signal, $u(\boldsymbol{k})$ our sampling or "comb" function, $w(\boldsymbol{k})$ the DCF, $C(\boldsymbol{k})$ the regridding kernel, and $r(\boldsymbol{k})$ a function that defines a Cartesian grid[8], then

$$s_c(\boldsymbol{k}) = (((s(\boldsymbol{k}) \cdot u(\boldsymbol{k}) \cdot w(\boldsymbol{k})) * C(\boldsymbol{k})) \cdot r(\boldsymbol{k})) *^{-1} C(\boldsymbol{k}) \tag{1.52}$$

where $s_c(\boldsymbol{k})$ is the regridded k-space data, and $*^{-1}$ is the inverse convolution operation. While many methodologies exist for the DCF (including numerical methods [21]) a common form combines the sampling and regridding functions

$$w(\boldsymbol{k}) = \frac{u(\boldsymbol{k})}{u(\boldsymbol{k}) * C(\boldsymbol{k})} \tag{1.53}$$

[8] $r(\boldsymbol{k}) = \sum_i \sum_j \delta(k_x - i, k_y - j)$

As for the regridding kernel $C(\boldsymbol{k})$, the most common technique is to use the analytic Kaiser-Bessel function, although considerable literature exists in this area [11] [3]. By applying the inverse Fourier transform to (1.52), we obtain our complex image $\hat{\rho}(\boldsymbol{r})$:

$$\hat{\rho}(\boldsymbol{r}) = \mathcal{F}^{-1}\left\{((s(\boldsymbol{k}) \cdot u(\boldsymbol{k}) \cdot w(\boldsymbol{k})) * C(\boldsymbol{k})) \cdot r(\boldsymbol{k})\right\} \cdot \frac{1}{c(\boldsymbol{r})} \tag{1.54}$$

where the final term in (1.54) is the *deapodization* term, and is used to remove the effect of the convolution kernel in the image. We denote the Fourier transform of the convolution kernel $C(\boldsymbol{k})$ as $c(\boldsymbol{r})$.

The k-space trajectory, sampling, and truncation operations, in addition to any density compensation or regridding required for non-Cartesian modalities, have the effect of a filter on the sampled k-space data. If the k-space data is $s(\boldsymbol{k})$, and our object is described by $\rho(\boldsymbol{r})$, we can describe the cumulative effect of our k-space operations as $H(\boldsymbol{k})$, and write

$$\begin{aligned} \hat{\rho}(\boldsymbol{r}) &= \mathcal{F}^{-1}\{s(\boldsymbol{k})H(\boldsymbol{k})\} \\ &= \rho(\boldsymbol{r}) * h(\boldsymbol{r}) \end{aligned} \tag{1.55}$$

The function $h(\boldsymbol{r})$ is called the *point spread function* (psf) of our imaging operations, and it can be easily seen that $\hat{\rho}(\boldsymbol{r}) = h(\boldsymbol{r})$ if we are imaging a delta function. The psf is a useful metric for comparing different MR imaging techniques, with the theoretical ideal being a delta function (*ie.* k-space operations have no effect on the produced image, such that $\hat{\rho}(\boldsymbol{r}) = \rho(\boldsymbol{r})$). Of course this is impossible in reality – widening of the central peak of a psf results in blurring in the resultant image, and regular structures outside the main peak (*ie.* rings) can result in aliasing in the final image. We can identify the image resolution from the width of the centre peak of the psf, as the full-width at half maximum (FWHM).

1.12 SNR, CNR, SAR Calculations

MR images contain noise, both systematic and random. With regards to Durga, we have noise from both effects, particularly since we are actively under-sampling k-space in order to decrease the scan time, with the trade-off being an increase in random noise. We begin our analysis by stating

that white noise has a frequency spectrum that is evenly distributed and uncorrelated. If our true signal is $s(\boldsymbol{k})$, and a white noise factor $\epsilon(\boldsymbol{k})$ is added to give our measured signal, then

$$s_m(\boldsymbol{k}) = s(\boldsymbol{k}) + \epsilon(\boldsymbol{k}) \tag{1.56}$$

We can describe the factor $\epsilon(\boldsymbol{k})$ as being Gaussian or normally distributed by the central limit theorem, since it is sum of a large number of independent factors, with an equal probability of $\epsilon(\boldsymbol{k})$ being either positive or negative. Thus we can say that our Gaussian white noise has a mean of zero and a variance of σ_m^2. From [9] we can write the standard deviation of the image domain of k-space as

$$\sigma_{image}(p\Delta x) = \sqrt{\frac{\sigma_m}{N}} \tag{1.57}$$

where N is the number of samples. Note that (1.57) is only for one dimension, but we can easily expand that to multiple directions, with N_x, N_y, and N_z representing the number of samples taken in the x, y, and z directions respectively.

Intuitively, we might imagine that the signal for any given voxel would be dependent on the volume size – the larger the voxel, the more precessing protons contained in that volume, and hence a larger number of spins available for the FID signal. Formally given in [9], we can write

$$signal \propto \Delta x \Delta y \Delta z \tag{1.58}$$

A useful metric for evaluating the quality of an image is the signal to noise ratio, or SNR. By unifying (1.57) and (1.58), and noting that $noise \propto \sqrt{BW_{readout}}$, we can write:

$$SNR/voxel \propto \frac{\Delta x \Delta y \Delta z}{\sqrt{\frac{BW_{readout}}{N_x N_y N_z}}} \tag{1.59}$$

From (1.29), the signal is proportional to ω_0, and thus from (1.10), B_0. However, also from (1.29) we also see that the signal is dependent on the transverse magnetization vector, which is also proportional to ω_0. Thus we can write

$$SNR \propto {B_0}^2 \tag{1.60}$$

However, this simplistic case is not completely accurate in the general sense. Consult [9] for a more thorough discussion, although for the remainder of this paper, it is assumed that (1.60) holds.

A simple definition for SNR is:

$$SNR_{\text{AB}} = S_A/\sigma_{\text{image}} \tag{1.61}$$

This is commonly used to define SNR over a region of interest, and is the method used in this paper. Another important metric is the contrast-to-noise ratio, or CNR. Contrast allows us to differentiate tissues (*ie.* diseased versus healthy tissue) – excellent SNR is not useful from a clinical perspective if the contrast between different tissues is poor. If we define the contrast between two tissues A and B as their difference in signals, or $C_{AB} = S_A - S_B$, then the CNR becomes:

$$\begin{aligned} CNR_{\text{AB}} &= \frac{S_A - S_B}{\sigma_{\text{image}}} \\ &= SNR_A - SNR_B \end{aligned} \tag{1.62}$$

There are a variety of techniques used in clinical MRI to enhance the contrast between tissues, such as the use of contrast agents (*ie.* gadolinium compounds), selection of TE and TR imaging parameters, MR techniques (*ie.* fat saturation), *etc.* The expressions given above for SNR and CNR can be used for both magnitude and complex images, but in this paper only magnitude images will be used.

Specific Absorption Rate, or SAR, is a measure of the amount of heat accrued in a patient by the deposition of energy from RF sources, and it is measured in W/kg. There are strict limits for both average and peak SAR allowed during a scan – the FDA limits are $3W/kg$ and $4W/kg$ respectively. Estimates or measurements of SAR are quite complicated, since they involve a variety of factors: Larmor frequency, type of RF pulse (shape and flip angle), TR, type of coil, volume of tissue imaged, anatomical configuration *etc.* However, from [20] we present a proportionality

$$SAR \propto B_0^2 \alpha^2 d \tag{1.63}$$

where α is the flip angle of the RF pulse, and d is the duty cycle of the pulse (ratio of the average to the peak of the pulse). Although SAR estimation is complicated by the factors listed above, from [20] we can use

$$SAR = \|\boldsymbol{E}\|^2 \frac{\sigma}{2\rho} \frac{\tau N_P N_S}{TR} \tag{1.64}$$

where $\boldsymbol{E}$ is the electric field, σ and ρ are the tissue dependencies (representing the conductivity and density respectively), τ the pulse duration, and N_P and N_S the number of pulses and number of slices in the particular pulse sequence.

Chapter 2

Methods and Materials

2.1 MR hardware

A 3.0T MRI scanner (GE Healthcare®, Milwaukee, WI) shown in Figure 2.1 was used in developing the Durga pulse sequence. It is located in the Imaging Research Centre at St. Joseph's Healthcare, Hamilton, Ontario, Canada. Relevant statistics and capabilities are related in Table 2.1. Owing to the complexity of a multi-coil SENSE-like reconstruction, a single-channel coil was used in these experiments.

2.2 Durga Pulse Sequence

Anand *et al* [2] have formulated a new k-space sampling scheme entitled Durga, which uses asymmetric pseudo-random trajectories (see Figure 2.3) formulated by the optimization of second order cone problems. Durga under-samples k-space, less than the Nyquist criterion; however, this does not result in aliasing, since random trajectories tend to produce non-coherent artifacts (*ie.* noise) instead of aliased images [19]. Some of the key features of this pulse sequence design are: (for a complete review, consult [2])

- incorporation of gradient waveform constraints (peak, slew rate) in the trajectory design
- 3D, volumetric modality

Magnetic Field Strength (T)	3.0
Water Centre Frequency (Hz)	127 799 074
Maximum FOV (cm)	60
Shimmed area (cm)	45×48
Gradient Rise Time (μs)	268
Gradient Max Amplitude (mT/m)	40
Gradient Slew Rate (T/m/s)	150
Sampling Bandwidth (kHz)	500
Software OS Version	12M5

Table 2.1: MR hardware parameters of GE Signa Excite 3.0T

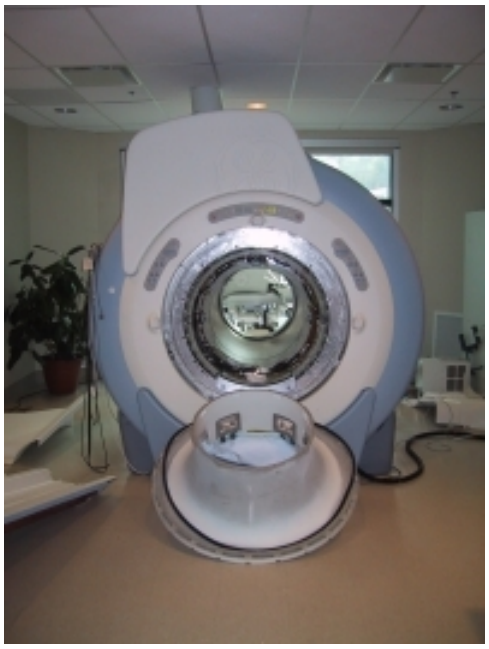

Figure 2.1: 3.0T scanner located in the Imaging Research Centre

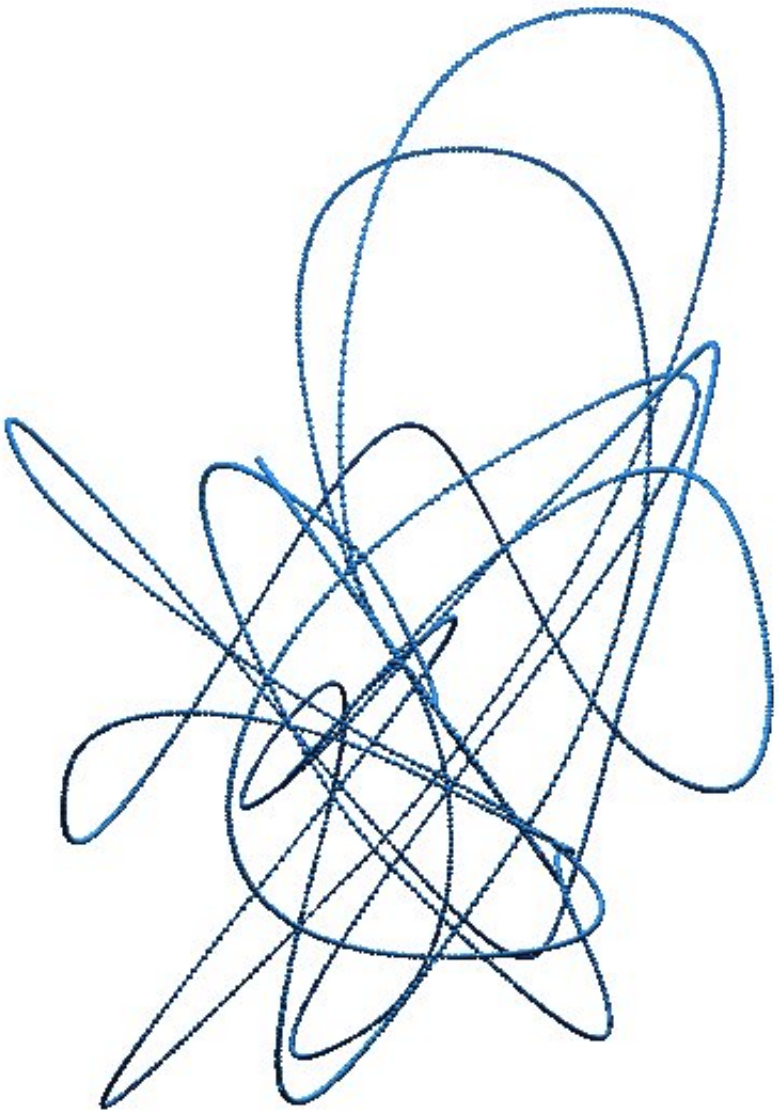

Figure 2.2: Single Durga trajectory – length 12.7 ms.

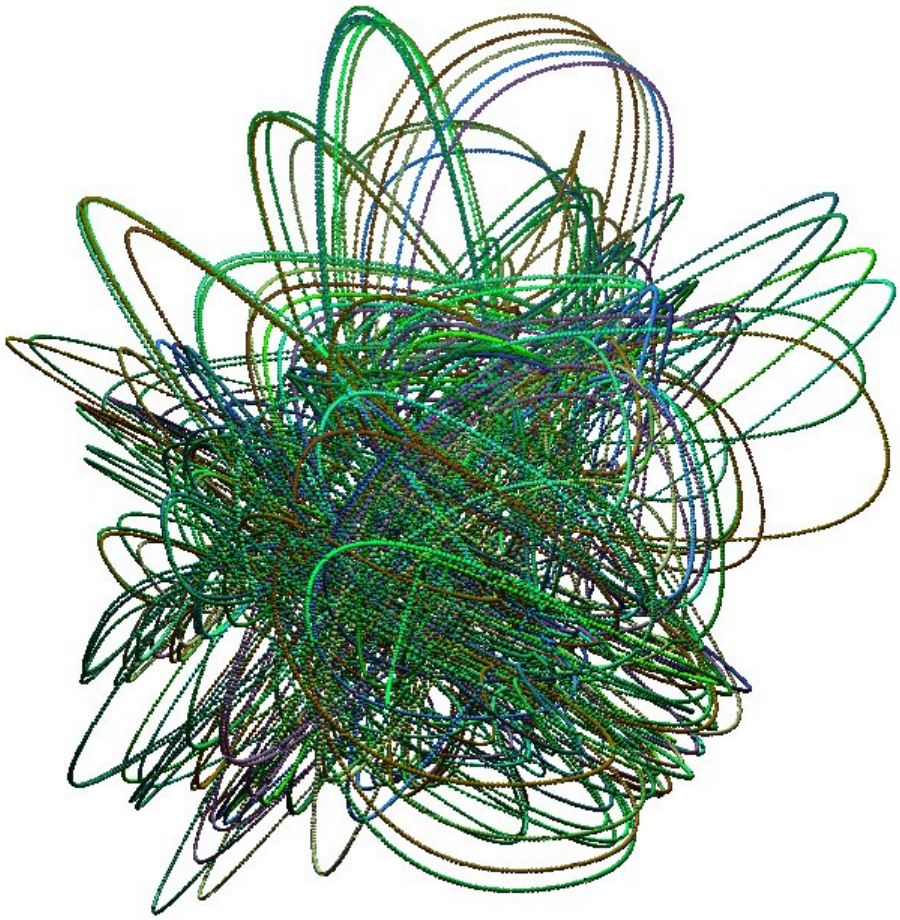

Figure 2.3: Subset of Durga trajectories. Colour added for viewing ease.

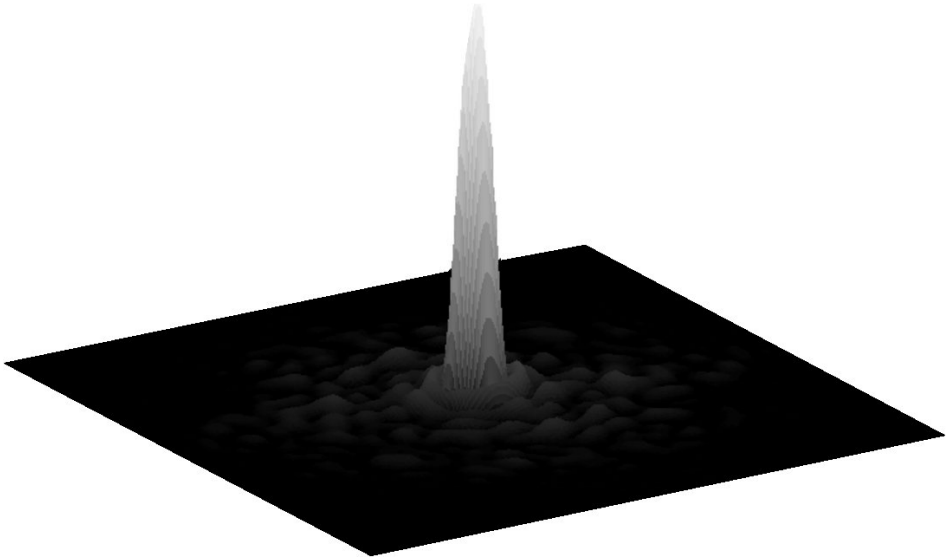

Figure 2.4: Durga point spread function.

- elimination of the slice rewinders and integration of first-moment nulling in the readout gradients resulting in increased sampling efficiency
- randomness of trajectories means that the Nyquist criteria is not applicable, so that under-sampling appears as noise – in fact, we can trade off noise for reduced scan time
- point spread function performance is excellent, with a FWHM value of 1 mm, a slight aliasing ring at 8% of the peak height, plus other smaller peaks which will appear as noise (see Figure 2.4)
- excellent frame rate performance (five-fold decrease in scan time) when compared to other fast sequences for dynamic imaging (blood flow, cardiac imaging)

Durga is inherently sensitive to magnet inhomogeneities and gradient eddy currents since it is designed as a steady-state sequence. These can combine to disrupt the steady-state system, resulting in increased artifacts, decreased SNR, loss of contrast and overall loss of image quality.

Gradient Parameter	Set A	Set B	Set C
TR	16.7 ms	16.7 ms	16.7 ms
Gradient Points / Trajectory	1210	2068	3176
Sampling Points / Trajectory	2420	4136	6352
Sampling Time / Trajectory	4.84 ms	8.272 ms	12.704 ms
Sampling Efficiency (compared to TR)	29.0 %	49.5 %	76.1 %
Number of Trajectories	525	296	177
Total Scan Time	8.77 s	4.94 s	2.96 s

Table 2.2: Durga Gradient Set details

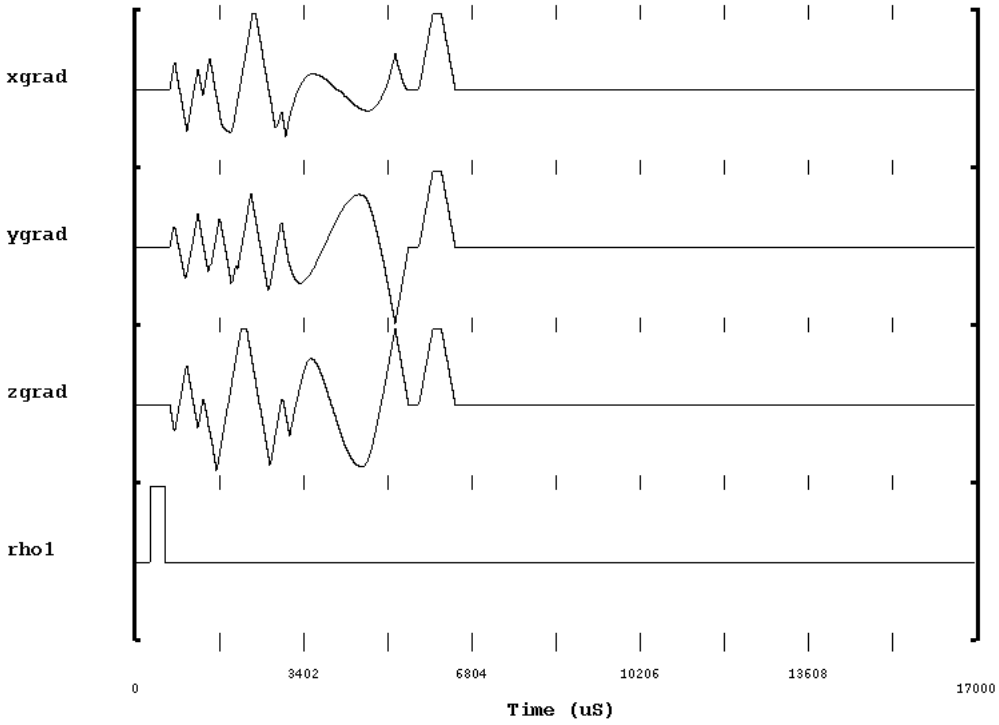

Figure 2.5: Durga pulse diagram Set A for one TR (16.7 ms)

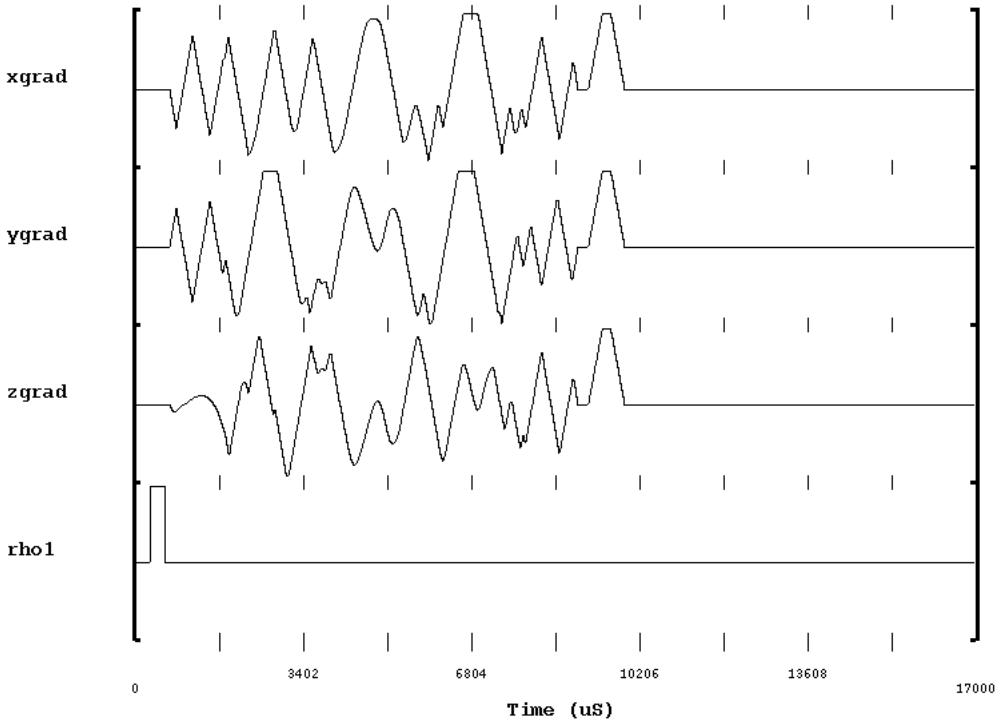

Figure 2.6: Durga pulse diagram Set B for one TR (16.7 ms)

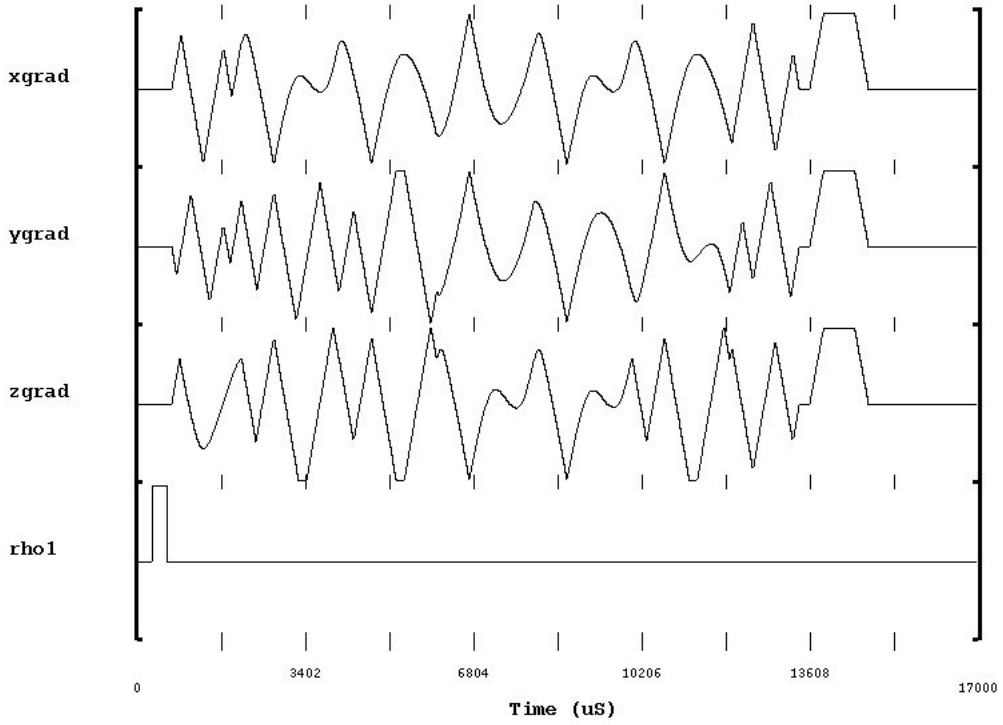

Figure 2.7: Durga pulse diagram Set C for one TR (16.7 ms)

Any given set of trajectories is the result of only one invocation of the Durga algorithm; this fact allows Durga to be tuned to particular applications as required. Three different gradient sets were designed for these experiments, all of which are spoiled, steady-state sequences. The details of the different sets (hereafter referred to as *Gradient Sets A, B, and C*) are found in Table 2.2. All sets are designed such that they are to be run sequentially, each trajectory beginning and ending at $\mathbf{k} = 0$. The implementations use a short, hard RF pulse (300 μs) for excitation, random crusher gradients at the end of each trajectory to disperse the transverse magnetization, and a fixed number of disabled acquisitions at the beginning of the sequence to "build-up" to steady-state conditions before sampling begins. The reconstructions in the paper were to a resolution of 256×256, resulting in Durga acquiring about 18% of the total k-space data available, or a speed-up factor of five. See Figures 2.5, 2.6 and 2.7 for the pulse sequence diagrams implemented on the GE scanner.

2.3 Software

There are four distinct software stages required to obtain Durga images from the GE scanner:

1. create GE compatible gradient files from files output by the Durga trajectory design algorithm
2. run the pulse sequence on the physical hardware and collect the k-space data
3. process the raw data and create the k-space trajectory information, by presenting the data in a format to be read by the reconstruction algorithm
4. reconstruct the data

Parts 1 and 3 are handled by Perl scripts, and their details and the code is given in Appendix A. For 2 and 4, please see sections 2.3.1 and 2.3.2.

2.3.1 EPIC

Creating pulse sequences for GE scanners requires the use of their software package *Environment for Pulse programming In C* (EPIC). EPIC is a proprietary system, used in the Imaging Research Centre under license – as such, details of the software or code listings is prohibited. However, what can be said about EPIC is that it is essentially a C programming environment with additional macros, preprocessing and libraries packaged together. The result is that EPIC is not very well documented, not completely flexible, and is a trial to understand [26]. It works best when making small changes to the stock sequences which are written for use with the scanner.

Durga, with its very short TR and non-uniform trajectories is unlike any clinical sequence, and this necessitated writing the pulse sequence from scratch. This presented unique challenges – while this methodology avoided extraneous code which may have broken the sequence or had unintended consequences, it did require a more thorough understanding what is, and is not, necessary in creating a working GE pulse sequence. For further discussion of these challenges, see Section 2.4.

In order to run the Durga sequence on the scanner, gradient values for the trajectories needed to be read in by the sequence. The memory requirements for all three gradient boards cannot exceed 4 MB due to hardware limitations. This limits the size of each individual gradient file to be 1.3 MB, with some slack being left for overhead in the scanner software. Thus, the maximum number of gradient points that can be run by the scanner is 650000, although this number can be distributed among any number of trajectories (as long as each trajectory is equal in length). More details about the format of these files can be found in Appendix A.

Since an off-line data reconstruction scheme was used, the pulse sequence was designed to instruct the scanner to save the raw k-space data (or "P-File" in GE parlance) sequentially – that is, save the data in the same order as it was acquired. The sequence was created such that the trajectories were played out in sequence, acquiring a complex data point every 2 μs. As the gradients were designed at a resolution of 4 μs, a simple linear interpolation was used to generate k-space trajectory information at 2 μs. Each line of sampling along a trajectory produced (in GE terms) one *frame* of data, and the *number of frames* was equal to the number of trajectories.

For Set C and using the data from Table 2.2, we see that the k-space data from one scan produced 177 frames, with 6352 complex data points /

Parameter Description	Value	Variable
Grid Dimension	256	N
Regrid Matrix Size	$256 \times 256 \times 256$	A
Data Frame Size	2420, 4136, or 6352 $\Rightarrow 2 \times$ gradient points	frsize
Number of Data Frames	525, 296, or 177 $\Rightarrow$ number of trajectories	nframes
k-space Spatial Frequency Limits	± 0.5	kmin
Δk	$\frac{\lvert 2(kmin)\rvert}{N-1}$	deltak
Kaiser-Bessel kernel Beta	6	beta
Kaiser-Bessel kernel Window	4	Wn

Table 2.3: Reconstruction parameters

frame. Each complex point is arranged as a $[real, imaginary]$ pair in the P-File. With 2 bytes / value, we see that the raw data size for a single scan was: $177 \times 6352 \times 4$ bytes = 4497216 bytes. For a multiple-coil scan, the data size would be the number of coils $\times$ 4497216 bytes.

Using the same methodology, Set A gradients produce $525 \times 2420 \times 4$ bytes = 5082000 bytes for a single coil scan, and Set B data is 4897024 bytes for the one coil configuration.

2.3.2 Reconstruction

Imaging with Durga on the scanner in the Imaging Research Centre entails three major parts – the setup and running of the actual sequence, processing the gradient input files and k-space data, and the reconstruction itself which is done via a MATLAB® script. The reconstruction algorithm uses a general regridding algorithm, as outlined in 1.11. A brief description of the algorithm and the code follows here – for a full listing, refer to Appendix B.

Table 2.3 outlines the parameters used to define the k-space grid, k-space data size, and Kaiser-Bessel kernel parameters.

The regridding kernel ($C(\boldsymbol{k})$ from section 1.11) used is a standard Kaiser-Bessel window, as described by

$$w_n = \begin{cases} \frac{1}{W} \cdot \mathcal{I}_0 \left(\beta \left[1 - \left(\frac{2n}{W} \right)^2 \right] \right) & -W \leq 2n \leq W \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

The code defined k-space on a grid ranging from $[-0.5, 0.5]$ for each orthogonal direction, while MATLAB matrix indices must be positive integers $(1, 2, \ldots)$. MATLAB functions `tok.m` and `tomatrix.m` converted the matrix index between one coordinate system and the other.

After reading in the trajectory information processed from the `grads` file, the code linearly interpolated gradient information to a resolution of 2 μs, as the sampling rate was 500 kHz or one complex data point every 2 μs. Complex k-space data was read in a similar fashion and stored in an internal MATLAB array. These intermediate arrays were saved as files in MATLAB format to facilitate debugging and decrease run time upon subsequent executions.

Before the regridding operation can be started, an appropriate DCF was calculated using the standard form shown in Equation 1.53. A similar block of code then utilized the DCF and the k-space data to regrid onto a 256^3 Cartesian grid. Both phase and magnitude images were available to the user in the MATLAB environment by using the correct command syntax. Deapodization or "rolloff correction" was provided to the image by taking the Fourier transform of the kernel window. Images presented in this document use the MATLAB reconstruction code as described above.

2.4 Challenges

Understanding the overall system architecture was a natural and obvious place to begin the research. Because of the software license in place in the Imaging Research Centre, it is not possible to describe in any detail the various components (hardware, software components) involved in running the Durga sequence on the 3.0T magnet. However, this section will give an overview of selected problems and solutions encountered in the development of Durga.

1. Design the gradient sequence execution.

    - With the advice given by [26], it was decided to create the gradients in three separate gradient files in EPIC compatible for-

mat (see Appendix A), and load these files in memory just before the sequence is executed. While the scan is executing, a pointer switches to subsequent memory locations in real-time to run the various trajectories. An alternative methodology was considered, with each RF pulse and trajectory running concurrently in the same TR. However, this "long TR" idea was quickly abandoned in favour of the "short TR" sequence (*ie.* 1 RF pulse and trajectory per TR) described above because of the difficulties in getting the long TR sequence to download to the scanner hardware.

2. Overrange gradient current errors.

   - The Durga-designed gradients proved troublesome to run. Although designed within the magnet's specifications, the gradient points were at a lower resolution than the gradient boards' resolution of 4 μs. The scanner does no automatic interpolation for the gradient boards. If a sequence attempts to run gradients which are too extreme for the gradient boards, it produces either a *Gradient overrange* error or warning, depending on the severity. An error stopped the executing sequence immediately. A warning allowed the sequence to continue, but since the trajectory was likely warped or altered, the frame of k-space data associated with that trajectory was bad or invalid.

     Simple linear interpolation helped matters to some extent, but it was still not possible to run the gradients "full-strength", and some measure of attenuation was required. Eventually, the Durga gradient software was changed to design gradients at a resolution of 4 μs, and this resolved the gradient current errors.

3. Invalid data acquisition and noisy / missing k-space echoes.

   - After the initial problem of getting the sequence to run correctly was solved, there remained the difficulties of collecting valid echoes (*ie.* where $\mathbf{k} = 0$). After determining a method of arranging the data in the P-file (see 2.3.1) and the correct EPIC function call, the data collected was essentially "white noise" with no discernible echoes. It was eventually determined through trial-and-error that the pulse width was too wide (3200 μs) for a volumetric sequence,

and that a narrower width was required for adequate signal. The current version of the sequence uses a hard pulse width of 300 μs.

4. Echoes in data not aligning with $\mathbf{k} = 0$ in k-space

   - Comparing the k-space data and the trajectory information indicated that the echoes were not located at the correct location in the frame data, based on where they should have appeared as a consequence of the k-space trajectory position. Due to a misunderstanding with how this variable is set, the sequence was acquiring at the default rate of just over 31 kHz instead of the desired 500 kHz. This was confirmed when comparing the $\mathbf{k} = 0$ positions with the echoes in the data (they were off by a geometric factor). Correcting this bug fixed this echo position problem.

     After setting the receiver bandwidth correctly, the acquisition delay still had to be set correctly. The acquisition delay corrects for the amount of time between when the gradients are turned "on", and when they are actually running. This delay occurs because of the internal system electronics and gradient hardware being unable to force a large current instantaneously through the system. The data acquisition subsystem, on the other hand, has no such delay. Because Durga acquires data the entire time that the gradients are active, it is vital that this delay be set correctly such that the echoes appear in the k-space frame precisely where the trajectory data indicates they should. This was determined heuristically, analyzing the k-space echoes and gradient information, and confirmed through subsequent trials and tests.

5. Reconstructed data producing unrecognizable images.

   - The initial Durga gradients contained a Cartesian portion of k-space, named a "fly-by", where the data could be easily extracted and Fourier-transformed into low-resolution images. This was done for testing purposes and to try to isolate bugs in the sequence from potential bugs in the reconstruction. However, the initial data obtained from imaging a spherical phantom produced unrecognizable, obviously wrong images, and the phase images had many phase changes over the FOV. Deducing that the problem was in the data itself, a simple projection view sequence was

substituted into the code in place of the Durga gradients. This data was then 1-D Fourier transformed. The result was still incorrect, indicating a more fundamental problem with the sequence.

The projection gradient was then placed into a simple, working gradient echo stock sequence, and the data from this produced the correct 1-D profile. Comparing this working projection sequence with the Durga sequence indicated two missing lines of code in the Durga sequence – these lines are necessary to set an intermediate demodulation frequency, critical in correctly setting up the data collection procedure. Without this frequency set, the data was probably shifted by most of a Nyquist band, potentially explaining the erratic phase term. Changing the demodulation frequency shifts the reconstructed object in the encoding direction, which in the case of Durga would have unpredictable consequences based on the nature of the trajectories.

6. Unable to reduce the effective TR below a time of 16.7 ms.

   - By examining the running Durga gradients with an oscilloscope, it was determined that the *effective* TR was set to value of about 16.7 ms, regardless of what the TR parameter was set to in the sequence. As other GE sequences run with TRs of a shorter duration, it is evident that this should be a solvable problem, and thus shorter TRs possible. At the time of writing, this problem remains outstanding as a "known bug".

7. Evident oval artifact in some images.

   - Examining some of the images obtained with Durga, there is an "oval artifact" present, especially in regular, high proton-density images (*ie.* spherical phantom). The artifact seems regularly positioned, and appears in approximately the same positions between scans of the same phantom. Although this issue is still outstanding, two possible sources of this artifact exist:
     - an underlying bias with the Durga trajectories - perhaps with the implementation of the algorithm itself
     - an error in the reconstruction code

Of these two possibilities, the latter seems most likely to be the source. A simple debugging technique would be to reconstruct Durga images with other software, either a regridding technique or a more formal iterative method. If this eliminates the artifact, then the fault lies with the reconstruction code used for these images.

Chapter 3

Results

3.1 Phantoms

Three phantoms were used to evaluate the effectiveness of the Durga algorithm, with representative images generated by the three trajectory sets presented. Each dataset is accompanied by reference images from approximately the same slice positions, collected by a stock GE pulse sequence. ROIs accompany the images - the ROI positions are consistent for each image in the dataset, and the ROI positions are indicated in the reference image for each set. ROIs were square, and are dimensioned 16 x 16 for Sets A and B, and 12 x 12 for Set C, unless otherwise noted. The discrepancy is accounted for by the difference in FOV between Sets A, B, and C (see 3.2), such that ROIs represent approximately equal amounts of signal regardless of the gradient trajectories. See equations (1.61) and (1.62) for definitions of SNR and CNR.

The phantoms' specifications are as follows:

- Spherical GE provided phantom. Plastic exterior, water-filled, with a mass of 2.9 kg and diameter of 18.1 cm.
- GE resolution phantom. Cylindrical, plastic exterior and water-filled. Measures 26.8 cm long and 11 cm in diameter with a mass of 8 kg.
- MRA constructed phantom. Plastic tube, with plastic stoppers on each end. Water doped with Gadolinium to approximate the T_1 of hemoglobin. Measures 59.5 cm long, outer diameter of 1 cm, inner

Trajectories	% of Image	FOV (diameter / % of Image)
Set A	51.9	35 cm
Set B	53.1	34 cm
Set C	39.4	46 cm

Table 3.1: Effective FOV for Trajectories A, B, and C

diameter of 0.8 cm, and a mass of 100 g. Some air bubbles inside the tube were unavoidable during construction.

The following sections detail Durga results using the three trajectory sets over the three phantoms. The sections follow a similar format – images from approximately the same position in the phantom are shown for each trajectory set, and these are accompanied by one reference image for each phantom location. A table summarizes the SNRs and CNRs for the various ROIs. Details of the effective FOVs for the three trajectories are provided in the next section.

3.2 Field of View Calculations

Durga, because of its iterative, algorithmic design, does not have an easy theoretical method to calculate FOV, unlike more regular non-Cartesian trajectories (*ie.* 2-D spiral). However, by imaging objects of a known size Durga's effective FOV, based on the trajectory set, can be determined. For simplicity, images of the spherical phantom at the centre of the image set were chosen for these calculations. Consult Table 3.1 for the results.

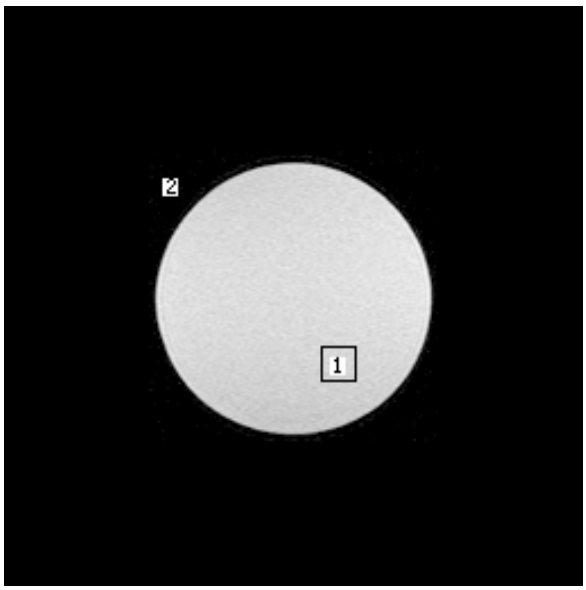

Figure 3.1: Reference image for spherical phantom with ROIs.

3.3 Spherical Phantom Measurements

Images were acquired axially with two ROIs used for measurements on this simple phantom. ROI_1 is in the interior, while ROI_2 is in the location of the oval artifact described in section 2.4. Images of sets A, B, and C are provided by Figures 3.2, 3.3, and 3.4. See Figure 3.1 for a reference.

Examining the images and Table 3.2, we can see that the results mirror a qualitative analysis. The images from Set A produced the best SNR and CNR, Set C the worst with B in between. Ideally, ROI_2 would have a low SNR being outside the image proper – the coherent oval artifact is manifested in the low CNR_{12} for all three sets. The dimpling artifact in A may indicate a spike in the k-space data for this group of images, since data from this set as a whole do not show this artifact. Conversely, the streaking in the Set C image is prevalent in other images from these trajectories, and indicates that these are inherent to the set. The oval artifact is apparent in images from all three sets, with the major axis oriented top left to bottom right.

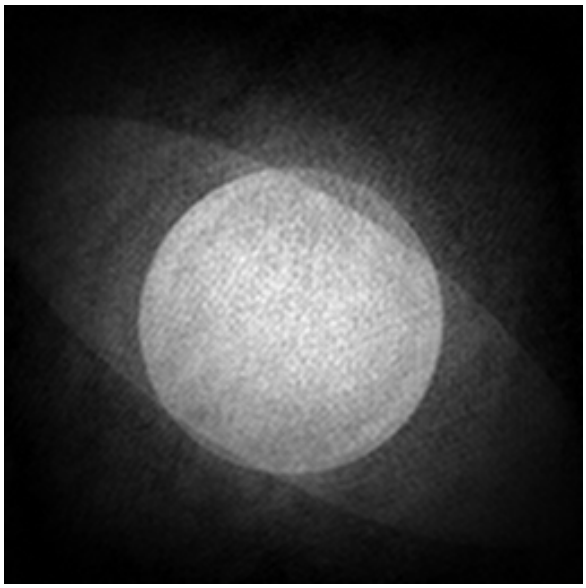

Figure 3.2: Set A image for spherical phantom.

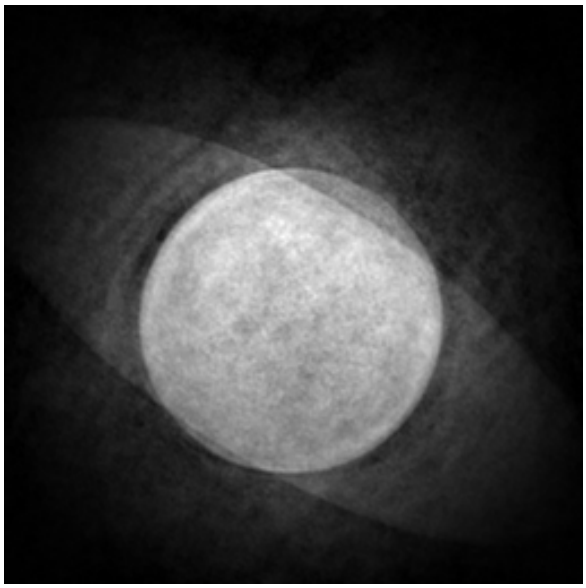

Figure 3.3: Set B image for spherical phantom.

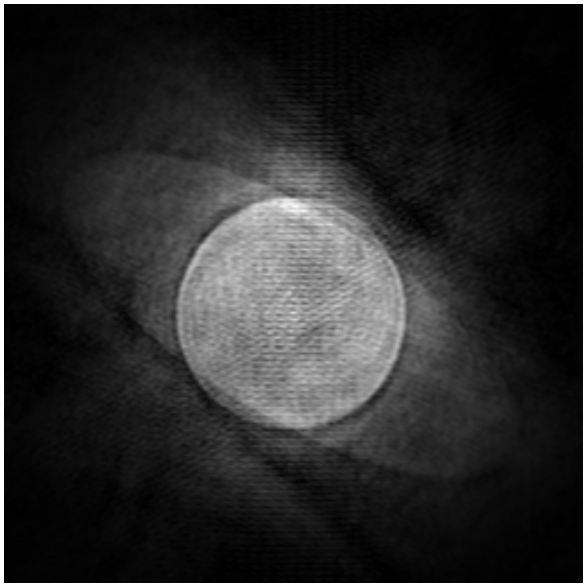

Figure 3.4: Set C image for spherical phantom.

Trajectories	SNR_1	SNR_2	CNR_{12}
Set A	36.3	13.3	23.0
Set B	27.2	10.6	16.6
Set C	21.9	9.8	12.0

Table 3.2: Spherical phantom imaging results

Trajectories	SNR_1	SNR_2	SNR_3	CNR_{14}	CNR_{24}	CNR_{34}
Set A	14.4	14.0	18.4	11.4	11.5	15.9
Set B	9.9	7.8	12.5	7.6	5.5	10.2
Set C	8.4	9.4	8.7	5.5	6.5	5.7

Table 3.3: GE resolution phantom, Image 1 results

3.4 GE Phantom Measurements

This resolution phantom is provided by GE to test stock and custom pulse sequences. Axially, it is comprised of various slices with vastly differing proton densities. Two representative reference images are selected (Figures 3.5 and 3.9), as these exhibit sparse and intense signal respectively. As before, images are compared between all three trajectory sets.

Comparing the images for Figures 3.6, 3.7, and 3.8, qualitatively the images from Sets A and C produce the best images. The comb-like structure in the upper left is plainly visible, as is the small capital 'A' roughly in the centre of the image, although the GE logo in the upper right is partially smeared by noise. Although not as high quality (and with a larger FOV), Figure 3.8 from Set C still produces a reasonable image, with the major structures identifiable, if not clear. Set B suffers from considerable smearing and produces an image of low quality. The ROI data for these images is presented in Table 3.3, and given the sparse nature of these images the measurements for SNR and CNR are understandably low.

The images presented in Figures 3.10, 3.11, and 3.12, are higher quality images than their counterparts for the previous slice location. It should be noted than the reference image in Figure 3.9 is not in the precise location of the data images – there is a horizontal dark bar not in evidence in the reference image because of slice location. Images for Sets A and B produce good images with reasonable CNRs, keeping in mind that ROI_2 should be a dark location, producing high contrast with ROI_1. Dimpling and noise obscure Set C's image, although the major features are identifiable. Full ROI measurements are contained in Table 3.4.

Figure 3.5: Reference image 1 for GE phantom with ROIs.

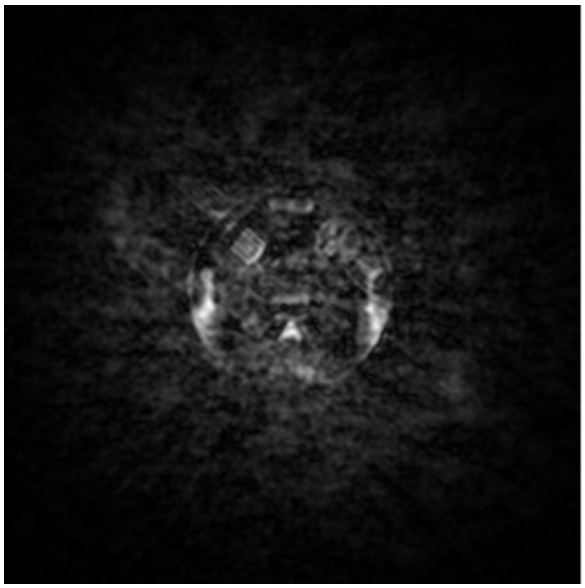

Figure 3.6: Set A, image 1 for GE phantom.

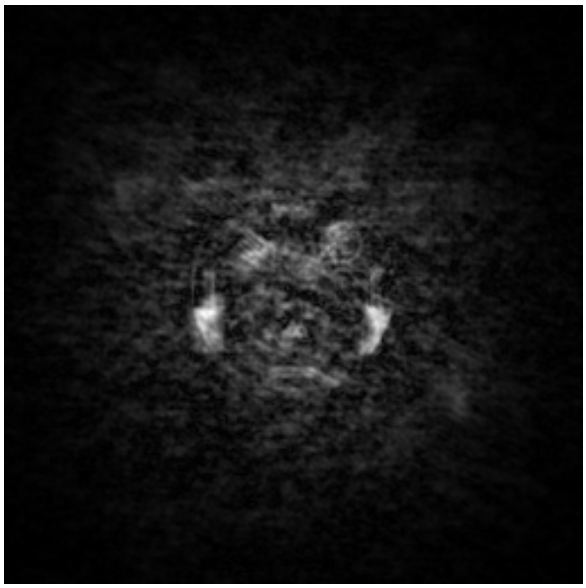

Figure 3.7: Set B, image 1 for GE phantom.

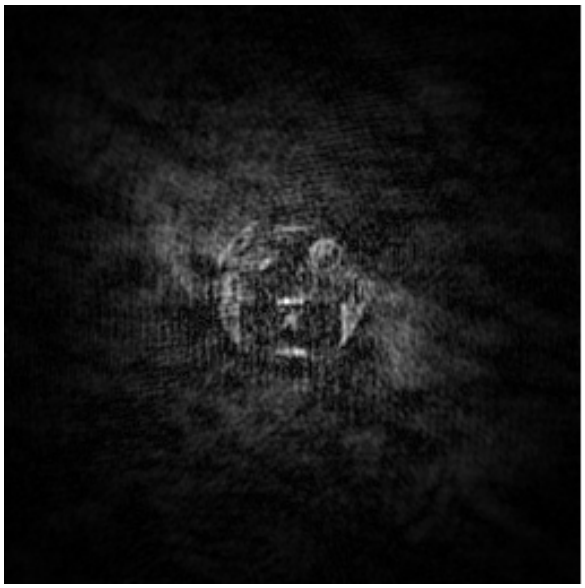

Figure 3.8: Set C, image 1 for GE phantom.

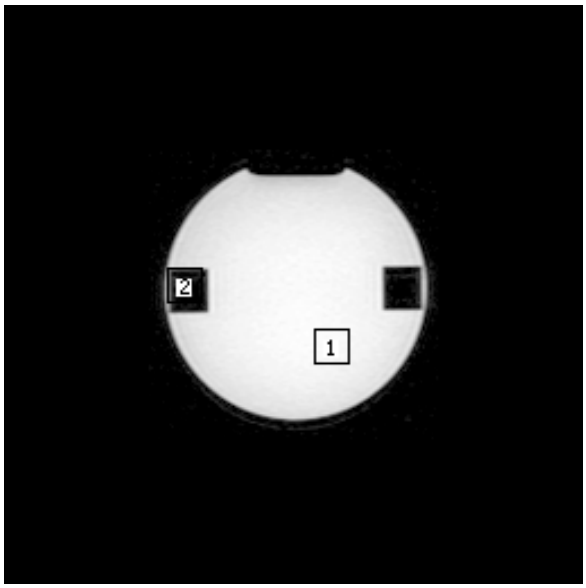

Figure 3.9: Reference image 2 for GE phantom with ROIs.

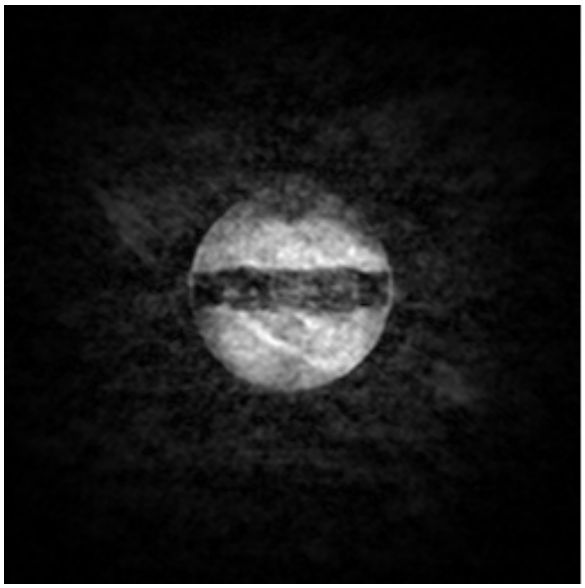

Figure 3.10: Set A, image 2 for GE phantom.

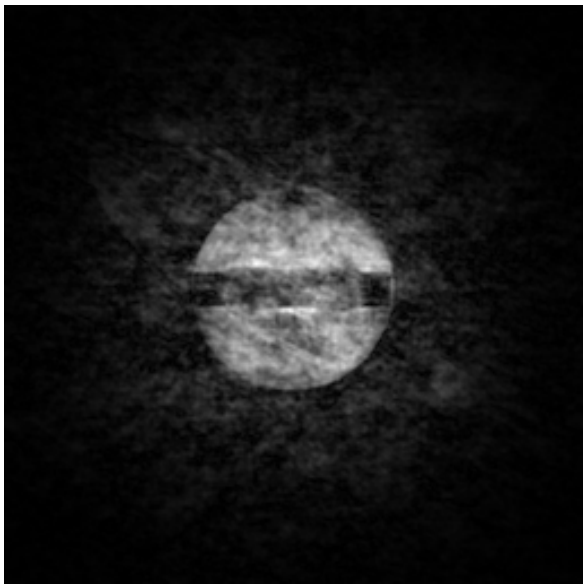

Figure 3.11: Set B, image 2 for GE phantom.

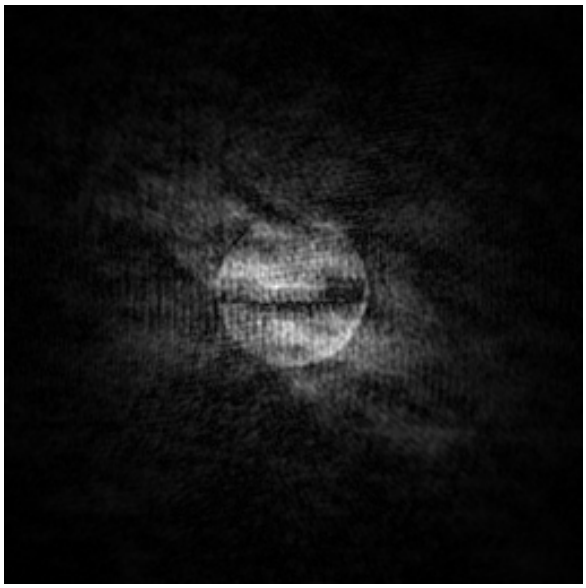

Figure 3.12: Set C, image 2 for GE phantom.

Trajectories	SNR_1	SNR_2	CNR_{12}
Set A	49.8	10.4	39.5
Set B	27.8	7.4	20.4
Set C	18.2	13.3	4.9

Table 3.4: GE resolution phantom, Image 2 results

Trajectories	SNR		
	Image 1	Image 2	Image 3
Set A	14.9	23.1	36.0
Set B	19.7	16.1	17.3
Set C	30.7	23.4	14.8

Table 3.5: MRA resolution phantom results

3.5 MRA Phantom Measurements

The MRA phantom was coiled in the 1-channel array such that the phantom was roughly circular in the coronal plane. However, it was also helically aligned anteriorly/posteriorly such that different coronal slices could image different parts of the helix.

Reference images are provided to a give a general overview of the phantom, but ROI locations are not provided because of the difficulty in aligning the reference scans to the trajectory images. Instead, SNR measurements are given in Table 3.5, and are calculated by comparing an ROI centred on a brightest portion of the image, and an equal area of noise in the middle of the coiled phantom. Unlike the previous measurements, ROIs are square with an area of 4 x 4, to account for the small image size and the desire to have an ROI completely in the signal portion of the image.

SNR measurements are hampered by evident noise in the images across all the trajectories, although the main features of the slice in relation to the signal in the phantom can be discerned fairly easily. The measurements themselves are highly reliant on the positioning of the ROI and the noise area respectively, and are given only as general guidelines to the image quality.

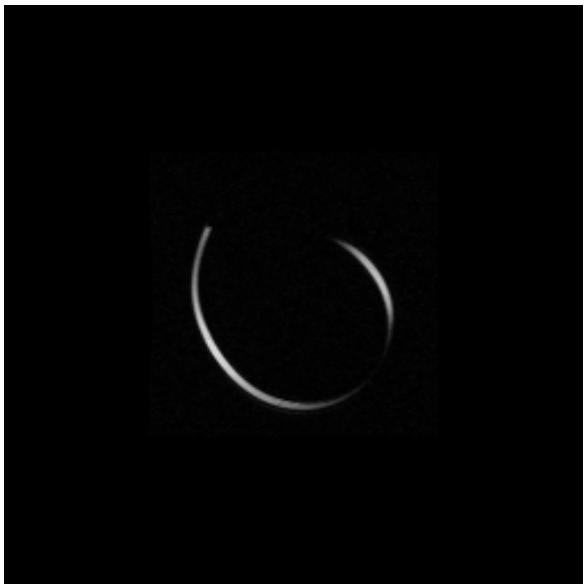

Figure 3.13: Reference image 1 for MRA phantom.

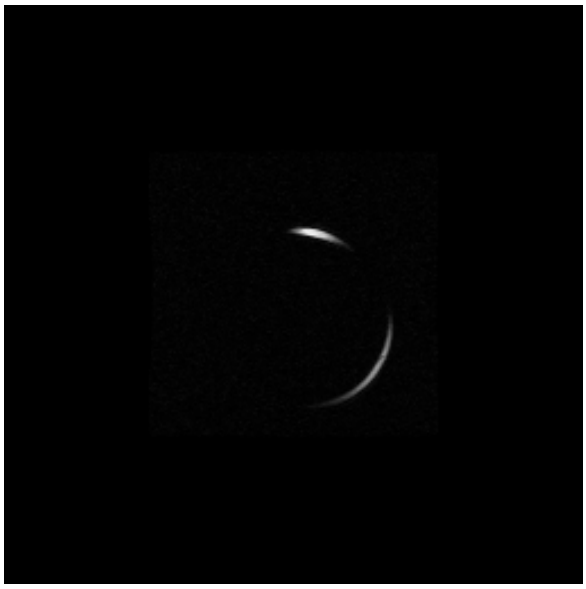

Figure 3.14: Reference image 2 for MRA phantom.

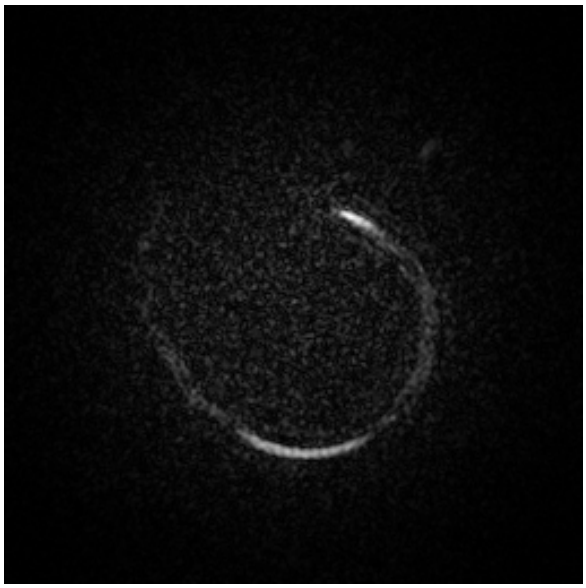

Figure 3.15: Set A, image 1 for MRA phantom.

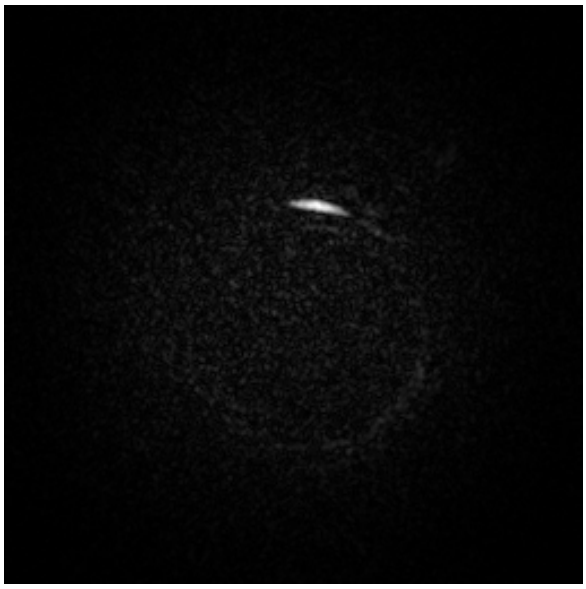

Figure 3.16: Set A, image 2 for MRA phantom.

Figure 3.17: Set A, image 3 for MRA phantom.

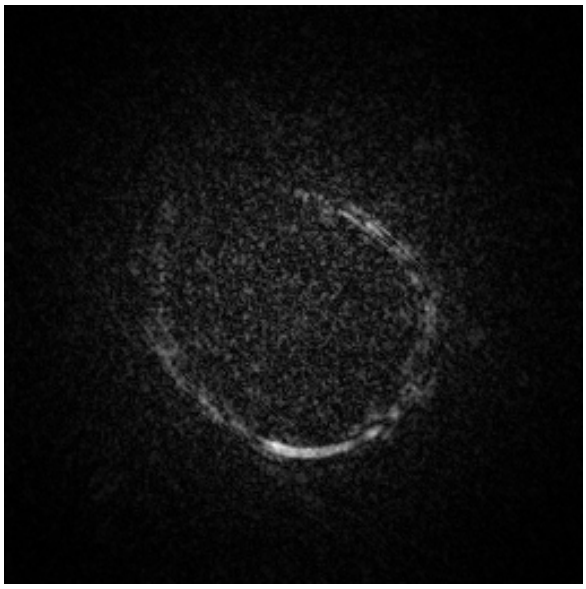

Figure 3.18: Set B, image 1 for MRA phantom.

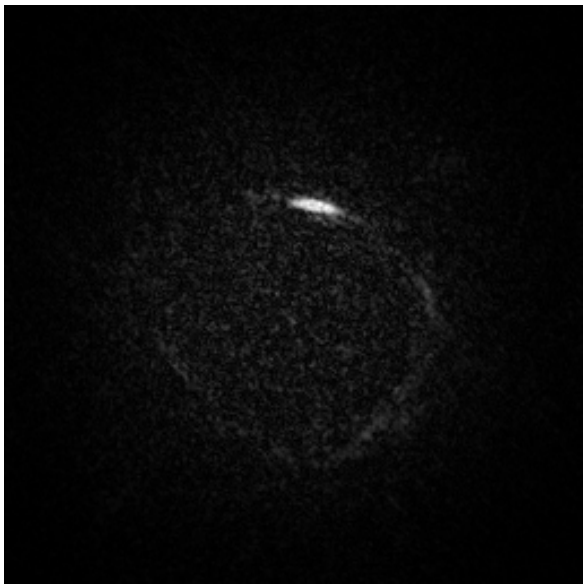

Figure 3.19: Set B, image 2 for MRA phantom.

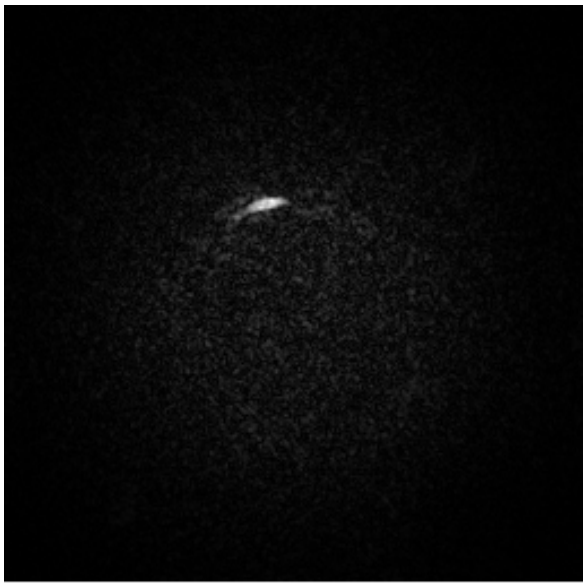

Figure 3.20: Set B, image 3 for MRA phantom.

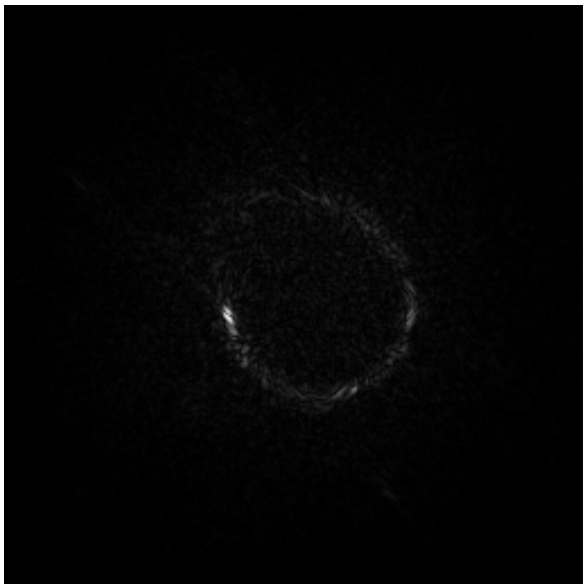

Figure 3.21: Set C, image 1 for MRA phantom.

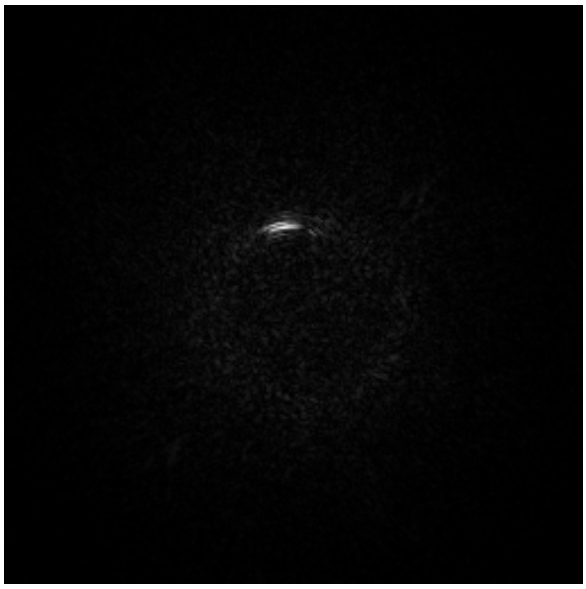

Figure 3.22: Set C, image 2 for MRA phantom.

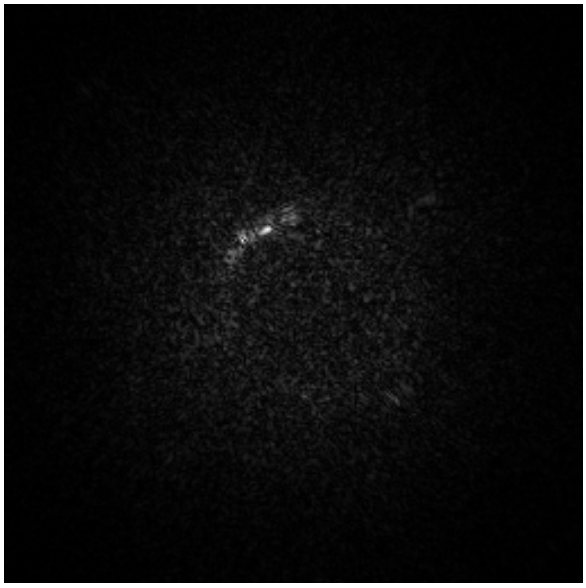

Figure 3.23: Set C, image 3 for MRA phantom.

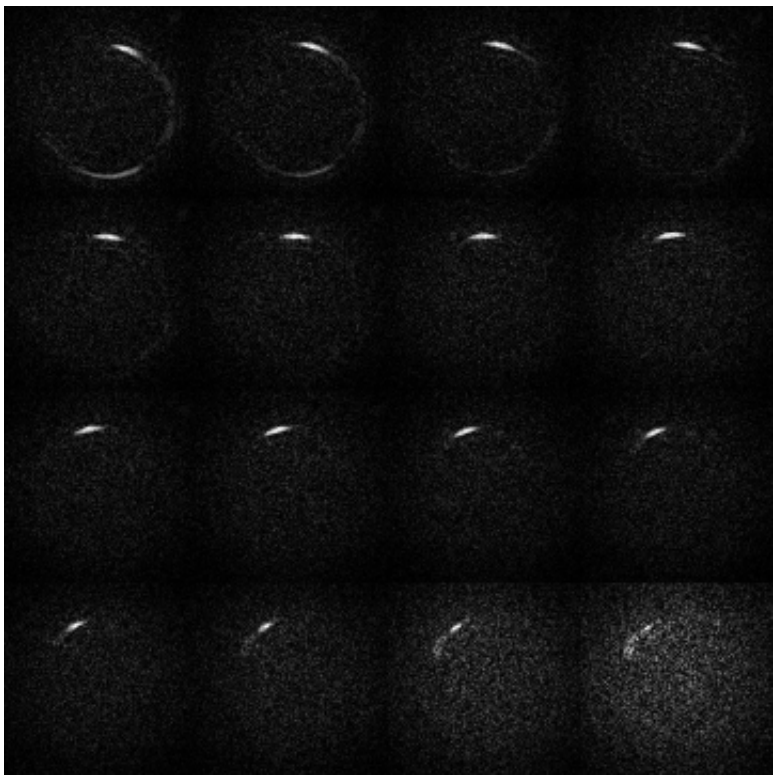

Figure 3.24: Consecutive MRA slices, Set A.

3.6 Consecutive MRA slices

This section details consecutive coronal slices of the MRA phantom for all trajectory sets and the same section of phantom. Although this data is perhaps better presented as a series of slides in a video-like fashion, they are given here as static images. Slides run in order across and then down, starting with the image in the upper left. It should be noted that since Sets A and B both have a smaller FOV than Set C, they produce a greater number of images over the volume of interest, and this is reflected in the number of slides presented and the subsequent differences between consecutive images.

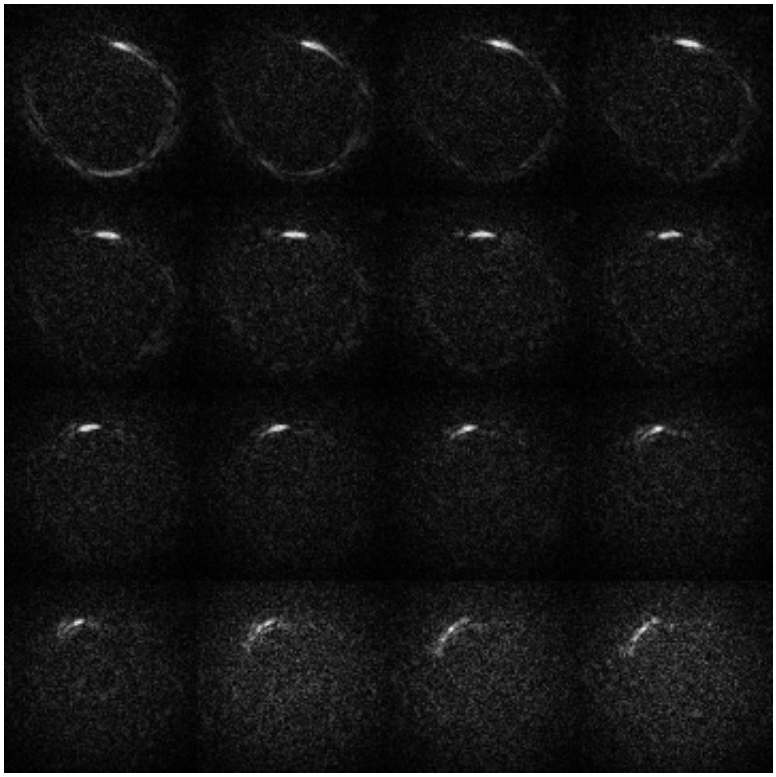

Figure 3.25: Consecutive MRA slices, Set B.

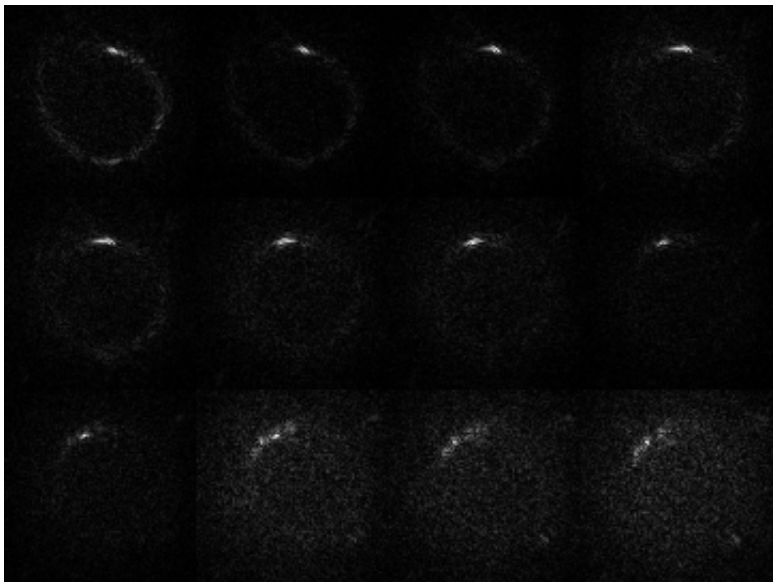

Figure 3.26: Consecutive MRA slices, Set C.

Chapter 4

Discussion and Future Research

Examining the data from the previous section, it is clear that of the three phantoms, Durga performs best with the sphere, followed by the MRA, and the poorest results are from the GE resolution phantom. This is not altogether surprising – the finer details present in the resolution phantom were always the most likely to be obscured by the levels of noise we are prepared to accept in Durga as a trade-off for shorter scan times. In particular, the images from location 1 (Figures 3.6, 3.7, and 3.8) are indicative of the problems these Durga trajectories have in imaging objects with fine details.

Conversely, Durga does comparatively well with the more homogeneous phantoms, with better SNR and overall image quality evident in the spherical phantom images (Figures 3.2, 3.3, and 3.4), and location 2 from the GE phantom (Figures 3.10, 3.11, and 3.12). Tempering these results is the evident oval artifact presented in Section 2.4. It would appear that the intensity of the artifact is positively correlated with either the spin density or the symmetry present in the object, as the artifact is less noticeable or absent in the sparser images of the GE or MRA phantoms.

Comparing the trajectories' images from across all phantoms, we can see that Sets A and B were approximately equal with regards to SNR, CNR, and overall image quality, and were superior in these regards to Set C. This is perhaps to be expected – Sets A and B have longer periods of "dead-time" after the imaging gradients as is shown in Section 2.2, and this is reflected in their associated sampling efficiencies. Balanced against the inferior image quality from Set C is its short sampling time which makes it particularly attractive to future experiments in MRA.

Investigating the source of the image quality differences, one is led to

the difference in sampling times between the three sets. It is possible that a phase error is built up during the readout of the gradients, and thus the longer readout time present in Set C produces more error and inferior images to those of Sets A and B. Another explanation concerns the physical characteristics of the gradients. It was observed during development that sometimes Durga would cause gradient current errors or warnings on the physical scanner, even though the gradients themselves were at or below the stated hardware specifications. Relaxing the demands on the hardware by attenuating the gradient amplitudes seemed to alleviate this problem, although it is possible that there was still a mismatch between the prescribed k-space position, and the actual one played out by the gradients. This would account for some of the image distortions evident in the images. However, it is perhaps unwise to draw too many conclusions from comparisons for only three sets of Durga trajectories – clearly more exploration needs to be done in these areas.

Imaging the MRA phantom produced some interesting results, as all of the trajectories were able to adequately image the same "loop" of signal in the upper portion of the phantom images. Examining the consecutive coronal slices, we see that if we arrange the images in a flipbook fashion, we can create something like a MRA "movie", in that the images appear to be tracking the passage of a bright bolus through the phantom. Of course, this is merely an illusion, since the Gadolinium-doped water is static, and the "movement" of the signal is simply an optical trick from the helical positioning of the phantom. However, it does suggest a potential clinical application in angiography imaging. There would be significant hurdles to overcome, and it is unclear how well Durga would do in regards to imaging moving spins, although its inherent velocity compensation would indicate a degree of optimism. The images of the MRA phantom present an interesting clinical progression for Durga – the data indicates a starting point for Durga's efficacy in MRA applications, although more experiments need to be conducted and the trajectories refined. Durga's low SNR images indicate that it is better suited to high contrast imaging, where the presence of noise can be easily ignored or overlooked by the eye. Conversely, applications that are SNR sensitive (*ie.* thermometry) are not well-suited for Durga trajectories.

Durga shows potential for use in non-Cartesian SENSE techniques [22] – although this would require an iterative reconstruction technique, it promises a shorter scan time coupled with increased image quality. With regards to reconstruction, a more advanced reconstruction technique (*ie.* iterative tech-

niques, Sparse MRI[15]) would improve Durga significantly. These techniques would allow us to remove much of the noise inherent in the images, increasing overall image quality.

It is evident that Durga benefits from a DCF [6], although what the optimum function might be is an area for further research. The procedure described in Section 1.11 (convolution with a kernel, and FFT) is a simple and general strategy, although it is likely a superior, more Durga-specific method exists.

With regards to imaging humans, Durga presents an interesting challenge. Heating limits are well established, and thus it is necessary to ensure accurate SAR calculations to guarantee that the sequence is within those limits. However, the issue with dB/dt is not as clear – the FDA states that they consider MRI to be a significant risk when dB/dt is sufficient to produce severe discomfort or painful nerve stimulation [7], while other research indicates that a limit of 20 T/s be observed [24]. Patient comfort levels must be monitored particularly closely, especially in initial experiments, because of Durga's unique gradient design.

Bibliography

[1] C. B. Ahn, J. H. Kim, and Z. H. Cho. High-speed spiral-scan echo planar nmr imaging. *IEEE Transactions on Medical Imaging*, MI-5(1), March 1986.

[2] C. K. Anand, A. T. Curtis, and R. Kumar. Durga: A heuristically-optimized data collection strategy for volumetric magnetic resonance imaging. *Engineering Optimization*, 40(2):117–136, 2008.

[3] C. K. Anand, T. Terlaky, and B. Wang. Rapid, embeddable design method for spiral magnetic resonance image reconstruction resampling kernels. *Optimization and Engineering*, 5(4):485–502, December 2004.

[4] Andrew V Barger, Walter F Block, Yuriy Toropov, Thomas M Grist, and Charles A Mistretta. Time-resolved contrast-enhanced imaging with isotropic resolution and broad coverage using an undersampled 3d projection trajectory. *Magn Reson Med*, 48(2):297–305, Aug 2002.

[5] Matt A. Bernstein, Kevin F. King, and Xiaohong Joe Zhou. *Handbook of MRI Pulse Sequences*. Academic Press, 1 edition, 9 2004.

[6] A. T. Curtis and C. K. Anand. Random volumetric mri trajectories via genetic algorithms. *International Journal of Biomedical Imaging*, 2008, 2008.

[7] US Food and Drug Administration. Guidance for Industry and FDA Staff : Criteria for Significant Risk Investigations of Magnetic Resonance Diagnostic Devices. Online PDF, July 2003. `http://www.fda.gov/cdrh/ode/guidance/793.pdf`.

[8] G. H. Glover. Simple analytic spiral k-space algorithm. *Magn Reson Med*, 42(2):412–415, Aug 1999.

[9] E. M. Haacke, R. W. Brown, M. R. Thompson, and R. Venkatesan. *Magnetic Resonance Imaging : Physical Principles and Sequence Design*. John Wiley & Sons, Inc., 1999.

[10] J. P. Hornak. The basics of mri, 2007. `http://www.cis.rit.edu/htbooks/mri/index.html` [Online; accessed 11-July-2007].

[11] J. Jackson, C. Meyer, D. Nishimura, and A. Macovski. Selection of a Convolution Function for Fourier Inversion Using Gridding. *IEEE Transactions on Medical Imaging*, 10(3):473–478, 1991.

[12] S. Lai and G. H. Glover. Three-Dimensional Spiral fMRI Technique: A Comparison with 2D Spiral Acquisition. *Magnetic Resonance in Medicine*, 39:68–78, 1998.

[13] P. C. Lauterbur. Image formation by induced local interactions: Examples employing magnetic resonance. *Nature*, 242:190–191, 1973.

[14] S. Ljunggren. A Simple Graphical Representation of Fourier-Based Imaging Methods. *Journal of Magnetic Resonance*, 54:338–343, 1983.

[15] Michael Lustig, David Donoho, and John M Pauly. Sparse mri: The application of compressed sensing for rapid mr imaging. *Magn Reson Med*, 58(6):1182–1195, Dec 2007.

[16] P. Mansfield and P. G. Morris. *Advances in Magnetic Resonance, the Principles of Biological and Medical Imaging by NMR*. Academic Press, 1982.

[17] Donald W. McRobbie, Elizabeth A. Moore, Martin J. Graves, and Martin R. Prince. *MRI from Picture to Proton*. Cambridge University Press, 2 edition, 4 2007.

[18] A. C. Melissinos. *Experiments in Modern Physics*. Academic Press, 1966.

[19] K. Nayak and D. Nishimura. Randomized trajectories for reduced aliasing artifact. In *ISMRM 1998 Conference Proceedings*, volume 1998/S1, page 670. International Society for Magnetic Resonance in Medicine, 1998.

[20] M. Noseworthy. Lecture Notes for BME-701. `http://physics.stjosham.on.ca/~mikenose/`, 2007. McMaster University, School of Biomedical Engineering.

[21] J. Pipe and P. Menon. Sampling Density Compensation in MRI: Rationale and an Interative Numerical Solution. *Magnetic Resonance in Medicine*, 41:179–186, 1999.

[22] K. P. Pruessmann, M. Weiger, P. Bornert, and P. Boesiger. Advances in sensitivity encoding with arbitrary k-space trajectories. *Magn Reson Med*, 46(4):638–651, Oct 2001.

[23] K. P. Pruessmann, M. Weiger, M. B. Scheidegger, and P. Boesiger. Sense: sensitivity encoding for fast mri. *Magn Reson Med*, 42(5):952–962, Nov 1999.

[24] D. J. Schaefer, J. D. Bourland, and J. A. Nyenhuis. Review of patient safety in time-varying gradient fields. *J Magn Reson Imaging*, 12(1):20–29, Jul 2000.

[25] Liewei Sha, Hua Guo, and Allen W Song. An improved gridding method for spiral mri using nonuniform fast fourier transform. *J Magn Reson*, 162(2):250–258, Jun 2003.

[26] J. Stainsby. Stainsby's Lectures on EPIC - For the Lx Software Release, September 2001. copyright: 1995, 1998, 1999, 2000, 2001.

[27] D. Twieg. The k-trajectory formulation of the NMR imaging process with applications in analysis and synthesis of imaging methods. *Medical Physics*, 10(5):610–621, 1983.

[28] Wikipedia. Spin-1/2 — wikipedia, the free encyclopedia, 2007. `http://en.wikipedia.org/w/index.php?title=Spin-1/2` [Online; accessed 21-June-2007].

[29] Wikipedia. Spin (physics) — wikipedia, the free encyclopedia, 2007. `http://en.wikipedia.org/w/index.php?title=Spin_(physics)` [Online; accessed 21-June-2007].

Appendix A

Scripts for Gradient and Raw Data Processing

A.1 makegrads.pl – create scanner gradient files

The `makegrads.pl` script creates the binary gradient files used by the pulse sequence when executing on the scanner. It expects four command-line arguments:

- *filename format* - base filename (see below)
- *N* - filename index (see below)
- *source resolution* - in μs, this is legacy and is usually 4, which is the resolution of gradient boards on the scanner
- *omit level* - legacy of previous testing and development, not used anymore – should be set to 0

There is one trajectory's worth of data in each input file.

Input filenames are in a format such that they are: *basename* + *num* + *num* + *gradient board* + .GRD. For example:

```
test001004X.GRD
test001004Z.GRD
test010001Y.GRD
...
```

The basename in the above case would be "test", while N is one more that the maximum number used in the input files. For example, continuing the example above, if the last gradient file was named `test015015Z.GRD`, then N would be set to 16.

The script will ignore (with a warning) any "missing" input files – that is, any files that are missing in a strict numerical sense (*ie.* `test002004X.GRD` exists, but not `test002005X.GRD`).

The GE scanner expects all gradient information to be in 16-bit signed, even, and big-endian format, and that each trajectory has an even number of points (the script will add a 0 at the end of each trajectory if it has an odd number of points). The script will make all gradients even (rounding toward 0), and will crop data (with a warning) where $|data| > 32666$. It also checks to ensure that the slew rate is not violated between successive points, and finally gives a warning whenever the number of gradient points changes between different trajectories – these checks ensure that the Durga code designing the gradients presents the information in a coherent and sane format.

There is one gradient file created per gradient board, named `xgrad`, `ygrad`, and `zgrad`. After successful completion of the `makegrads.pl` script, the number of created trajectories and gradient points per trajectory is noted, and the gradient files can be uploaded onto the scanner for use in the Durga pulse sequence.

Listing A.1: makegrads.pl code

```
#!/usr/bin/perl

use strict;

use Bit::Vector;

use constant TRUE => 1;
use constant FALSE => 0;
use constant MAX => 32766;
use constant MIN => -32766;
use constant MAX_GRAD => 0.040; # T/m
use constant MAX_SLEW => 155;   # T/m/s
```

```
use constant GRAD_RES => 4e-6;  # 4 us

my $numargs = $#ARGV + 1;

if ($numargs != 4) {
    print "Usage: $0 [filename format] [N (as NxN)]
       [source resolution (in us)] [omit level]\n";
    exit 1;
}

my $format = $ARGV[0];
my $N = $ARGV[1];
my $res = $ARGV[2];
my $omitnum = $ARGV[3];

$| = 1; # don't buffer STDOUT;

my $cpts;
for (my $i=1; $i<=$omitnum; $i++) {
    $cpts += $i;
}

my @grads = ("X", "Y", "Z");
my ($grad_res, $ntraj);
my $last_grad_res = 0;

print "Number of trajectories to omit: ".(4 *
   $cpts)."\n";
print "Total number of trajectories to create: ".($N *
   $N - 4 * $cpts)."\n";

# set omit matrix to FALSE
# ie. all trajectories are included in ?grad files
my @omit;
for (my $i=0; $i<$N; $i++) {
    for (my $j=0; $j<$N; $j++) {
        $omit[$i][$j] = FALSE;
    }
```

```
}

# remove trajectories from gradient files
for (my $i=0; $i<$N/2; $i++) {
    for (my $j=0; $j<$N/2; $j++) {
        my $vsum = $i + $j + 2;
        if ($vsum <= $omitnum + 1) {
            # eliminate trajectory from each
                corresponding corner
            $omit[$i][$j] = TRUE;
            $omit[$i][$N - $j - 1] = TRUE;
            $omit[$N - $i - 1][$j] = TRUE;
            $omit[$N - $i - 1][$N - $j - 1] = TRUE;
        }
    }
}

# Right now, the script is dumb, and assumes in source
   resolution is
# 16us if not 4us
if ($res > 4) {
    print "Expand gradient files to 4us... ";

    # Expand gradients to 4us first -- write to
       seperate output files
    mkdir("4us");

    foreach my $grad (@grads) {
        while (glob("16us/".$format."*".$grad.".GRD"))
           {
            my $in = $_;
            /^16us\/($format(\d{3})(\d{3}).*)$/;
            my $out = "4us/".$1;
            my $row = int($2);
            my $col = int($3);

            next if ($omit[$row][$col]);    # skip
               omitted trajectories
```

```
open (INFILE, $in) || die("$!\n");
open (OUTFILE, ">$out") || die("$!\n");

my @lines = readline INFILE;
my $outcount = 0;

my $last;
my $set = FALSE;
for (my $i=0; $i<=$#lines; $i++) {
    next if ($lines[$i] =~ /^#/);

    my $l = $lines[$i];
    chomp $l;
    my ($val, $tmp) = split(" ", $l);

    if ($set) {
        my $step = ($val - $last) / 4;

        for (my $j=1; $j<=4; $j++) {
            my $v = int($last + $j *
                $step);
            print OUTFILE $v."\n";
            $outcount++;
        }
    }
    else {
        $set = TRUE;
        print OUTFILE $val."\n";
        $outcount++;
    }

    $last = $val;
}

# add 1 line to make the number of points
    even
if (($outcount % 2) == 1) {
```

```
                print OUTFILE "0\n";
                $outcount++;
            }

            $grad_res = $outcount;

            if ($last_grad_res != $grad_res) {
                print "Warning: Gradient resolution
                    changed from: $last_grad_res to
                    $grad_res -- $out\n";
            }

            $last_grad_res = $grad_res;

            close OUTFILE;
            close INFILE;
        }
    }

    print "done\n";
}
else {
    rename("4us", "4us-orig");
    mkdir("4us");

    foreach my $grad (@grads) {
        while (glob("4us-orig/".$format.
            "*".$grad.".GRD")) {

            my $in = $_;
            /^4us-orig\/($format(\d{3})(\d{3}).*)$/;
            my $out = "4us/".$1;
            my $row = int($2);
            my $col = int($3);

            open (INFILE, $in) || die("$!\n");
            open (OUTFILE, ">$out") || die("$!\n");
```

```
my $outcount = 0;
my $lastval = 0;
my $maxslew = 0;
my $maxslewline = 0;

while(<INFILE>) {
    next if ($_ =~ /^#/);
    my ($val, $tmp) = split(" ", $_);
    $outcount++;

    if ($val > MAX) {
        $val = MAX;
        print "Warning: MAX value cropping
            -- file: $in, line:
            $outcount\n";
    }
    elsif ($val < MIN) {
        $val = MIN;
        print "Warning: MIN value cropping
            -- file: $in, line:
            $outcount\n";
    }

    my $slew = (($val - $lastval) / MAX *
        MAX_GRAD) / (GRAD_RES);

    if ($slew > $maxslew) {
        $maxslew = $slew;
        $maxslewline = $outcount;
    }

    $lastval = $val;

    print OUTFILE "$val\n";
}

# ensure last point in outfile is 0
if ($lastval != 0) {
```

```
                print OUTFILE "0\n";
                $outcount++;
            }

            # make number of points even
            if (($outcount % 2) == 1) {
                print OUTFILE "0\n";
            }

            if ($maxslew > MAX_SLEW) {
                print "Warning: Maximum slew violated
                    -- $out: ".int($maxslew).", line:
                    $maxslewline\n";
            }

            $grad_res = $outcount;

            if ($last_grad_res != $grad_res) {
                print "Warning: Gradient resolution
                    changed from: $last_grad_res to
                    $grad_res -- $out\n";
            }

            $last_grad_res = $grad_res;

            close OUTFILE;
            close INFILE;
        }
    }

    print "done\n";
}

print "Create binary gradient files... ";

# Create xgrad, ygrad, and zgrad binary files
foreach my $grad (@grads) {
    my $out;
```

```
if ($grad eq "X") {
    $out = "xgrad";
}
elsif ($grad eq "Y") {
    $out = "ygrad";
}
else {
    $out = "zgrad";
}

open (OUTFILE, ">$out") || die ("Could not write
   $out: $!");
$ntraj = 0;

for (my $i=0; $i<$N; $i++) {
    for (my $j=0; $j<$N; $j++) {

        next if ($omit[$i][$j]);   # skip omitted
            trajectories

        my $in = sprintf("4us/test%.3d%.3d%s.GRD",
            $i, $j, $grad);

        if (-e $in) {
            open (INFILE, $in) || die("$!\n");
        }
        else {
            print "Warning: skipping $in file --
                Does not exist\n";
            next;
        }

        $ntraj++;

        while (<INFILE>) {
            my $val = $_;
            chomp $val;
            $val = makeeven($val);
```

```
                my $vec = Bit::Vector->new_Dec(16,
                    $val);
                print OUTFILE pack("H*", $vec->to_Hex);
                #print $vec->to_Hex."\n";
            }

            close INFILE;
        }
    }

    close OUTFILE;

    print "$out ";
}

print "\n";
#printtraj(\@traj);

print "Gradient Resolution: $grad_res\n";
print "Number of Trajectories: $ntraj\n";

print "done\n";

sub makeeven($) {
    my $x = shift;

    if (($x % 2) == 1) {
        return ($x > 0) ? $x - 1 : $x + 1;
    }

    return $x;
}

sub printtraj($) {
    my $ref = shift;
    my @traj = @$ref;
```

```
    for (my $i=0; $i<$N; $i++) {
        for (my $j=0; $j<$N; $j++) {
            ($traj[$i][$j]) ? print "O " : print "  ";
        }

        print "\n";
    }
}
```

A.2 concatgrads.pl – create k-space position from gradients

The `concatgrads.pl` script takes as inputs the `?grad` files created with `makegrads.pl` and gives as an output a `grads` file, which is used by the reconstruction code as the k-space positions of the data. The script requires that the gradient files are located in the same directory, and the number trajectories as a command-line argument. To calculate the k-space positions, the code uses a simple running total (*ie.* zeroth moment). The code assumes that there is no accumulated phase between trajectories (*ie.* each new trajectory starts its count at 0).

Listing A.2: concatgrads.pl code

```
#!/usr/bin/perl

use strict;

use Bit::Vector;
use File::stat;

use constant SIGNED => "s*";
use constant UNSIGNED => "S*";
use constant BIG => "n*";

my $numargs = $#ARGV + 1;
```

```
if ($numargs != 1) {
    print "Usage: $0 [number of trajectories]\n";
    exit 1;
}

my $traj = $ARGV[0];

open (XGRAD, "xgrad") || die("$!\n");
open (YGRAD, "ygrad") || die("$!\n");
open (ZGRAD, "zgrad") || die("$!\n");

open (OUTFILE, ">grads") || die("$!\n");

my $stats = stat("xgrad");
my $grad_pts = ($stats->size / 2) / $traj;

for (my $i=0; $i<$traj; $i++) {
    my ($kx, $ky, $kz);
    $kx =
        $ky =
        $kz = 0;

    for (my $j=0; $j<$grad_pts; $j++) {
        my ($xval, $yval, $zval);

        read XGRAD, $xval, 2;
        read YGRAD, $yval, 2;
        read ZGRAD, $zval, 2;

        my $x = unpack SIGNED, pack UNSIGNED, unpack
            BIG, $xval;
        my $y = unpack SIGNED, pack UNSIGNED, unpack
            BIG, $yval;
        my $z = unpack SIGNED, pack UNSIGNED, unpack
            BIG, $zval;

        $kx += $x;
```

```
            $ky += $y;
            $kz += $z;

            print OUTFILE "$kx $ky $kz\n";
        }
}

close OUTFILE;

close XGRAD;
close YGRAD;
close ZGRAD;
```

A.3 raw.pl – process raw P-File

The `raw.pl` script takes as an input the name of the "P-File" that was used in the Durga scan. Besides writing an output file(s) in the format `pfile-coil#-complex`, the script also displays an assortment of data that is packaged along with the "P-File", among these the scan date, frame size and number of frames, the number of receive coils, and the raw data size (see section 2.3.1). Data is organized in the output files as a *real, imaginary* pair, one data point per line. One output file is created for each receive coil used in the scan.

Listing A.3: raw.pl code

```
#!/usr/bin/perl

use strict;

use Time::Local;
use Time::localtime;
use Getopt::Std;

use constant HEADER_REC_SIZE            => 2048;
use constant HEADER_PER_PASS_TAB_SIZE   => 4096;
```

```
use constant HEADER_UNLOCK_RAW_SIZE      => 4096;
use constant HEADER_DATA_ACQ_TAB_SIZE    => 20480;
use constant HEADER_NEX_TAB_SIZE         => 2052;
use constant HEADER_NEX_ABORT_TAB_SIZE   => 2052;
use constant HEADER_TOOL_SIZE            => 2048;
use constant HEADER_EXAM_SIZE            => 1040;
use constant HEADER_SERIES_SIZE          => 1028;
use constant HEADER_IMAGE_SIZE           => 1044;

use constant DATA_START => 66072;

use constant SEQ_NUM_OFF => 8;
use constant SEQ_NUM_SIZE => 2;
use constant RUN_NUM_OFF => 10;
use constant RUN_NUM_SIZE => 6;
use constant SCAN_DATE_OFF => 16;
use constant SCAN_DATE_SIZE => 10;
use constant SCAN_TIME_OFF => 26;
use constant SCAN_TIME_SIZE => 8;
use constant RAW_DATA_TYPE_OFF => 44;
use constant RAW_DATA_TYPE_SIZE => 2;
use constant SCAN_TYPE_OFF => 54;    # 15 bits
use constant SCAN_TYPE_SIZE => 2;

use constant NUM_SLICES_OFF => 68;
use constant NUM_SLICES_SIZE => 2;
use constant NUM_ECHOES_OFF => 70;
use constant NUM_ECHOES_SIZE => 2;
use constant NUM_EXCITATIONS_OFF => 72;
use constant NUM_EXCITATIONS_SIZE => 2;
use constant NUM_FRAMES_OFF => 74;   # yres
use constant NUM_FRAMES_SIZE => 2;
use constant FRAME_SIZE_OFF => 80;   # xres
use constant FRAME_SIZE_SIZE => 2;
use constant NUM_BASELINES_OFF => 76;
use constant NUM_BASELINES_SIZE => 2;

use constant NUM_3D_VOLS_OFF => 86;
```

```
use constant NUM_3D_VOLS_SIZE => 2;

use constant HEADER_RAW_PASS_SIZE_OFF => 116;
use constant HEADER_RAW_PASS_SIZE_SIZE => 4;

use constant RECEIVE_COILS_OFF => 200;
use constant RECEIVE_COIL_START=> 0;
use constant RECEIVE_COIL_STOP => 2;
use constant RECEIVE_COIL_NUMBER_SIZE => 2;

use constant IMAGE_SIZE_Y_OFF => 858;
use constant IMAGE_SIZE_Y_SIZE => 2;
use constant FREQ_KSPACE_STEP_OFF => 864;
use constant FREQ_KSPACE_STEP_SIZE => 4;
use constant PHASE_KSPACE_STEP_OFF => 870;
use constant PHASE_KSPACE_STEP_SIZE => 4;

use constant ASCII => "A*";
use constant FLOAT => "f*";
use constant SHORT => "v*";
use constant SIGNED => "s*";
use constant LONG => "V*";

my $numargs = $#ARGV + 1;
if ($numargs != 1) {
    print "Usage: raw.pl [P-file]\n";
    exit;
}

my $infile = $ARGV[0];
open (INFILE, $infile) || die("Could not open $infile:
    $!\n");

my $seq_num = GetHeaderVal(*INFILE, SEQ_NUM_OFF,
    SEQ_NUM_SIZE, ASCII);
my $run_num = GetHeaderVal(*INFILE, RUN_NUM_OFF,
    RUN_NUM_SIZE, ASCII);
```

```
my $scan_date = GetHeaderVal(*INFILE, SCAN_DATE_OFF,
    SCAN_DATE_SIZE, ASCII);
my $scan_time = GetHeaderVal(*INFILE, SCAN_TIME_OFF,
    SCAN_TIME_SIZE, ASCII);
my $tm = ProcessDateTime($scan_date, $scan_time);
my $lt = localtime($tm);

my $data_type = GetHeaderVal(*INFILE,
    RAW_DATA_TYPE_OFF, RAW_DATA_TYPE_SIZE, SHORT);

my $nslices = GetHeaderVal(*INFILE, NUM_SLICES_OFF,
    NUM_SLICES_SIZE, SHORT);
my $nechoes = GetHeaderVal(*INFILE, NUM_ECHOES_OFF,
    NUM_ECHOES_SIZE, SHORT);
my $nex = GetHeaderVal(*INFILE, NUM_EXCITATIONS_OFF,
    NUM_EXCITATIONS_SIZE, SHORT);
my $framesize = GetHeaderVal(*INFILE, FRAME_SIZE_OFF,
    FRAME_SIZE_SIZE, SHORT);
my $nframes = GetHeaderVal(*INFILE, NUM_FRAMES_OFF,
    NUM_FRAMES_SIZE, SHORT);
my $nbaselines = GetHeaderVal(*INFILE,
    NUM_BASELINES_OFF, NUM_BASELINES_SIZE, SHORT);

my @rcvcoils = GetReceiveCoilInfo(*INFILE);

my $raw_pass_size = GetHeaderVal(*INFILE,
    HEADER_RAW_PASS_SIZE_OFF,
    HEADER_RAW_PASS_SIZE_SIZE, LONG);

my $image_size_y = GetHeaderVal(*INFILE,
    IMAGE_SIZE_Y_OFF, IMAGE_SIZE_Y_SIZE, SHORT);
my $freq_kspace_step = GetHeaderVal(*INFILE,
    FREQ_KSPACE_STEP_OFF, FREQ_KSPACE_STEP_SIZE, FLOAT);
my $phase_kspace_step = GetHeaderVal(*INFILE,
    PHASE_KSPACE_STEP_OFF, PHASE_KSPACE_STEP_SIZE,
    FLOAT);

###
```

```
# Print Header Information

print "P-File: $infile\n";

print "Data File Number: $run_num\n";

my $data_type_name;
if ($data_type == 0) {
    $data_type_name =  "EMP";
}
elsif ($data_type == 1) {
    $data_type_name = "NOREC";
}
else {
    $data_type_name = "NOPROC";
}
print "Data Type: $data_type_name\n";

# Assume just the first structure is used...
print "Receive Coil Numbers:
   ".$rcvcoils[0]->{"start"}." to
   ".$rcvcoils[0]->{"stop"}."\n";
print "Total Coils Used: ".($rcvcoils[0]->{"stop"} -
   $rcvcoils[0]->{"start"} + 1)."\n";

print "Number of Slices: $nslices\n";
print "Number of Echoes: $nechoes\n";
print "Number of Excitations: $nex\n";
print "Frame Size (xres): $framesize\n";
print "Number of Frames (yres): $nframes\n";
print "Number of Baselines: $nbaselines\n";

print "Raw Size: $raw_pass_size\n";

#exit 1;

# Extract Real/Imaginary information
my (@real, @imag);
```

```
my $pos;
my ($rval, $ival, $r, $i);
my $datalen = $framesize * $nframes;

print "\n";

# Skip first frame (?) -- no?
# Organize data coil first, then rhnframes, then
    rhfrsize
for (my $coil=$rcvcoils[0]->{"start"};
    $coil<=$rcvcoils[0]->{"stop"}; $coil++) {
     my (@treal, @timag);

     for (my $i=0; $i<$nframes; $i++) {
          $pos = DATA_START + (4 * $coil * $datalen) +
             (4 * $i * $framesize);
          seek INFILE, $pos, 0;

          for (my $j=0; $j<$framesize; $j++) {
               read INFILE, $rval, 2;
               read INFILE, $ival, 2;
               my $r = unpack(SIGNED, $rval);
               my $i = unpack(SIGNED, $ival);
               push @treal, $r;
               push @timag, $i;
          }
     }

     @real[$coil] = [ @treal ];
     @imag[$coil] = [ @timag ];

     print "Processed Coil data: $coil\n";
}

close INFILE;

print "\n";
```

```
# Output to data files
for (my $coil=$rcvcoils[0]->{"start"};
    $coil<=$rcvcoils[0]->{"stop"}; $coil++) {
     my $out = "$infile-$coil-complex";
     open (OUT, ">$out") || die ("Could not open $out:
        $!\n");

     for (my $i=0; $i<$datalen; $i++) {
          print OUT $real[$coil][$i]."
             ".$imag[$coil][$i]."\n";
     }

     close OUT;

     print "Finished writing: $out\n";
}

close OUT;


sub GetHeaderVal($$$$) {
     my $in = shift;
     my $off = shift;
     my $size = shift;
     my $temp = shift;
     my $val;

     seek $in, $off, 0;
     read $in, $val, $size;

     return unpack($temp, $val);
}

sub ProcessDateTime($$) {
     my $date = shift;
     my $time = shift;

     $date =~ /(\d+)\/(\d+)\/(\d+)/;
```

```
    my $mon = $1 - 1;
    my $day = $2;
    my $year = 1900 + $3;

    $time =~ /(\d+):(\d+)/;
    my $hour = $1;
    my $min = $2;

    return timelocal(0, $min, $hour, $day, $mon,
        $year);
}

sub GetReceiveCoilInfo($) {
    my $in = shift;
    my $val;
    my @ret;

    for (my $i=0; $i<4; $i++) {
        seek $in, RECEIVE_COILS_OFF + ($i * 4) +
            RECEIVE_COIL_START, 0;
        read $in, $val, RECEIVE_COIL_NUMBER_SIZE;
        $ret[$i]->{"start"} = unpack(SHORT, $val);

        seek $in, RECEIVE_COILS_OFF + ($i * 4) +
            RECEIVE_COIL_STOP, 0;
        read $in, $val, RECEIVE_COIL_NUMBER_SIZE;
        $ret[$i]->{"stop"} = unpack(SHORT, $val);
    }

    return @ret;
}
```

Appendix B

Reconstruction Software

This appendix contains the reconstruction code as described in section 2.3.2.

B.1 recon.m

Listing B.1: recon.m code

```
%%%
% k-space regridding algorithm
%
% Part of research conducted by:
%
% Paul Polak
%
% for use in a thesis for:
%
% M.A.Sc Biomedical Engineering
% McMaster University
% Autumn 2008
%%%

% define grid
global kern W nkernval;

N = 256;
```

```
A = zeros(N, N, N);
density = zeros(N, N, N);
frsize = 6352;
grad_pts = frsize / 2;
nframes = 177;
skip = 1;
trikern = 0;

kmin = -0.5;
delta = (abs(2*kmin)) / (N - 1);

beta = 8;
Wn = 4;
W = Wn * delta;

% data files
basename = '06144-0';
kdata = strcat(basename, '-complex');
ksave = strcat(basename, '-data.mat');
Asave = strcat(basename, '-A.mat');
densitysave = 'density-data.mat';
gradsave = 'grads-data.mat';
gradsin = 'grads';
rolloffsave = 'rolloff.mat';

tic;
starttime = toc;
initial = toc;

disp('Starting reconstruction...');

% make kernel look-up array
kernwidth = 16;
kernstep = 1e6;
n = (-kernwidth/2*delta):(kernwidth/kernstep):
    (kernwidth/2*delta);

if (trikern == 0)
```

```
    disp('KB kernel');
    kern = (1/W) * besseli(0, beta * (1 - (2*n/W).^2))
       .* rect(2*n/W, 1);
else
    disp('Triangle kernel');
    height = 100;
    m = height / W;

    k = 1;
    for j = (-kernwidth/2*delta):(kernwidth/kernstep):
        (kernwidth/2*delta)

        if (abs(j) > W)
            kern(k) = 0;
        else
            if (j < 0)
                kern(k) = m * j + height;
            else
                kern(k) = -m * j + height;
            end
        end

        k = k + 1;
    end
end

now = toc;
disp(sprintf('Finished populating kernel lookup.
   Time: %d seconds', round(now - initial)));

% read in trajectories if necessary
if ((exist(gradsave, 'file')) ~= 2)
    disp(sprintf('Read trajectory information from
       file "%s"... (%d lines)', gradsin, grad_pts *
       nframes));

    in = fopen(gradsin, 'r');
    in_traj = zeros(grad_pts * nframes, 3);
```

```
    x = 1;

    for u=1:nframes
        for v=1:grad_pts
            line = fgetl(in);

            if (line == -1)
                disp('EOF reached?');
                return;
            end

            [kx ky kz] = strread(line, '%f %f %f');

            in_traj(x, 1) = kx;
            in_traj(x, 2) = ky;
            in_traj(x, 3) = kz;
            x = x + 1;

            if (mod(x, 10000) == 0)
                disp(sprintf('Read %d lines', x));
            end
        end
    end

    fclose(in);

    save(gradsave, 'in_traj');

    disp('Saved trajectory data');
else
    load(gradsave, 'in_traj');
    disp('Loaded trajectory data');
end


disp('Create interpolated trajectory matrix');

% Get value to standardize kspace to [-kmin, kmin];
```

```
kmaxval = max(max(abs(in_traj))) / (abs(kmin));

traj = zeros(2 * length(in_traj), 3);

% interpolate between k space values
for j = 1:nframes
    for k = 1:(grad_pts - 1)
        idx = (j-1) * grad_pts + k;

        for l = 1:3
            traj(2*idx - 1, l) = in_traj(idx, l) /
               kmaxval;
            traj(2*idx, l) = ((in_traj(idx, l) +
               in_traj(idx+1, 1)) / 2) / kmaxval;
        end
    end

    idx = j * grad_pts;

    for l = 1:3
        traj(2*idx - 1, l) = in_traj(idx, l) / kmaxval;
        traj(2*idx, l) = traj(2*idx - 1, l);
    end
end

clear in_traj;

disp('Finished interpolated trajectories');


if ((exist(ksave, 'file')) ~= 2)
    % read in k-space data
    disp(sprintf('Read k-space data from file "%s"...
       (%d lines)', kdata, frsize * nframes));

    in = fopen(kdata, 'r');
    data = zeros(frsize * nframes, 1);
```

```
    x = 1;

    for u=1:nframes
        for v=1:frsize
            line = fgetl(in);

            if (line == -1)
                disp('EOF reached?');
                return;
            end

            [re im] = strread(line, '%f %f');
            data(x) = re + i*im;
            x = x+1;

            if (mod(x, 10000) == 0)
                disp(sprintf('Read %d lines', x));
            end
        end
    end

    fclose(in);

    save(ksave, 'data');

    disp('Saved k-space data');
else
    load(ksave, 'data');
    disp('Loaded k-space data');
end


initial = toc;

if ((exist(densitysave, 'file')) ~= 2)
    disp('Begin density calculation...');
    initial = toc;
```

```
for j=1:skip:length(traj)
    if (mod(j, 1000) == 0)
        disp(sprintf('Finished density gridding %d
            points', j));

        if (mod(j, 10000) == 0)
            curtime = toc;
            disp(sprintf('Complete in %d seconds',
                round((length(data) - j) / j *
                (curtime - initial))));
        end
    end

    % Trajectory points
    trajx = traj(j, 1);
    trajy = traj(j, 2);
    trajz = traj(j, 3);

    % Translate these points to matrix indices
    xbase = tomatrix(trajx, kmin, delta);
    ybase = tomatrix(trajy, kmin, delta);
    zbase = tomatrix(trajz, kmin, delta);

    % start and end points for x,y,z
    xst = max(-Wn/2 + xbase, 1);
    xed = min(Wn/2 + xbase, N);

    yst = max(-Wn/2 + ybase, 1);
    yed = min(Wn/2 + ybase, N);

    zst = max(-Wn/2 + zbase, 1);
    zed = min(Wn/2 + zbase, N);

    % make distance matrix first
    dist = zeros(length(xst:xed), length(yst:yed),
        length(zst:zed));
    for u = xst:xed
        for v = yst:yed
```

```
                    for w = zst:zed
                        grx = kmin + (u - 1) * delta;
                        gry = kmin + (v - 1) * delta;
                        grz = kmin + (w - 1) * delta;
                        dist(u-xst+1, v-yst+1, w-zst+1) =
                            sqrt((trajx - grx)^2 + (trajy -
                            gry)^2 + (trajz - grz)^2);
                    end
                end
            end

            % put into density
            %density(xst:xed, yst:yed, zst:zed) =
                density(xst:xed, yst:yed, zst:zed) +
                KernelValue(beta, W, dist);
            density(xst:xed, yst:yed, zst:zed) =
                density(xst:xed, yst:yed, zst:zed) +
                kern(round((dist + kernwidth / 2 * delta) /
                kernwidth * kernstep) + 1);
        end

        save(densitysave, 'density');
        disp('Saved density data');
    else
        load(densitysave, 'density');
        disp('Loaded density data');
    end

    now = toc;
    disp(sprintf('Finished calculating density.  Time: %d
        seconds', round(now - initial)));


    disp('Begin calculating DCF...');
    initial = toc;

    DCF = zeros(length(traj), 1);
```

```
for j=1:skip:length(traj)
    % Get trajectory position
    trajx = traj(j, 1);
    trajy = traj(j, 2);
    trajz = traj(j, 3);

    xmin = 1 + floor((trajx - kmin) / delta);
    xmax = 1 + ceil((trajx - kmin) / delta);
    ymin = 1 + floor((trajy - kmin) / delta);
    ymax = 1 + ceil((trajy - kmin) / delta);
    zmin = 1 + floor((trajz - kmin) / delta);
    zmax = 1 + ceil((trajz - kmin) / delta);

    grxmin = kmin + ((floor((trajx - kmin) / delta)) *
       delta);
    grxmax = kmin + ((ceil((trajx - kmin) / delta)) *
       delta);
    grymin = kmin + ((floor((trajy - kmin) / delta)) *
       delta);
    grymax = kmin + ((ceil((trajy - kmin) / delta)) *
       delta);
    grzmin = kmin + ((floor((trajz - kmin) / delta)) *
       delta);
    grzmax = kmin + ((ceil((trajz - kmin) / delta)) *
       delta);

    val = [
        grxmin grymin grzmin density(xmin, ymin,
           zmin); grxmin grymin grzmax density(xmin,
           ymin, zmax);
        grxmin grymax grzmin density(xmin, ymax,
           zmin); grxmin grymax grymax density(xmin,
           ymax, zmax);
        grxmax grymin grzmin density(xmax, ymin,
           zmin); grxmax grymin grzmax density(xmax,
           ymin, zmax);
        grxmax grymax grzmin density(xmax, ymax,
           zmin); grxmax grymax grzmax density(xmax,
```

```
            ymax, zmax);
    ];

    for k=1:length(val)
        dist = sqrt((val(k,1) - trajx)^2 + (val(k,2) -
            trajy)^2 + (val(k,3) - trajz)^2);
        DCF(j) = DCF(j) + (sqrt(3) - dist) * val(k, 4)
            / 8;
    end
end

now = toc;
disp(sprintf('Finished calculating DCF.  Time %d
    seconds', round(now - initial)));

disp('Begin gridding data...');

initial = toc;

if ((exist(Asave, 'file')) ~= 2)

    for j=1:skip:length(traj)
        if (mod(j, 1000) == 0)
            disp(sprintf('Finished gridding %d
                points', j));

            if (mod(j, 10000) == 0)
                curtime = toc;
                disp(sprintf('Complete in %d seconds',
                    round((length(data) - j) / j *
                    (curtime - initial))));
            end
        end

        % skip empty values
        if (data(j) == 0)
            continue;
        end
```

```
trajx = traj(j, 1);
trajy = traj(j, 2);
trajz = traj(j, 3);

xbase = tomatrix(trajx, kmin, delta);
ybase = tomatrix(trajy, kmin, delta);
zbase = tomatrix(trajz, kmin, delta);

% Create window around basepoint to grid
% k-space data
xst = max(-Wn/2 + xbase, 1);
xed = min(Wn/2 + xbase, N);

yst = max(-Wn/2 + ybase, 1);
yed = min(Wn/2 + ybase, N);

zst = max(-Wn/2 + zbase, 1);
zed = min(Wn/2 + zbase, N);

% make distance matrix first
dist = zeros(length(xst:xed), length(yst:yed),
   length(zst:zed));
for u = xst:xed
    for v = yst:yed
        for w = zst:zed
            grx = kmin + (u - 1) * delta;
            gry = kmin + (v - 1) * delta;
            grz = kmin + (w - 1) * delta;
            dist(u-xst+1, v-yst+1, w-zst+1) =
               sqrt((trajx - grx)^2 + (trajy -
               gry)^2 + (trajz - grz)^2);
        end
    end
end

datamat = ((data(j) / DCF(j)) *
   ones(xed-xst+1, yed-yst+1, zed-zst+1)) .*
```

```
            kern(round((dist + kernwidth / 2 * delta) /
                kernwidth * kernstep) + 1);
            A(xst:xed, yst:yed, zst:zed) = A(xst:xed,
                yst:yed, zst:zed) + datamat;

            %A(xst:xed, yst:yed, zst:zed) = A(xst:xed,
                yst:yed, zst:zed) + (KernelValue(dist) *
                data(j) / DCF(j));
            %density(xst:xed, yst:yed, zst:zed) =
                density(xst:xed, yst:yed, zst:zed) +
                KernelValue(dist);
            %datamat = (data(j) / DCF(j)) *
                ones(xed-xst+1, yed-yst+1, zed-zst+1);
            %curdata = A(xst:xed, yst:yed, zst:zed);
            %kernel = KernelValue(beta, W, dist);
            %kernel = KernelValue(dist);
            %curdata = curdata + datamat .* kernel;
            %A(xst:xed, yst:yed, zst:zed) = curdata;

%           for u = 1:xed-xst+1
%               for v = 1:yed-yst+1
%                   for w = 1:zed-zst+1
%                       if (DCF(j) == 0)
%                           continue;
%                       end
%
%                       %curdata(u,v,w) = curdata(u,v,w)
    + (kernel(u, v, w) * data(j));
%                       curdata(u,v,w) = curdata(u,v,w)
    + (kernel(u, v, w) * data(j) / DCF(j));
%                       %curdata(u,v,w) = curdata(u,v,w)
    + (kernel(u, v, w) * 1 / DCF(j));
%                   end
%               end
%           end

    end
```

```
    save(Asave, 'A');
    disp('Saved regridded k-space data');
else
    load(Asave, 'A');
    disp('Loaded regridded k-space data');
end

now = toc;
disp(sprintf('Finished gridding data.  Time: %d
   seconds', round(now - initial)));

initial = toc;
disp('Create rolloff correction matrix...');

if ((exist(rolloffsave, 'file')) ~= 2)
    x = zeros(N, N);

    for u = 1:N
        for v = 1:N
            x(u, v) = sqrt((u-N/2)^2 + (v-N/2)^2);
        end
    end

    arg = pi^2 * W^2 .* x.^2 - beta^2;
    ro = sin(sqrt(arg)) ./ sqrt(arg);

    now = toc;
    save(rolloffsave, 'ro');

    disp('Saved rolloff matrix');
else
    load(rolloffsave, 'ro');
    disp('Loaded rolloff matrix');
end

disp(sprintf('Finished.  Time: %d seconds', round(now
   - initial)));
```

```
disp('create FFT, image and phase matrices');

fftA = fftshift(ifftn(ifftshift(A)));
image = abs(fftA);
phase = angle(fftA);

endtime = toc;
disp(sprintf('Total time for reconstruction: %d
    seconds', round(endtime - starttime)));

disp('Done');
```

B.2 rect.m

Listing B.2: rect.m code

```
function r = rect(x, range)
%%%
%  The rectangular function is defined to be 1
%  on [-range, range] and 0 elsewhere
%%%
% initialize output matrix to be all zeros, same size
    as input
r = zeros(size(x));

myset = find(abs(x) <= range);
r(myset) = ones(size(myset));
```

B.3 tomatrix.m

Listing B.3: tomatrix.m code

```
function ret = tomatrix(x, min, delta)
%%%
```

```
% Returns integer index from k-space
% index of the regridding matrix
%%%
    ret = 1 + round((x - min) / delta);
end
```

B.4 tok.m

Listing B.4: tok.m code

```
function ret = tok(x, min, delta)
%%%
% Returns the k-space index from integer
% index of the regridding matrix
%%%
    ret = min + ((x - 1) * delta);
end
```