Physics Tutorial 1: Exercises in Image Formation

Originally written by Karla Miller
Last Modified: Mark Chiew, Nadine Graedel, Tom Okell, Oct 2014

In this tutorial, you will be exploring different aspects of image formation using MATLAB with your tutor. Interspersed throughout the tutorial, there will be questions for you consider. Questions without any asterisks are those that should be attempted by everyone, whereas more challenging questions are marked with one (*) or two (**) asterisks, and should be considered optional. Take these opportunities to think about these questions, and then discuss your answers with your tutor.

At the end of the tutorial period, you will receive a "take–home tutor", which is an annotated version of this tutorial guide that will help you complete the tutorial at home if you don't manage to make it all the way through with your tutor, or if you missed the tutorial session.

Attendance will be taken by the tutors, and marks will be given by participation. If you would like additional feedback or clarification on the tutorial material, you are welcome to submit your questions or comments to Weblearn, and a tutor will provide written feedback for you. Those unable to attend the tutorial must submit answers to all unstarred questions to receive credit for the tutorial.

Part 0 – Getting Started

0.1 Starting MATLAB

Download and unzip the file containing the tutorial resources from Weblearn, or if you have access to the FMRIB internal network, copy them into your current directory from here:

```
~mchiew/GradCourse/1_Image_Formation
```

Start MATLAB, and make sure you're inside the tutorial directory (i.e., the folder containing all the tutorial files).

Note to jalapeno00 users*:* please start with the –nojvm option, "matlab -nojvm"; *this should reduce server CPU load if lots of people are trying to use jalapeno00 simultaneously*

Appendix 1 contains a MATLAB primer. Consider spending 10 – 15 minutes covering basic MATLAB operation.

Part 1 – K-Space Sampling

This section uses data contained in the file `brain.mat` which you can load using:

```
>> load brain.mat
```

1.1 Properties of k-space [10-15 minutes]

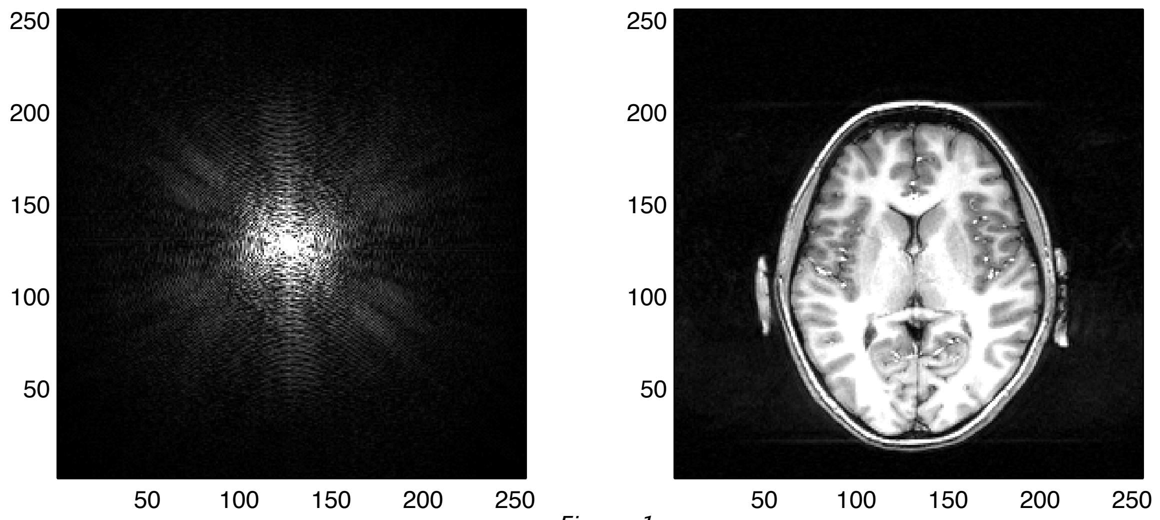

Figure 1

Figure 1 shows an image of a single slice through the brain on the right, and its corresponding k-space magnitude image on the left. The relationship between k-space and image space is governed by Fourier transform properties/relationship.

Consider showing students examples of other image/k-space pairs, even non-brain images, like the mandrill (which you can access with the `load mandrill` command). For more technically inclined students, try generating random noise images (zero mean and non-zero mean) as well. You can demonstrate any of the following properties on any dataset you choose. The `brain.mat` file contains square, circle and Gaussian shapes as well. Note that show_pair might not always scale images sensibly.

Some properties of k-space:

- The central point in k-space (**k**=0) is typically the greatest in magnitude
  Use the data cursor to highlight k-space magnitude at the centre
- The k-space of a magnitude image (or any real image) has (conjugate) symmetry
  Demonstrate the conjugate symmetry by selecting k-space conjugate pairs (remember that generally, the points are symmetric about the `[m/2+1,n/2+1]` point in an m by n k-space. If the students don't know what complex conjugates are, just show them that the magnitudes are symmetric. More advanced students can be told that conjugate symmetry holds only for purely real images.
- Shifting an image around has no effect on the magnitude of k-space (and vice versa)
  Shift an image by using the `circshift` function, and show that the magnitude of k-space is identical with respect to any and all shifts. Try shifting both dimensions!
- The size of a k-space matrix is the same as the size of the image matrix
  It should be sufficient to point out that the output of ifft2c is always the same size as the input. For more advanced students, you can point out that the discrete Fourier transform is a linear, invertible transform (representable by a square matrix) and is unitary under the right scaling factors. This means that the input and output vectors are always the same size.

Question 1.1.1*
What do you think happens to k-space when you *rotate* an image, instead of shifting it?
Simple answer is that rotates objects or images produce k-spaces rotated at the same angle. Why is this the case? More technical students may find it interesting to learn about the Projection Slice Theorem.

Question 1.1.2**
Describe an image that could have a k-space that does not peak in magnitude at **k**=0
This is impossible for magnitude data. However, in reality, data are actually complex (well not really, but the magnetization in each voxel is a 2D vector with a magnitude and phase,

and we use complex numbers to represent this), and complex data can have zero-mean. Since the central point of k-space represents the unweighted complex sum across all voxels, any object with the right phase variation will have low central k-space magnitude. Data that have been demeaned will show a null k-space centre as well.

1.2 Δk and Field-of-View (FOV) [15-20 minutes]

$$FOV \propto \frac{1}{\Delta k}$$

The image field-of-view (FOV), and the spacing between k-space points have an inverse relationship. That is, *reducing* the space between k-space samples *widens* the FOV, and *increasing* the space between samples *shrinks* the FOV.

It is important to note that Δk is a property of the sampling performed by the imaging pulse sequence (as you'll see in Part 2), and not something that is evident from the k-space matrices themselves. That is, two k-space matrices with the same number of rows and columns can have very different Δk, and hence represent images that have very different FOVs.

We can simulate the effect of changing Δk by sub-sampling some k-space data. For example, selecting every other point in k-space effectively doubles Δk, which should halve the effective FOV in the corresponding image.

Demonstrate the effect of selecting every other line in k-space.
```
>>   k_sub = k_data(:,2:2:end);
>>   x_sub = ifft2c(k_sub);
>>   show_pair(k_sub, x_sub);
```
For more advanced students, try even lines, and then odd lines and then ask students if they expect a difference in the resulting image. Point out that the even and odd sampling are shifted versions of each other, and that phase variation along the shifted sampling direction can cause signal loss due to phase cancellation in aliased regions.

Question 1.2.1
What happens to the image FOV when every third line is selected? Or every fourth?
It should be apparent that the FOV shrinks proportionally to Δk.

Extra Information
In reality the physical object in the scanner obviously does not suffer from the overlap that you see in the reduced FOV images. Clearly then, the FOV does not actually refer to an actual physical interval over which the image is captured, otherwise you would not see the "aliasing", or unwanted overlap. What the FOV actually refers to, and what Δk actually controls, is the distance between successive *copies* of the image. This distance dictates the extent over which points in space are uniquely spatially encoded.

To illustrate this, try illustrating (on paper or in MATLAB) the relationship between the copy distance (i.e. the FOV) and the Δk parameter. It should become apparent that the distance between successive copies is also the width of the image over that contains unique (non-redundant) information. Also, it should become clear that this window can be arbitrarily placed without losing any information (compare this property to Q0.2.2).

This is why, practically speaking, the phase-encoding FOV must be set at least as large as the object is in that direction, because you cannot exclude portions of the object by modifying sampling alone.

You may need to remind students that one of the defining differences between phase-encoding (PE) and frequency-encoding (RO) is that in the PE-direction a gradient "blip" is *used to prepare a different fixed phase offset for each physical position, whereas in the RO-direction, a constant gradient is used to impose a position-dependent resonant frequency. Do note that while this difference exists, it is mostly a practical difference, and conceptually successive points in the RO-direction represent successive position-dependent phase offsets. This can be easily seen by considering the "constant" RO gradient as a series of "blips" with no gap.*

Question 1.2.2*
We typically only observe aliasing to occur along the phase-encoding direction. Given what you know about the difference between how the phase-encoding and frequency-encoding (readout) data are collected, why might this be?
Practically speaking, because data along the readout direction are acquired continuously, anti-aliasing filters can be applied that reject frequencies outside a certain band corresponding to the portion of the object that you want to record signal from. This means it is trivial to prevent aliasing in that direction, hence the term anti-aliasing filter. This means of signal rejection cannot be applied in the phase-encoding direction because phase data variations are measured discretely (in consecutive lines), and it is impossible to distinguish between different rates of phase accrual (frequencies) beyond what your discrete sampling rate permits. Note that this is precisely the classical case of aliasing according to Shannon-Nyquist sampling theory. Enterprising students may recognize the Δk as a sampling period, and the FOV as the maximum "bandwidth" permitted by such sampling.

1.3 k_{max} and Resolution [15-20 minutes]

$$\Delta x \propto \frac{1}{k_{max}}$$

If Δk is the distance between samples in k-space, then k_{max} is the maximum extent to which k-space is sampled. What k_{max} represents is the highest spatial frequency measured, which in turn dictates the resolution (or voxel size) in the corresponding image.

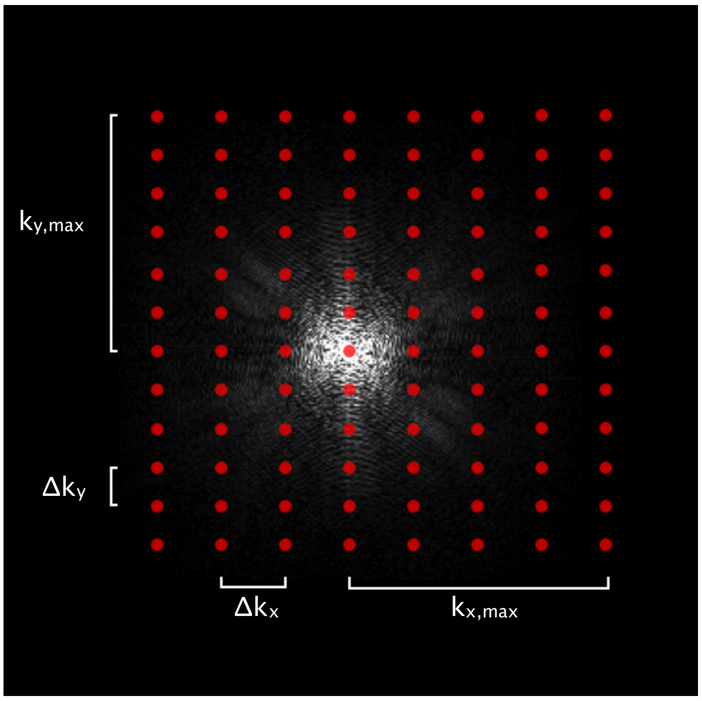

Figure 2

Figure 2 illustrates how the continuous and infinite 2D k-space is discretely sampled to form a 2D image. Different sampling parameters along the x- and y-directions lead to different image properties along these directions.

Spend some time exploring Fig. 2. In particular, you might want to note that our k-space matrices are only a discrete sampling of a continuous and infinite k-space. Changing our sampling properties only changes what portion of k-space we view.
For more advanced students, you may want to also note the asymmetric sampling of k-space, which is simply a convention in MRI, not a fundamental necessity.

We can again simulate the effect of sampling to different values of k_{max} by sub-sampling a pre-sampled k-space, and observe how the image changes with the different sampling patterns.

Consider various subsets of k-space, including restricted windows along x, y and both x & y
```
>>  k_subx   = k_data(64+(1:128), :);          % central 128 x-points
>>  k_suby   = k_data(:, 64+(1:128));          % central 128 y-points
>>  k_subxy = k_data(64+(1:128), 64+(1:128));   % central 128x128 square
```
Make sure to explore the produced images, and make it clear that things that affect k_x sampling affect the x-direction of the image, and similarly for k_y and the y-direction.
Set the 3rd optional parameter of `show_pair, fov_square` to 1 in order to force the second subplot to a square FOV.
```
>> show_pair(k_subx, ifft2c(k_subx), 1)         % square image FOV
```

An important thing to remember is that the resolution in an image is related to the amount of spatial frequency content sampled in k-space, and not simply the number of pixels or voxels in the image. For example, we can "zero-pad" our k-space to any desired matrix size. In zero-padding, empty or zero-filled k-space entries are generated to pad out the matrix, without adding any information about the sample. The resulting image is has more pixels, but no additional resolution.

In the previous examples, it is "obvious" to the student that the sampling produces lower resolution because of the lower matrix size, or "bigger pixels". Generate some zero-padding examples, and point out that because k_{max} does not change with zero-padding, image resolution does not change, despite having "smaller pixels".
```
>> k_xy_zeropad = zeros(256);
>> k_xy_zeropad(65:192,65:192) = k_subxy; % 128x128 square zero padded
```

For the following question, it may be useful to show students how to change the windowing on their MATLAB figures.

Question 1.3.1
Consider a zero-padded image. What features do you notice appearing near the sharp edges and boundaries? What are these features called, and what causes them?
Close inspection of the images should reveal ringing artefacts at boundaries and edges. These artefacts are called "Gibbs ringing", and are caused by sudden truncation of the k-space data. Two ways to alleviate Gibbs ringing are acquisition of a greater extent in k-space (higher resolution), or apodization (multiplying or filtering your k-space signal to force it down to zero at the edges).

Extra Information
Another feature of sampling that affects resolution is the total width of the k-space sampling window. Two sampling patterns with identical k_{max}, but different sampling windows can have different effective image resolutions. Larger sampling windows lead to higher image resolutions. Consider the extreme example of 2 k-spaces, one with *N* samples extending to k_{max}, and the second with a single sample at k_{max} itself. While both sampling patterns have the same k_{max}, the second has an infinitesimally small window (a single point), and conveys no spatial information (along that direction).

This may need some additional explanation or diagrams, depending on how in-depth you want to go. Ultimately, discussion of resolution can be related to two things:

i. The distance between image sampling points (pixel size), which is related to k_{max}

ii. The amount of spatial information encompassed by each sampling point (point spread function), which is related to the width of the sampling window

These things are normally not separate concerns, because in conventional imaging, $k_{max} = (N/2)*\Delta k$, and and window size $W = N*\Delta k$, so that $k_{max}=W/2$ and changing one changes the other one to match. This way, pixel samples that are 1 mm apart also have point spread functions of exactly 1 mm.

We can violate this relationship in a number of ways, however, like in zero-padding. There, we make $k_{max} \neq N*\Delta k$ (where k_{max} is the greatest non-zero k-space position), and so while the distance between samples decreases, the point spread function does not. For example, consider pixels that are now 0.5 mm apart, but still have point spread functions with width 1 mm. The resolving power is still restricted by the 1mm point spread, despite more finely spaced samples.

Question 1.3.2
Since k-space sampling takes time, we would often like to reduce the amount of sampling required to form an image to reduce acquisition times. What property of k-space can be exploited to enable reduced k-space coverage without a corresponding loss of image resolution? What is this technique called?
The property of k-space from section 1.1, that states that MR images can have conjugate symmetry (under certain conditions). With this property, we can collect half of k-space and use the symmetry property to fill in the other half without needing to measure it. This technique is called Partial Fourier (reconstructed with conjugate synthesis).
For students that are aware that conjugate symmetry only holds for purely real images, you might discuss that conjugate synthesis is not actually for reconstruction in practice, and that zero-filling is actually a common and robust reconstruction method, although one that does cost spatial resolution.

Part 2 – K-Space Trajectories

This section uses data contained in the file `epi_traj.mat`

2.1 Gradient Waveforms [10-15 minutes]

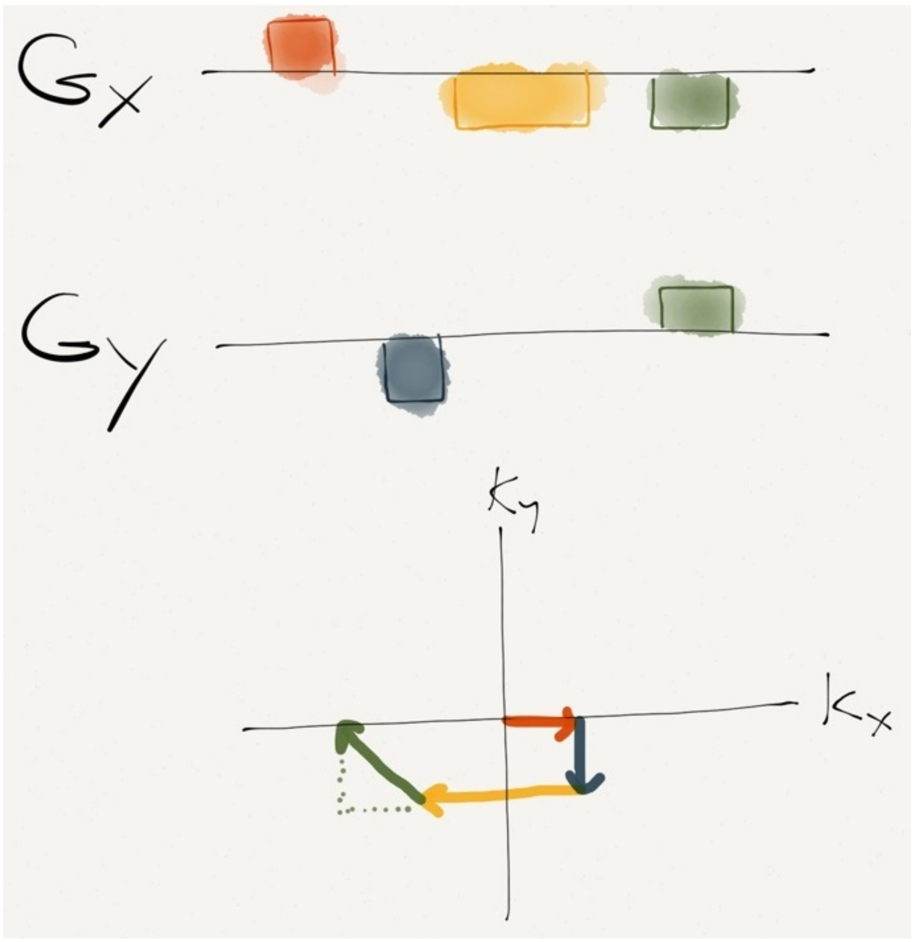

Figure 3

Sampling in k-space is achieved through the use of magnetic field gradients. Mathematically, the position you occupy in k-space is equal to the net area under the magnetic field gradient waveforms. This rule dictates the position along each direction independently, i.e., the x-gradient controls k_x position, the y-gradient controls k_y. Note that these waveforms are not what the magnetic fields look like, rather they are the time-varying amplitudes of the spatially linear gradient magnetic fields.

Walk through Fig. 3 step-by-step. Explain how positive area under the G_x waveform moves in the moves your sampling point in the positive k_x-direction (rightwards), negative area moves you in the negative k_x-direction (leftwards), and no gradients leave you in the current k_x-position. Similarly for G_y and k_y. Take some care to explain that the gradient waveforms they see in pulse sequence diagrams are not representations of what the magnetic fields look like. A useful analogy may be like driving a car across k-space, where the magnitude of G_x represents the strength of the x-gradient that moves you in k-space along the x-direction, and the polarity dictates whether you're moving in forward or reverse.

*Furthermore you may explain that G_x is like a velocity term, and the distance you cover in k-space is velocity*time (i.e., the area under the velocity curve). For advanced students, you may explain that in addition to speeding up the rate at which you cover k-space, increasing gradient strength also literally increases the spread in magnetic field magnitude, which corresponds to increasing the spread of resonant frequencies (also known as the bandwidth). This is why the speed of gradient traversal through k-space is controlled by the bandwidth parameter on the scanner, as it implicitly controls the strength of the gradient. Higher bandwidth = faster sampling. Next tutorial, students will learn more about bandwidth and its relationship to SNR.*

Question 2.1.1

Consider the G_Y gradient waveform provided (see Fig. 4), and divide its amplitude by 2. What would result from a scan that used this new Gy gradient, in terms of k-space and the resulting image FOV and resolution?
Scaling the G_Y gradient by half means the speed of traversal in the y-direction is halved. That means for the same amount of travel time, you only go half the distance, and k-space would be compressed by a factor of 2 in the k_Y-direction.
While the number of lines N stays the same, both Δk and k_{max} are also halved. This means that the FOV in the y-direction doubles and so does the size of the pixels.

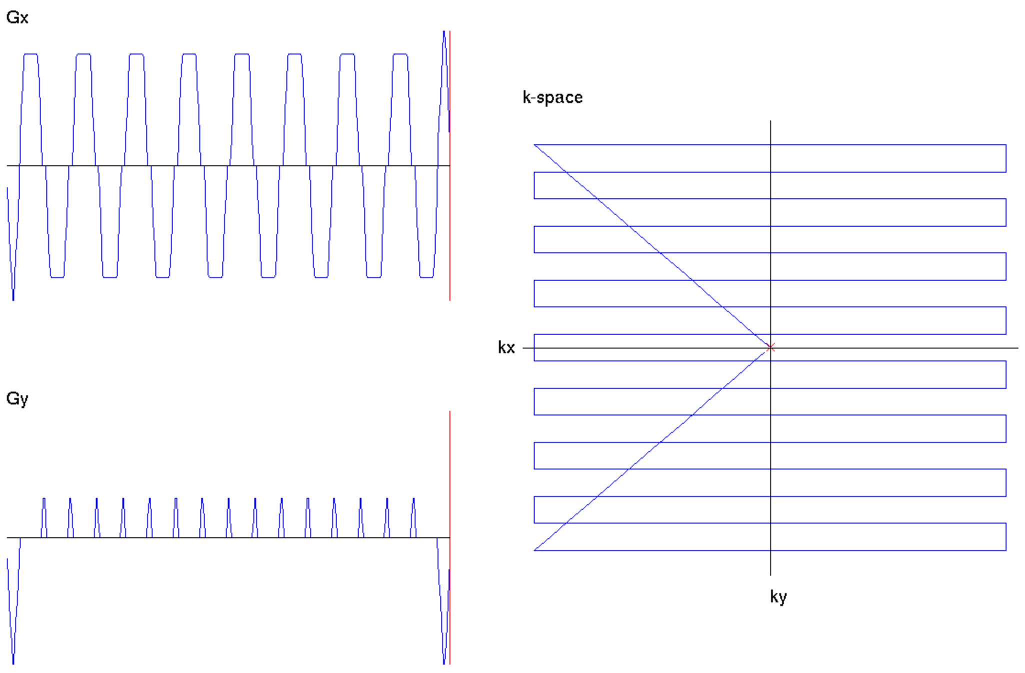

Figure 4

Nothing in the k-space formalism requires you to travel in straight lines across k-space. In fact, although the geometry of the k-space sampling patterns theoretically completely dictates image properties, in practice, the way you go from sample-to-sample (i.e. the trajectory) in k-space also contributes to different image features or artefacts. Spiral trajectories, for instance, cover 2D k-space in a completely different way than standard (Cartesian) trajectories do.

Consider generating some different geometric trajectories, or even random trajectories:
```
>>  Gx_sp = (1:1000).*cos((pi/100)*(1:1000));   %simple spiral x-gradient
>>  Gy_sp = (1:1000).*sin((pi/100)*(1:1000));   %simple spiral y-gradient
>>  show_traj(Gx_sp, Gy_sp);
>>
>>  Gx_rand = randn(1, 1000);       %simple stochastic x-gradient
>>  Gy_rand = randn(1, 1000);       %simple stochastic y-gradient
>>  show_traj (Gx_rand, Gy_rand);
>>
>>  show_traj (Gx_sp/1000, Gy_rand);      %hybrid trajectory
>>
>>  a = 17;
>>  b = 23;
>>  t = 0:0.005:4;
>>  G_rose = (a+b)*exp(1j*(a+b)*t)+(a-b)*exp(-1j*(a-b)*t); %simple rosette
>>  show_traj(real(G_rose), imag(G_rose));
```

In the discussion of different gradient trajectories, you may want to discuss various practical considerations of different trajectories, such as whether they require gridding, gradient slew-rate limitations, maximum gradient amplitude, efficiency, etc.

Extra Information
This relationship between gradient waveform amplitude and k position is not a neat trick or coincidence, but is actually linked to how k space is defined. The time average gradient waveform dictates the spatial frequency of a sinusoidal signal variation, and moving around in k space amounts to imposing various different sinusoidal patterns on the object before sampling. Remember that data is acquired as a sum of the signals across the entire object (or slice) to produce a single number. If we did not impose different gradients prior to sampling, that number would never change, and we would not be able to spatially resolve objects at all. Technically, the signal variations we impose on our object need not be sinusoidal (although they must satisfy other mathematical properties), but fortuitously, we find that the sinusoidal variations produced by linear gradients correspond exactly to Fourier basis functions, which enables us to use the Fourier transform to reconstruct our images easily.

Question 2.1.2**
Compare MRI acquisition with x-ray and CT (computed tomography) imaging. Why are you unable to get 3D data from a simple x-ray? What feature of CT imaging allows 3D information to be recovered? How might you replicate an x-ray projection type image using MRI? *NB. If you are unfamiliar with CT, it is essentially a scanner that obtains multiple x-ray images at different angles around the subject.*
You cannot extract 3D data from a simple x-ray because it is a projection of the object along the beam axis. All you can know is the mean attenuation along the beam. You need information from different projections, or different angles in order to extract 3D information (i.e. information parallel to the beam axis), which is what CT provides with its multiple views.
From the projection slice theorem, you know that the 1D Fourier transform of the projection of an object at some angle θ is equal to the 2D Fourier transform of the object evaluated along a line at the same angle θ. This means to collect that particular projection, you need to traverse k-space along that angled trajectory, which can easily be achieved with the right ratio of G_x and G_y gradients.

Appendix 1 – MATLAB Primer

A.1 Some Useful MATLAB Commands

Load `brain.mat` `epi_traj.mat` and demonstrate the use of the following 4 commands on the data in `k_data`, `Gx` and `Gy`.

`ifft2c`	This function calculates the inverse fast Fourier transform (FFT) of a 2D matrix (e.g. a 2D image). For example, the following calculates the inverse FFT of a 2D matrix k and puts the result in the 2D matrix `im`: `>> im = ifft2c(k);`
`show_pair`	Displays a pair of matrices side-by-side. For example, the following will show `im1` on the left and `im2` on the right: `>> show_pair(im1,im2);`

`zeros`	Creates a matrix of a specified size containing all zeros. For example, to create a matrix of zeros, **A**, with 64 rows and 32 columns: `>> A = zeros(64,32);`
`show_traj`	Given gradient waveforms, displays both the waveforms and the corresponding k-space trajectory. For example, to display the trajectory traversed by waveforms Gx and Gy: `>> show_traj(Gx,Gy);`

Question A.1.1*

Given that `ifft2c` is the *inverse* fast Fourier transform, what do you think the forward fast Fourier transform (`fft2c`) does?

The forward transform takes an object from "image space" into its corresponding k-space representation. In effect, the MRI measurement is a "physical" forward transform, in that it takes an object and returns its k-space.

For the following question, encourage students to try running fft2 vs. fft2c on some data, and observe the difference. You could also sketch a diagram of the periodic nature of discrete k-space, and how the k-space "window" is positioned differently in the centric vs. non-centric cases.

Question A.1.2*

The "c" in `(i)fft2c` stands for "centric". What is the difference between the centric transforms, and the built-in default `(i)fft2` MATLAB functions?

The centricity refers to where the k=0 position is located in MATLAB's representation of the k-space data. Because the discrete Fourier transform is periodic, there is no difference data-wise whether k-space is represented as k=0,1,2,…,N-1 or as k=-N/2, -N/2+1,…,0,…,N/2+1. The centric transforms use the latter convention, whereas the default non-centric transforms use the former.

0.2 MATLAB matrix indexing

Generate some random test matrices and demonstrate the following indexing rules:

- *`M = randn(16);        %random 256x256 matrix`*
- *`M(3,11)               %element in row 3, column 11`*
- *`M(3,:)                %elements in row 3`*
- *`M(:,11)               %elements in column 11`*
- *`M(:,3:2:13)           %odd columns between 3 and 13`*
- *`M(1:4:end,:)          %every 4th row`*
- *`M(17)                 %first entry of 2nd column`*

You may or may not find it necessary to explain that the `end` keyword replaces the size of that dimension.

Also that linear indexing works by counting along the row index first, starting at the left-most coiumn and working your way right.

Accessing a single element	The element of a matrix M, in the ith row and jth column is M(i,j)
Accessing a row or column	To refer to the (entire) ith row or jth column, MATLAB uses the shorthand: M(i,:) or M(:,j). The colon (:) represents "all elements along this dimension"
Accessing intervals	You can create an index for matrices with the notation `min:step:max`, which creates an array of numbers between

min and max in increments of step. For example, to access the odd columns between 30 and 40 in a matrix M, you would use M(:,31:2:39).

Encourage students to use the internet or MATLAB's built in help to perform operations that are not obvious to them.

Question A.2.1*

If `M` is a square matrix, how would you get all the entries along the diagonal in a single command?

This question serves mostly to illustrate that `M(1:end,1:end)` does not work. MATLAB slicing works by "reading" the row and column indices independently, not at the same time. Therefore the above command returns the entire matrix. `diag(M)` works well for this.