EEE473-573 Medical Imaging Project Paper
MRI Simulator

Ayhan Okuyan
21601531
Bilkent University
Ankara, Turkey
ayhan.okuyan@bilkent.edu.tr

Can Bayar
21602767
Bilkent University
Ankara, Turkey
can.bayar@ug.bilkent.edu.tr

Contents

Abstract

YouTube Link: https://www.youtube.com/watch?v=Bm7a1uIMPu0
In this project, we have devised an MRI simulator that simulates the MRI image of a discrete anatomical model from its transverse axis, using many non-idealities. We have followed the three stages of MRI imaging, namely the Slice Selection, k-Space Acquisition and Reconstruction. For slice selection, we have used a non-realistic slice selection in order to decrease the computational power required. We have used a dynamic slice selection procedure, and slice thickness so that the observations could be observed more clearly. For slice selection, we have simulated two imaging sequences as the Gradient Echo (GRE) and the Spin Echo (SE). For the reconstruction, we have applied a two-dimensional inverse Fourier transform and used various contrast models (T_1, T_2, T_2^* and Proton Density weighted). Furthermore, we have wrapped the implementation around a GUI interface to make the various hyperparameters easily configurable.

1. Introduction

Magnetic Resonance Imaging (MRI) has been one of the most important medical imaging techniques, since its discovery, due to its configurable and the ability to show contrast between soft tissue elements, unlike radiography-based techniques such as X-Ray or CT imaging. The underlying theory is based on the Nuclear Magnetic Resonance (NMR) that explains the magnetization of the odd numbered elements (mostly Hydrogen) when a magnetic field is applied on them.
In account of all these, we have created an MRI Simulator that accepts a discrete anatomical model of the human brain and the magnetic resonance parameters of separate these various types of tissues and generates a realistic MRI image. For the anatomical model, we have used the Brain Web Dataset [2]. For slice selection we have chosen to build non-realistic sinc pulse since it is easier to configure the tip angle parameters. Furthermore, we have built a k-space acquisition algorithm, that doesn't assume short data acquisition window principle. We have configured many realistic parameters and non-idealities, namely the B_0 variation, the chemical shift, the proton density, the T_2 decay. We have simulated the gradient echo and spin echo pulse sequences with two and three types of contrasting options, with realistic parameters. Then, we have used MATLAB's GUIDE API to build a UI application that enables us to use many configurations that we introduced into the algorithm.

2. Methods

2.1. Dataset

For the dataset, we have taken advantage of the Brain Web dataset [2] that provides anatomical models for both the normal brain and brains with various MS legions. We have preferred this dataset since it contained the tissue information regarding the T_1, T_2, chemical shift and the proton density values of the respective tissues. The models, we have used are discrete models that are given the majority label to a $1x1x1mm$ voxel. The tissue types used in the data are given as Background, Cerebrospinal Fluid (CSF), Grey Matter, White Matter, Fat, Muscle/Skin, Skin, Skull, Glial Matter, Connective Tissue and the MS Legions. The parameters can be found through . For the chemical shift properties, we have discovered that the shift happens mostly in the environments with a high fat density, hence, we only used a chemical shift value of $3.5ppm$ in the voxels labeled as 'Fat', which are measured in a $1.5T$ MRI machine.
To load the minc files, which is the format of the anatomical data used, we have used the **loadminc** function of Laszlo Balkay [1]. All other code, is written by us from scratch.

2.2. Slice Selection

For slice selection, we have used a non-realistic approach in the form of an infinite RF pulse. While this is not the most comprehensive approach, we have done it such that the resulting images can be observed better in contrast. Furthermore, this approach enabled us to declare the α tip angle more easily, as we have adjusted the tip angle according to the contrast mechanism that we have used for imaging. The RF pulse used is given as follows.

$$B_1(t) = A\Delta\nu sinc(\Delta\nu t)e^{-2\pi j\bar{\nu}t} \tag{1}$$

This envelope carrier combination results in a $\alpha(\nu)$ that can be presented as follows.

$$\alpha(\nu) = \gamma \int B_1^e(t)e^{-2\pi j\bar{\nu}t}e^{2\pi j\nu t}dt \tag{2}$$

$$= \gamma Arect\left(\frac{\nu - \bar{\nu}}{\Delta\nu}\right) \tag{3}$$

where $\bar{\nu}$ is the frequency center and $\Delta\nu$ is the frequency range. Then, this slice information can be embedded into the z-axis using the z-gradient since, ν is defined as follows,

$$\nu = \bar{\gamma}(B_0 + G_z * z) \tag{4}$$

where the $\bar{\gamma}$ is the gyromagnetic ratio of H_1^1, equal to $43.58MHz/T$. Then, we can represent the $\alpha(z)$ as follows.

$$\alpha(z) = \gamma Arect\left(\frac{\bar{\gamma}(B_0 + G_z * z) - \bar{\gamma}(B_0 + G_z * \bar{z})}{\bar{\gamma}(B_0 + G_z * \Delta z)}\right) \tag{5}$$

[1] https://www.mathworks.com/matlabcentral/fileexchange/32644-loadminc

Where $\bar{z}$ is the slice center and the Δz is the slice thickness. Here, we can see that the frequency range that, we can work is from γB_0 to $\gamma(B_0 + G_z * Z_{max})$, where Z_{max} corresponds to the number of slices for our data, which is $181mm$. Then, we have used a scale factor to determine the slice center, and used the odd values of $1, 3, 5$ and $7mm$ as slice thickness options.
For the tip angle, we observed that in [2], for different contrast mechanisms, different tip angles should be applied. For spin echo imaging, we have used a tip angle of $\alpha = \pi/2$, whereas for the T_1 and T_2^* weighted gradient echo imaging, we have used a tip angle of $\pi/9$ degrees that corresponds to $20°$. Hence, we have defined as A as $\frac{\pi}{2\gamma}$ or $\frac{\pi}{9\gamma}$ to be able to capture the right angle.
Furthermore, we have not used the variation on the static B field and the effects of chemical shift in the slice selection phase in order to work with geometrically uniform slices.

2.3. k-Space Acquisition

After the selection of slices, we have implemented a k-space acquisition algorithm that is based on the . The k-space acquisition algorithm uses G_x and G_y pulses that are simulated over time based on the selected Spin Echo sequence, it then generates the received MRI k-space data using the simulated pulses in order to sample the entirety of the k-space.

We have used the following equation in order to compute the k-space data.

$$s_0(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)e^{-2\pi jk_x(t)x}e^{-2\pi jk_y(t)y}dxdy \tag{6}$$

Where $f(x, y)$ is the calculated received MRI signal. The computation of this signal depends on a few non-idealities namely, T2(or T2*) decay, chemical shift, proton density and simulated noise. The computation of this signal can seen below.

$$f(x, y) = M_0 \frac{e^{-\frac{t}{T_2^*}}(1 - e^{-\frac{TR}{T_1}})}{1 - cos(\alpha)e^{-\frac{TR}{T_1}}} \tag{7}$$

Where α is the tip angle and M_0 is the magnetization which can be found using,

$$M_0 = \frac{\hat{B_0} * \gamma^2 * \hbar^2}{4 * k * T}PD \tag{8}$$

where, γ is the gyromagnetic ratio, $\hbar$ is the Planck's constant, k is the Boltzmann's constant and T is the temperature in K, while PD stands for the proton density. For our simulator, we have used $300K$, which is equal to $27°C$, around the room temperature. Also, $\hat{B_0}$ denotes the chemical shifted B_0

2.3.1 Gradient Echo (GRE) Sequence

Gradient echo sequence is the readout sequence that enables us to read the k-space data line by line. It is used such that at each repetition, the y and x gradients are used to move to the starting point of the read sequence. Then the x gradient is applied to traverse in parallel to the x-axis from the starting point while the ADC is open, meaning the data acquisition is done. A representative pulse sequence graph is provided in Figure 1 [4].

For GE, we have provided two main choices of contrasting namely the T_1 and T_2^* contrast. For the T_1 contrasting, we have set the parameters TE, TR as $14, 500ms$ and α tip angle to $\pi/2$ and for the T_2^* weighting, we have used the TE, TR as $14, 500ms$ and adjusted the α tip angle to $\pi/9$.

2.3.2 Spin Echo (SE) Sequence

Spin echo however, is more complex-in terms of application since for a $\pi/2$ RF pulse, a π RF pulse is applied at $TE = TR/2$, where TE is the echo time and TR is the repetition time. This reverses the effects of T_2^* decay by refocusing the magnetization on the same axis but in the opposite direction. The most important part of spin echo is that when applied, the decay that we see is in the form of T_2 decay, which enhances the image quality. This is a longer and more intricate process in the MRI system than gradient echo [3, 5], however from a black-box point of view, it is easy to simulate. We have used the same mechanism as we have constructed for the gradient echo design, however, we have adjusted the imaging parameters to use the T_2 decay and also adjusted the α tip angle accordingly. A sample pulse sequence diagram that illustrates the operation is provided in Figure 2 [3].

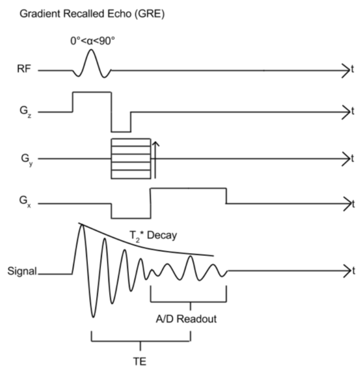


Figure 1: Gradient Echo (GRE) Pulse Sequence Diagram

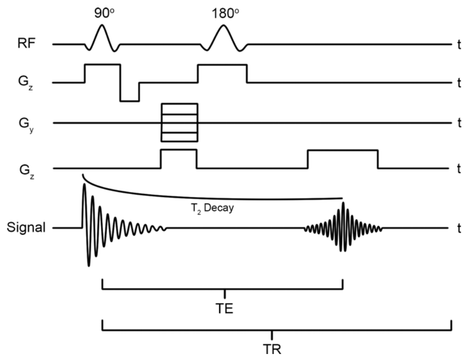


Figure 2: Spin Echo (SE) Pulse Sequence Diagram

The overall MRI Sequence parameters that we have used can be found below in Table 1. We have selected them to be in compliance with the lecture notes and the [1].

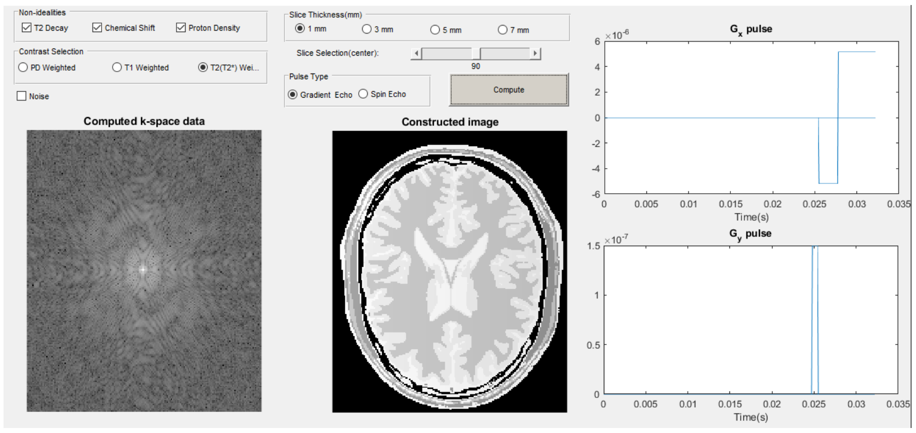
	Gradient Echo (GRE)		Spin Echo (SE)		
	T_1 weighted	T_2^* weighted	T_1 weighted	T_2 weighted	PD weighted
α	$\frac{\pi}{2}$	$\frac{\pi}{9}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$
TR (ms)	500	4000	500	4000	6000
TE (ms)	14	30	14	100	14

Table 1: Used MRI Sequence Parameters

2.4. Reconstruction

For the image reconstruction, we have configured a fairly simple approach. Since, we have acquired a k-space data, in a rectangular format, which corresponds to a Fourier domain, we have used a 2D inverse Fourier transform to obtain the image in Cartesian coordinates. For our first iteration, we have constructed and used our own function that implements as follows.

$$f(x,y) = \frac{1}{FOV_x FOV_y} \sum_v \sum_u f(u,v) e^{2\pi j\left(k_x \frac{v}{FOV_y} + k_y \frac{u}{FOV_x}\right)} \tag{9}$$


Non-idealities: [✓ T2 Decay] [✓ Chemical Shift] [✓ Proton Density]

Contrast Selection: PD Weighted, T1 Weighted, [✓ T2(T2*) Wei...]

Noise: ☐

Slice Thickness(mm): [✓ 1 mm] 3 mm 5 mm 7 mm

Slice Selection(center): 90

Pulse Type: [✓ Gradient Echo] Spin Echo

Figure 3: UI Overview of the MRI Simulator

where, $f(u,v)$ represents the k-space data. Here we can clearly observe that this is an inverse 2D discrete Fourier transform, hence, we have decided to use the MATLAB's own *ifft2* function instead in order to improve the algorithm in terms of execution speed. Note that, with slice thickness greater than $1mm$, we have processed each slice independent from each other.

2.5. GUI Application

After the individual implementations, we have created a UI-based application that enables the user to choose the data, the slice selection parameters, the k-space sequence, and the contrast mechanism. Also we have made it such that the generated x and y gradients, the k-space image and the reconstructed image can be observed. The configuration is presented in Figure 3.

3. Results

For the observation of our results, we have run an ablation study, where we have observed the the similar conditions with the absence of the controlled variable and questioned its effectiveness in the algorithm. Since we have used many non-idealities, we have looked at each of them separately and placed some of the results through discussion.

3.1. Static and Spatially Varying B_0 Magnetic Field

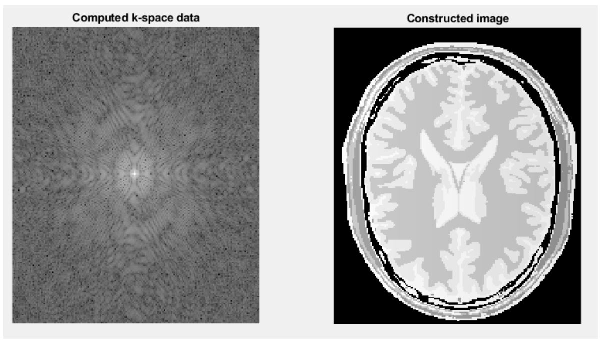
From lectures, we have known that the magnetic field of B_0 is hard to construct and is the primary reason behind the cost of MRI machines. However, we have also learned that there is $1ppm$ variation allowed in each mm^3. Hence, we have incorporated the same functionality in our own implementation, by adding a Gaussian noise to each voxel with 0 mean and $1.5\mu T$ variance. we have illustrated the process where we use and don't use the noise component is presented in 4. By looking at the figure, we are able to observe that the pixels appear to be more-realistic.

3.2. Effect of T_2 Decay

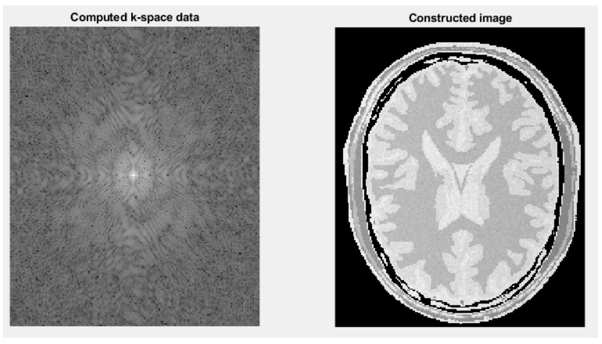
Also, we have learned that T_2 is one of the most important aspect of MRI imaging, and is one of the sources for creating contrast in the image. While this is more obvious, without the implementation of a T_2 or T_2^* decay, we have expected a black and white image with less contrast since the resulting image will only cover the effects of T_1 decay. The governing imaging equation then can be presented as follows.

$$f(x,y) = M_0 \frac{1 - e^{-\frac{TR}{T_1}}}{1 - cos(\alpha)e^{-\frac{TR}{T_1}}} \tag{10}$$

which is the steady-state magnetization. Here, we present another case, where we show and hide the information governed by T_2. The results are shown in Figure 5. The difference is clearly observable as T_2 decay is an important aspect of MRI imaging.

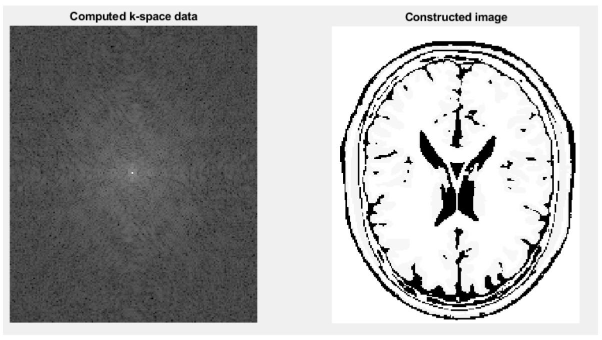


(a) No Noise

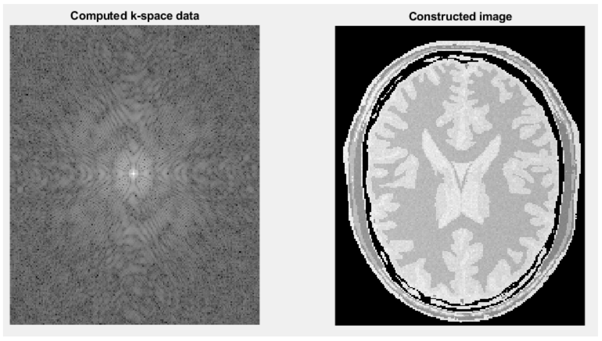


(b) Noise

Figure 4: Reconstructed MRI images with static B_0 and spatially varying B_0



(a) Without T_2 Decay



(b) With T_2 Decay

Figure 5: Reconstructed MRI images with static B_0 and spatially varying B_0

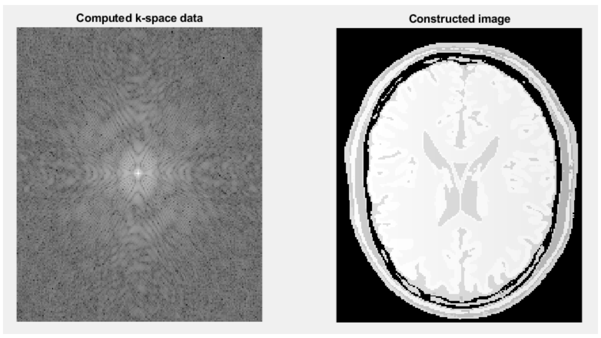
3.3. Effect of Chemical Shift

Chemical shift happens due to the bonding of H atoms in fat structures, since H comes into contact with many type of bonding in these structures, changing its Larmor frequency. When its fundamental frequency changes, we observe the change given below.

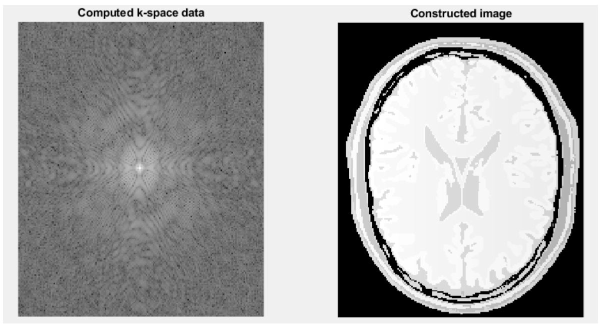
$$\hat{\nu_0} = \nu_0(1-\xi) \tag{11}$$

$$\hat{B_0} = B_0(1-\xi) \tag{12}$$

where ξ is named as the chemical shift, since it causes a spatial shift in the reconstructed image. Hence, we have also used this non-ideality in our design for the k-space acquisition, and we present the results with and without the chemical shift in Figure 6.



(a) Without Chemical Shift



(b) With Chemical Shift

Figure 6: Sample images with and without Chemical Shift

From this figure, the difference, is hard to observe except the subtle changes in the k-space data, however, in Figure 7, we show the difference between these two images, which makes it easier to understand. Here, we can observe the spatial shift in detail.

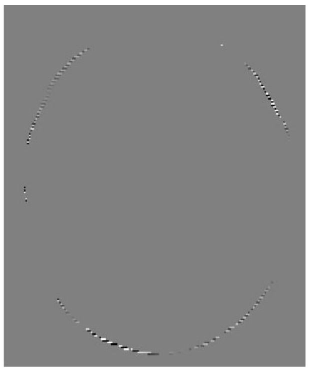


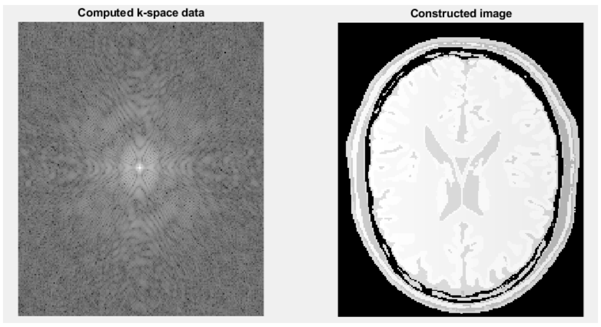
Figure 7: Difference between the images in Figure 6

3.4. Effect of Proton Density

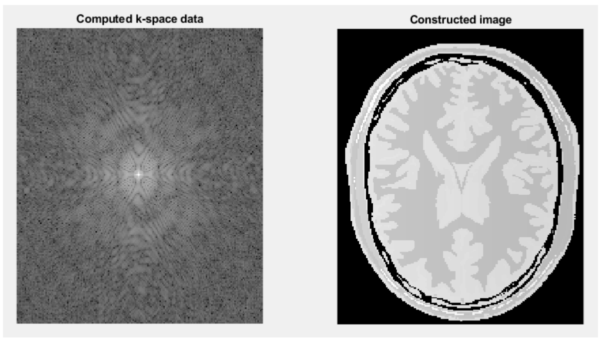
Proton Density is the term that is used to denote density of hydrogen particles inside a tissue voxel of $1mm^3$. As we have observed in Eq. 8, the magnetization largely depends on the presence of proton density. As obvious as it seems, we have neglected the direct effect of proton density, which is why we have included it as a non-ideality. Figure 8 shows a pair of T_2 weighted images with and without the use of proton density in the imaging equation. From figure, we can easily observe that the image without the proton density is way worse in terms of contrast and the it is nearly impossible to differentiate between the white and gray matter.

3.5. Effect of Slice Thickness

The anatomical data we have used had voxels of size 1x1x1 mm. In our program we have allowed users to select transversal slices with variable thickness of size 1, 3, 5, 7 mm. During computation we compute the reconstructed images of thickness 1 mm and then average the resulting images in order to get the desired thickness. As seen in Figure 9 the reconstructed image where a thickness of 5mm was selected is more blurry and the edges in the image are less prevalent than the 1 mm thick version with the same parameters. The reduction of the resolution is caused by the averaging operation described above.

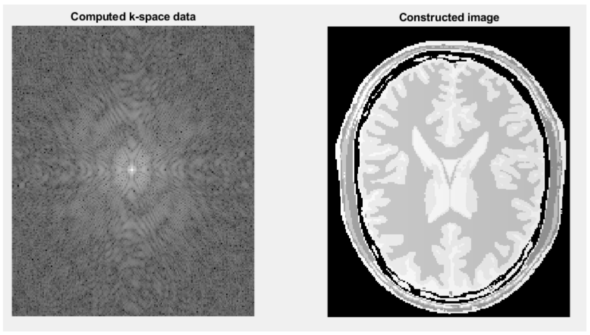


(a) Without PD

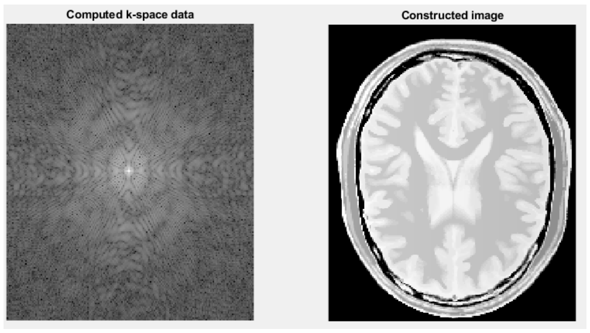


(b) With PD

Figure 8: Sample images with and without the Proton Density



(a) Slice with $1mm$ thickness



(b) Slice with $5mm$ thickness

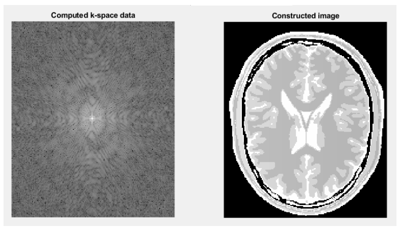
Figure 9: Sample Images with different Slice Thicknesses

3.6. Effect of Contrasting

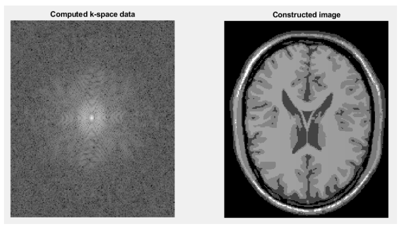
Our MRI simulator provides different types of contrasting options. Namely T1 weighted, T2* weighted for gradient echo weighted and proton density weighted additionally to these two for spin echo sequence. Since T1, T2 and T2* values differ for different types of tissue, contrasting effects the final image by amplifying different parts of the tissue.

3.7. Spin Echo Contrasting Mechanisms

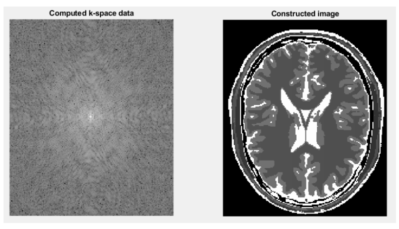
In Figure 11, we present the same slice, imaged with the same parameters except the type of contrasting mechanism, which highlights different tissue types.



(a) PD Weighted



(b) T_1 Weighted



(c) T_2 Weighted

Figure 10: Sample Spin Echo images with three different contrast mechanisms

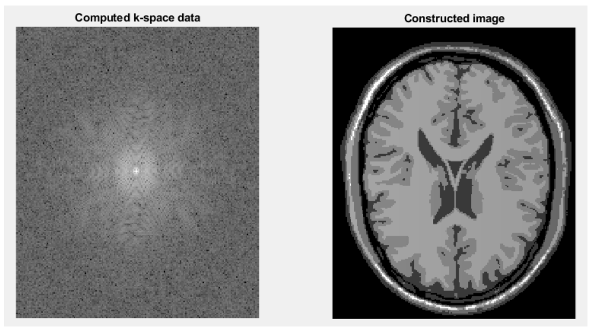
We present the Table 2 below that hold information about the three main tissue types of the brain and their corresponding color descriptions.

	PD Weighted	T_1 Weighted	T_2 Weighted
White Matter	Dark Gray	Light Gray	Dark Gray
Gray Matter	Light Gray	Gray	Gray
CSF	Bright	Dark Gray	Very Bright

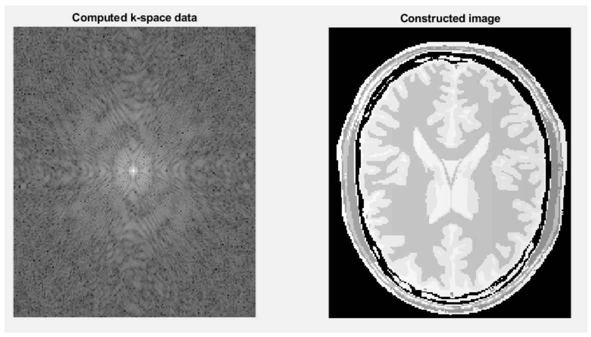
Table 2: Here we see that the PD weighted contrast highlights the CSF regions and the global contrast is worse than the other options. For the T_1 weighted image, we see that the bone marrow is highlighted, however, we clearly see that the CSF is darker than gray matter, which is darker than white matter, since T_1 values are inversely proportional to the brightness. For the T_2 weighted image, we see a clear contrast that highlights the CSF regions better. All three mechanisms are easy to distinguish, which makes this part of the project a success.

3.8. Gradient Echo Contrasting Mechanisms

In Figure 11, we present the same slice, imaged with the same parameters except the type of contrasting mechanism, which highlights different tissue types.



(a) T_1 Weighted



(b) T_2^* Weighted

Figure 11: Sample Gradient Echo images with two different contrast mechanisms

From figure, we can see that the resulting images are highly different from each other. We repeat the previous part and comment on the results.

	T_1 Weighted	T_2^* Weighted
White Matter	Light Gray	Gray
Gray Matter	Gray	Bright
CSF	Dark Gray	Very Bright

Table 3: Here we see that the T_1 weighted contrast highlights the fat regions. However, we see that the inverse relationship the we observe still holds regarding the T_1 values of tissues. However, for the T_2^* decay, we observe that the although the T_2^* value of CSF is smaller than the others it still comes as the brightest. After thorough investigation, we have found that this is due to the effect of the proton density compensating. We see the CSF brightest, then gray matter and white matter respectively. Although this was not the expected outcome, this functionality is similar to the functionality of T_2 weighted contrast in the spin echo sequence.

4. Discussion

Overall, in this project, we have built a realistic MRI simulator application that uses many non-idealities and various configurations. Overall all of our implementations have shown successful results. One specific details that we could not foresee in the classroom was the proton density neglecting the effect of T_2^* decay and increasing the contrast, which was an irregular observation in our opinion.
There are two modifications to the project that could have been done for improvement. One is the simulation of a realistic slice selection algorithm that takes into account the rect function used as a window for the RF pulse. One other improvement that could be done would be to use fuzzy data matrices instead of a discrete model, then the images should be smoother since there would be no single label assigned to a voxel. However, we believe that, if the slice are selected bigger than $1mm$, the resulting images are highly realistic.

References

[1] M. Andrew. Mrı sequence parameters. Available at https://radiopaedia.org/articles/mri-sequence-parameters, version 1.6.0.

[2] D. L. Collins, A. P. Zijdenbos, V. Kollokian, J. G. Sled, N. J. Kabani, C. J. Holmes, and A. C. Evans. Design and construction of a realistic digital brain phantom. *IEEE Transactions on Medical Imaging*, 17(3):463–468, 1998.

[3] T. U. of Calgary. Pulse sequences. Available at http://199.116.233.101/index.php/Pulse_Sequences, version 1.6.0.

[4] D. C. Preston. Magnetic resonance imaging (mri) of the brain and spine: Basics. Available at https://case.edu/med/neurology/NR/MRI%20Basics.htm, version 1.6.0.

[5] J. P. Ridgway. *Gradient Echo Versus Spin Echo*, pages 91–95. Springer International Publishing, Cham, 2015.

Appendix A - MATLAB Code

GUI Code

```
function varargout = mriGUI(varargin)
% MRIGUI MATLAB code for mriGUI.fig
%      MRIGUI, by itself, creates a new MRIGUI or raises the existing
%      singleton*.
%
%      H = MRIGUI returns the handle to a new MRIGUI or the handle to
%      the existing singleton*.
%
%      MRIGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MRIGUI.M with the given input arguments.
%
%      MRIGUI('Property','Value',...) creates a new MRIGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before mriGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to mriGUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help mriGUI

% Last Modified by GUIDE v2.5 10-Jan-2021 22:59:03

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @mriGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @mriGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before mriGUI is made visible.
function mriGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to mriGUI (see VARARGIN)

% Choose default command line output for mriGUI
handles.output = hObject;

% [T1map, T2map, T2smap, PDmap, CSHmap] = load_data();
% G_z = 4e-2; %4G/cm in T/m
% v_bar_frac = 0.5;
% slice_thic = 1;
% tp = 1e-6;
```

```
%
% [handles.T1sl, handles.T2sl, handles.T2ssl, handles.PDsl, handles.CSHsl, handles.alpha] = ...
%     slice_select(T1map, T2map, T2smap, PDmap, CSHmap, G_z, v_bar_frac, slice_thic, tp);
xFov = 181;
yFov = 217;
handles.s0 = zeros(yFov, xFov);
handles.image = zeros(yFov, xFov);
handles.tspace = 0:0.0001:constants.TE + constants.TS_1 * xFov / 2;
handles.gx = zeros(length(handles.tspace));
handles.gy = zeros(length(handles.tspace));
handles.non_ideal_mode = [0; 0; 0];
handles.slider1.Value = 0.5;
handles.curr_slice = 0.5;
handles.slice_text.String = floor(handles.curr_slice * 180);
handles.slider_min = 0;
handles.slider_max = 1;
handles.thickness = 1;
% handles.contrast_group.Buttons = [handles.t1_button handles.t2_button handles.pd_button];

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using mriGUI.
if strcmp(get(hObject,'Visible'),'off')
    axes(handles.k_axes);
    imshow(abs(handles.s0), []);
    title('Computed k-space data')
    axes(handles.image_axes);
    imshow(handles.image, []);
    title('Constructed image')
    axes(handles.pulse_axes_x);
    plot(handles.tspace, handles.gx);
    title('G_x pulse')
    xlabel('Time(s)')
    axes(handles.pulse_axes_y);
    plot(handles.tspace, handles.gy);
    title('G_y pulse')
    xlabel('Time(s)')
end

% UIWAIT makes mriGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = mriGUI_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

% --- Executes on button press in compute_button.
function compute_button_Callback(hObject, eventdata, handles)
% objHandles = guihandles(hObject);
f = waitbar(0.1, 'Loading Data...');

[T1map, T2map, T2smap, PDmap, CSHmap] = load_data();
G_z = 4e-2; %4G/cm in T/m
v_bar_frac = handles.curr_slice; % str2double( get(handles.slice_selection_bar, 'String') );
slice_thic = handles.thickness;
tp = 1e-6;

if handles.gradient_button.Value == 1
    pulse_type = 'grad';
elseif handles.spin_button.Value == 1
    pulse_type = 'spin';
end

if handles.t1_button.Value == 1
```

```matlab
    contrast_mode = 'T1';
elseif handles.t2_button.Value == 1
    contrast_mode = 'T2';
elseif handles.pd_button.Value == 1
    contrast_mode = 'PD';
end

[handles.T1sl, handles.T2sl, handles.T2ssl, handles.PDsl, handles.CSHsl, handles.alpha] = ...
    slice_select(T1map, T2map, T2smap, PDmap, CSHmap, G_z, v_bar_frac, slice_thic, ...
    contrast_mode, pulse_type);

guidata(hObject, handles);

waitbar(.60, f, 'Computing k-space');

disp(handles.non_ideal_mode)


[handles.s0, handles.gx, handles.gy, handles.tspace] = compute_k(...
    handles.alpha, handles.T1sl, handles.T2sl, handles.T2ssl, ...
    handles.PDsl, handles.CSHsl, handles.non_ideal_mode, contrast_mode, pulse_type, handles.noise_box.Value);

% popup_sel_index = get(handles.mode_menu, 'Value');

% disp(popup_sel_index)
% switch popup_sel_index
%     case 1 %% T2
%         [handles.s0, handles.gx, handles.gy] = compute_k(...
%             handles.alpha, handles.T1sl, handles.T2sl, handles.T2ssl, ...
%             handles.PDsl, handles.CSHsl, 'T2Decay');
%
%     case 2 %% chemical shift
%         [handles.s0, handles.gx, handles.gy] = compute_k(...
%             handles.alpha, handles.T1sl, handles.T2sl, handles.T2ssl, ...
%             handles.PDsl, handles.CSHsl, 'Chemical Shift');
%
%     case 3 %% proton density
%         [handles.s0, handles.gx, handles.gy] = compute_k(...
%             handles.alpha, handles.T1sl, handles.T2sl, handles.T2ssl, ...
%             handles.PDsl, handles.CSHsl, 'Proton Density');
% end

waitbar(.90, f, 'Constructing Image');
guidata(hObject, handles);
handles.image = reconstruct_image(handles.s0);
guidata(hObject, handles);

axes(handles.k_axes);
cla;
imshow(log(abs(fftshift(handles.s0, 2))), []);
title('Computed k-space data')

axes(handles.image_axes);
cla;
% imshow(abs(fftshift(handles.image)), []);
imshow(abs(handles.image), []);
title('Constructed image')

axes(handles.pulse_axes_x);
cla;
plot(handles.tspace, handles.gx);
title('G_x pulse')
xlabel('Time(s)')

axes(handles.pulse_axes_y);
cla;
plot(handles.tspace, handles.gy);
```

```
title('G_y pulse')
xlabel('Time(s)')

close(f)
guidata(hObject, handles);

% --------------------------------------------------------------------
function FileMenu_Callback(hObject, eventdata, handles)

% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
printdlg(handles.figure1)

% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
                     ['Close ' get(handles.figure1,'Name') '...'],...
                     'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)

% --- Executes on selection change in mode_menu.
function mode_menu_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function mode_menu_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end

set(hObject, 'String', {'T2 Decay', 'Chemical Shift', 'Proton Density'});

function thickness_bar_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function thickness_bar_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function slice_selection_bar_Callback(hObject, eventdata, handles)

function slice_selection_bar_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in t2_decay_box.
function t2_decay_box_Callback(hObject, eventdata, handles)
% hObject    handle to t2_decay_box (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.non_ideal_mode(1) = get(hObject, 'Value');
guidata(hObject, handles)
```

```
% Hint: get(hObject,'Value') returns toggle state of t2_decay_box


% --- Executes on button press in csh_box.
function csh_box_Callback(hObject, eventdata, handles)
% hObject    handle to csh_box (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.non_ideal_mode(2) = get(hObject, 'Value');
guidata(hObject, handles)
% Hint: get(hObject,'Value') returns toggle state of csh_box


% --- Executes on button press in pd_box.
function pd_box_Callback(hObject, eventdata, handles)
% hObject    handle to pd_box (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.non_ideal_mode(3) = get(hObject, 'Value');
guidata(hObject, handles)
% Hint: get(hObject,'Value') returns toggle state of pd_box


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% handles.slider1.Min = floor(handles.thickness / 2) / 180;
% handles.slider1.Max = 1 - floor(handles.thickness / 2) / 180;
handles.curr_slice = get(hObject, 'Value');
handles.slice_text.String = floor(handles.curr_slice * 180) + 1;
guidata(hObject, handles)
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in mm1_button.
function mm1_button_Callback(hObject, eventdata, handles)
% hObject    handle to mm1_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(hObject,'Value') == 1
    handles.thickness = 1;
    handles.slider1.Min = floor(handles.thickness / 2) / 180;
    handles.slider1.Max = 1 - floor(handles.thickness / 2) / 180;
    handles.slider1.Value = 0.5;
    handles.curr_slice = 0.5;
    handles.slice_text.String = "90";
end
guidata(hObject, handles)
% Hint: get(hObject,'Value') returns toggle state of mm1_button


% --- Executes on button press in mm3_button.
```

```
function mm3_button_Callback(hObject, eventdata, handles)
% hObject    handle to mm3_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(hObject,'Value') == 1
    handles.thickness = 3;
    handles.slider1.Min = floor(handles.thickness / 2) / 180;
    handles.slider1.Max = 1 - floor(handles.thickness / 2) / 180;
    handles.slider1.Value = 0.5;
    handles.curr_slice = 0.5;
    handles.slice_text.String = "90";
end
guidata(hObject, handles)
% Hint: get(hObject,'Value') returns toggle state of mm3_button


% --- Executes on button press in mm5_button.
function mm5_button_Callback(hObject, eventdata, handles)
% hObject    handle to mm5_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(hObject,'Value') == 1
    handles.thickness = 5;
    handles.slider1.Min = floor(handles.thickness / 2) / 180;
    handles.slider1.Max = 1 - floor(handles.thickness / 2) / 180;
    handles.slider1.Value = 0.5;
    handles.curr_slice = 0.5;
    handles.slice_text.String = "90";
end
guidata(hObject, handles)
% Hint: get(hObject,'Value') returns toggle state of mm5_button


% --- Executes on button press in mm7_button.
function mm7_button_Callback(hObject, eventdata, handles)
% hObject    handle to mm7_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(hObject,'Value') == 1
    handles.thickness = 7;
    handles.slider1.Min = floor(handles.thickness / 2) / 180;
    handles.slider1.Max = 1 - floor(handles.thickness / 2) / 180;
    handles.slider1.Value = 0.5;
    handles.curr_slice = 0.5;
    handles.slice_text.String = "90";
end
guidata(hObject, handles)
% Hint: get(hObject,'Value') returns toggle state of mm7_button


% --- Executes on button press in noise_box.
function noise_box_Callback(hObject, eventdata, handles)
% hObject    handle to noise_box (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of noise_box
```

Slice Selection

```
function [T1sl, T2sl, T2ssl, PDsl, CSHsl, alpha_mat] = slice_select(T1map,T2map,T2smap, ...
                                        PDmap,CSHmap, G_z, v_bar_frac, ...
                                        slice_thic, contrast_mode, pulse_type, tp)
    % get data dimensions
    dims = size(T1map);
    Z_max = dims(3);

    % find the frequrncy range for the anatomical model
    V_LOW = constants.GMR * constants.B_0;
    V_DIF = constants.GMR * G_z * Z_max;
    V_HIGH = V_LOW + V_DIF; %281MHz

    % set location using a multiplier on the min-max frequency range
    v_bar = V_LOW + V_DIF * v_bar_frac; %Hz

    % set slice thickness over unit slice  (1mm)
    delta_v_unit = constants.GMR * G_z; %1703200;
    delta_v = delta_v_unit * slice_thic;

    % set the freqency axis
    v_base = linspace(V_LOW, V_HIGH, dims(3));

    % set the realistic rect width for realistic slice selection (NOT USED)
    % tp = 1e10; %s

    % set alpha
    if strcmp(contrast_mode, 'T2') && strcmp(pulse_type, 'grad')
        A = pi / 9;
    else
        A = pi / 2;
    end

    % find the alpha angles for z-axis
    if nargin == 10
        [alpha, z] = rf(v_base, A, delta_v, v_bar, G_z);
    elseif nargin == 11
        [alpha, z] = rf(v_base, A, delta_v, v_bar, G_z, tp);
    else
        error('Wrong number of input arguments in slice selection');
    end

    % select slices
    T1sl = T1map(:,:,alpha>0);
    T2sl = T2map(:,:,alpha>0);
    T2ssl = T2smap(:,:,alpha>0);
    PDsl = PDmap(:,:,alpha>0);
    CSHsl = CSHmap(:,:,alpha>0);

    alpha = alpha(alpha > 0);
    alpha_mat = zeros(dims(1),dims(2), slice_thic);
    for z = 1:slice_thic
        alpha_mat(:,:,z) = alpha(z);
    end

end
```

RF Pulse

```
function [alpha,zs] = rf(v_base, A, delta_v, v_bar, G_z, tp)
    if nargin == 6
        [alpha,zs] = rf_r(v_base, A, delta_v, v_bar, G_z, tp);
    elseif nargin == 5
        [alpha,zs] = rf_nr(v_base, A, delta_v, v_bar, G_z);
    else
        error('Number of parameters are incorrect for RF')
    end
end

% realistic slice selection
function [alpha_z,z_base] = rf_r(v_base, A, delta_v, v_bar, G_z, tp)
```

```
    z_base = (v_base ./ (constants.GMR) - constants.B_0)./G_z;
    z_bar = (v_bar ./ (constants.GMR) - constants.B_0)./G_z;
    delta_z = delta_v ./ (constants.GMR.*G_z);
    z_shift = floor(length(z_base/2));
    sinc_inf = tp.* sinc(tp.*constants.GMR.*(constants.B_0 + G_z .* (z_base-z_shift)));
    % figure();plot(sinc_inf);title('sinc_inf')
    trunc = sinc_inf(z_shift-tp:z_shift+tp);
    % figure();plot(trunc);title('trunc')
    alpha_z = conv(rect(z_base, A, z_bar, delta_z),trunc, 'same');
end

% non-realistic slice selection
function [alpha_z,z_base] = rf_nr(v_base, A, delta_v, v_bar, G_z)
    z_base = (v_base ./ (constants.GMR) - constants.B_0)./G_z;
    z_bar = (v_bar ./ (constants.GMR) - constants.B_0)./G_z;
    delta_z = delta_v ./ (constants.GMR.*G_z);
    alpha_z = rect(z_base, A, z_bar, delta_z);
end

% to define the function Arect((x-b)/c)
function r=rect(x, A, b, c)
    %x is the base unit range vector (x,y,z,t)
    r = zeros(size(x));
    r(abs((x-b)./c) <= 0.5) = A;
end

\subsection*{Data Load}
% first, add the folder resource to your path by Home > Environment > Set
% Path > Add with Subfolders and select the 'resource' and 'data' folders.
function [T1map,T2map,T2smap,PDmap,CSHmap]=load_data()
    TISSUE_NUM = 11;
    %Load the anatomical phantom
%     [anat,info] = loadminc('data/phantom_1.0mm_normal_crisp.mnc');
    [anat,info] = loadminc('phantom_1.0mm_msles1_crisp.mnc');

    sizeVol = size(anat);

    T1map = zeros(sizeVol);
    T2map = zeros(sizeVol);
    T2smap = zeros(sizeVol);
    PDmap = zeros(sizeVol);
    CSHmap = zeros(sizeVol);

    for i = 1:TISSUE_NUM
        T1map(anat == i-1) = nmrparams.T1(i);
        T2map(anat == i-1) = nmrparams.T2(i);
        T2smap(anat == i-1) = nmrparams.T2s(i);
        PDmap(anat == i-1) = nmrparams.PD(i);
        CSHmap(anat == i-1) = nmrparams.CSH(i);
    end

    disp(info)
end

\subsection{Constants}
classdef constants
    properties (Constant = true)
        GMR = 42.58 * 10^6; %Gyromagnetic Ratio of 1H Hz/T
        GMR_nor = 42.58 * 10^6 / (2 * pi);
        PLANCK = 6.62607004 * 10^-34; %Planck's constant
        BOLTZMANN = 1.38064852 * 10^-23; %Boltzmann's constant
        T = 300; %IDK YET
        TR = 0.6; % Test value
        TE = 0.02; % Test value
        TS_1 = 25e-6; % Time spent sampling 1 mm of kspace
        xFov = 181;
        yFov = 217;
        % Tissue values are measured in 1.5T scanner,
        B_0 = 1.5 %Static Magnetic Field T
        B_0_var = 3e-6 % Magnetic Variation
    end
end
```

K-Space Acquisition

```
function [s0, gx_t, gy_t, t_space] = compute_k(alpha, T1sl, T2sl, T2ssl, PDsl, CSHsl, mode, contrast, pulse_type,
    % Define TR, TE
    switch contrast
        case 'T1'
            TE = 0.014;
            TR = 0.5;
        case 'T2'
            TE = 0.03;
            TR = 4;
            if strcmp(pulse_type, 'spin')
                TE = 0.1;
            end
        case 'PD'
            TE = 0.014;
            TR = 6;
    end

    % Define space variables
    xFov = 181;
    yFov = 217;
    xSpace = -90:90;
    ySpace = -108:108;
    [kX, kY] = meshgrid(xSpace, ySpace);

    % Define G values corresponding to kspace coords
    Gx_0 = 1 / (xFov * constants.GMR * constants.TS_1);

    Gy_0 = 1.5e-7;
    Gy_area = 1 / (constants.GMR * yFov);
    Gy_duration = Gy_area / Gy_0;

    timestamps = TE + kX .* constants.TS_1; % only depends on kX since we switch to a new line every TR

    % Plot Gy, Gx pulses
    t_space = 0:0.0001:TE + constants.TS_1 * constants.xFov / 2;
    gx_t = zeros(length(t_space));
    gy_t = zeros(length(t_space));

    for i = 1:length(t_space)
        [gx_t(i), gy_t(i)] = generate_g_t(Gx_0, Gy_0, Gy_duration, TE, pulse_type, t_space(i));
    end

    if add_noise == 1
        noise = sqrt(constants.B_0 * 1e-3) .* randn(yFov, xFov);
    else
        noise = 0;
    end

    % Acquire f(x, y)
    f_xy = (noise + constants.B_0) .* (1 - exp( - TR ./ T1sl)) .* ...
        (1 ./ (1 - cos(alpha) .* exp( - TR ./ T1sl)) );

    if mode(1) == 1
        if strcmp(pulse_type, 'grad')
            f_xy = f_xy .* exp( - timestamps ./ T2ssl);
        elseif strcmp(pulse_type, 'spin')
            f_xy = f_xy .* exp( - timestamps ./ T2sl);
        end
    end

    if mode(2) == 1
        f_xy = (1 - CSHsl) .* f_xy;
    end

    if mode(3) == 1
        f_xy = f_xy .* constants.GMR_nor.^2 * constants.PLANCK.^2 ./ ...
                (4 * constants.BOLTZMANN * constants.T) .* PDsl;
    end

    % Compute k_space data
    s0 = zeros(yFov, xFov); % delta x = delta y = 1
    for y = 1:yFov
        for x = 1:xFov
            s0(y, x) = sum(f_xy .* exp(- 2 * pi * 1j * constants.GMR * ...
                ((- Gx_0 * (xFov * constants.TS_1 / 2)) + kX .* Gx_0 * constants.TS_1 * x + ...
                kY .* Gy_0 * (y - floor(yFov / 2)) * Gy_duration)), 'all');
        end
    end

end

function [gx_t, gy_t] = generate_g_t(Gx_0, Gy_0, Gy_duration, TE, pulse_type, t)
    switch pulse_type
        case 'grad'
            if (t > TE - constants.TS_1 * constants.xFov - Gy_duration && t <= TE - constants.TS_1 * constants.xFo
                gx_t = 0;
                gy_t = Gy_0;
            elseif (t > TE - constants.TS_1 * constants.xFov && t <= TE - constants.TS_1 * constants.xFov / 2)
                gx_t = - Gx_0;
                gy_t = 0;
            elseif (t > TE - constants.TS_1 * constants.xFov / 2 && t <= TE + constants.TS_1 * constants.xFov / 2)
                gx_t = Gx_0;
                gy_t = 0;
            else
                gx_t = 0;
                gy_t = 0;
            end
        case 'spin'
            if (t > 0 && t <= Gy_duration)
                gx_t = 0;
                gy_t = Gy_0;
            elseif (t > Gy_duration && t <= Gy_duration + constants.TS_1 * constants.xFov / 2)
                gx_t = Gx_0;
                gy_t = 0;
            elseif (t > TE - constants.TS_1 * constants.xFov && t <= TE - constants.TS_1 * constants.xFov / 2)
                gx_t = Gx_0;
                gy_t = 0;
            elseif (t > TE - constants.TS_1 * constants.xFov / 2 && t <= TE + constants.TS_1 * constants.xFov / 2)
                gx_t = Gx_0;
                gy_t = 0;
            else
                gx_t = 0;
                gy_t = 0;
            end
    end
end
```