

EE369C: Assignment 2

Due Thursday, Oct 18

Introduction This assignment introduces the basic operations in a gridding reconstruction algorithm. Starting from a very simple routine, you will add pre-weighting density correction, k-space oversampling, and deapodization.

The data set for this assignment is a simulated phantom using a small spiral acquisition with 6 interleaves of 1536 samples. This was the default real-time imaging sequence for many years. With a repetition time T_R of 24 ms, this produces a new image every 144 ms, at a rate of 7 images/s. The k-space trajectory, pre-weighting function, and simulated data are in the file:

http://www.stanford.edu/class/ee369c/mr_data/rt_spiral_03.mat.

The complex k-space data is in the matlab variable `d`, and the pre-weighting function is in the matlab variable `w`. The k-space trajectory (k_x, k_y) is stored as $k = k_x + i * k_y$ in the complex matlab variable `k`. `k` is scaled relative to a $\pm k_{max}$ of ± 0.5 . The k-space trajectory doesn't actually reach $\pm k_{max}$, but is scaled to produce the right field-of-view when reconstructed as a 128x128 image.

We will start with a very basic gridding algorithm that doesn't do density correction, uses a very simple separable triangular kernel, uses a 1X grid, and doesn't do deapodization. This m-file is available at

<http://www.stanford.edu/class/ee369c/mfiles/grid1.m>.

Note that this only does the gridding operation. You still need to do an inverse 2DFFT to produce an image.

```
function m = grid1(d,k,n)

% function m = grid1(d,k,n)
%   d -- k-space data
%   k -- k-trajectory, scaled -0.5 to 0.5
%   n -- image size

% convert to single column
d = d(:);
k = k(:);

% convert k-space samples to matrix indices
nx = (n/2+1) + n*real(k);
ny = (n/2+1) + n*imag(k);

% zero out output array
m = zeros(n,n);

% loop over samples in kernel
for lx = -1:1,
    for ly = -1:1,

        % find nearest samples
        nxt = round(nx+lx);
```

```

nyt = round(ny+ly);

% compute weighting for triangular kernel
kwx = max(1-abs(nx-nxt),0);
kwy = max(1-abs(ny-nyt),0);

% map samples outside the matrix to the edges
nxt = max(nxt,1); nxt = min(nxt,n);
nyt = max(nyt,1); nyt = min(nyt,n);

% use sparse matrix to turn k-space trajectory into 2D matrix
m = m+sparse(nxt,nyt,d.*kwx.*kwy,n,n);
end;
end;

% zero out edge samples, since these may be due to samples outside
% the matrix
m(:,1) = 0; m(:,n) = 0;
m(1,:) = 0; m(n,:) = 0;

```

This loops over the gridding kernel, which is relatively small. For each sample of the kernel, the gridding operation for the entire data vector is done with the `sparse()` matrix call. This sets up an $n \times n$ sparse matrix with values $d \cdot k_{wx} \cdot k_{wy}$ at matrix locations (n_x, n_y) . This automatically gets converted to a full 2D matrix when it is added to m .

This is just one possible way to code the gridding algorithm. If this seems obscure, wrong, or if you find a faster approach, you are welcome to recode it. I'd be particularly interested in faster versions!

Questions

1. *Simple Gridding Reconstruction*

Reconstruct an 128×128 image of the simulated phantom data with this algorithm. There is a dominant low frequency artifact. What is it due to? Display your reconstruction.

2. *Pre-weighted Gridding Reconstruction*

Extend the algorithm to use the preweighting function w that has been provided. Display your reconstruction.

3. *Oversampled Gridding Reconstruction*

Extend the algorithm to reconstruct on a 2X grid. This is a grid that is sampled twice as finely in k-space, and has twice the FOV in image space. The kernel should extend for ± 2 samples on the 2X grid. Display your 2X reconstruction. What artifacts have been reduced or eliminated?

4. *Deapodization Correction*

The kernel we are using is a separable triangle function in k_x and k_y . Compute the apodization produced by this kernel for the 2X oversampled reconstruction, and divide it out of the reconstructed image. Plot a cross-section through the phantom before and after correction. Display your corrected reconstruction.

At this point you should have an algorithm that does a pretty reasonable job. In the next assignment we will look at better kernel functions, computing the density correction, and, non-integer oversampling factors.