EE369C: Assignment 5

Due Thurs Oct 29

The problems this week will again be concerned with parallel imaging. We will look at coil compression, and k-space parallel reconstruction. The data set is the same as last week.

1. Coil Compression Even though we have eight receive channels, we can only get a much smaller acceleration factor. Here we will look at the reasons for this. We can exploit this to reduce the number of channels we need to reconstruct, and this can be important for large 3D array coils.

a. Eigencoils The coils sensitivities are not orthogonal, and span a space that has a lower dimension than the number of channels. To see this we'll compute the eigencoils that correspond to the set of coil sensitivities.

We first need to create a 2D matrix that describes the coils

```
[nx ny nc] = size(map);
m2 = reshape(map,nx*ny,nc)
```

This is a matrix that has an entire vectorized map in each column, with one column per coil. The eigen decomposition is

```
[ev ed] = eig(m2'*m2);
```

The eigenvalues are on the diagonal of `ed`, and the corresponding eigenvectors as columns of `ev`.

Each eigenvalue and eigenvector correspond to a combination of coils, with the largest eigenvalue the most important combination. Reconstruct the eigencoil maps by

```
ecmap= reshape(m2*ev,nx,ny,nc);
```

Display the magnitude and phase of the eigencoil maps, in descending order of significance. The energy in one of the maps is the eigenvalue. How many terms do you need before we get to less than 1% of the energy? Note that as the total energy decreases, the maps become more localized to the edge of the head. Also, note that the phase maps have phase wraps in the angular direction. How many wraps does the j^{th} map have?

b. g-Factor It looks like there are only four or five significant eigencoils resulting from the eight array coils. Compare the g-factor maps for all eight array coils compared to the most significant 4, 5, and 6 eigencoils. Choose an acceleration of `Rx=Ry=2`.

c. Coil Compression We can compute a reduced data set the same way as we computed the eigencoil maps. Make images corresponding to the different eigencoils, and display them on the same scale.

d. SENSE Reconstruction You can speed up your reconstruction by only using a subset of the eigencoils. Undersample the data from the previous part by `Rx=Ry=2`, and reconstruct it using 4, 5, and 6 of the eigencoils.

2. Parallel Imaging with SPIRiT There are several different parallel reconstruction algorithms that operate in k-space. We'll look at the SPIRiT algorithm that we talked about in class. This is an iterative algorithm that doesn't require calibration for a specific sampling pattern. This will be useful for combining parallel imaging, and compressed sensing.

To help you get started, we'll give you two routines that are available on the web site. The first is

```
sk = spirit_kernel(mc,Nk)
```

This takes a calibration region `mc` and a kernel size `Nk`, and returns the SPIRiT kernel `sk`. The kernel has an `Nk` by `Nk` by `Nc` kernel for every coil. The second routine is

```
ms = apply_spirit(m,sk)
```

This takes k-space data `m` and applies the kernel `sk`, and returns the result.

a) Generate the SPIRiT kernel The data you have is in image space. Transform it to k-space and save it in a variable `m`. It may be useful to define functions

```
function y = fft2c(x)
y = fftshift(fft2(fftshift(x)))
```

and the corresponding `ifft2c()` to perform the centered transforms.

For the calibration region, extract the central 24 by 24 by 8 block of the k-space data, and call it `mc`. Compute a 5 by 5 SPIRiT kernel,

```
sk = spirit_kernel(mc,5)
```

Ideally if we apply this kernel to the full data set, we should get the same data back. Apply the kernel for 10 iterations. Display one of the original coil images, the reconstructed coil image after 10 iterations, and the difference image multiplied by 10.

b) SPIRiT Reconstruction Write a function that takes an undersampled k-space data set and a calibration data set, and computes the SPIRiT reconstruction. It will alternately apply data consistency, and consistency with the kernel. Since this is iterative, you need to choose a stopping criteria. Explain your reasoning.

Apply your function to the `Rx=Ry=2` data. Use the calibration region and kernel size from (a). Display the reconstructed coil data, the difference image multiplied by 10, and the square root of sum of squares reconstruction.

Usually the calibration region is included in the initial data. Repeat the reconstruction and see if this helps.

c) Random Sampling Generate a randomly sampled data set with

```
% choose random phase encodes
msk1 = (rand(size(m(:,:,1))) > 0.5);
% copy to all coils
msk = repmat(msk1,[1 1 8]);
% mask the data off
mr = m.*msk;
```

This randomly deletes half of the samples. Reconstruct using the same calibration region as above. Display one of the coil images, and the difference multiplied by 10. This would be a difficult reconstruction to do with GRAPPA!

You can adjust the fraction of the data that is deleted by changing the threshold value from 0.5. Estimate the point at which the reconstruction become unacceptable. What happens to the number of iterations you need?