EEE 475/575 Medical Image Reconstruction & Processing
Homework 2
30 March 2020, Monday at 23:59

GUIDELINES FOR HOMEWORK SUBMISSION

Instructions:

1. NO submission via E-MAIL (all emails will be discarded).
2. NO submission of RAR or ZIP files (rar or zip files will be discarded).
3. You should upload your solutions to Moodle as two separate files. One PDF file (should be named name_surname_id_hw#.pdf), and one MATLAB m file (should be named name_surname_id_hw#.m).
4. Submission system will remain open for 1 day after the deadline. No points will be lost if you submit your assignment within 12 hours of the deadline. There will be a 50% penalty if you submit after 12 hours but within 24 hours past the deadline. No submissions beyond 24 hours past the deadline.

MATLAB File Guidelines

1. It should be a single m file containing the codes for all questions (if you upload many m files, we will not evaluate any of them).
2. There is a template on Moodle to help you organize your solution in one m file, you should use it in all HWs.
3. Read the guidelines in the template file and follow each and every step.
4. If you do not upload your m file, your homework will not be evaluated (i.e., 0 pts). There will not be any exceptions to this rule.
5. If your m file gives runtime errors, your homework will not be evaluated (i.e., 0 pts). There will not be any exceptions to this rule.

PDF Report Guidelines

1. You get your points mainly from the PDF report file. The report should be typeset (no handwriting allowed). The PDF file should contain all results and plots. Unclear presentation of results will be penalized heavily. No partial credits to unjustified answers.
2. Maximum file size that can be uploaded to Turnitin is 40 MB. Pay attention to this before submitting your file.
3. You should properly name your pdf file and upload it to Turnitin upload area. We will discard all improperly named submissions (i.e., 0 pts).
4. After each question or each part you should properly display your MATLAB results, plots and MATLAB codes for that part (zero points for missing outputs).

PREPARATION

From the Moodle page of the class, download the following files:

- **gridkb.m:** Gridding code that chooses the optimum parameters based on Philip Beatty's paper (this paper is posted on Moodle). Type "help gridkb" in MATLAB to see how this function works.
- **voronoidens.m:** Calculates the area associated with each k-space point on a trajectory. Type "help voronoidens" in MATLAB to see how this function works.
- **spiral.mat:** Complex MRI data in k-space for an "interleaved" spiral k-space acquisition (with 6 interleaves). kdata is the data. ktraj is the k-space trajectory given as kdata = kx + i*ky as a complex variable. ktraj is scaled to ±kmax = ±0.5. The trajectory does not fully reach kmax.
- **shepplogan_radial_data.mat:** Complex MRI data in k-space for a radial k-space acquisition. Each acquisition starts from the center of k-space and extends out at an angle. There are 300 such radial lines.

PART I – GRIDDING (45 pts)

1.1) Load spiral.mat. Display the 2D k-space trajectory, where the x-axis is kx (real part of ktraj) and the y-axis is ky (imaginary part of ktraj). To do this, you can simply type:

```
>> plot(ktraj);
```

Clearly label both axes. Here, MATLAB treats the real part of ktraj as the x-axis and imaginary part of ktraj as the y-axis. What you see is a so-called "interleaved spiral trajectory", which has a total of 6 interleaves (shown with different colors in this plot). Each interleaf is acquired during a separate repetition time (TR) after the excitation of the 2D slice, and has 2048 data points. Hence, both ktraj and kdata have size 2048×6.

1.2) Calculating Density Compensation Filter: Next, use the voronoidens.m function to estimate the density on the area associated with each k-space point on the trajectory:

```
>> area = voronoidens(ktraj);
```

Note that the area directly gives the density compensation filter that we want to use (i.e., density compensation filter = 1/density = area). Plot the computed area:

```
>> plot(area(:));
```

Clearly label both axes. Also plot a zoomed-in version that shows the reasonable values in this plot. How many NaN values does the computed area have? Why are there NaN values? Why are some of the area values unreasonably large? Explain.

1.3) Correcting the Density Compensation Filter: Now, find a way to assign reasonable values to the computed areas. There are many ways to do this. For example, you could do a simple thresholding. Whatever method you use, make sure that there are no NaN values remaining. Plot the resulting area values.

1.4) Direct Summation: One way to reconstruct non-Cartesian data is the direct reconstruction technique, which can be expressed as follows for the 1D case, as given in class:

$$m[n] = \sum_{i=1}^{K} d_i M(k_i)\, e^{j2\pi k_i \Delta x\,(n-\tau)}\,, \quad n = 0, \dots N-1$$

where $M(k_i)$ is the k-space data at k-space position k_i, d_i are density compensation factors (i.e., area computed in Question 1.3) at k-space position k_i, and Δx is the size of a pixel. Note that since ktraj is scaled to ±kmax = ±0.5, Δx=1 in our case. In addition, τ indicates the center of the field-of-view relative to the image, and hence is equal to $\tau = N/2$.

This technique is used as the "gold standard" method in some papers. First, re-write the equation given above for the 2D case. Then, implement the direct summation technique in MATLAB. Reconstruct an image with a size of 128×128. Pay attention to use vector arithmetic instead of for loops whenever possible. Normalize the resulting image, and display it using the imshow command.

```
>> imshow(abs(ima_direct),[]);
```

Here, the [] argument tells imshow to automatically set the image display range. Comment on the problems that you see in the image. We will use this image as the "*reference*" image for the rest of Part I questions.

Display the central horizontal cross-section of this image:

```
>> plot(abs(ima_direct(end/2,:));
```

Also, display the central vertical cross-section of this image:

```
>> plot(abs(ima_direct(:,end/2));
```

Lastly, use the built-in "cputime" function of MATLAB to compute the CPU time needed for the direct summation technique. When using "cputime", only consider the part of your code where you are reconstructing the image (i.e., exclude the parts where you are displaying the results).

1.5) Simple Gridding Without Density Compensation: We will start with simple 1X gridding, meaning that we will use the minimum field-of-view (FOV). We will also not use any density compensation. Reconstruct an image with a 128×128 size, an oversampling factor (osf) of 1, a kernel width (wg) of 2.

```
>> w = ones(size(kdata));
>> ima = gridkb(kdata,ktraj,w, 128,1,2,'image');
```

Here, “ones(size(kdata))” gives a weight of 1 to each k-space data point (i.e., no density compensation). Normalize the resulting image, and display it using the imshow command. Display the error image (i.e., difference between this image and the reference image). Also, display the central horizontal and vertical cross-sections of this image. In these 1D plots, overlay the horizontal and vertical cross-sections of the reference image for visual comparison. Compute PSNR and SSIM. Compute CPU time needed for gridding reconstruction.

Comment on the problems that you see in the 1X gridded image when compared to the reference image. Comment on the CPU time when compared to that of the direct summation technique.

1.6) Simple Gridding with Density Compensation: Now, repeat Question 1.5 by assigning the k-space weighting, w, to be equal to the area computed via the Voronoi method. Display the image, error image, and the cross-sections (with cross-sections of the reference image overlayed). Compute PSNR, SSIM, CPU time. Comment on the improvements on the image. Comment on the problems that you still see in the image.

1.7) 2X Gridding: Repeat Question 1.6 for an oversampling factor of osf = 2. Note that you also need to increase the kernel width to wg = 4, so that the kernel covers the same radius in k-space as it did for the 1X grid. First, display the full images that result from this gridding. Actually, we only care about the central 128×128 image, so display that central part, as well. Also, display the error image and the cross-sections of the central part (with cross-sections of the reference image overlayed). Compute PSNR, SSIM, CPU time. Comment on the differences that you see between this image and the result from Question 1.6.

1.8) Reduced Oversampling for Gridding: Instead of 2X Gridding, it may be sufficient to use slightly smaller oversampling to reduce both computation time and memory usage. Using Figure 3 in the Beatty paper, choose a kernel width suitable for an oversampling factor of 1.25, such that the relative aliasing amplitude is limited to 10^{-3} or smaller. Reconstruct the data for these parameters and display the results as before (full image, central part, error image, overlayed cross-sections). Compute PSNR, SSIM, CPU time. Comment on the differences between this image and the result from Question 1.7.

1.9) Effect of Deapodization: The gridkb.m function automatically does the deapodization when you return an image. To see what the image would look like without deapodization, use the last input to gridkb as ‘k-space’ instead of ‘image’. This returns the k-space data without deapodization. To calculate the resulting image, type:

```
>> ima = ifft2c(kd);
```

where kd is the result of the gridkb with opt = ‘k-space’. Display the results as before (full image, error image, central part, overlayed cross-sections). Compute PSNR, SSIM, CPU time. Comment on the differences between this image and the result from Question 1.8.

<u>PART II – GRIDDING VS. BACKPROJECTION FOR PROJECTION DATA (35 pts)</u>

2.1) Generating Projections: In Matlab, the built-in "radon" function generates the projections for a given input (i.e., computes its Radon Transform). First, generate a 256×256 phantom image and then its projections for angles from 0 degree to 179 degree:

```
>> P = phantom('Modified Shepp-Logan',256);
>> proj = radon(P,[0:179]);
```

Display the phantom and its central horizontal and vertical cross-sections. We will use the phantom itself as the "*<u>reference</u>*" image for <u>both Part II and Part III</u> questions. Display the sinogram (i.e., proj). Here, proj matrix has size 367x180, where 180 is the number of projections. One axis is l and the other axis is θ. Clearly label both axes of the sinogram.

2.2) Backprojection: Using "iradon", reconstruct the image from its projections (i.e., compute the inverse Radon Transform). Use the default filter in iradon (called Ram-Lak in Matlab), which is the ramp filter that we have covered in class. Note that iradon will yield an image of size 258×258. Normalize this image, and use the central 256×256 of this image as your result. Display the image and the error image (i.e., difference between image and P). Display the central cross-sections (with cross-sections of the reference image overlayed). Compute PSNR and SSIM. Comment on the result.

2.3) Naïve Backprojection: Repeat Question 2.2 for the case of no filter (i.e., 'none' option for the filter in iradon). This corresponds to naïve backprojection. Display the results as before (image, error image, overlayed cross-sections). Compute PSNR and SSIM. Comment on the differences between this image and the result from Question 2.2.

2.4) Generating k-space Data from Projections: Now, using the Projection-Slice Theorem, compute the k-space data (a 367×180 matrix) and the corresponding k-space trajectory (a 367×180 matrix) for this case. For the k-space trajectory, we have

```
>> ktraj = kx + i*ky;
```

Here, use the maximum k-space radius to be 0.5, so that your data will be compatible with the gridkb function. Plot the k-space trajectory. Display the magnitude of the k-space data (i.e., the **<u>1D</u>** Fourier transform of the sinogram along the l-axis) as an image (using *log* format as explained in Homework 1). Here, the k-space data is a 367×180 image, where one axis is k_r (or ρ) and the other axis is θ. Clearly label both axes.

2.5) Calculating Density Compensation Filter: Using the geometric approach for the radial scanning case (as covered in class during gridding reconstruction), compute the area associated with each k-space point. This will also be a 367×180 matrix. Plot the resulting area values for the first angle only.

Side note: Voronoi method will give an error for this case. Voronoi method expects unique positions for each data point, but the trajectory passes through the center of k-space for all projections.

***Note:** For Questions 2.6-2.8 below, the resulting images may be flipped/transposed due to flipped k-space axes. So, flip/transpose them back to compare to the original phantom.*

2.6) Direct Summation: Compute the direct summation image using the k-space data, trajectory, and the density compensation filter that you computed. Display the results as before (image, error image, overlayed cross-sections). Compute PSNR and SSIM. Comment on the differences between this image and the one from filtered backprojection.

2.7) 1X Gridding Reconstruction: Perform 1X gridding reconstruction, like you have done in Part I. Use 256×256 as the image size for gridkb function, since that was the original image size. Display the results as before (image, error image, overlayed cross-sections). Compute PSNR and SSIM. Comment on the results.

2.8) 2X Gridding Reconstruction: Perform 2X gridding reconstruction, like you have done in Part I. First, display the full 512×512 image that results from this gridding. Also, display the central 256×256 image, error image, and overlayed central cross-sections. Compute PSNR and SSIM. Comment on the results.

PART III – GRIDDING FOR RADIAL DATA (20 pts)

3.1) Load shepplogan_radial_data.mat. Here, both the trajectory and k-space data are 129×300 matrices, where 129 is the number of samples for each line, and 300 is the number of radial lines. Display the 2D k-space trajectory. Clearly label both axes.

3.2) Calculating Density Compensation Filter: This data is slightly different than the k-space data of a projection. Here, the radial lines start at the center of k-space and extend out in one direction only. How should you modify the density compensation filter? Explain. Using a modified version of the geometric approach, compute the area associated with each k-space point. This will also be a 129×300 matrix. Plot the resulting area values for one radial line only.

***Note:** For Questions 3.3-3.5 below, the resulting images may be flipped/transposed due to flipped k-space axes. So, flip/transpose them back to compare to the original phantom.*

3.3) Direct Summation: Compute the direct summation image using the k-space data, trajectory, and the density compensation filter that you computed. Display the results as before (image, error image, overlayed cross-sections). Compute PSNR and SSIM. Comment on the results.

3.4) 1X Gridding Reconstruction: Perform 1X gridding reconstruction. Use 256×256 as the image size for gridkb function. Display the results as before (image, error image, overlayed cross-sections). Compute PSNR and SSIM. Comment on the results.

3.5) 2X Gridding Reconstruction: Perform 2X gridding reconstruction. First, display the full 512×512 image that results from this gridding. Also, display the central 256×256 image, error image, and overlayed cross-sections. Compute PSNR and SSIM. Comment on the results.