EE-591 MAGNETIC RESONANCE IMAGING

TERM PROJECT

RECONSTRUCTION OF NON-CARTESIAN DATA USING BURS/RBURS ALGORITHM

Zheng Li

Department of Electrical Engineering

December 5, 2004

1. INTRODUCTION

There are many alternatives to 2DFT acquisition methods. These include spiral scans, radial scans, Lissajou trajectory scan and so on (Figure 1) [1]. Many of these have specific advantages over spin-warp, such as speed and SNR efficiency. The main disadvantage with these methods is the difficulty of reconstructing the resulting data sets. There are many choices for non-Cartesian data sets image reconstruction. The first approach is to collect the non-Cartesian data in a way that a previously known reconstruction method can be applied. For example Filtered Back Projection (FBP) can be applied for radial scans data set. While this solves the reconstruction problem, it usually requires compromises in data acquisition. Second, the non-Cartesian data can be demodulated point-by-point with the conjugate phase reconstruction. But this method is very slow. The most computationally efficient method of reconstruction is to resample the data onto a Cartesian grid, which enable the subsequent use of inverse fast Fourier transform (IFFT), and post compensation, if necessary.

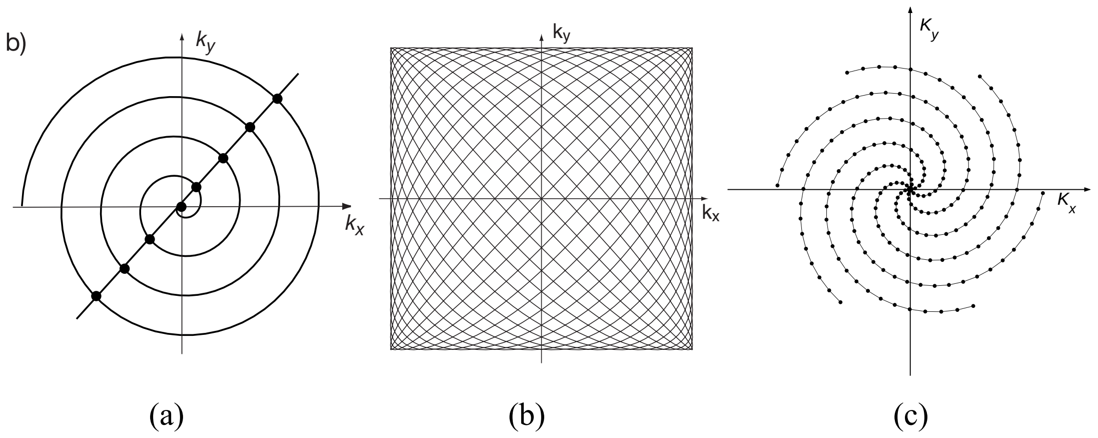


Figure 1. Some alternative acquisition methods. (a) Constant angular rate spiral, which can use projection reconstruction method. (b) Lissajou trajectory (c) Spiral trajectory used in this project

In MRI, the most widely used resampling algorithm is gridding. Usually, the gridding methods consist of four steps: 1) pre-compensation for varying sampling density; 2) convolution with a Kaiser-Bessel window onto a Cartesian grid; 3) IFFT; 4) post-compensation by dividing the image by the transform of the window. In this paper, the Block Uniform Re-Sampling (BURS) and regularization Block Uniform Re-Sampling (rBURS) are used to interpolate the non-Cartesian scan data. BURS and rBURS are both optimal/suboptimal and computationally efficient. Comparing to the conventional gridding, neither pre- nor post-compensation are required, and the results were shown to be of excellent accuracy.

2. THEORY

In this section, the theories for BURS algorithm will be introduced first. Then the theoretical analysis of noise for BURS will be addressed. Finally, one noise reduction solution for BURS, namely regularization BURS (rBURS), will be provided.

2.1 BLOCK UNIFORM RESAMPLING (BURS) ALGORITHM

The BURS algorithm can be summarized as follows:

1. Initialize an N by M matrix $\mathbf{A}^{\#}$ with zeros (N and M represent the number of the Cartesian grid points and the number of the non-uniformly sampled data points, respectively)

2. For each Cartesian grid point k_i, $(i = 1,\cdots,N)$:

   2.a. Select the $\overline{M}_i$ non-uniformly sampled points in a δ_k neighborhood of k_i.

   2.b. Select the $\overline{N}_i$ Cartesian grid points in a Δ_k neighborhood of k_i.

   2.c. Form a $\overline{M}_i \times \overline{N}_i$ matrix $\overline{\mathbf{A}}$ of the interpolation coefficients based on the sinc function.

   2.d. Compute $\overline{\mathbf{A}}^{\#}$, the truncated singular value decomposition (SVD) pseudo-inverse matrix of $\overline{\mathbf{A}}$.

   2.e. Transfer the row of $\overline{\mathbf{A}}^{\#}$ corresponding to the point k_i to the *i-th* row of $\mathbf{A}^{\#}$.

3. The uniform samples are calculated as $\mathbf{x} = \mathbf{A}^{\#}\mathbf{b}$, where $\mathbf{b}$ is a column vector containing the non-uniform data measurements.

4. Perform an inverse Fourier transform (IFT) on the resulting uniform samples.

Figure 2 illustrates how to select the $\overline{M}_i$ non-uniformly sampled points in a δ_k neighborhood of k_i and $\overline{N}_i$ Cartesian grid points in a Δ_k neighborhood of k_i. The δ_k and Δ_k neighborhoods of the k_i are illustrated as circle regions in the Figure 2. But in the implementation, other shapes of neighborhood maybe used. For example, square neighborhood can be used in Cartesian coordinate for computational efficiency and easier implementation. Square and circular shapes of neighborhood are tested in our simulations.

In BURS algorithm, the selections of values for Δ_k and δ_k will dramatically affect the final results (which will be shown in the simulations). When $\overline{N}_i > \overline{M}_i$, the pseudo-inverse can be computed as:

$$\overline{\mathbf{A}}^{\#} = (\overline{\mathbf{A}}^{\mathbf{T}}\overline{\mathbf{A}})^{-1}\overline{\mathbf{A}}^{\mathbf{T}} \tag{1}$$

When $\overline{N}_i < \overline{M}_i$, the pseudo-inverse can be computed as:

$$\overline{\mathbf{A}}^{\#} = \overline{\mathbf{A}}^{\mathbf{T}}(\overline{\mathbf{A}}\overline{\mathbf{A}}^{\mathbf{T}})^{-1} \tag{2}$$

The simulation results give some examples of how the reconstruction results varies with different combination of $\overline{N}_i$ and $\overline{M}_i$.

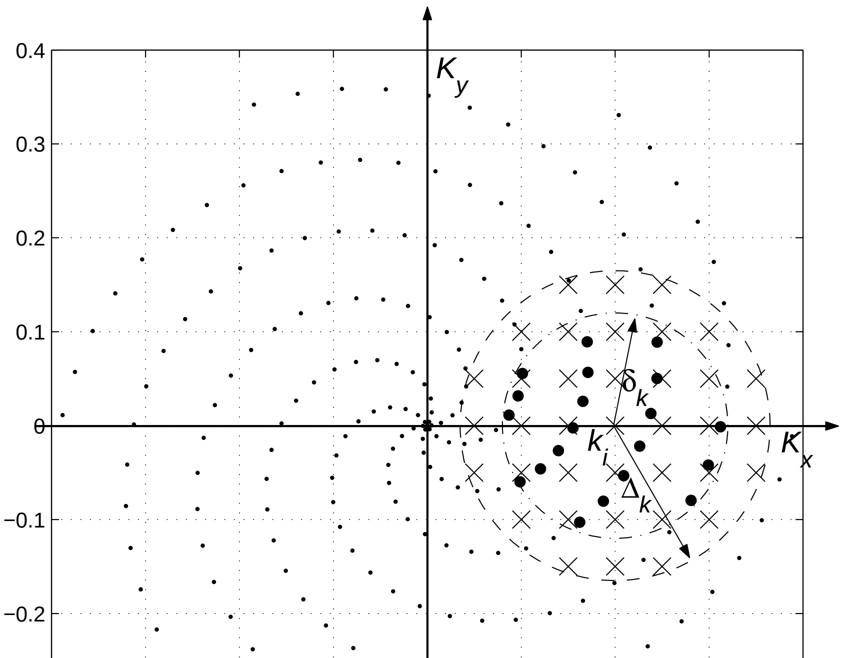


Figure 2. The illustration for BURS algorithm. The δ_k and Δ_k neighborhoods of the k_i in this plot are defined as a circle regions. The big dots represents $\overline{M}_i$ non-uniformly sampled points in a δ_k neighborhood of k_i ; the big cross signs represent $\overline{N}_i$ Cartesian grid points in a Δ_k neighborhood of k_i.

2.2 EFFECT OF NOISE

Several papers have reported that although the BURS algorithm is very accurate, it is also sensitive to the noise. As a consequence, even in the presence of a low level of measurement noise, the resulting image is often highly contaminated with noise.

In the gridding process, each uniform output point at location k_i (i=1,…,N) is linearly interpolated using $\overline{M}_i$ known data of non-uniform samples $\{k_m^i, m = 1, \cdots, \overline{M}_i\}$ which are within δ_k neighborhood of k_i:

$$f(k_i) = \sum_{m=1}^{\overline{M}_i} a_{im} f(k_m^i) \tag{3}$$

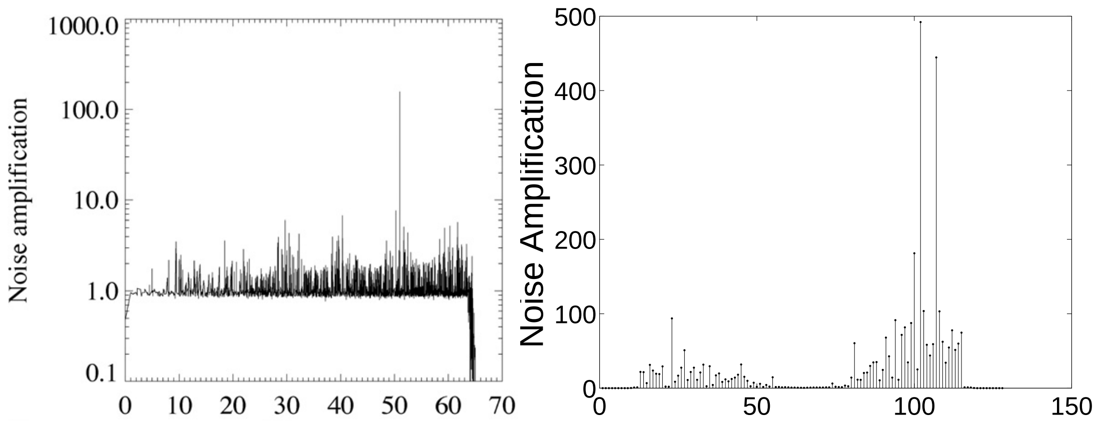
where $f(k_m^i)$ is non-uniform (non-Cartesian) input data; $f(k_i)$ is the interpolated uniform output(Cartesian) at k_i; a_{im} are the interpolation coefficients, in BURS algorithm, these coefficients are derived by pseudo-inverse. Assuming the noise is additive and consists of zero mean white Gaussian noise with variance σ^2, using above equation, it can be derived that the noise of the interpolated data $f(k_i)$ is additive Gaussian noise with zero mean and the variance σ_i^2:

$$\sigma_i^2 = \sigma^2 \sum_{m=1}^{\overline{M}_i} a_{im}^2 \tag{4}$$

Because the interpolation coefficients vary as k_i changes, the noise level σ_i^2 is space dependent in k-space domain, even if we assume the noise is i.i.d in original non-Cartesian k-space data. In Rosenfeld's paper [4], the k-space "noise amplification" is defined as:

$$\Omega_i = \sigma_i / \sigma = \sqrt{\sum_{m=1}^{\overline{M}_i} a_{im}^2} \tag{5}$$

Rosenfeld [4] tested this noise effect of BURS by using a four-interleaf spiral trajectory. The Ω_i was calculated for each uniform point. We also did the test on our spiral trajectory and get similar results, which are show in Figure 3.



(a) (b)

Figure 3. Noise amplification Ω_i using BURS for spiral trajectories. (a) the result from [4], x-axis represents the distance from the origin of the k-plane. (b) The Ω_i values for the row $k_y = 0$ based on our own spiral trajectory. The x-axis represents the k_x coordinate.

Both results show that most points have a noise amplification of about unity, however a substantial number of points have extreme high noise amplification number. This is the reason that cause the reposted noise contaminated result for BURS algorithm. Although only small part of k-space point have very high noise level, after the Fourier transform, the noise will distributed across the whole image.

Equation (5) shows that the high noise amplification coefficients are due to the high value of interpolation coefficients, which is the row of $\overline{\mathbf{A}}^{\#}$ corresponding to the point k_i. We know that the solution of an inverse problem is unstable, which means that small changes in the input data may lead to large perturbations in the results (ill-posed problem). So it becomes clear that, the ill-conditioned matrixes $\overline{\mathbf{A}}^{\mathbf{T}}\overline{\mathbf{A}}$ cause the large perturbations in the coefficients and finally result in large noise level in reconstructed image.

2.3 REGULARIZED BLOCK UNIFORM RESAMPLING ALGORITHM

The basic ideal of the rBURS is to stabilize the matrix inversion solution by modifying the problem in such a way that the inversion solution becomes less sensitive to small perturbations in the data. At the same time, the solution to the modified problem must remain close to the original solution. Thus the original solution $\mathbf{x} = \mathbf{A}^{\#}\mathbf{b}$ is replaced by the approximate solution $\mathbf{x} = \mathbf{A}_{\rho}^{\#}\mathbf{b}$ such that

$$\lim_{\rho \to 0} \mathbf{A}_{\rho}^{\#}\mathbf{b} = \mathbf{A}^{\#}\mathbf{b} \tag{6}$$

where ρ is a positive smoothing parameter. We now focus on one type of regularization technique, referred to as "*spectral windowing*". By using equation (1),

$$\overline{\mathbf{A}}^{\#}\mathbf{b} = (\overline{\mathbf{A}}^{\mathbf{T}}\overline{\mathbf{A}})^{-1}\overline{\mathbf{A}}^{\mathbf{T}}\mathbf{b} = \sum_{k} \alpha_k^{-1}(\mathbf{v}_k^T\mathbf{b})\mathbf{u}_k \tag{7}$$

where $\mathbf{v}_k$ are eigenvectors of $\mathbf{A}\mathbf{A}^T$; $\mathbf{u}_k$ are eigenvectors of $\mathbf{A}^T\mathbf{A}$; $\alpha_k, \alpha_1 \geq \alpha_2 \geq \cdots$ are singular values. $\mathbf{A}_{\rho}^{\#}\mathbf{b}$ is computed as:

$$\overline{\mathbf{A}}_{\rho}^{\#}\mathbf{b} = \sum_{k} W_{\rho k}\alpha_k^{-1}(\mathbf{v}_k^T\mathbf{b})\mathbf{u}_k \tag{8}$$

where $W_{\rho k}$ is called the "*window coefficients*". There are many different definitions for these coefficients including "*Truncated singular system expansion*" and "*Tikhonov filter*" which are defined separately as:

"*Truncated singular system expansion*" : $$W_{\rho k} = \begin{cases} 1 & k < (1/\rho) \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

"*Tikhonov filter*" : $$W_{\rho k} = \frac{\alpha_k^2}{\alpha_k^2 + \rho} \tag{10}$$

When $W_{\rho k}$ is defined as (10), it can be proved that $\mathbf{x} = \mathbf{A}_{\rho}^{\#}\mathbf{b}$ can be computed as:

$$\mathbf{x} = \overline{\mathbf{A}}_{\rho}^{\#}\mathbf{b} = \sum_{k} W_{\rho k}\alpha_k^{-1}(\mathbf{v}_k^T\mathbf{b})\mathbf{u}_k = (\mathbf{A}^T\mathbf{A} + \rho\mathbf{I})^{-1}\mathbf{A}^T\mathbf{b} \tag{11}$$

In our implementation, equation (11) is employed for regularization.

3. IMPLEMENTATION

In the real system, given the non-Cartesian k-space trajectory, δ_k and Δ_k, by using the BURS algorithm described in section 2.1, the matrix $\mathbf{A}^{\#}$ can be calculated and saved pre reconstruction. Whenever the data sampling is done and reconstruction is needed, the matrix $\mathbf{A}^{\#}$ can be reloaded and used directly. By this way, the computational time is shortened dramatically. But this method needs to process the huge size matrix $\mathbf{A}^{\#}$, which makes the data handling not so easy. In addition, in order to test the different parameter combinations in this paper, the parameters δ_k and Δ_k change from time to time, which makes $\mathbf{A}^{\#}$ change each time. So, in our simulation, instead of storing the huge matrix $\mathbf{A}^{\#}$ and interpolating all points one time, the Cartesian point interpolation is done point by point through the whole image.

3.1 Implementation of BURS/rBURS algorithm

1. Initialize an $\mathrm{N} \times \mathrm{N}$ matrix $\mathbf{M}$ with zeros (the size of the image is $\mathrm{N} \times \mathrm{N}$)
2. For each Cartesian grid point $\mathbf{M}_{ij}$, $(i = 1,\cdots,N;\ j = 1,\cdots,N)$:
    - 2.a. Select the $\overline{M}_{ij}$ non-uniformly sampled points in a δ_k neighborhood of $\mathbf{M}_{ij}$. Form a $\overline{M}_{ij} \times 1$ column vector $\mathbf{d}_{ij}$ using $\overline{M}_{ij}$ known non-uniformly sampled data.
    - 2.b. Select the $\overline{N}_{ij}$ Cartesian grid points in a Δ_k neighborhood of $\mathbf{M}_{ij}$.
    - 2.c. Form a $\overline{M}_{ij} \times \overline{N}_{ij}$ matrix $\overline{\mathbf{A}}$ of the interpolation coefficients based on the sinc function.
    - 2.d-BURS. For BURS algorithm, $\overline{\mathbf{A}}^{\#}$=pseudo-inverse matrix of $\overline{\mathbf{A}}$.
    - 2.d-rBURS. For rBURS algorithm, $\overline{\mathbf{A}}^{\#} = (\overline{\mathbf{A}}^{\mathbf{T}}\overline{\mathbf{A}} + \rho \cdot \mathbf{I})^{-1}\overline{\mathbf{A}}^{\mathbf{T}}$.
    - 2.e. Let $\overline{\mathbf{a}}^{\#}$ = row of $\mathbf{A}^{\#}$ corresponding to the point $\mathbf{M}_{ij}$, $\mathbf{M}_{ij} = \overline{\mathbf{a}}_{ij}^{\#} \cdot \mathbf{d}_{ij}$
3. Perform an inverse Fourier transform (IFT) on the $\mathbf{M}$.

3.2 Shape of the Neighborhood

In the real implementation, the neighborhood of the point (x_0, y_0) within radius r can be defined at least in two different ways:

1. Circular Neighborhood with radius r_C:

$$\{neighhorhood\} = \{(x, y) \mid \|(x, y) - (x_0, y_0)\| \leq r_C\} \tag{12}$$

2. Square Neighborhood with radius r_S:

$$\{neighhorhood\} = \{(x, y) \mid \max(|x - x_0|, |y - y_0|) \leq r_S\} \tag{13}$$

Notice that, when the radiuses have the same value, the square neighborhood has larger coverage area than that of circular neighborhood. To make both definitions have the same coverage area, r_C and r_S should satisfy:

$$\pi \cdot r_C^2 = (2r_S)^2 \quad \Rightarrow \quad r_S = r_C\sqrt{\frac{\pi}{4}} \tag{14}$$

Circular neighborhood has the advantage that the closest (in the sense of norm2) points from the center of the neighborhood are selected. Square neighborhood will select some points (in the corner of the square) not so close to the center, but square neighborhood is easier to implement and computational more effective. In the simulation, two neighborhood definitions are tested and compared.

To make the comparison equitable, same "effective" radius r is used for different shapes, then r_C and r_S are computed using equation (14). Suppose both neighborhood have same "effective" radius r, then:

For circular neighborhood: $r_C = r$; For square neighborhood: $r_S = r\sqrt{\frac{\pi}{4}}$ (15)

4. Simulations and Results

The data set used here is a simulated phantom using a spiral acquisition with 6 interleaves of 1536 samples. Center part of the trajectory is illustrated in Figure 2. Four problems are studied in our simulation:

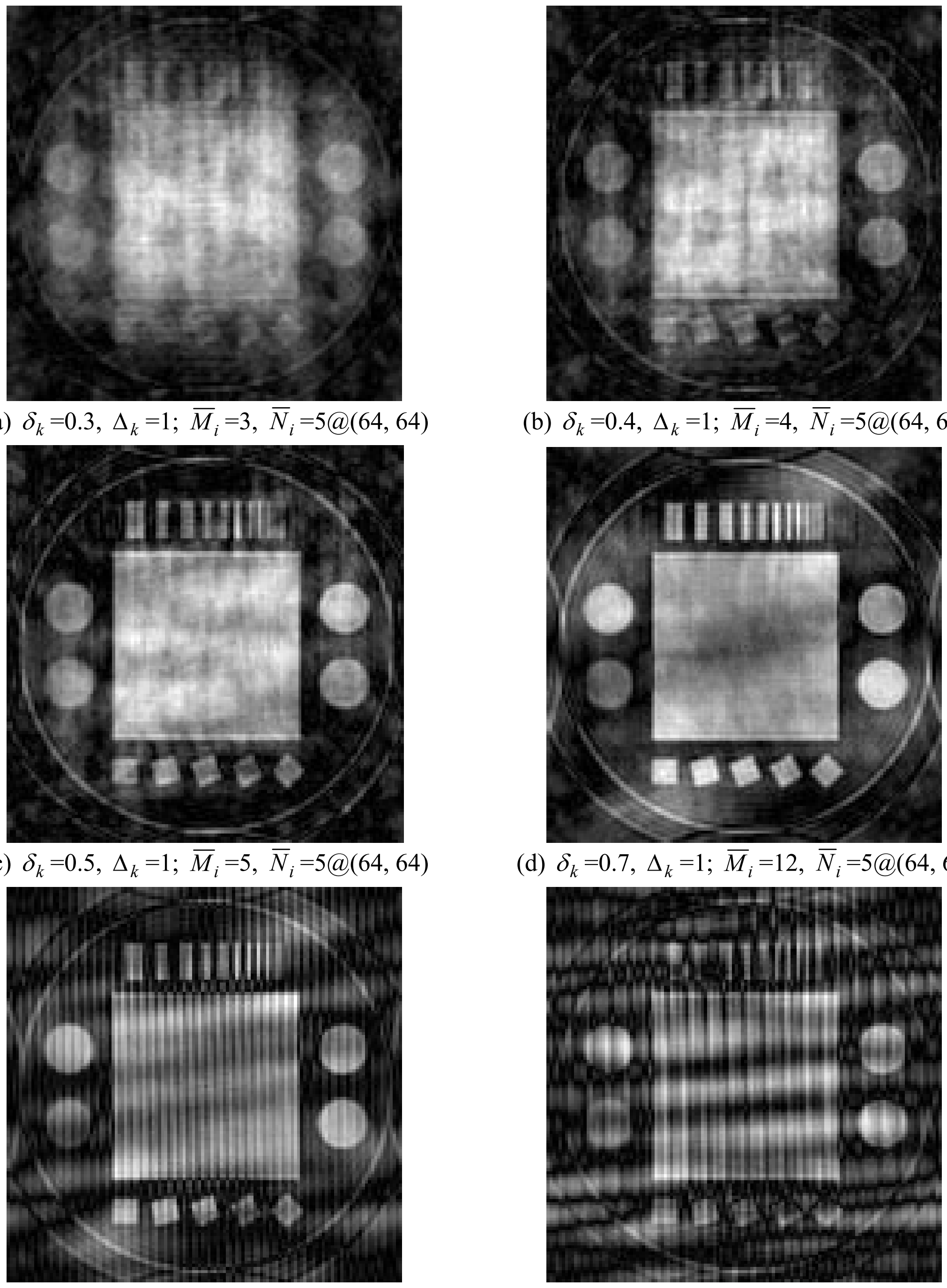
1) *How the reconstruction result changes with* Δ_k *and* δ_k. Figure 4 shows the results for Δ_k=1 while δ_k varies from 0.3~1. Figure 5 shows the results for Δ_k=2 while δ_k varies from 0.5~1.4. Beyond these δ_k ranges, the results become unacceptable.

2) *How the shape of neighborhood (circular vs. square) affect the results*. In the simulation, the circular neighborhood is always used for δ_k (non-Cartesian), circular AND square neighborhoods are tested for Δ_k (Cartesian points). Set the effective radius Δ_k=1, 2, 3 respectively, δ_k values are chosen such that the best reconstruction achieved for each case. Circular and square neighborhoods are tested with same effective radius Δ_k and δ_k settings. Figure 6 shows the results.

3) *BURS vs. rBURS algorithm*. One image with high-SNR and one with low-SNR are tested using BURS and rBURS algorithm respectively. The low-SNR image is produced by adding Gaussian noise to K-space spiral sampled data.

4) *Compare the result of BURS/rBURS with "true" image.* The image reconstructed by gridding w/ Pre-Density Compensation & Deapodization is used as the "original"

image. Then we compare the best results produced by BURS and rBURS with the "original" image. Figure 8 shows the images and the difference images. Figure 9. shows the profile of the images.



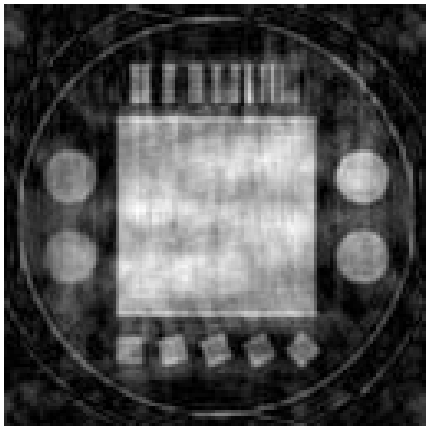
(a) δ_k =0.3, Δ_k =1; $\overline{M}_i$ =3, $\overline{N}_i$ =5@(64, 64) (b) δ_k =0.4, Δ_k =1; $\overline{M}_i$ =4, $\overline{N}_i$ =5@(64, 64)

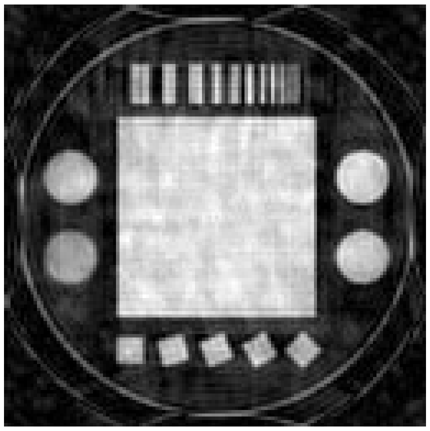
(c) δ_k =0.5, Δ_k =1; $\overline{M}_i$ =5, $\overline{N}_i$ =5@(64, 64) (d) δ_k =0.7, Δ_k =1; $\overline{M}_i$ =12, $\overline{N}_i$ =5@(64, 64)

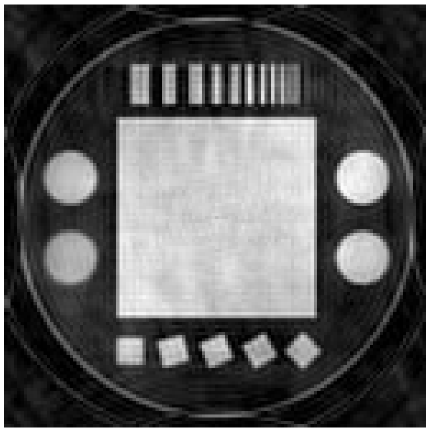
(e) δ_k =0.9, Δ_k =1; $\overline{M}_i$ =35, $\overline{N}_i$ =5@(64, 64) (f) δ_k =1, Δ_k =1; $\overline{M}_i$ =37, $\overline{N}_i$ =5@(64, 64)

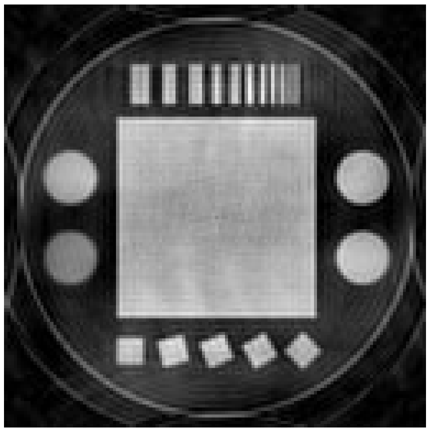
Figure 4. Comparing different Δ_k and δ_k combinations for BURS algorithm. Circular neighborhoods are used for Δ_k and δ_k . Fix Δ_k =1, δ_k value is changed from 0.3 to 1. $\overline{M}_i$ and $\overline{N}_i$ values @ (k_x, k_y) =(64,

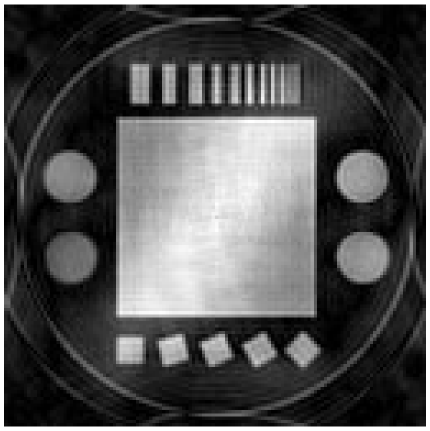
64) are provided for each case. It shows that when bad underdetermined case ($\overline{M}_i >> \overline{N}_i$) occurs, some artifacts will appear in the reconstructed image.


(a) δ_k =0.5, Δ_k =2; $\overline{M}_i$ =5, $\overline{N}_i$ =13@(64, 64)


(b) δ_k =0.7, Δ_k =2; $\overline{M}_i$ =12, $\overline{N}_i$ =13@(64, 64)


(c) δ_k =0.9, Δ_k =2; $\overline{M}_i$ =35, $\overline{N}_i$ =13@(64, 64)


(d) δ_k =1, Δ_k =2; $\overline{M}_i$ =37, $\overline{N}_i$ =13@(64, 64)


(e) δ_k =1.2, Δ_k =2; $\overline{M}_i$ =43, $\overline{N}_i$ =13@(64, 64)

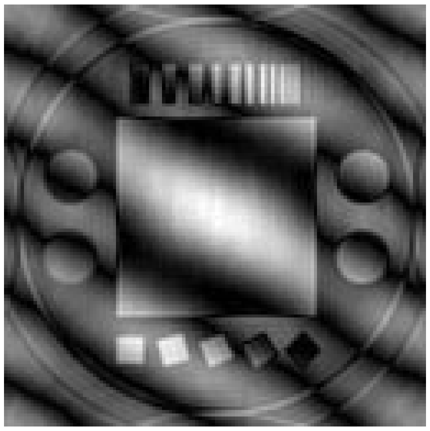
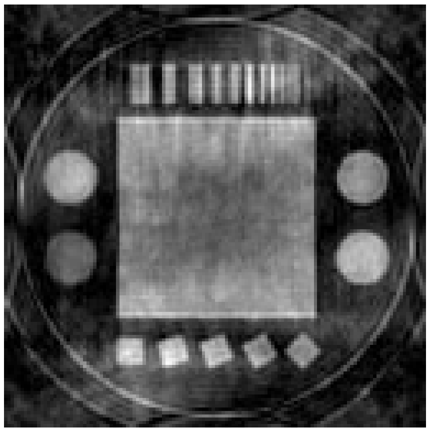
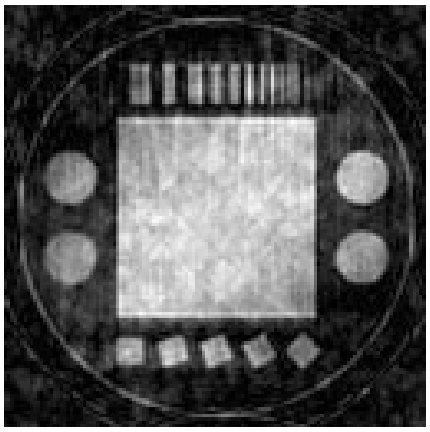
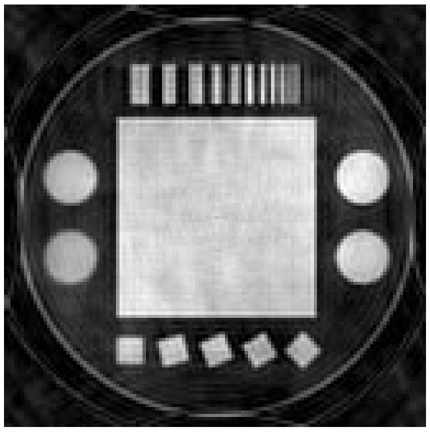
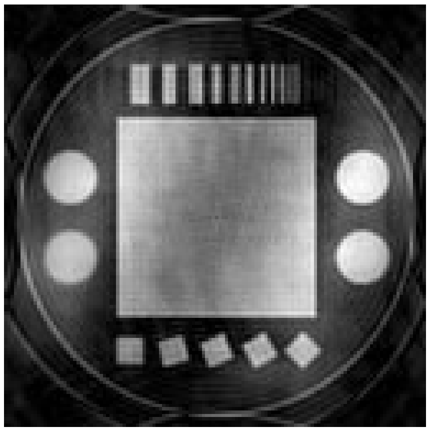

(f) δ_k =1.4, Δ_k =2; $\overline{M}_i$ =51, $\overline{N}_i$ =13@(64, 64)

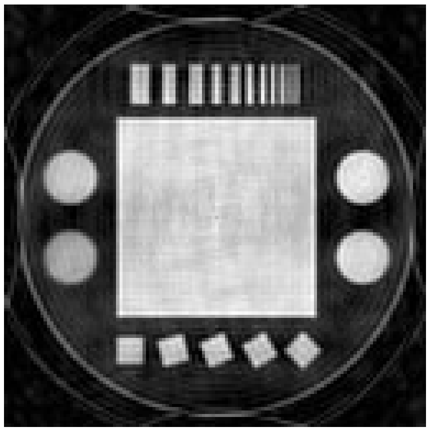
Figure 5. . Comparing different Δ_k and δ_k selections for BURS algorithm. Circular neighborhoods are used for Δ_k and δ_k . Fix Δ_k =2, δ_k value is changed from 0.4 to 1.4. $\overline{M}_i$ and $\overline{N}_i$ values @ (k_x, k_y) =(64, 64) are provided for each case. It shows that when bad underdetermined case ($\overline{M}_i >> \overline{N}_i$) occurs, some artifacts will appear in the reconstructed image.


(a) δ_k =0.6, Δ_k =1, Circular


δ_k =0.6, Δ_k =1, Square


δ_k =0.9, Δ_k =2, Circular


δ_k =0.9, Δ_k =2, Square


δ_k =1.3, Δ_k =3, Circular

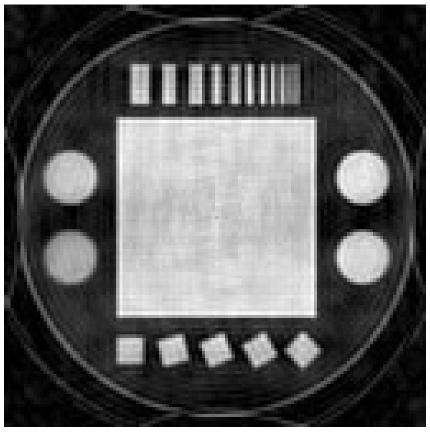
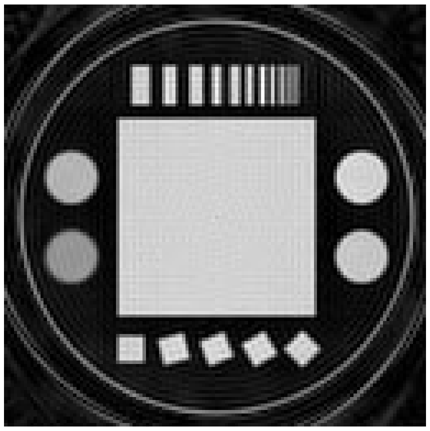
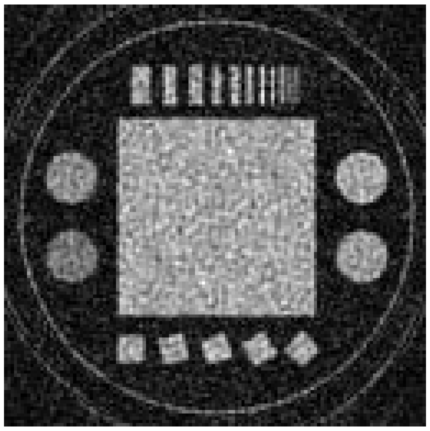
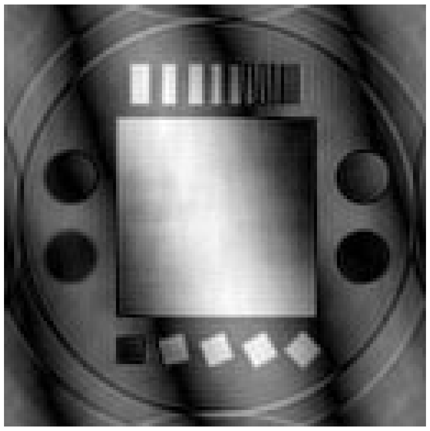
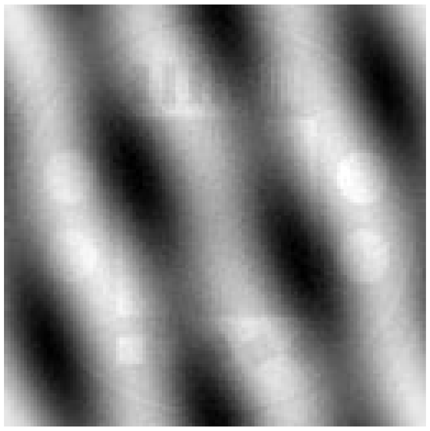

δ_k =1.3, Δ_k =3, Square

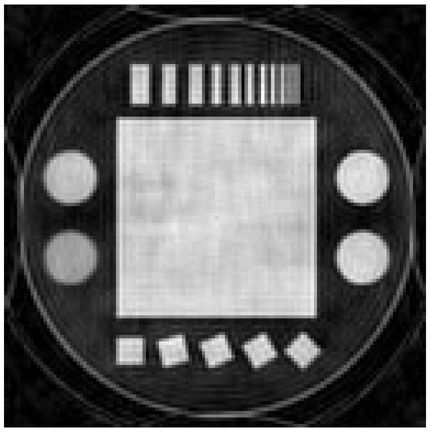
Figure 6. Comparing "circular" neighborhood with "square" neighborhood for BURS algorithm. Circular neighborhoods are always used for δ_k ; circular and square neighborhoods are tested for Δ_k . Fix the "effective" neighborhood radius Δ_k =1, 2 and 3, δ_k values are selected such that best reconstruction result is achieved for each case. The results show that different shapes of neighborhood have some but limited effect (circular neighbor is little bit better) on reconstructed image.


(1a) Original w/ High SNR


(2a) Original w/ Low SNR


(1b) BURS result w/ High SNR


(2b) BURS result w/ Low SNR


(1c) rBURS result w/ High SNR

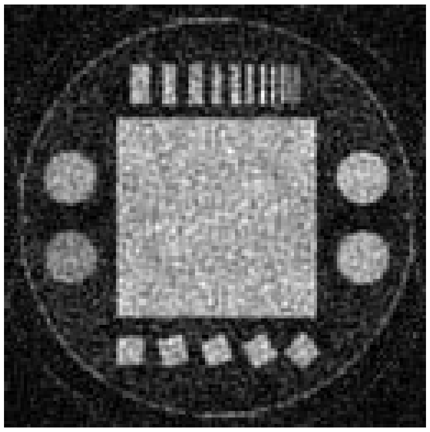
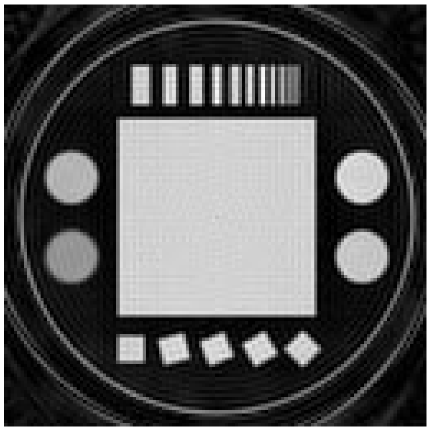
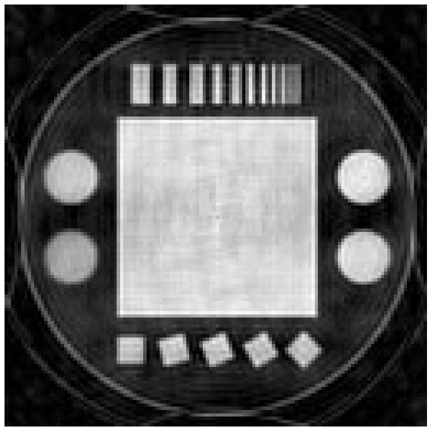
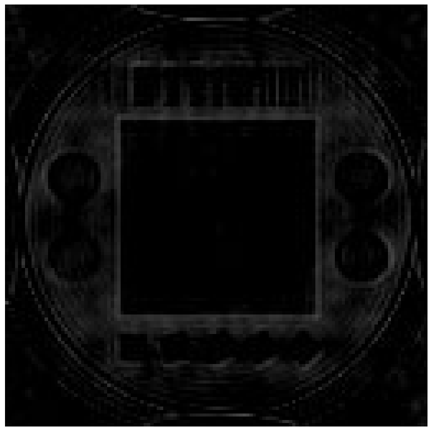
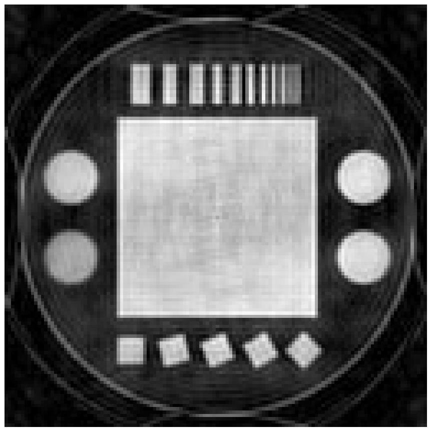

(2c) rBURS result w/ Low SNR

Figure 7. Compare BURS with rBURS algorithm. Left column is for High SNR case, right column is for Low SNR case. The "original" image is produced by gridding with Pre-Density Compensation & Deapodization. Low SNR image is produced by adding Gaussian noise in K-space. For all BURS/rBURS reconstructions, set δ_k =1.5, Δ_k =3. Regularization smoothing parameter ρ=0.01.


(a) Original Image


(b) BURS


(c) Difference Image
between BURS and “original” image


(d) rBURS

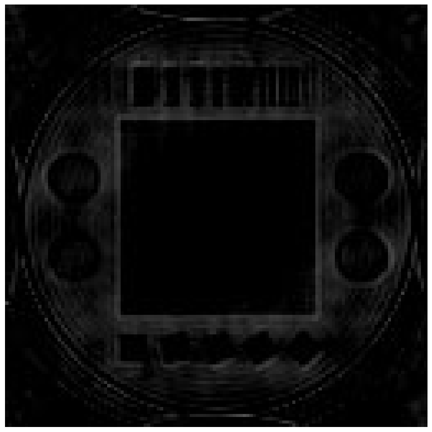
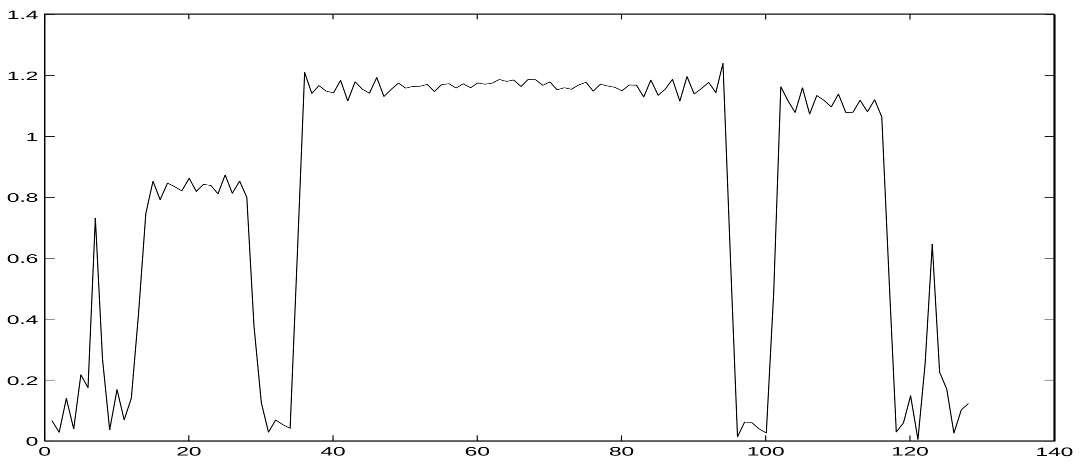
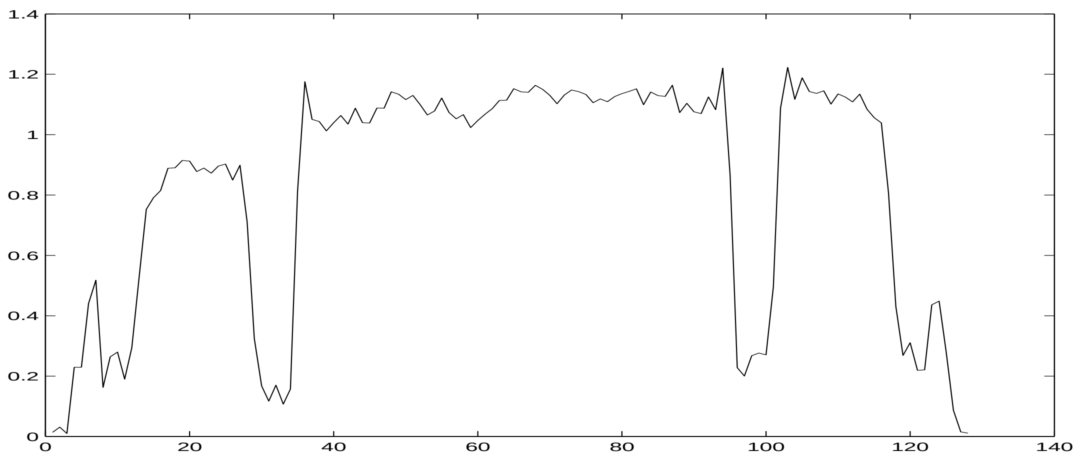

(e) Difference Image for rBURS
between rBURS and “original” image

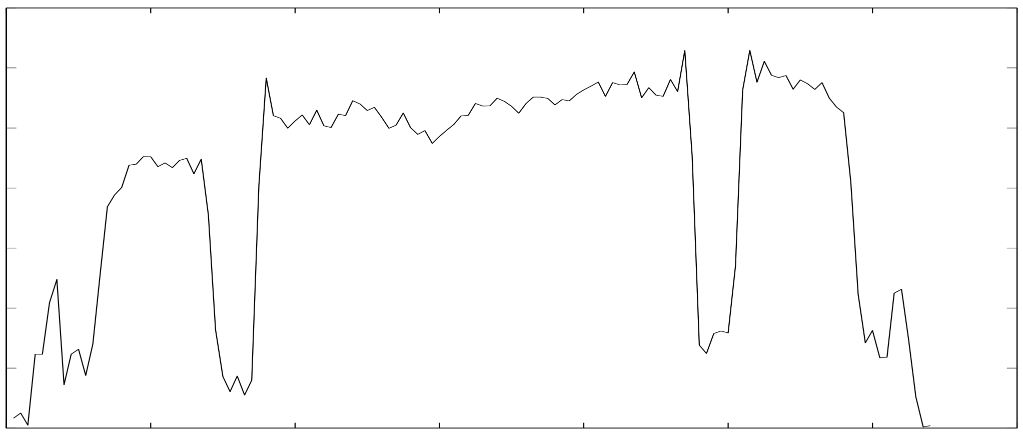
Figure 8. Compare the best BURS and best rBURS results with “original” image. The “original” image is produced by gridding with Pre-Density Compensation & Deapodization. The results shown here for BURS and rBURS are the best results we get during the simulation. The difference image shown on the right is the difference between BURS/rBURS with the original image.



(a) Profile for Original Image @ row x=78



(b) Profile for BURS Image @ row x=78



(c) Profile for rBURS Image @ row x=78

Figure 9. The profiles for different images shown in Figure 8.

5. CONCLUSIONS

(1) *Effect of neighborhood radius* δ_k *and* Δ_k .

Figure 4 and 5 show that (i) If δ_k is too small (δ_k <0.3), then no matter how large the Δ_k is, we can not get very good result. (ii) Keep Δ_k fixed, when δ_k increases from a very small number (around 0.3), the result will become better first, then become worse. For the tested cases, when $\delta_k \approx \Delta_k / 1.5 \sim \Delta_k / 2.0$, the BURS produces best result. (iii) When δ_k fixed, increasing the value of Δ_k , the result becomes better.

If we check the BURS algorithm more carefully, we will find that although δ_k , Δ_k will affect the result, they are not the root of the reason. In fact, it is $\overline{M}_i$ and $\overline{N}_i$ values who really affect the result! In order to produce good results, $\overline{M}_i$ should NOT exceed $\overline{N}_i$ too much. If $\overline{M}_i >> \overline{N}_i$ occurs for some points (often occurs around the origin in k-plane, because our spiral data is more dense around the origin which makes $\overline{M}_i$ achieve it's maximum value around the origin), we can still get the result, however, there will be some low frequency artifacts in the images (see Figure 4e, 4f, 5e, 5f). Now we can explain the (i)~(iii) listed above based on $\overline{M}_i$ and $\overline{N}_i$ values. (i) $\overline{N}_i$ and $\overline{M}_i$ should not be too small. (ii) $\overline{M}_i$ can not exceed $\overline{N}_i$ too much all the time, otherwise the result will have some low-frequency artifact. (iii) the bigger the $\overline{N}_i$ and $\overline{M}_i$, the better the result.

(2) *Effect of the shape of the neighborhood.*

Our results show that BURS with circular neighborhood will produce a little bit better results than that of square neighborhood, but the differences are small (Figure 6).

(3) *BURS vs. rBURS.*

BURS is sensitive to the high level of noise as well as underdetermined case (Figure 7-1b, 2b). On the contrary, the rBURS is robust to the high level of noise as well as underdetermined case (Figure 7-1c, 2c). rBURS is also robust in the case of combination of high noise and underdetermined matrix. Even in this worst case, the result of rBURS (Figure 7-2c) is still very close to the "original" image (Figure 7-2a), which is produced using gridding with Pre-Density Compensation & Deapodization.

(4) *Fidelity of BURS/rBURS*

By checking the reconstructed images, difference images (Figure 8) and the profiles of the reconstructed images (Figure 9), we can conclude that (I) The best results produced by BURS and rBURS are very close to the "original" image. (II) There are some small errors occur in high frequency components, i.e. some errors around the edges.

REFERENCES

[1] John Pauly. Image Reconstruction Textbook (in progress), Chaper 5: Reconstruction of non-Cartesian Data.

[2] Rosenfeld D. An optimal and efficient new gridding algorithm using singular value decompositioin. Magnetic Resonance in Medicine 1998; 40:12-23

[3] Moriguchi H, Wendt M, Duerk JL. Applying the uniform resampling (URS) algorithm a a Lissajous trajectory: fast image reconstruction with optimal gridding. Magnetic Resonance in Medicine 2000; 44:766-781

[4] Rosenfeld, Daniel. New Approach to Gridding using Regularization and Estimation Theory, Magnetic Resonance in Medicine 2002; 48: 193-202

APPENDIX MATLAB CODES

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EE591 MRI                                   %
% Term Project    BURS & rBURS                %
% Zheng Li, Dec. 2004                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;       close all;    n=128;

load rt_spiral.mat;    %{d: data; k: sampling kernel; w: weight}
%load noise_spiral;     %{nd: additive Gausian noise} load same random noise data each time
%d=nd+d;

erc=3;     % (effective) radius of delta-k neighborhood in Cartesian coordinate
rk=1.3;    % radius of delta-k neighborhood in Non-Cartesian coordinate
shape='c'; % shape of the neighorhood, 'c'-->circular; 's'-->square

% for sqare neighorhood, rc=effective r * sqrt(pi/4)
if isequal(shape, 's')
   rc=erc*sqrt(pi/4);
   disp(strcat('Square Neighborhood, Radius=', num2str(rc)));
end;
% for circular neighorhood, rc=effective r
if isequal(shape, 'c')
   rc=erc;
   disp(strcat('Circular Neighborhood, Radius=', num2str(rc)));
end;

if (0) %BURS
   [MB, OMB]= gridBURS(d,k,n, rk, rc, shape);  % call BURS gridding function;
   imgB=ift(MB);
   figure; imagesc(abs(imgB));
   axis square;   colormap('gray');   colormenu;  axis off;
else % rBURS
```

```
   [MrB, OMrB]=gridrBURS(d, k, n, rk, rc, 0.01, shape); % call rBURS gridding function
   imgrB=ift(MrB);
   figure; imagesc(abs(imgrB));
   axis square;    colormap('gray');   colormenu;  axis off;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%
%             function    BURS                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%
function [M, OM] = gridBURS(d,k,n,rk,rc,shape)

% function [M, OM] = gridBURS(d,k,n,rk,rc)
% Block Uniform ReSampling method for gridding
%       d -- k-space data
%       k -- k-trajectory, scaled -0.5 to 0.5
%       n -- image size
%       rk-- non-cartesian kernel radius
%       rc-- cartesian kernel radius
%       shape-- choose circle (=='c') neighborhood
%            or square neighborhood (=='s') for Cartesian points
%
%       M -- K-space interpolated data
%       OM-- noise amplification (defined in Rosenfeld 2002 Magn Reson Med)
%
% Zheng Li, Nov. 2004

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

% convert k-space samples to matrix indices
nxk=(n+1)/2 + (n-1)*real(k);
nyk=(n+1)/2 + (n-1)*imag(k);
% initialize the output matrix
M=zeros(n,n);
OM=zeros(n,n);

% change the cartesian coordinate to one column, so that we can find
% the cartesian point withing "rc" easily.
[mxc, myc]=meshgrid(1:n, 1:n);
mxc=mxc(:);
myc=myc(:);

% main loop, compute the BURS gridding value for each point
for xc=ceil(1+rc):floor(n-rc)
   for yc=ceil(1+rc):floor(n-rc)
      if shape=='s'
         [mxdc, mydc]=meshgrid(ceil(xc-rc):floor(xc+rc), ceil(yc-rc):floor(yc+rc));
         % get the Cartesian points in "square" neighborhood of (xc,yc)
         xyc=mxdc(:)+i*mydc(:);
         indc=ones(length(xyc),1);     % just to make to !=[]
      elseif shape=='c'
         % find the index of the Cartesian points in the "rc" neighborhood of (xc+yc*i)
         indc=find( ((mxc-xc).^2 + (myc-yc).^2) <=rc^2+eps );
         if ~(isempty(indc))
            % get the Cartesian points in "circular" neighborhood of (xc,yc)
            xyc=mxc(indc)+i*(myc(indc));
         end;
      end;
      % find the index of the Non-Cartesian points in the "kc" neighborhood of (xc+yc*i)
      indk=find( ((nxk-xc).^2 + (nyk-yc).^2) <=rk^2+eps );
      % print the N (# of Cartesian points), and M (# of Non-Cartesian points)
      % @ several positions along ky=0 axis.
      if (yc==64) & (mod(xc, 16)==0)
         disp(strcat('N=', num2str(length(indc)), ...
                  ',  M=', num2str(length(indk)), ...
                  '  @(', num2str(xc), ',', num2str(yc), ')'));
      end;
      if ~(isempty(indk)) & ~(isempty(indc))
         % get the Non-Cartesian points in "circular" neighborhood of (xc,yc)
         xyk=nxk(indk)+i*(nyk(indk));
         A=interp2sinc(xyc, xyk);
         pA=pinv(A);     % pinv can handle over/under determined cases automatically
         ind0=find( (xyc==xc+yc*i) );
         M(xc, yc)=pA(ind0, :)*d(indk);
         OM(xc, yc)=sqrt( sum(pA(ind0, :).^2) );
      end;
   end;
end;
%end

function [M, OM] = gridrBURS(d,k,n,rk,rc,r,shape)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              function        rBURS                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 function [M, OM] = gridrBURS(d,k,n,rk,rc,r,shape)
% regularized Block Uniform ReSampling method for gridding
%      d -- k-space data
%      k -- k-trajectory, scaled -0.5 to 0.5
%      n -- image size
%      rk-- non-cartesian kernel radius
%      rc-- cartesian kernel radius
%      shape-- choose circle (=='c') neighborhood
%            or square neighborhood (=='s') for Cartesian points
%      r -- regularization smoothing parameter
%
%      M -- K-space interpolated data
%      OM-- noise amplification (defined in Rosenfeld 2002 Magn Reson Med)
%
% Zheng Li, Nov. 2004

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

% convert k-space samples to matrix indices
nxk=(n+1)/2 + (n-1)*real(k);
nyk=(n+1)/2 + (n-1)*imag(k);
% initialize the output matrix
M=zeros(n,n);
OM=zeros(n,n);

% change the cartesian coordinate to one column, so that we can find
% the cartesian point withing "rc" easily.
[mxc, myc]=meshgrid(1:n, 1:n);
mxc=mxc(:);
myc=myc(:);

% main loop, compute the BURS gridding value for each point
for xc=ceil(1+rc):floor(n-rc)
   for yc=ceil(1+rc):floor(n-rc)
      if shape=='s'
         [mxdc, mydc]=meshgrid(ceil(xc-rc):floor(xc+rc), ceil(yc-rc):floor(yc+rc));
         % get the Cartesian points in "square" neighborhood of (xc,yc)
         xyc=mxdc(:)+i*mydc(:);
         indc=ones(length(xyc),1);     % just to make to !=[]
      elseif shape=='c'
         % find the index of the Cartesian points in the "rc" neighborhood of (xc+yc*i)
         indc=find( ((mxc-xc).^2 + (myc-yc).^2) <=rc^2+eps );
         if ~(isempty(indc))
            % get the Cartesian points in "circular" neighborhood of (xc,yc)
            xyc=mxc(indc)+i*(myc(indc));
         end;
      end;
      % find the index of the Non-Cartesian points in the "kc" neighborhood of (xc+yc*i)
      indk=find( ((nxk-xc).^2 + (nyk-yc).^2) <=rk^2+eps );
      % print the N (# of Cartesian points), and M (# of Non-Cartesian points)
      % @ several positions along ky=0 axis.
```

```
        if (yc==64) & (mod(xc, 16)==0)
            disp(strcat('N=', num2str(length(indc)), ...
                    ',  M=', num2str(length(indk)), ...
                    '  @(', num2str(xc), ',', num2str(yc), ')'));
        end;
        if ~(isempty(indk)) & ~(isempty(indc))
            % get the Non-Cartesian points in "circular" neighborhood of (xc,yc)
            xyk=nxk(indk)+i*(nyk(indk));
            A=interp2sinc(xyc, xyk);
            % compute the regularized pseduo-inverse using "Tikhonov" window coefficients
            if length(indk)<=length(indc)   % underdetermined case
                pA=A'*inv(A*A'+r*eye(length(indk)));
            else % overdetermined case
                pA=inv(A'*A+r*eye(length(indc)))*A';
            end;
            ind0=find( (xyc==xc+yc*i) );
            M(xc, yc)=pA(ind0, :)*d(indk);
            OM(xc, yc)=sqrt( sum(pA(ind0, :).^2) );
        end;
    end;
end;
%end

%%%%%%%%%%%%%%%%%%%%%%%%%
%              function interp2sinc                  %
%%%%%%%%%%%%%%%%%%%%%%%%%
function A=interp2sinc(xyc, xyk);
% function A=interp2sinc(xyc, xyk)
% 2D interpolation using sinc function.
% xyc: the column vector of Cartesian point positions,
%       each position is represented by a complex number.
% xyk: the column vector of Non-Cartesian point postion.
% A: the linear transform matrix s.t. the (DATA@xyk)=A*(DATA@xyc);
%
% Zheng Li, Nov. 2004

% pxy* is the postions in Cartesian(c) and Non-cartian(k)
[pxyc, pxyk]=meshgrid(xyc, xyk);
% the distances between each points of Cartesian and Non-Cartesian
dist=pxyc-pxyk;
A=sinc(real(dist)).*sinc(imag(dist));

% end
```