Biomedical Imaging Exercise – Week 8

Name: Alkinoos Sarioglou

Student ID: 20-947-743

1. Task 3.1

a) The number of photons N_0 incident to the object during an X-ray burst can be calculated by first calculating the number of electrons liberated by the cathode per unit time depending on the tube current: $Ne = \frac{Ia}{e}$, where I_a, the tube current and e, the electron charge. Then, the number of photons that will be released by the tube and will reach the body is calculated by: $N_0 = N_e \cdot yield \cdot U_A \cdot dt$, where U_A, the anode voltage and dt, the burst duration. The code used on MATLAB to calculate this is shown in the following figure:

```
% TASK 3.1 UNCOMMENT AND FILL IN HERE
e               = 1e-19*1.6;            % electron charge     [C]
Ne              = (Ia)/e;               % #electrons          [1/s]
N0              = Ne*yield*Ua*dt;       % #photons            []
```

b) The number of photons transmitted through the objects depend on the linear attenuation coefficients and as a result on the sum of the bin intensities of each projection angle. In this part, three new matrices are defined to help in the calculation of the detected photons. These are: N, which includes the number of detected photons for each projection angle without noise, N_noisy, which includes the number of detected photons for each projection angle with noise and sum_proj, which calculates the sum of projections of all the bins in each specific projection angle. These are defined as following:

```
N_noisy = zeros(1,length(projection_angles));    % Number of detector's incident photons with noise
N = zeros(1,length(projection_angles));          % Number of detector's incident photons
sum_proj = zeros(1,length(projection_angles)); % Sum of projections in each angle
```

The higher the values of linear attenuation coefficients are, the less photons are detected. Also, there is an exponential decrease in the number of detected photons for increase in linear attenuation coefficients values (Beer-Lamberts Law). Therefore, the following code snippet is written to calculate the number of detected photons without noise in every projection angle:

```
% ------------------------------------------------------------------
% Calculate photons incident to detector using Beer-Lambert's law
% ------------------------------------------------------------------
% TASK 3.1 FILL IN HERE
sum_proj(1,idx) = sum(phantom.sino(:,idx),1);
N(1,idx) = N0*exp(-(sum_proj(1,idx)));
```

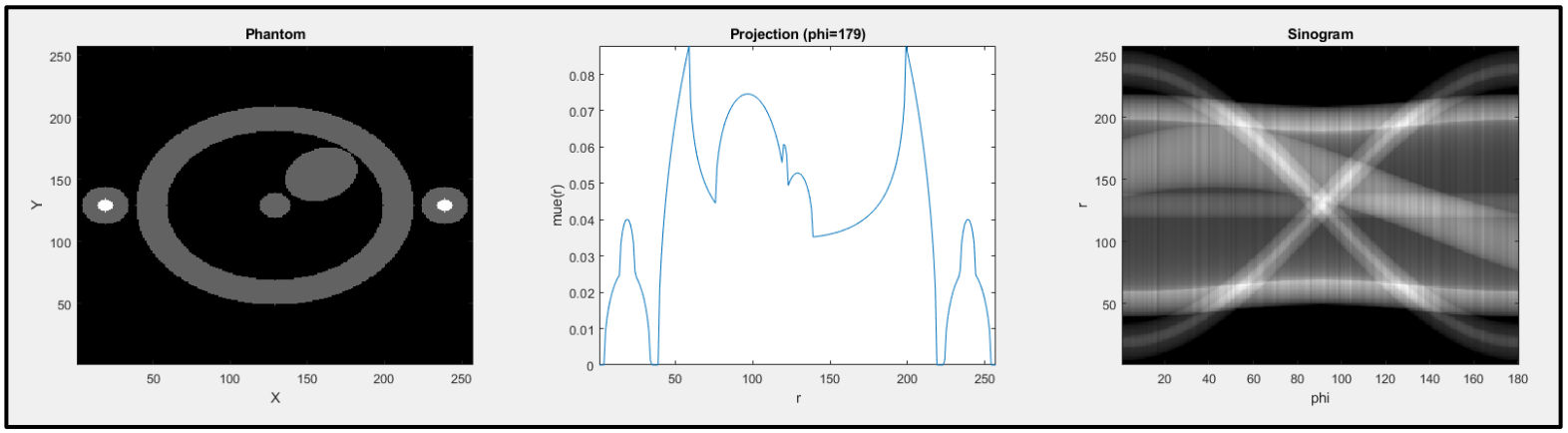
c) The *AddNoise()* function is manipulated as following in order to introduce quantum noise according to a Poisson distribution:

```
%=========================================================================
function [projection] = AddNoise(projection)

    % ------------------------------------------------------------------
    % Create Poisson noise
    % ------------------------------------------------------------------
    % TASK 3.1 FILL IN HERE
    projection = poissrnd(projection);

end
```

The main code is then changed to add the quantum noise in the projections as following:

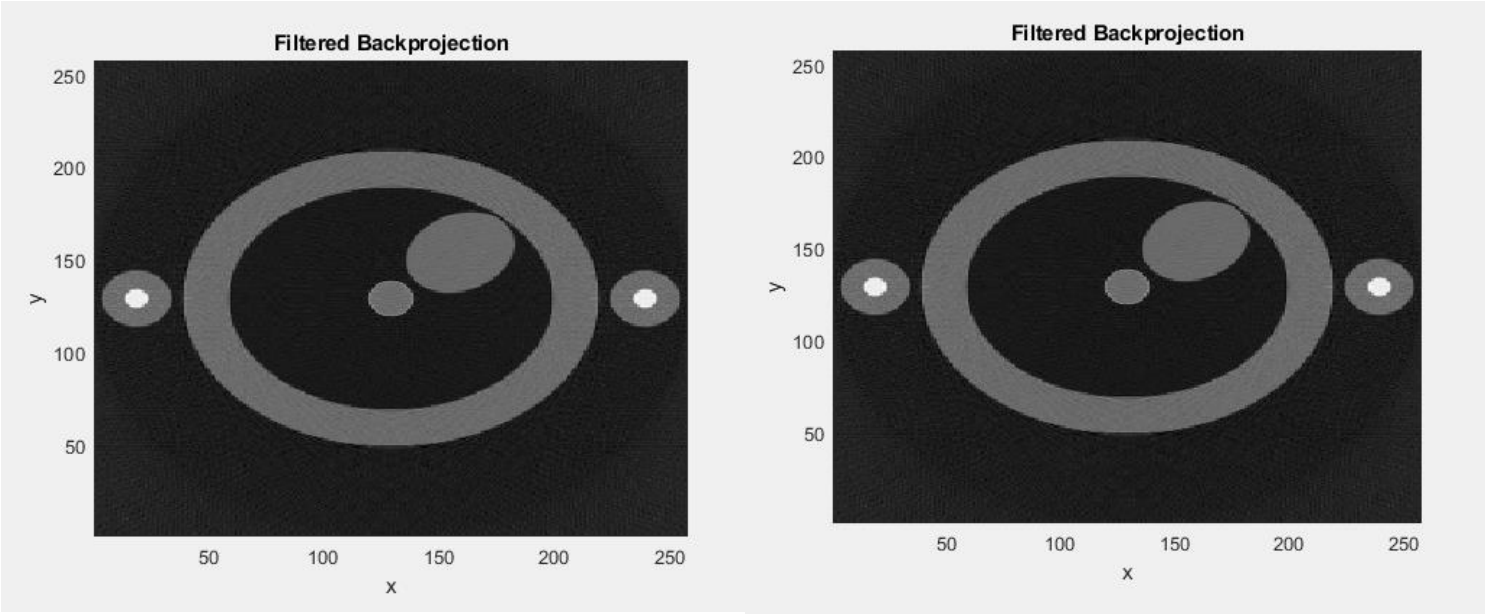
```
% ------------------------------------------------------------------
% Add noise
% ------------------------------------------------------------------
% TASK 3.1 UNCOMMENT
N_noisy(l,idx) = AddNoise(N(l,idx));

% ------------------------------------------------------------------
% Calculate projection including noise
% ------------------------------------------------------------------
% TASK 3.1 UNCOMMENT AND FILL IN HERE
phantom.sino(:,idx) = phantom.sino(:,idx).*(N(l,idx)/N_noisy(l,idx));

% ------------------------------------------------------------------
% Display projection and sinogram
% ------------------------------------------------------------------
DisplayData(phantom.sino(:,idx),[2,3,2]);
title(sprintf('Projection (phi=%d)',phi)); xlabel('r'); ylabel('mue(r)');

DisplayData(phantom.sino,[2,3,3]);
title('Sinogram'); xlabel('phi'); ylabel('r'); drawnow;

idx = idx+1;
```

d) Then the simulation is run in order to observe the results of noise on the projections and the sinogram:



A comparison between the image with noise and without noise can be seen in the following figures:

Without noise **With noise**



The results do not show significant differences between the two images apart from a slight difference in contrast due to the noise in the second picture. However, the effect is obvious in the sinogram. This is caused because of the accumulation of noise for every point in the image as the projections increase and are taken under different angles. The solution is to perform 2D Gaussian filtering in order to remove the noise and reconstruct a better version of the image.

2. Task 3.2

a) The general form of a 1D Gaussian filter centered around 0 is: $G(x) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{\frac{-x^2}{2\sigma^2}}$. In this case, the FWHM is equal to: $FWHM = \sigma\sqrt{8 \cdot \ln(2)}$. For this part, σ is chosen so that the value at x=0 is 1. Therefore, $\frac{1}{\sigma \cdot \sqrt{2\pi}} = 1$.

b) In order to be able to manipulate the width of the filter a resolution reduction factor is added in the formula. As a result, the final version of the filter is represented as:

$$G(x) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{\frac{-x^2}{2\sigma^2 \cdot factor}}$$

Then, a Gaussian filter is designed across the horizontal direction and one across the vertical direction. These are the combined by multiplication to give the 2D Gaussian filter. The final filter in the spatial frequency domain has the following form:

$$G(x) = \frac{1}{\sigma^2 \cdot 2\pi} \cdot e^{\frac{-(p^2+q^2)}{2\sigma^2 \cdot factor}}$$

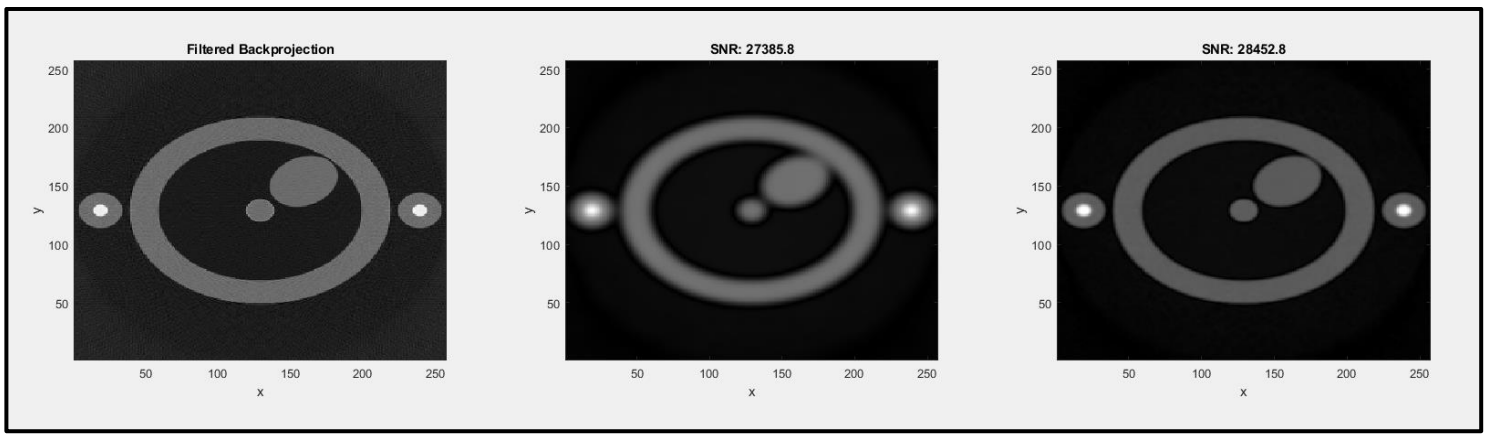
The MATLAB code written for this filter is given here:

```
%=========================================================================
function phantom = ApplyGaussianFilter(phantom,factor)

    matrix      = size(phantom.fbp,1);
    [p,q]       = meshgrid(-fix(matrix/2):+fix(matrix/2));
    phantom.img = phantom.fbp;

    % ---------------------------------------------------------------------
    % Define sigma based on full-width-at-half-maximum (FWHM) and the
    % resolution reduction factor
    % ---------------------------------------------------------------------
    % TASK 3.2 FILL IN HERE
    sigma = 1/sqrt(2*pi);  % Sigma calculated so that maximum value at center is 1
    sigma = 2*((sigma)^2); % Denominator Factor

    % ---------------------------------------------------------------------
    % Calculate Gaussian filter
    % ---------------------------------------------------------------------
    % TASK 3.2 FILL IN HERE
    filter_horizontal = zeros(matrix,matrix);
    filter_vertical = zeros(matrix,matrix);

    for i=1:matrix
        filter_horizontal(i,:) = exp((-(p(i,:).^2))/(sigma*factor));    % Horizontal Filter
        filter_vertical(:,i) = exp((-(q(:,i).^2))/(sigma*factor));      % Vertical Filter
    end

    filter = filter_horizontal.*filter_vertical;  % 2D Gaussian Filter

    % ---------------------------------------------------------------------
    % Filter data and store into phantom.img
    % ---------------------------------------------------------------------
    % TASK 3.2 UNCOMMENT HERE
    phantom.img = U2R(R2U(phantom.fbp).*filter);

end
```

c) The final 2D Gaussian filter is applied on the noisy image by executing the following code:

```
% ---------------------------------------------------------------------
% Reduce noise using Gaussian filter
% ---------------------------------------------------------------------
resolfact   = 1000;
phantom     = ApplyGaussianFilter(phantom,resolfact);
snr         = CalcSNR(phantom,8,Ia,dt);           % 8 = heart (index in list)

DisplayData(phantom.img,[2,3,5]); xlabel('x'); ylabel('y'); title(sprintf('SNR: %3.1f',snr));


% ---------------------------------------------------------------------
% Reduce resolution 4-fold using Gaussian filter
% ---------------------------------------------------------------------
resolfact   = 8000;
phantom     = ApplyGaussianFilter(phantom,resolfact);
snr         = CalcSNR(phantom,8,Ia,dt);           % 8 = heart (index in list)

DisplayData(phantom.img,[2,3,6]); xlabel('x'); ylabel('y'); title(sprintf('SNR: %3.1f',snr));
```

The resulting images are shown below:



It is obvious that the Gaussian filter significantly mitigates the effect of noise significantly. Furthermore, it is important to note that the middle image has a resolution factor of 1000 and the right-most image has a resolution factor of 8000. As a result, as the resolution reduction factor increases, the noise is even more suppressed. That is because, the resolution factor increases the width of the filter and allows more frequencies to be present in the image. That way, the surroundings are depicted better. Now, the resulting filter applied to the image is a band-pass filter. Therefore, the final image is seen to be less noisy and smoother at the edges.

3. Task 3.3

a) The SNR at the heart can be calculated by the following formula: $SNR = \frac{\bar{I}}{SD(I)}$, where $\bar{I}$ the mean value of the intensities at the points of the ROI and $SD(I) = \sigma = \sqrt{\frac{1}{Q \cdot \Delta z}}$, where $Q = Ia \cdot dt$, with Ia, the tube current and dt, the burst duration. Also Δz, is the thickness of the detector, which is taken to be 1mm here. The function implemented in MATLAB to calculate the SNR is the following:

```
%==========================================================================
function [snr] = CalcSNR(phantom,k,Ia,dt)

    snr = 0;
    % ---------------------------------------------------------------------
    % Find region-of-interest (ROI)
    % ---------------------------------------------------------------------
    [x,y]    = meshgrid(-fix(size(phantom.fbp,1)/2):+fix(size(phantom.fbp,1)/2));
    theta    = phantom.ellipse(k,5)*pi/180;

    X0       = [x(:)'-phantom.ellipse(k,1);y(:)'-phantom.ellipse(k,2)];
    D        = [1/(phantom.ellipse(k,3)-3) 0;0 1/(phantom.ellipse(k,4)-3)];
    Q        = [cos(theta) sin(theta); -sin(theta) cos(theta)];

    equ      = sum((D*Q*X0).^2);
    i        = find(equ<=1);          % indices of phantom.img inside ROI

    % ---------------------------------------------------------------------
    % Calculate signal-to-noise ratio (SNR)
    % ---------------------------------------------------------------------
    % TASK 3.3 FILL IN HERE
    mean_int = sum(phantom.img(i))/size(i,1);
    sd_int = sqrt(1/(Ia*dt));

    snr = mean_int/sd_int;


end
```

b) Then, the results are calculated for different values of Ia with constant dt and for different values of dt with constant Ia.

The results are the following:

- **For dt = 0.01s:**

Ia = 100 mA: SNR (fact=1000) = 13770.3

Ia = 100 mA: SNR (fact=8000) = 14162.5

Ia = 200 mA: SNR (fact=1000) = 19576.6

Ia = 200 mA: SNR (fact=8000) = 20135.9

Ia = 300 mA: SNR (fact=1000) = 24014.7

Ia = 300 mA: SNR (fact=8000) = 24700.0

- **For $Ia = 100\ mA$:**

dt = 0.01 s: SNR (fact=1000) = 13770.3

dt = 0.01 s: SNR (fact=8000) = 14162.5

dt = 0.02 s: SNR (fact=1000) = 19462.0

dt = 0.02 s: SNR (fact=8000) = 20015.7

dt = 0.04 s: SNR (fact=1000) = 27292.4

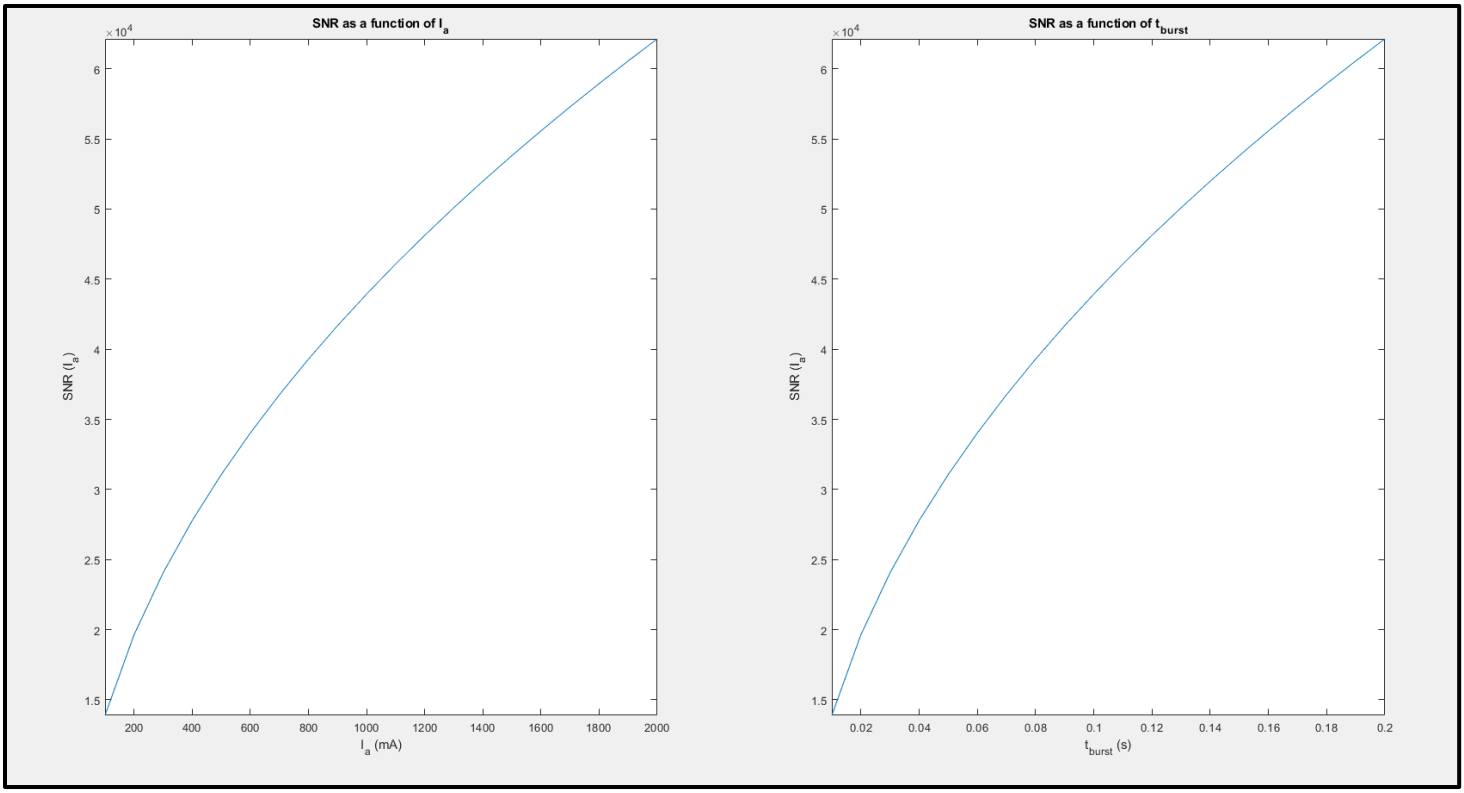
dt = 0.04 s: SNR (fact=8000) = 28356.4

It can be seen that the SNR is increased both for a higher tube current and for a longer burst duration.

c) In order to create the graphs of relation between the SNR and the tube current and the burst duration, the following code is used:

```
% TASK 3.3 FILL IN HERE
mean_int = sum(phantom.img(i))/size(i,1);
sd_int = zeros(1,20);
snr_graph = zeros(1,20);
i = 1;
for Ia = 100:100:2000

    sd_int(1,i) = sqrt(1/(Ia*dt));          % assuming detector thickness of 1 mm
    snr_graph(1,i) = mean_int/sd_int(1,i);
    i = i+1;
end
figure(2);
Ia = 100:100:2000;
subplot(1,2,1);plot(Ia,snr_graph);title('SNR as a function of I_a');xlabel('I_a (mA)');ylabel('SNR (I_a)');axis tight;

Ia = 100;
sd_int_2 = zeros(1,20);
snr_graph_2 = zeros(1,20);
i=1;
for dt = 0.01:0.01:0.2

    sd_int_2(1,i) = sqrt(1/(Ia*dt));
    snr_graph_2(1,i) = mean_int/sd_int_2(1,i);
    i = i+1;

end
dt = 0.01:0.01:0.2;
subplot(1,2,2);plot(dt,snr_graph_2);title('SNR as a function of t_b_u_r_s_t');xlabel('t_b_u_r_s_t (s)');ylabel('SNR (I_a)');axis tight;
%snr = mean_int/sd_int;

waitforbuttonpress;
```

d) The relation between the SNR and the tube current or the burst duration is illustrated in the following diagrams:



It is obvious that the SNR improves as we increase the tube current or the burst duration.

4. Task 3.4

a) The heart is now simulated to beat at a sinusoidal fashion. The formula used to implement this is:

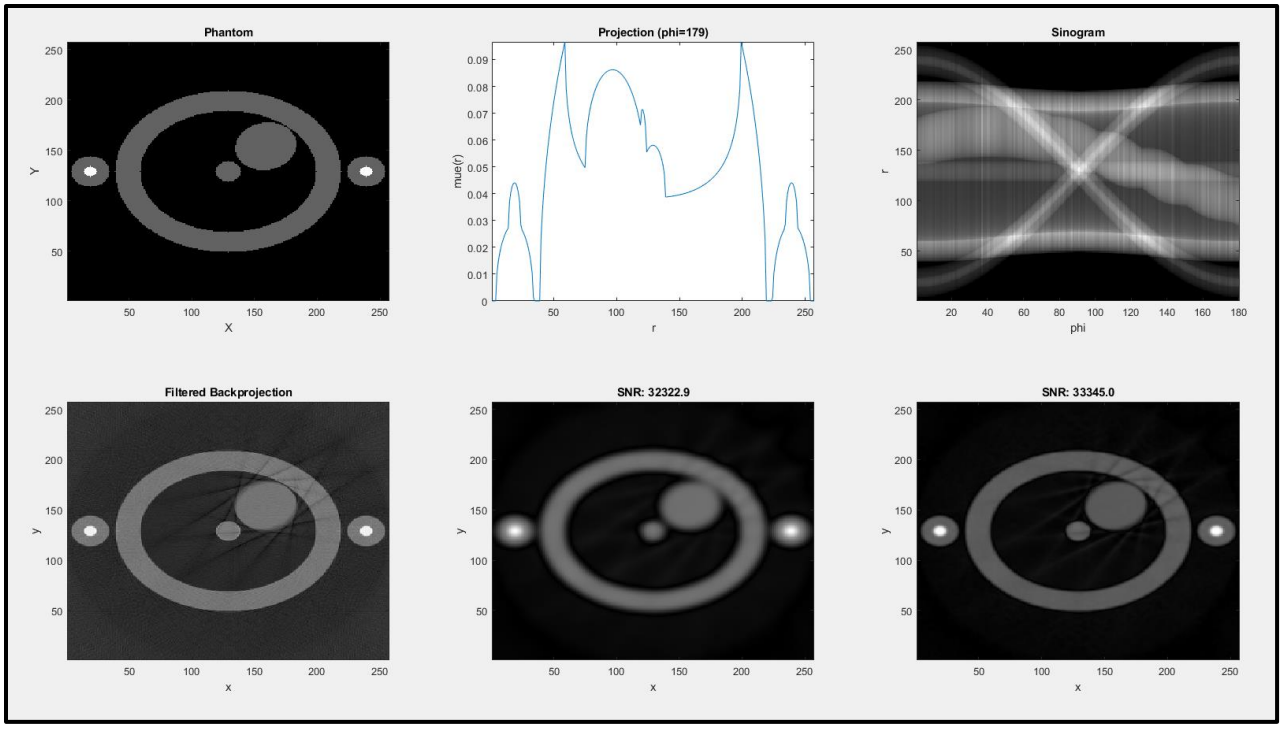
$$f(phi) = b \cdot (1 + |\sin(2\pi \cdot (0.5 \cdot dt \cdot phi))|)$$

With b, the vertical half-axis of the ellipse, phi, the angle of projection and dt, the scan time. For this formula it is true that each heartbeat corresponds to half of the period of the sinusoid.

This is implemented in the code as following:

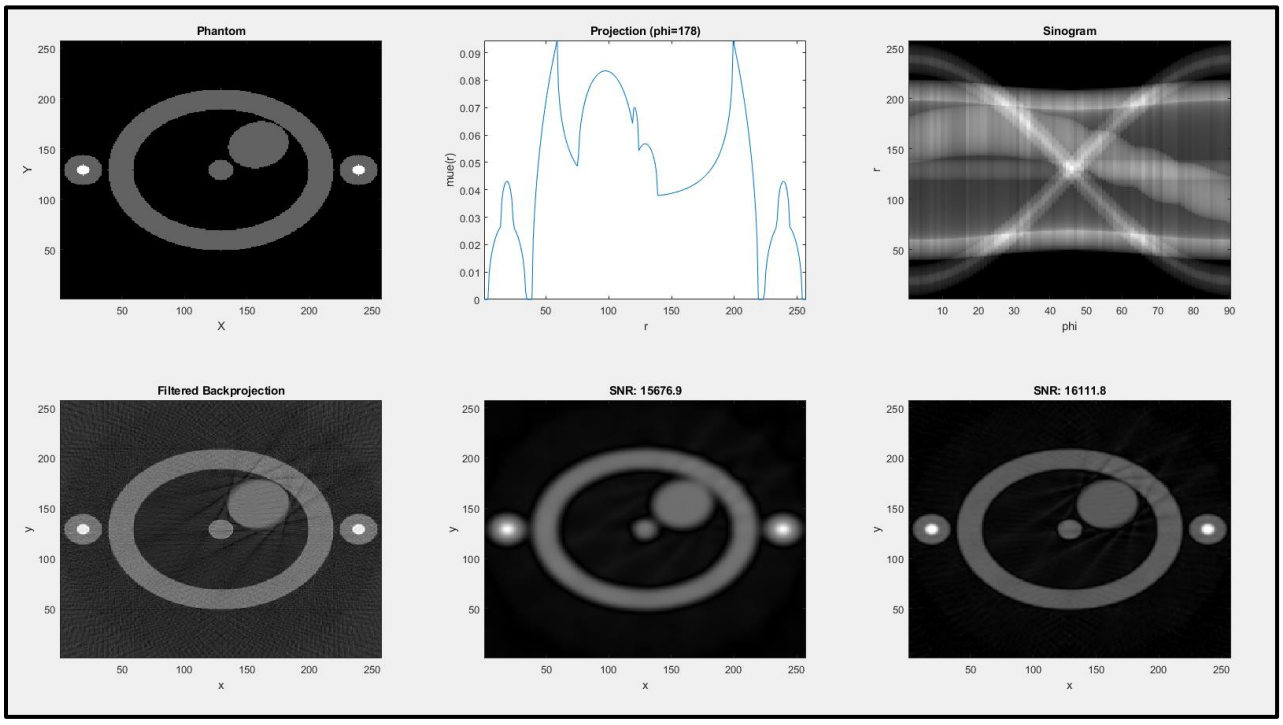
```
% ------------------------------------------------------------------
% Let the heart beat
% ------------------------------------------------------------------
% TASK 3.4 UNCOMMENT AND FILL IN HERE
phantom.ellipse(8,4) = heart_axis * (1 + 0.3*abs(sin(2*pi*0.5*dt*phi)));

phantom.discrete = CalcDiscretePhantom(x,y,phantom);
DisplayData(phantom.discrete,[2,3,1]); title('Phantom');
```

b) The resulting projections and sinogram show distortions in the part concerning the heart due to its motion and varying shape and size. The sinogram and the resulting image are shown in following figures:



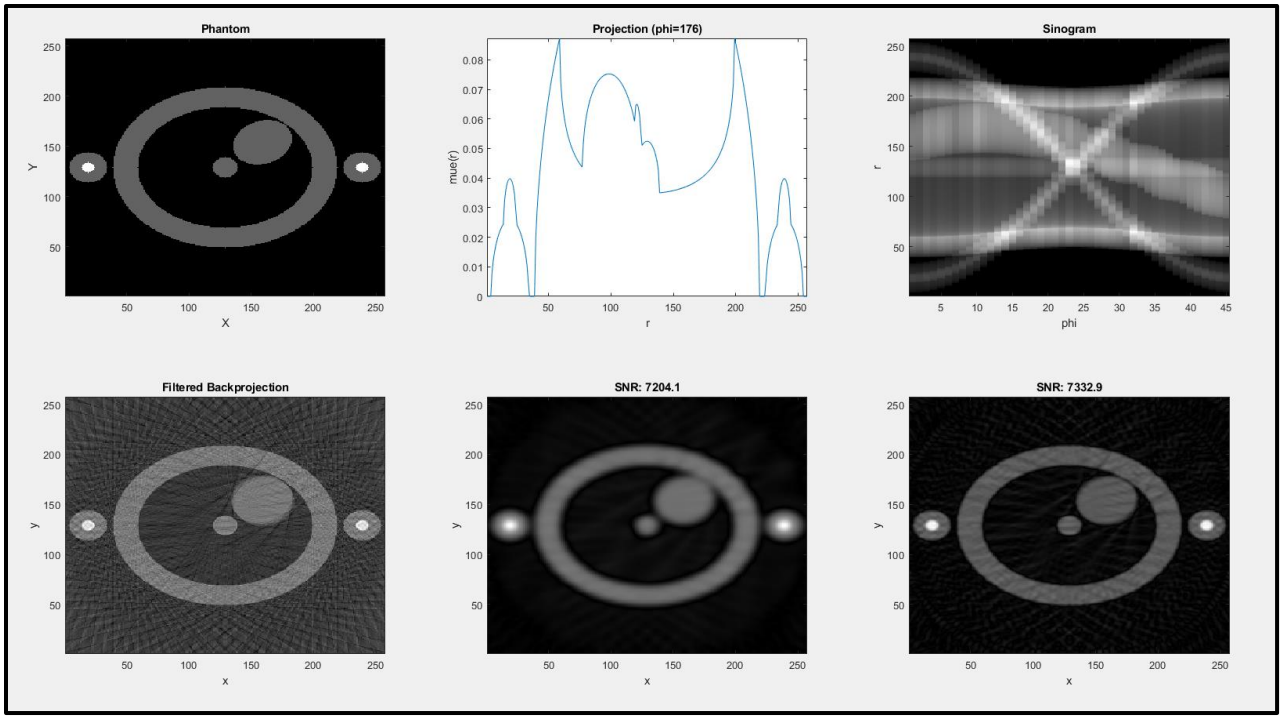
The sinogram demonstrates clearly that there is distortion in the line representing the heart due to its motion. Furthermore, on the image there are some dark lines around the heart, which are caused by photons which were received while the heart was in systole. As a result, there is significant distortion in the image and the sinogram due to the motion.

c) The undersampling factor is manipulated in order to try to improve the image quality. <u>For R=2:</u>



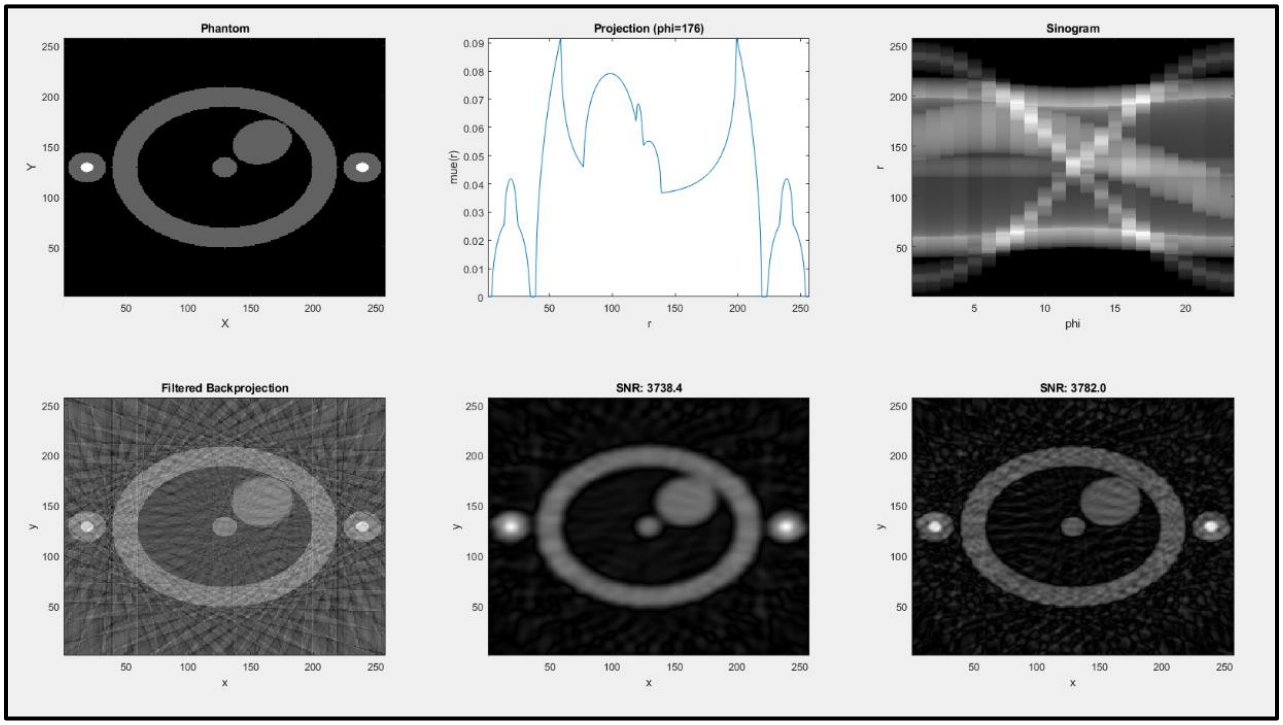
Here it can be seen that in the final image the intensity of the artefacts has decreased, but they are still visible. On the positive side, the imaging speed is faster than for R=1. Finally, the SNR has significantly dropped to half, therefore more noise has been introduced due to reduced number of projections.

For R=4:



In this case, the final image has significantly reduced the motion artefacts as the view of the heart is clearer. Also, the imaging speed is even faster. However, there is further noise introduced in the rest of the image, which significantly reduces the SNR even further.

For R=8:



The final image does not show any motion artefacts however it is obvious that significant amount of noise for the whole image has been introduced. Therefore, even though the imaging speed is very high, the extremely low SNR due to the high overall noise, does not constitute this undersampling factor a sensible option to reduce the motion artefacts.

d) As seen in the above results, there is a trade-off between the imaging speed, the amount of motion artefacts and the SNR of the final image. In that case, we need to consider what can be mitigated without the need of compromising the other factors significantly. That factor is obviously the motion. In a clinical setting, the motion of the patient's parts can be significantly reduced if we ask the patient to stand completely still during the time of the scan. In the case of the heart, the goal is to reduce the motion of the heart by probably suggesting relaxing breathing exercises to the patient right before the scan, in order to achieve low heart rate and therefore lower heart motion. Other than that, the SNR is more important than the imaging speed in order to get a satisfactory result, therefore it should be considered a priority. After testing the system with different imaging speeds/number of projections, which should always satisfy the Nyquist theorem by taking at least $\frac{\pi}{2} \cdot N$ projections for an $N \times N$ image, the scan time and the number of projections can be set to the minimum ones which achieve the desired SNR.