```
larger_pixel_in_Fov=100;
compressing_factor=2;

smaller_pixel_in_FoV=larger_pixel_in_Fov*compressing_factor;
total_smaller_pixel=smaller_pixel_in_FoV+compressing_factor-1;
```

```
% Getting input
image_series=create_shifted_images(larger_pixel_in_Fov,compressing_factor); % The
first parameter could be an image, or an integer to indicate the number of larger
pixels that we want in a row.
GT=image_series{1,1};
compressed_series=compress(image_series,compressing_factor);
```

```
% Setting up assumptions of redundants
% Assumptions are filled with numbers, while the unknowns remain as NaN.

all_pixels=zeros(total_smaller_pixel,total_smaller_pixel);
all_pixels(:)=NaN;

% Setting the right small pixels to the right values of compressed_series{1,end}
for i=1:larger_pixel_in_Fov
    all_pixels(1+
(i-1)*compressing_factor:i*compressing_factor,smaller_pixel_in_FoV+1:end)=compressed
_series{1,end}(i,end)/compressing_factor^2;
end

% Setting the lower right pixels
all_pixels(smaller_pixel_in_FoV+1:end,smaller_pixel_in_FoV+1:end)=compressed_series{
end,end}(end,end)/compressing_factor^2;

% Setting the bottom pixels
for i=1:larger_pixel_in_Fov
    all_pixels(smaller_pixel_in_FoV+1:end,1+
(i-1)*compressing_factor:i*compressing_factor)=compressed_series{end,1}(end,i)/
compressing_factor^2;
end
```

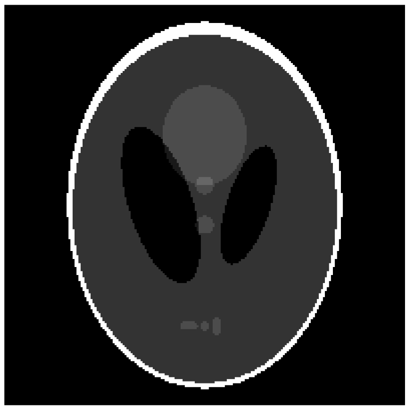
```
% generate A

% A is the actual image information that we got. It has a size of
(compressing_factor^2*larger_pixel_in_Fov^2)*(total_smaller_pixel^2).
% Each row contains the generating matrix of a larger pixel from some image. The
first larger_pixel_in_Fov^2 rows contains every pixel from row
% to row of the first input image (GT). The second larger_pixel_in_Fov^2 rows
contains the first image that is acquired after 1-unit horizontal
% shift.
```

```
A=generate_slider(larger_pixel_in_Fov,compressing_factor);
```

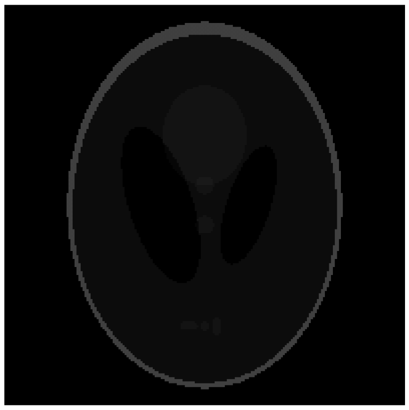
```
counter = 1
counter = 2
counter = 3
counter = 4
```

```
% Solve A*v=I.
% v is stored as an total_smaller_pixel^2*1 matrix; rows of total smaller pixels
are concatenated.
v=reshape(all_pixels',1,[])';
% I is stored an (compressing_factor^2*larger_pixel_in_Fov^2)*1 matrix. The first
larger_pixel_in_Fov^2 elements are rows of pixel values
% of input image 1 (GT); the second larger_pixel_in_Fov^2 elements are information
from the image that is acquired after 1 unit shift
% horizontally from GT.
I=get_I(compressed_series);
v_final=matrix_solver(A,v,I);
```

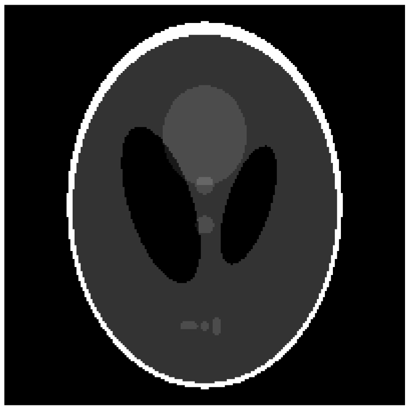
```
image_recon=reshape(v_final,total_smaller_pixel,total_smaller_pixel);
image_recon=image_recon(1:smaller_pixel_in_FoV,1:smaller_pixel_in_FoV);
figure;
imshow(GT);
```



```
figure;
imshow(image_recon');
```



```
image_recon_normalized=(image_recon-min(image_recon(:)))/(max(image_recon(:))-
min(image_recon(:)));
figure;
imshow(image_recon_normalized');
```



```
disp(sum(abs(GT-image_recon_normalized),"all")/smaller_pixel_in_FoV^2);
```

```
0.1257
```