# Final Examination

## Shuowen Wei

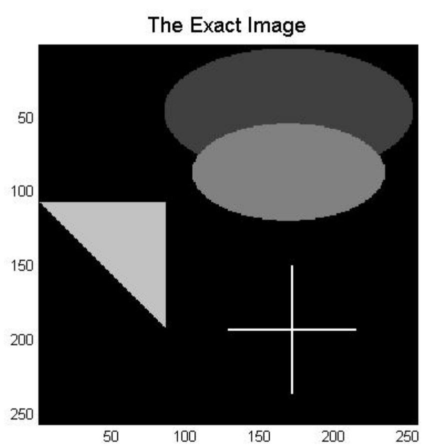
## 4/29/2012

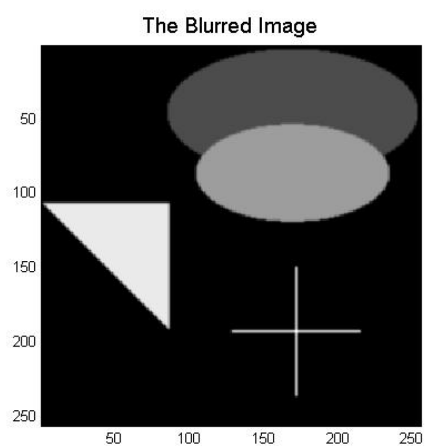
1.

We set N=256 and use blur.m to generate the matrix A, the blurred image b and the exact image x. Some codes are:

```
clc;clear;close all
format compact
N=256;
[A b x]=blur(N);
figure
E=reshape(x,N,N);%the exact image E
imagesc(E),title('The Exact Image')
axis image,colormap gray
B=reshape(b,N,N);%the blured image B
figure
imagesc(B),title('The Blurred Image')
axis image,colormap gray
```

then the exact image and the blurred image are as follows:

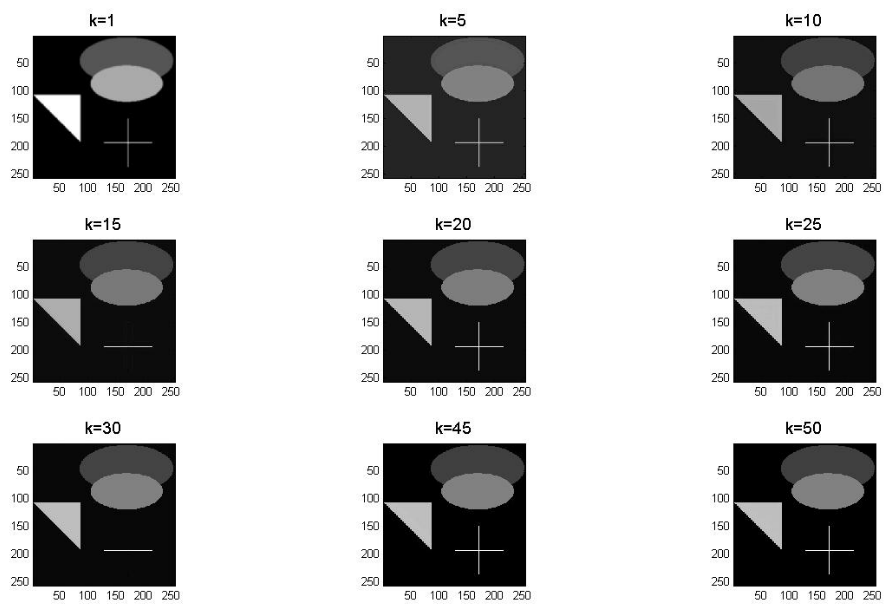


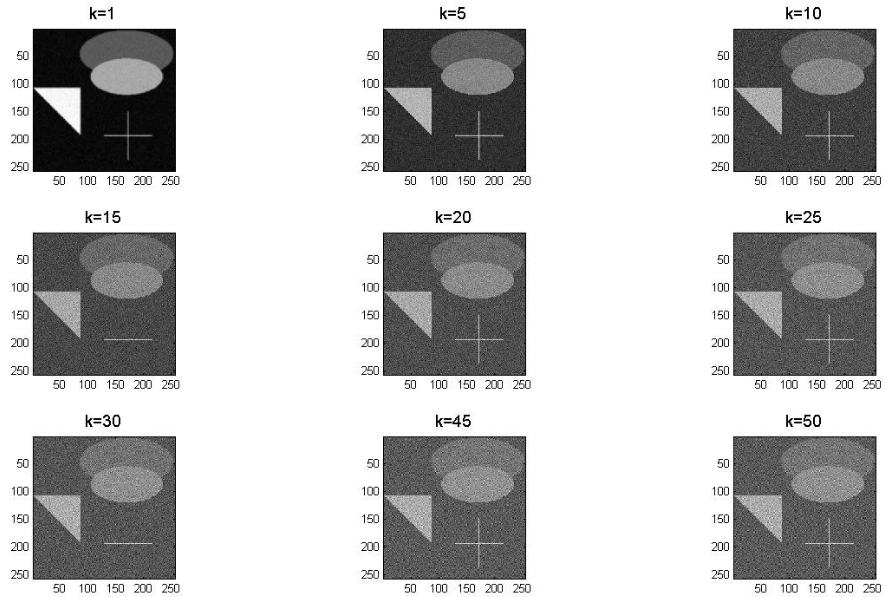
In order to illustrate the use regularizing properties of the CGLS algorithm, we then take iterations k=50 of CGLS, and we observed that the regularized image gets sharper and sharper as the number of iteration increases. We make an animation and codes are listed on the right:

```
% make an animation
k=50;
[X]=cgls(A,b,k);
X_image=reshape(X,N,N,k);
figure
for i=1:k
    imagesc(X_image(:,:,i));
    axis image
    colormap gray
    pause(0.2)
end
```

Thus, we pick up some regularized images after different numbers of iterations and show them as follows:




Now, we start to add some noise to the blurred image with noise level $\|e\|_2/\|b\|_2 \approx 0.1$, and then repeat the CGLS computations. Unfortunately, after a certain number of steps, we can see that the noise starts to dominate. We also pick up some regularized images after different numbers of iterations and show them as follows: [1]




2.

(a).

There mainly two kinds of noises, additive and multiplicative. Additive noise often has a probability density function of Normal distribution, like Gaussian White noise, while multiplicative noise often has a probability density function of Poisson distribution, like Poisson noise and Speckle noise.

After using ***doc imnoise.m***, we cited the types of noise from ‘MATLAB help’ that can be present in observed images as follow:

| Value | Description |
|---|---|
| `'gaussian'` | Gaussian white noise with constant mean and variance |
| `'localvar'` | Zero-mean Gaussian white noise with an intensity-dependent variance |
| `'poisson'` | Poisson noise |
| `'salt & pepper'` | On and off pixels |
| `'speckle'` | Multiplicative noise |

(b).

The SNR for images defined in terms of Decibels is by:

$$\mathrm{SNR} = 10\log_{10}\frac{||\mathrm{b}||}{||\mathrm{n}||}$$

where b is the signal and n is the noise.

(c).

The additive Gaussian noise is caused primarily by Johnson-Nyquist noise, also called the thermal noise, including that which comes from the reset noise of capacitors. [2]

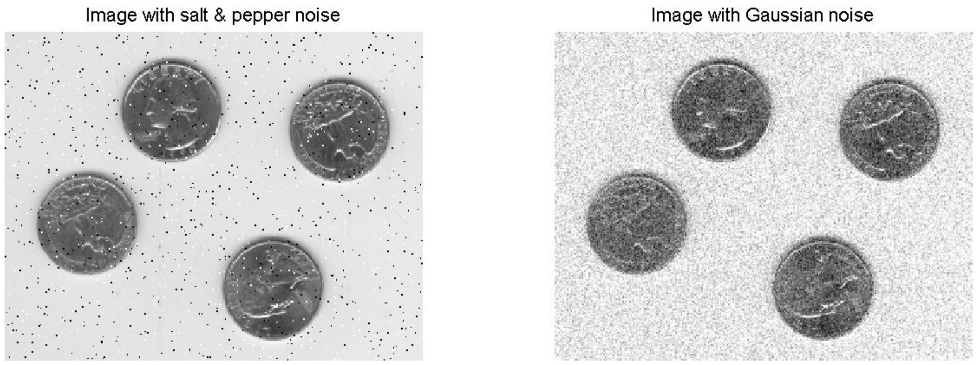
Poisson noise, or shot noise is a type of electronic noise which typically caused by the discrete nature of electric charge. The term also applies to photon counting in optical devices, where shot noise is associated with the particle nature of light. [3]

Salt and pepper noise, or spike noise can be caused by analog-to-digital converter errors, bit errors in transmission, etc. An image containing salt-and-pepper noise will have dark pixels in bright regions and bright pixels in dark regions. [4]

(d).

We load the image 'eight.tif' and save it in unit8 form. Then add Salt and pepper noise and Gaussian noise to it, the codes and the pictures are as follows:

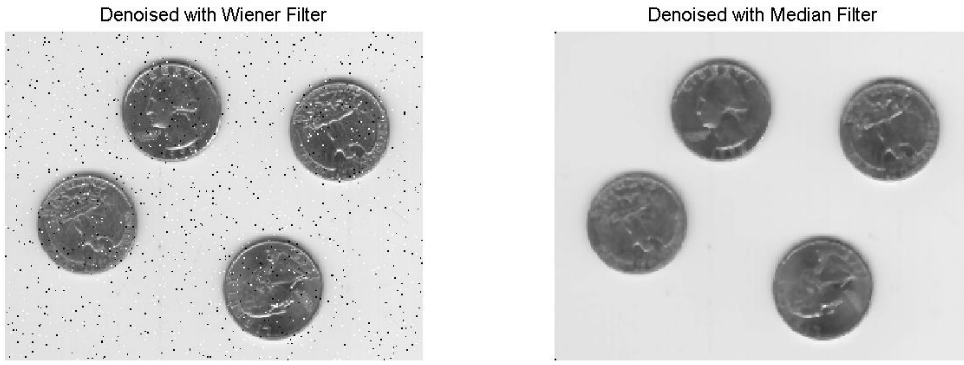
```
clc;clear;close all
x=imread('eight.tif');
imshow(x),title('\fontsize{14}the
original image')
figure
y1=imnoise(x,'salt & pepper',0.02);
subplot(1,2,1),imshow(y1)
title('\fontsize{16}Image with salt
& pepper noise')
y2=imnoise(x,'gaussian',0.02);
subplot(1,2,2),imshow(y2)
title('\fontsize{16}Image with
Gaussian noise')
```




Now, we are going to use command medfilt2 and wiener2 to try to remove the noise from the image above:

For removing the Salt and pepper noise, codes are:

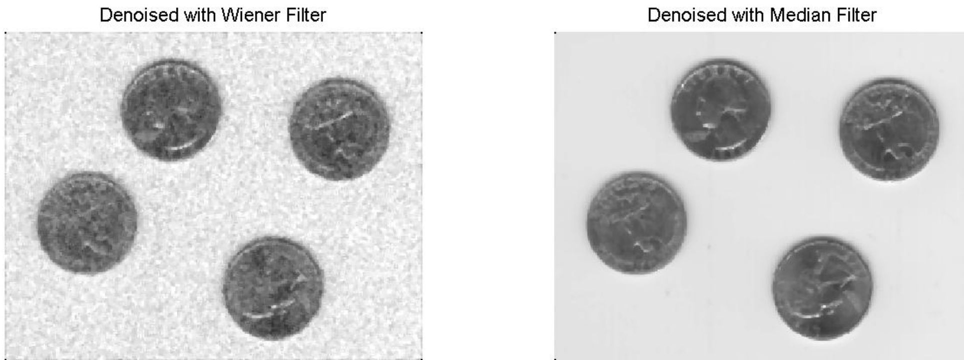
```
% denoise with two filters on salt and peper noise
figure
z11=wiener2(y1,[4 4]);
z12=medfilt2(y1);
subplot(1,2,1),imshow(z11)
subplot(1,2,1),imshow(y1)
title('\fontsize{16}Denoised with Wiener Filter')
subplot(1,2,2),imshow(z12)
title('\fontsize{16}Denoised with Median Filter')
```




We could obviously tell from the two graphs above that the median filter is better than the wiener filter in removing Salt and pepper noise, but it also makes the edges of objects in the image less sharper compared to that of original image.

For removing the Gaussian noise, codes are:

```
% denoise with two filters on gaussian noise
figure
z21=wiener2(y2,[4 4]);
z22=medfilt2(y2);
subplot(1,2,1),imshow(z21)
subplot(1,2,1),imshow(z22)
title('\fontsize{16}Denoised with Wiener Filter')
subplot(1,2,2),imshow(z12)
title('\fontsize{16}Denoised with Median Filter')
```





We could also tell from the two graphs above that the median filter is still better than the wiener filter in removing Gaussian noise, but they still have the same problem when removing salt and pepper noise, which is that both of them make the edge of objects less sharper than that of the original image.

3.

First of all, we generate the matrix $A$, $b$ and the true $x$ by gravity, and we then generate the first column c of the circulant matrix $C$, then compute $bc = Ax$ using fft and ifft. After getting the $bc$, we just compare their relative error using 2-norm, which turns out to be very small, thus $bc$ is very close to the true $b$. All the Matlab codes are as follows:

```
clc;clear
format compact
n=30;
[A b x]=gravity(n,1,0,1,0.5);
v=A(1,2:n)';
v=flipud(v);
c = [A(1:n,1)
     0
     v];
xzero=zeros(n,1);
xx=[x;xzero];
```

```
bcfft=ifft(fft(c).*fft(xx));
bc=bcfft(1:n);
[b,bc,bc-b]
relative_error=norm(bc-b)/norm(b)
```

We list our output bc and the true b as follows:

| $b$ | $bc$ | $bc - b$ |
|---|---|---|
| 1.44932819384894 | 1.44932819384894 | 2.22044604925031e-16 |
| 1.55774462279557 | 1.55774462279557 | 0 |
| 1.66451696377727 | 1.66451696377727 | -2.22044604925031e-16 |
| 1.76731190871261 | 1.76731190871261 | 0 |
| 1.86370033440002 | 1.86370033440002 | -2.22044604925031e-16 |
| 1.95127510049700 | 1.95127510049700 | 0 |
| 2.02777161050215 | 2.02777161050214 | -4.44089209850063e-16 |
| 2.09118074612541 | 2.09118074612541 | 0 |
| 2.13984549235943 | 2.13984549235943 | 8.88178419700125e-16 |
| 2.17253507729210 | 2.17253507729210 | -4.44089209850063e-16 |
| 2.18849316169201 | 2.18849316169201 | 4.44089209850063e-16 |
| 2.18745909333343 | 2.18745909333343 | 4.44089209850063e-16 |
| 2.16966325997973 | 2.16966325997973 | 4.44089209850063e-16 |
| 2.13579907298031 | 2.13579907298031 | 0 |
| 2.08697513358409 | 2.08697513358409 | -4.44089209850063e-16 |
| 2.02465176055705 | 2.02465176055705 | -4.44089209850063e-16 |
| 1.95056637551386 | 1.95056637551386 | 6.66133814775094e-16 |
| 1.86665231863715 | 1.86665231863715 | 2.22044604925031e-16 |
| 1.77495554785527 | 1.77495554785527 | 0 |
| 1.67755338770676 | 1.67755338770676 | -2.22044604925031e-16 |
| 1.57647905853768 | 1.57647905853768 | 0 |
| 1.47365514743067 | 1.47365514743067 | -2.22044604925031e-16 |
| 1.37083849637405 | 1.37083849637405 | 0 |
| 1.26957820356499 | 1.26957820356499 | 2.22044604925031e-16 |
| 1.17118759190725 | 1.17118759190725 | 0 |
| 1.07673013628725 | 1.07673013628725 | 2.22044604925031e-16 |
| 0.987018509663305 | 0.987018509663305 | 0 |
| 0.902625166585978 | 0.902625166585978 | 0 |
| 0.823902292813447 | 0.823902292813447 | 0 |
| 0.751008565720968 | 0.751008565720967 | -3.33066907387547e-16 |

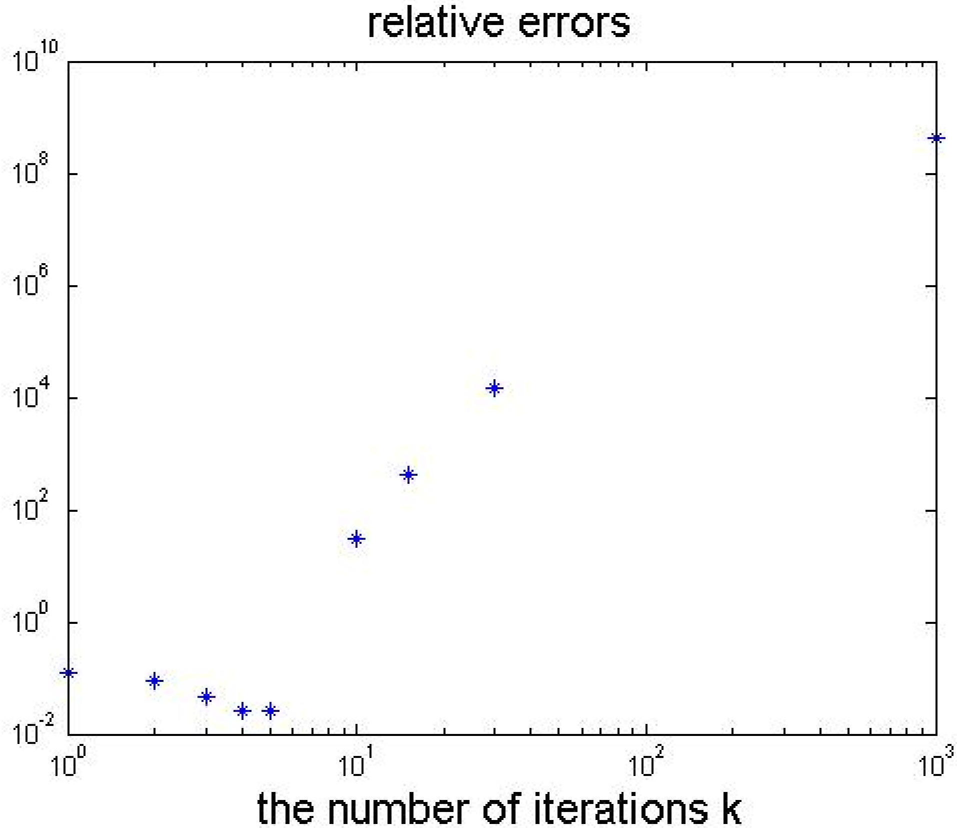
And their relative error is

$$relative\ error = \frac{\|bc - b\|}{\|b\|} = 1.8640 \times 10^{-16}$$

4.

(a).

We use gravity.m to generate the matrix $A$, true signal $b$ and the true solution $x$. Replace $b$ with $bn = b + noise$, tried different number of iterations $k$ and compute the corresponding relative error of $xc$, we use loglog to plot the errors out and we can clearly see the graph below:




A very small part of the codes are:

```
min_error=min(relativerror)
k_accurate=k(find(relativerror==min(relativerror)))
max_error=max(relativerror)
k_notaccurate=k(find(relativerror==max(relativerror)))
```

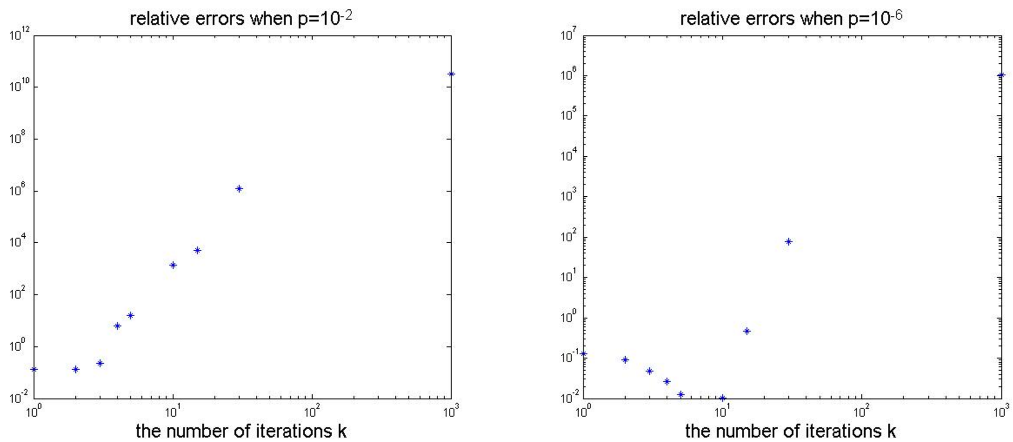
Output:

```
min_error =
    0.0248
k_accurate =
     5
max_error =
```

```
  4.3311e+008
k_notaccurate =
        1000
```

The outputs tell us that $k = 5$ gives us the most accurate solution, with the minimal relative error $0.0248$, while the largest value $k = 1000$ gives us the lease accurate solution, with the maximal relative error $4.3311 \times 10^8$, which is large.

Now we change the noise level $p$ a little bit to $p = 10^{-2}\ and\ p = 10^{-6}$, and run all of the above again, the results are showed below:




| | $p = 10^{-2}$ | $p = 10^{-4}$ | $p = 10^{-6}$ |
|---|---|---|---|
| most accurate $k$ | 2 | 5 | 10 |
| min relative error | 0.1316 | 0.0248 | 0.0104 |
| least accurate $k$ | 1000 | 1000 | 1000 |
| max relative error | $3.2725 \times 10^{10}$ | $4.3311 \times 10^8$ | $1.0737 \times 10^6$ |

The reason of these above is that, no matter what the level of the noise is, after a certain time the noise will start to dominate sooner or later. If the noise level is very small, like $p = 10^{-6}$, then it will not affect the iterations that much, so this process will have enough time to iterate more times to get more closer to the true solution, but when the noise level is large, like $p = 10^{-2}$, then the noise will quickly start to dominate before the iteration process get close enough to the true solution.

(b).

My modified **conjgrad.m** are as follows, we replace all the matrix times vector operations by fft and ifft, showed between “%%” below:

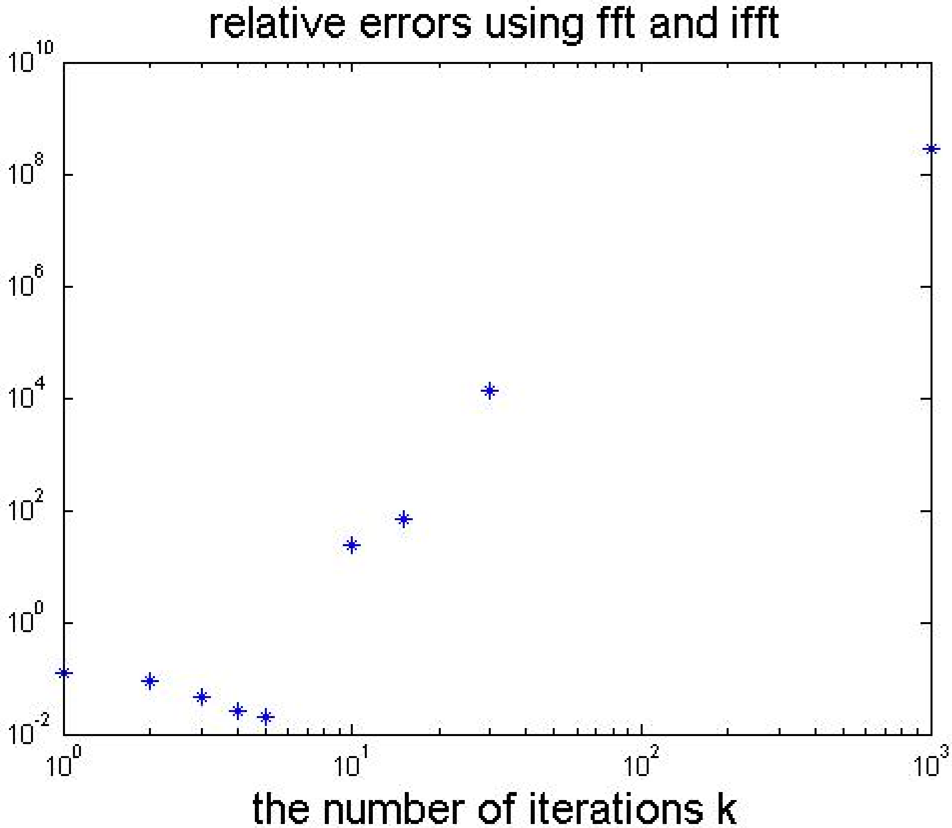
```
function [xc] = conjgradfft(A,b,x0,k)
% A is the symmetric positive definite (Toeplitz) matrix
% b is the right-hand-side
% x0 is the initial approximation to x
% xc is the computed approximate solution
   n=length(A);
   x = x0;
   %%
   v=A(1,2:30)';
   v=flipud(v);
   c = [A(1:30,1)
        0
        v];
   xx=[x;zeros(n,1)];
   bcfft=ifft(fft(c).*fft(xx));
   bc=bcfft(1:n);
   r = b - bc;%   r = b - A*x;
   %%
   w = -r;
   %%
   ww=[w;zeros(n,1)];
   bwfft=ifft(fft(c).*fft(ww));
   z=bwfft(1:n);%   z = A*w;
   %%
   a = (r'*w)/(w'*z);
   x = x + a*w;
   t = 0;

   for i = 1:k
      r = r - a*z;
  %   if( norm(r) < 1e-6 )
   %       break;
   %   end
      t = (r'*z)/(w'*z);
      w = -r + t*w;
      %%
      ww2=[w;zeros(n,1)];
      bw2fft=ifft(fft(c).*fft(ww2));
      z=bw2fft(1:n);%   z = A*w;
      %%
      a = (r'*w)/(w'*z);
      x = x + a*w;
   end
 xc = x;
end
```

Run what we do in 4(a) again and  get almost the same outputs:

Output:

```
min_error =
    0.0203
k_accurate =
     5
max_error =
  2.7814e+008
k_accurate =
        1000
```




# References:


[1] Hansan's Notes, Page 89-90

[2] Wikipedia, Image Noise http://en.wikipedia.org/wiki/Image_noise

[3] Wikipedia, Shot Noise http://en.wikipedia.org/wiki/Shot_noise

[4] Wikipedia, Image Noise http://en.wikipedia.org/wiki/Image_noise

# All Matlab Codes:

```
% #1 exercise 6.5.4
clc;clear;close all
format compact
N=256;
[A b x]=blur(N);
figure
E=reshape(x,N,N);%the exact image E
imagesc(E),title('\fontsize{14}The
Exact Image')
axis image,colormap gray
B=reshape(b,N,N);%the blurred image
B
figure
imagesc(B),title('\fontsize{14}The
Blurred Image')
axis image,colormap gray

% make an animation
k=50;
[X]=cgls(A,b,k);
X_image=reshape(X,N,N,k);
figure
for i=1:k
    imagesc(X_image(:,:,i));
    axis image
    colormap gray
    pause(0.2)
end

%pick 9 pictures of different k
kk=[1,5,10,15,20,25,30,45,50];
for i=1:length(kk)
    subplot(3,3,i)
    imagesc(X_image(:,:,kk(i)));

title(['\fontsize{14}k=',num2str(kk
(i))])
    axis image,colormap gray
end

%generate the noise
%ee=randn(size(b));
%ee=ee/norm(ee);
%bn=b+0.1*norm(b)*ee;
noise=randn(N^2,1);
e=0.1*noise*norm(b)/norm(noise);
bn=b+e;
%norm(e)/norm(b);

%pick 9 pictures added noise of
different k
[Xn]=cgls(A,bn,k);
Xn_image=reshape(Xn,N,N,k);
figure
for i=1:length(kk)
    subplot(3,3,i)
    imagesc(Xn_image(:,:,kk(i)));

title(['\fontsize{14}k=',num2str(kk
(i))])
    axis image,colormap gray
end


% #2
doc imnoise
doc wiener2

clc;clear;close all
x=imread('eight.tif');
imshow(x),title('\fontsize{14}the
original image')
figure
y1=imnoise(x,'salt & pepper',0.02);
subplot(1,2,1),imshow(y1)
title('\fontsize{16}Image with salt
& pepper noise')
y2=imnoise(x,'gaussian',0.02);
subplot(1,2,2),imshow(y2)
title('\fontsize{16}Image with
Gaussian noise')

% denoise with two filters on salt
and peper noise
figure
z11=wiener2(y1,[4 4]);
z12=medfilt2(y1);
subplot(1,2,1),imshow(z11)
subplot(1,2,1),imshow(y1)
title('\fontsize{16}Denoised with
Wiener Filter')
subplot(1,2,2),imshow(z12)
title('\fontsize{16}Denoised with
Median Filter')

% denoise with two filters on
gaussian noise
figure
z21=wiener2(y2,[4 4]);
z22=medfilt2(y2);
subplot(1,2,1),imshow(z21)
subplot(1,2,1),imshow(z22)
title('\fontsize{16}Denoised with
Wiener Filter')
subplot(1,2,2),imshow(z12)
```

```
title('\fontsize{16}Denoised with
Median Filter')



% #3
clc;clear
format compact
n=30;
[A b x]=gravity(n,1,0,1,0.5);
v=A(1,2:n)';
v=flipud(v);
c = [A(1:n,1)%generatethe the first
column of circulant matrix C
     0
     v];
xzero=zeros(n,1);
xx=[x;xzero];
bcfft=ifft(fft(c).*fft(xx));
bc=bcfft(1:n)
[b,bc,bc-b]
relative_error=norm(bc-b)/norm(b)



% #4 a
clc;clear;close all
format compact
[A b x]=gravity(30,1,0,1,0.5);
p=10^(-4);
bn=b+p*randn(30,1);
xo=bn;
k=[1 2 3 4 5 10 15 30 1000];
result=[];
relativerror=[];
for i=1:length(k)
    [xc]=conjgrad(A,bn,xo,k(i));
    result=[result,xc];
    error=norm(xc-x)/norm(x);

relativerror=[relativerror,error];
end
loglog(k,relativerror,'*')
title('\fontsize{16}relative
errors')
xlabel('\fontsize{16}the number of
iterations k')
min_error=min(relativerror)
k_accurate=k(find(relativerror==min
(relativerror)))
max_error=max(relativerror)
k_notaccurate=k(find(relativerror==
max(relativerror)))

% change p a little bit
% change p to 10^(-2)
p1=10^(-2);
bn1=b+p1*randn(30,1);
result=[];
relativerror=[];
for i=1:length(k)
    [xc]=conjgrad(A,bn1,xo,k(i));
    result=[result,xc];
    error=norm(xc-x)/norm(x);

relativerror=[relativerror,error];
end
figure;subplot(1,2,1)
loglog(k,relativerror,'*')
title('\fontsize{16}relative errors
when p=10^{-2}')
xlabel('\fontsize{16}the number of
iterations k')
min_error=min(relativerror)
k_accurate=k(find(relativerror==min
(relativerror)))
max_error=max(relativerror)
k_notaccurate=k(find(relativerror==
max(relativerror)))



% change p to 10^(-6)
p2=10^(-6);
bn2=b+p2*randn(30,1);
result=[];
relativerror=[];
for i=1:length(k)
    [xc]=conjgrad(A,bn2,xo,k(i));
    result=[result,xc];
    error=norm(xc-x)/norm(x);

relativerror=[relativerror,error];
end
subplot(1,2,2)
loglog(k,relativerror,'*')
title('\fontsize{16}relative errors
when p=10^{-6}')
xlabel('\fontsize{16}the number of
iterations k')
min_error=min(relativerror)
k_accurate=k(find(relativerror==min
(relativerror)))
max_error=max(relativerror)
k_notaccurate=k(find(relativerror==
max(relativerror)))



% #4 b
clc;clear;close all
format compact
[A b x]=gravity(30,1,0,1,0.5);
p=10^(-4);
```

```
bn=b+p*randn(30,1);
xo=bn;
k=[1 2 3 4 5 10 15 30 1000];
resultfft=[];
relativerrorfft=[];
for i=1:length(k)
    [xc]=conjgradfft(A,bn,xo,k(i));
    resultfft=[resultfft,xc];
    error=norm(xc-x)/norm(x);

relativerrorfft=[relativerrorfft,er
ror];
end
loglog(k,relativerrorfft,'*')
title('\fontsize{16}relative errors
using fft and ifft')
xlabel('\fontsize{16}the number of
iterations k')
min_error=min(relativerrorfft)
k_accurate=k(find(relativerrorfft==
min(relativerrorfft)))
max_error=max(relativerrorfft)
k_accurate=k(find(relativerrorfft==
max(relativerrorfft)))
```