Ben Postlethwaite 76676063

Table of Contents

Assignment 2

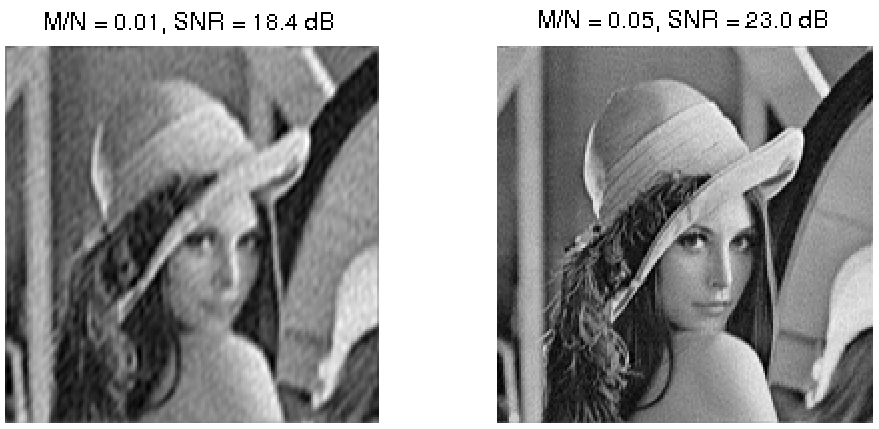
Setup

```
clear all; close all;
loadtools;
n = 512;
N = n*n;
f = rescale( load_image('lena', n) );
```

Exercise 1 - Image Approximation with Orthogonal Bases

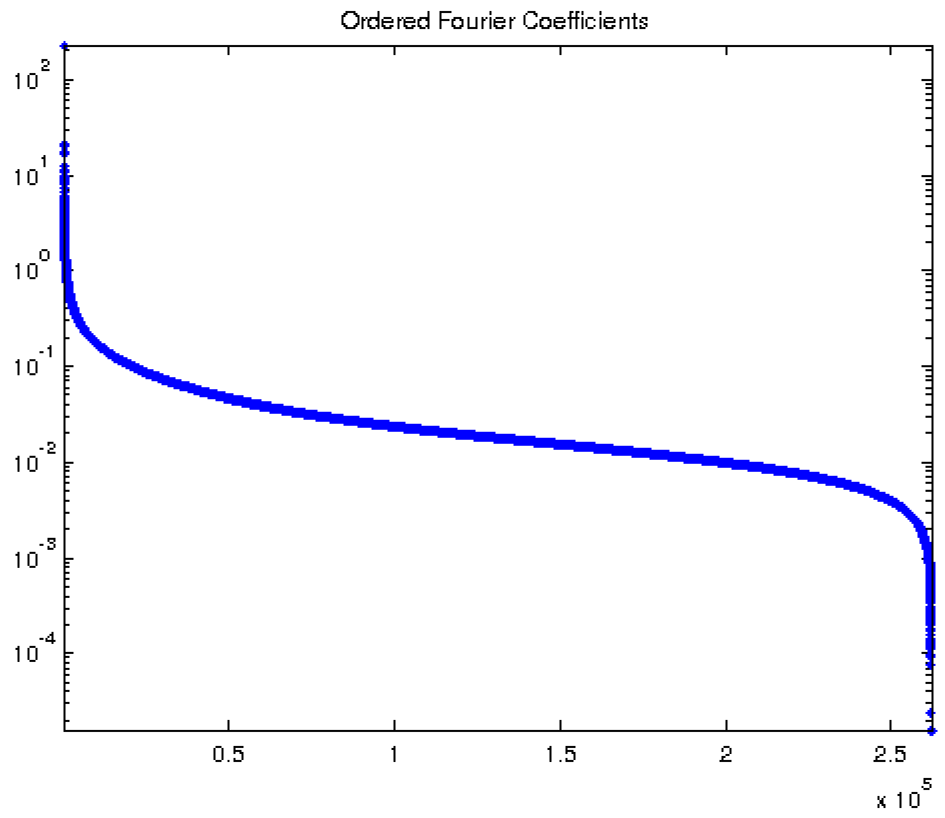
```
M = round(N*[0.01,0.05]);
fF = fft2(f)/n;
snark = sort(abs(fF(:)),1,'descend');
```

```
for ii = 1:2
    T = snark(M(ii));
    c = fF .* (abs(fF)>T);
    fM = real(ifft2(c)*n);
    figure(123)
    subplot(1,2,ii)
    imageplot(clamp(fM))
    title(sprintf('M/N = %0.2f, SNR = %2.1f dB',M(ii)/N,snr(f,fM)));
end
```



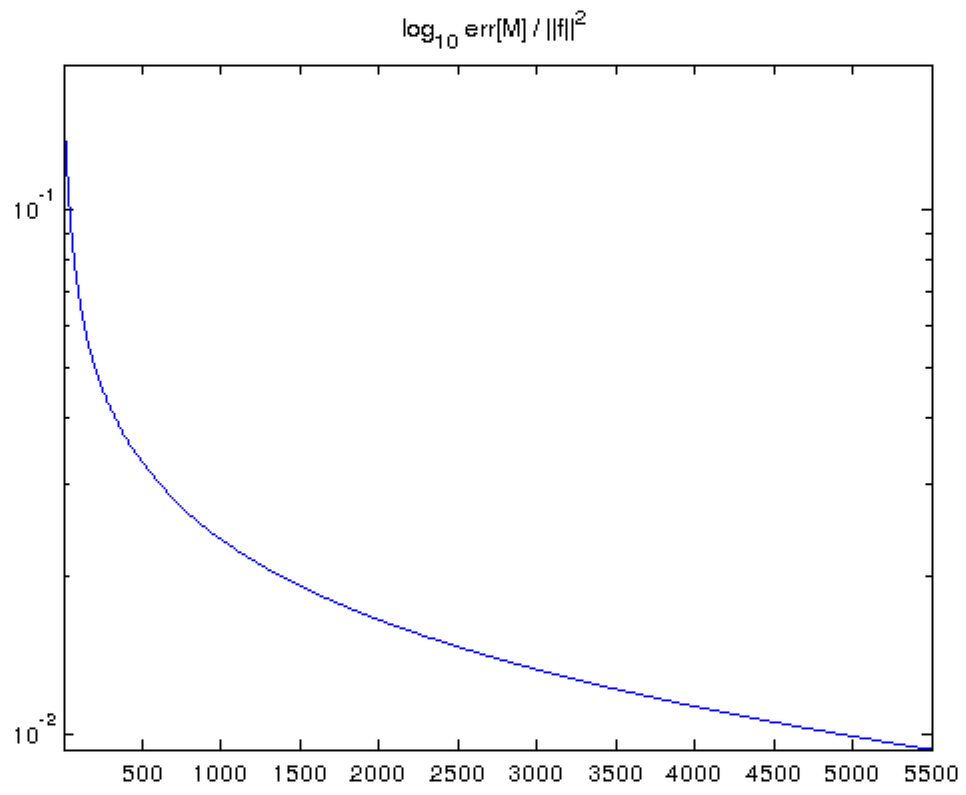
Exercise 2 - Image Approximation with Orthogonal Bases

```
figure(5423)
semilogy(snark,'.')
title('Ordered Fourier Coefficients')
axis tight
```



Exercise 3 - Image Approximation with Orthogonal Bases

```
err_fft = f(:)'*f(:) - cumsum(snark(1:5500).^2);
figure(1264);
semilogy(err_fft./(f(:)'*f(:)) )
title('log_1_0 err[M] / ||f||^2')
axis tight
```

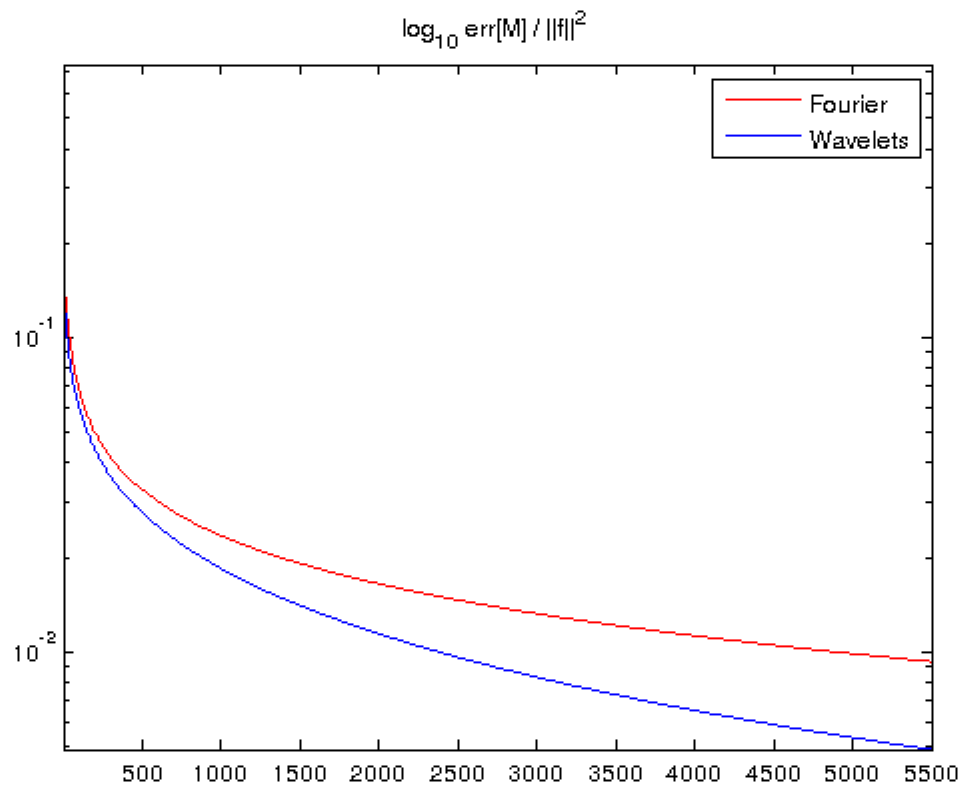


Exercise 4 - Image Approximation with Orthogonal Bases

```
Jmin = 1;
options.h = compute_wavelet_filter('Daubechies',10);
fW = perform_wavortho_transf(f,Jmin,+1, options);
snark = sort(abs(fW(:)),1,'descend');
for ii = 1:2
T = snark(M(ii));
c = fW .* (abs(fW)>T);
fM = perform_wavortho_transf(c,Jmin,-1,options);
figure(323)
subplot(1,2,ii)
    imageplot(clamp(fM))
    title(sprintf('M/N = %0.2f, SNR = %2.1f dB',M(ii)/N,snr(f,fM)));
end
```

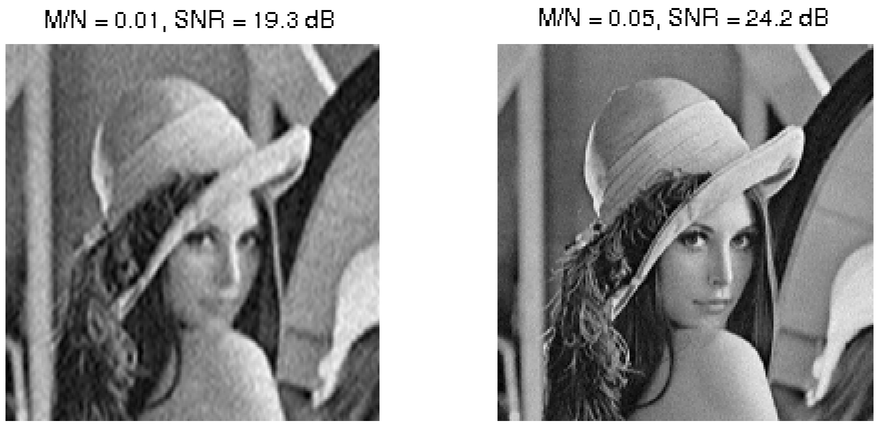
Exercise 5 - Image Approximation with Orthogonal Bases

```
err_wav = f(:)'*f(:) - cumsum(snark(1:5500).^2);
figure(1263);
semilogy([1:5500],err_fft./(f(:)'*f(:)),'r',[1:5500],err_wav./(f(:)'*f(:)),'b')
title('log_1_0 err[M] / ||f||^2')
axis tight
legend('Fourier','Wavelets')
```



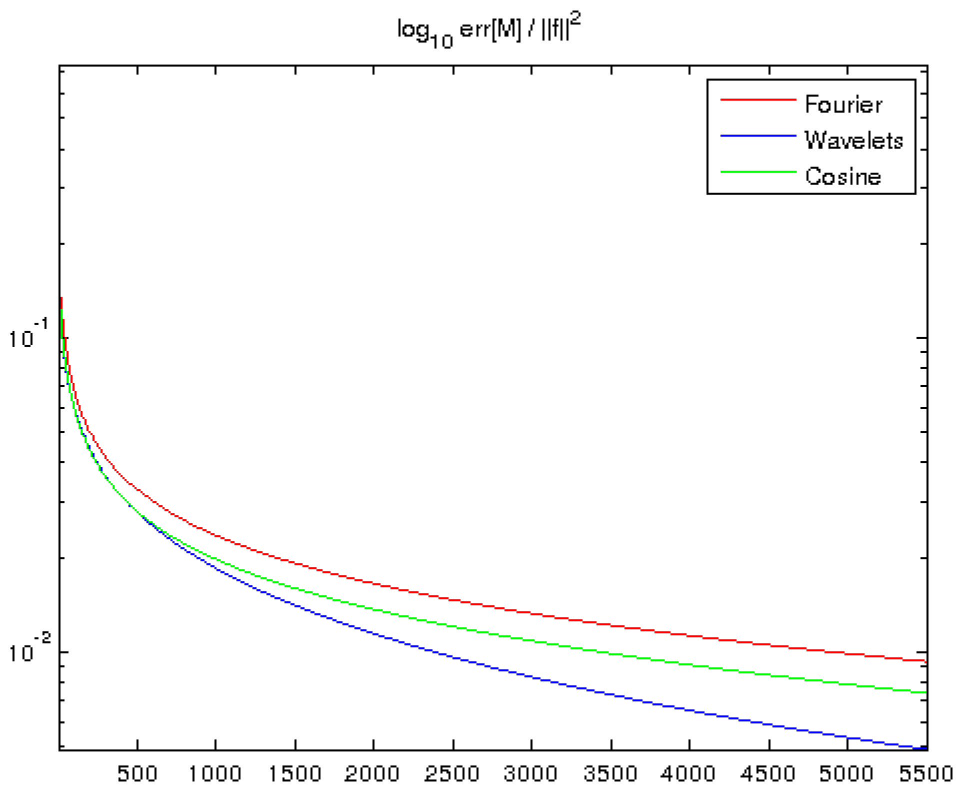
Exercise 6 - Image Approximation with Orthogonal Bases

```
fC = dct2(f);
snark = sort(abs(fC(:)),1,'descend');
for ii = 1:2
    T = snark(M(ii));
    c = fC .* (abs(fC)>T);
    fM = (idct2(c));
    figure(122)
    subplot(1,2,ii)
    imageplot(clamp(fM))
    title(sprintf('M/N = %0.2f, SNR = %2.1f dB',M(ii)/N,snr(f,fM)));
end
```



Exercise 7 - Image Approximation with Orthogonal Bases

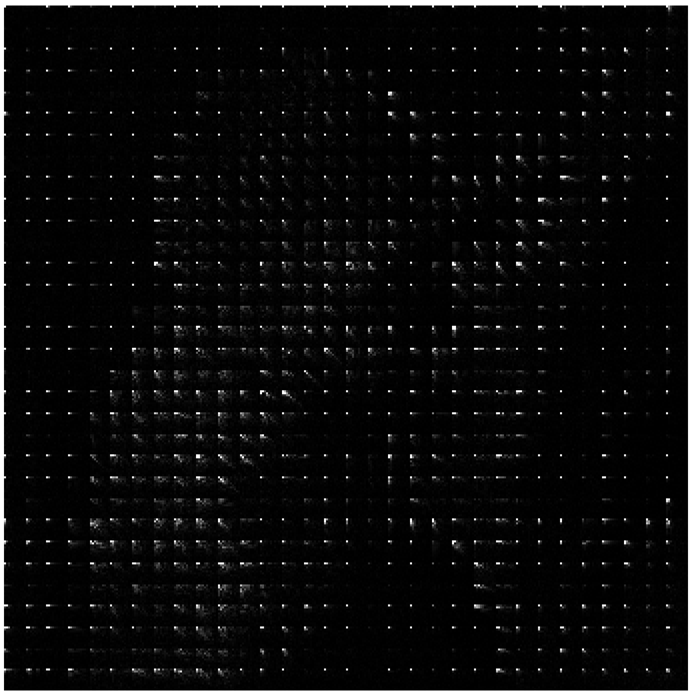
```
err_dct = f(:)'*f(:) - cumsum(snark(1:5500).^2);
figure(1263);
semilogy([1:5500],err_fft./(f(:)'*f(:)),'r',[1:5500],err_wav./(f(:)'*f(:)),...
    'b',[1:5500],err_dct./(f(:)'*f(:)),'g')
title('log_1_0 err[M] / ||f||^2')
axis tight
legend('Fourier','Wavelets','Cosine')
```



Exercise 8 - Image Approximation with Orthogonal Bases

```
w = 16;
fL = zeros(n,n);
for ii = 1:n/w
    for jj = 1:n/w
        seli = (ii-1)*w+1:ii*w;
        selj = (jj-1)*w+1:jj*w;
        P = f(seli,selj); % Select patch P
        fL(seli,selj) = dct2(P);
    end
end
figure(8534)
imageplot(min(abs(fL),.005*w*w));
title('Patch transformed Cosine Coefficients')
```

Patch transformed Cosine Coefficients



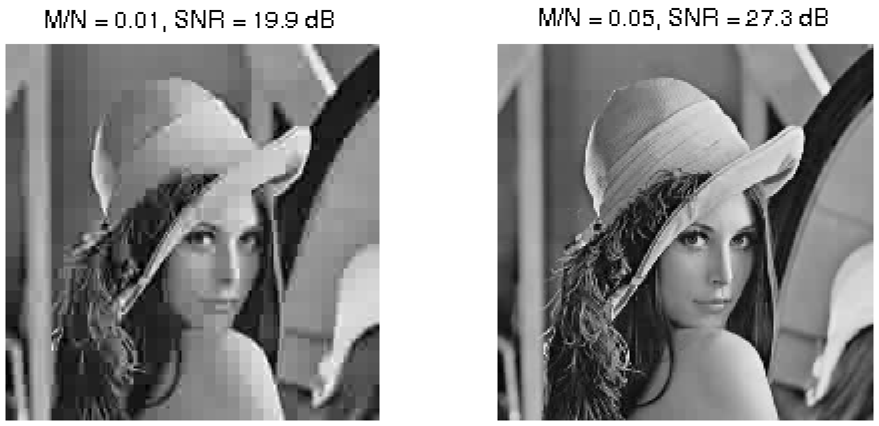
Exercise 9 - Image Approximation with Orthogonal Bases

```
for ii = 1:n/w
    for jj = 1:n/w
        seli = (ii-1)*w+1:ii*w;
        selj = (jj-1)*w+1:jj*w;
        P = fL(seli,selj); % Select patch P
        f1(seli,selj) = idct2(P);
    end
end

disp(strcat((['Error |M-M1|/|M| = ' num2str(norm(f(:)-f1(:))/norm(f(:)))])));
```

```
Error |M-M1|/|M| = 3.8496e-16
```

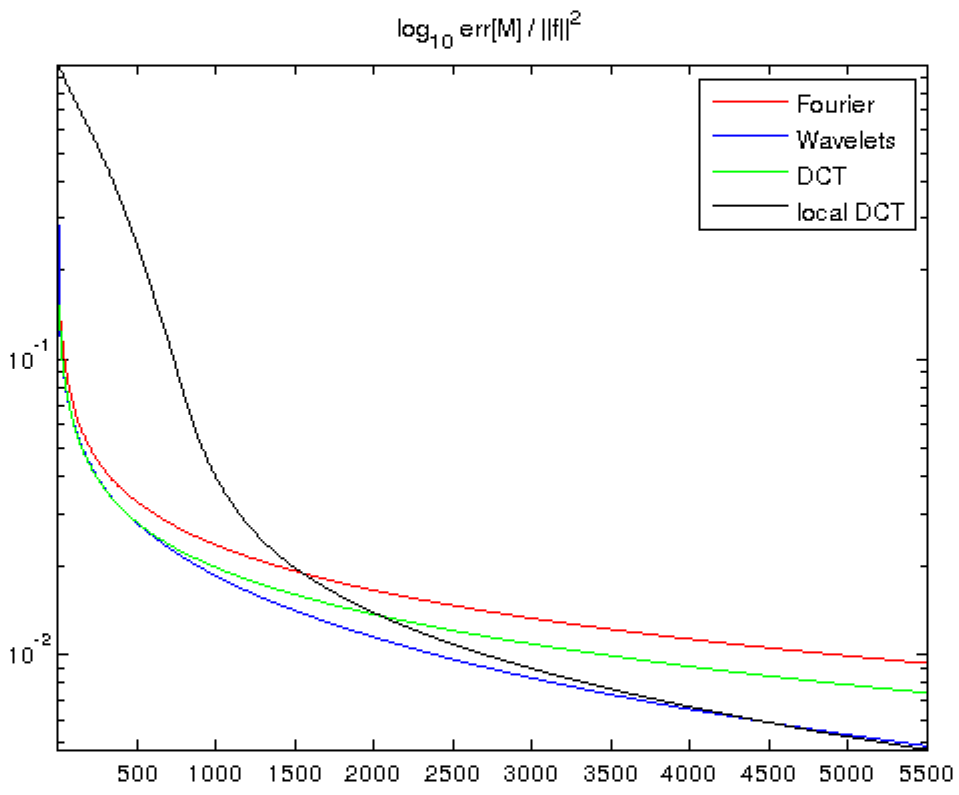
Exercise 10 - Image Approximation with Orthogonal Bases

```
snark = sort(abs(fL(:)),1,'descend');

for kk = 1:2

    T = snark(M(kk));
    c = fL .* (abs(fL)>T);

    for ii = 1:n/w
        for jj = 1:n/w
            seli = (ii-1)*w+1:ii*w;
            selj = (jj-1)*w+1:jj*w;
            P = c(seli,selj); % Select patch P
            fM(seli,selj) = idct2(P);
        end
    end

    figure(122)
    subplot(1,2,kk)
    imageplot(clamp(fM))
    title(sprintf('M/N = %0.2f, SNR = %2.1f dB',M(kk)/N,snr(f,fM)));
end
```



Exercise 11 - Image Approximation with Orthogonal Bases

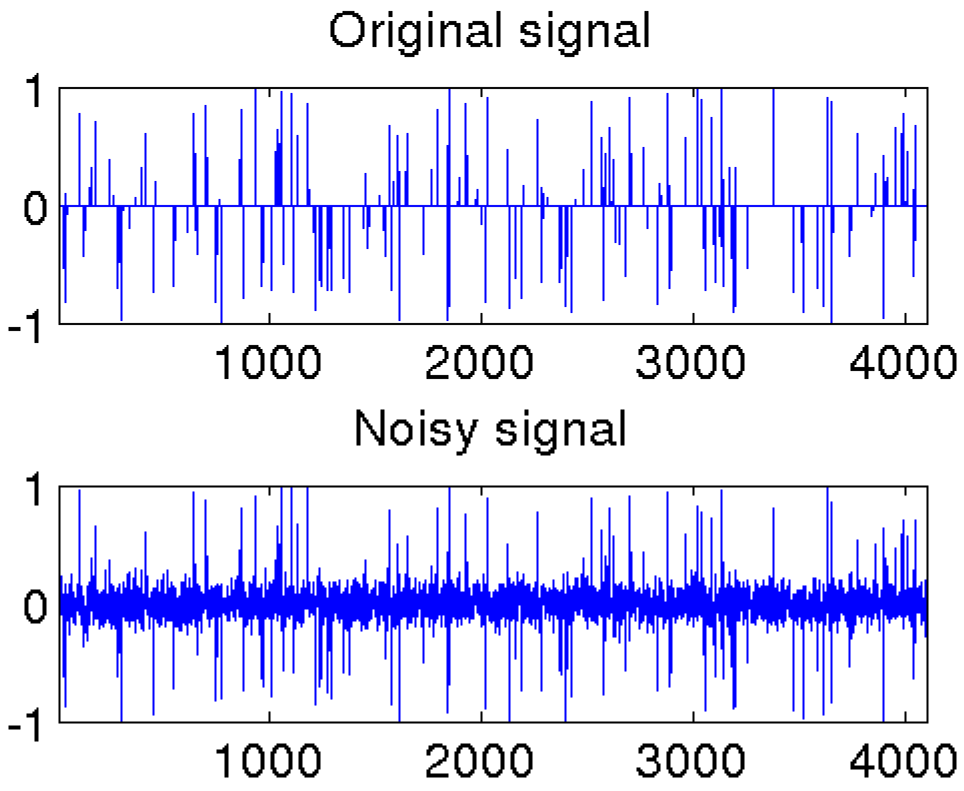
```
err_ldct = f(:)'*f(:) - cumsum(snark(1:5500).^2);
figure(1365);
semilogy([1:5500],err_fft./(f(:)'*f(:)),'r',[1:5500],err_wav./(f(:)'*f(:)),...
    'b',[1:5500],err_dct./(f(:)'*f(:)),'g',[1:5500],err_ldct./(f(:)'*f(:)),'k')
title('log_1_0 err[M] / ||f||^2')
axis tight
legend('Fourier','Wavelets','DCT','local DCT')
```

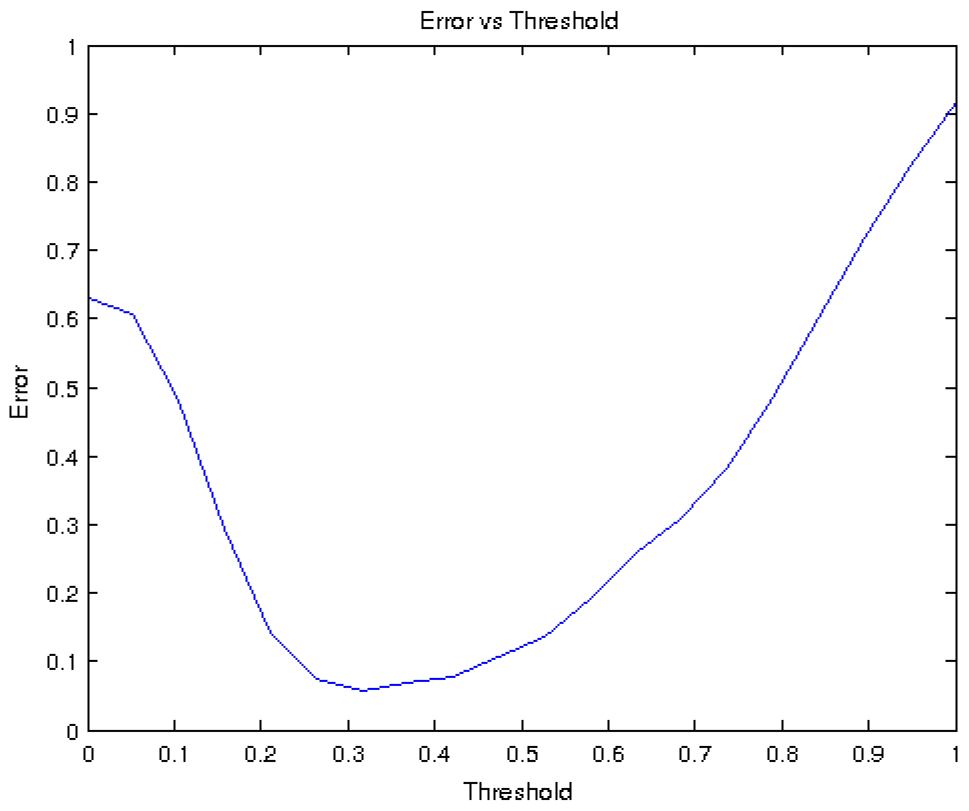


Exercise 1 - Signal and Image Noise Models

```
% dimension
n = 4096;
% probability of spiking
rho = .05;
% location of the spike
x0 = rand(n,1)<rho;
% random amplitude in [-1 1]
x0 = 2 * x0 .* ( rand(n,1)-.5 );

sigma = .1;
x = x0 + randn(size(x0))*sigma;

figure(6786)
subplot(2,1,1);
    plot(x0); axis([1 n -1 1]);
    set_graphic_sizes([], 20);
    title('Original signal');
    subplot(2,1,2);
    plot(x); axis([1 n -1 1]);
    set_graphic_sizes([], 20);
    title('Noisy signal');

X = kron(x,ones(1,20)); % Build repeat array
X0 = kron(x0,ones(1,20)); % Build repeat array
thresh = linspace(0,1,20); % Build thresh levels
T = kron(thresh,ones(length(x),1));
XT = X.*(abs(X) > T);
```

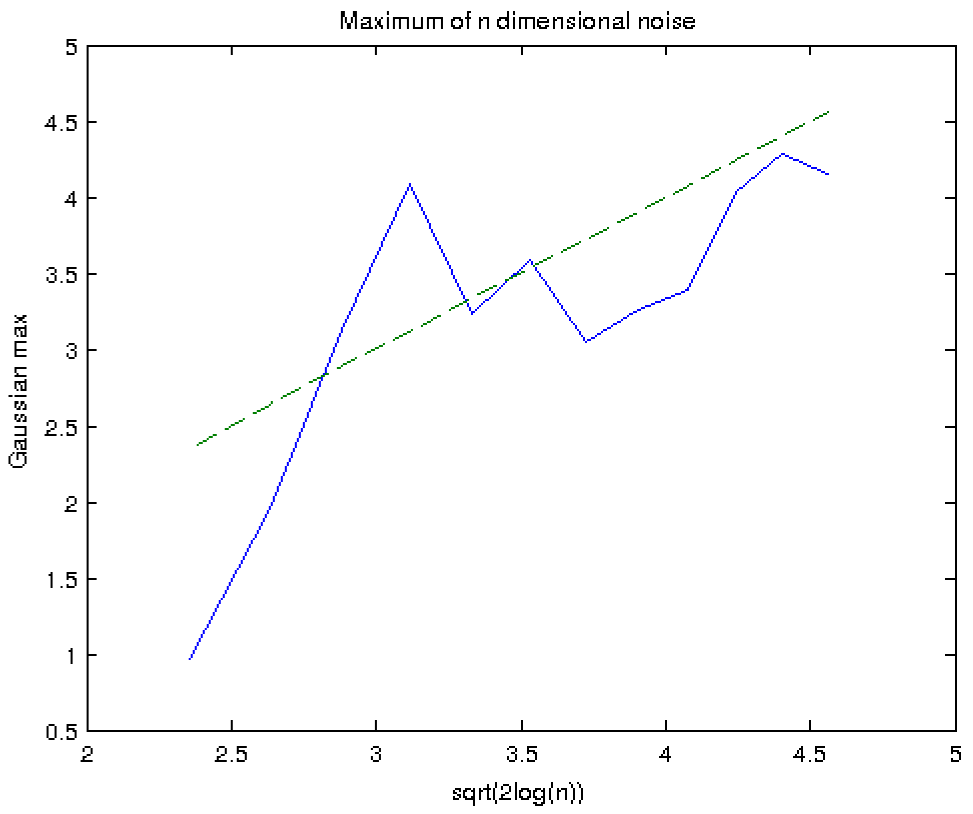
```
%imagesc(XT)

%XT = X.*(X >
err_T = sum( (XT - X0).^2,1) / norm(x0)^2;
figure(123423)
plot(thresh,err_T)
xlabel('Threshold')
ylabel('Error')
title('Error vs Threshold')
```





Exercise 2 - Signal and Image Noise Models

```
nn = 2.^[4:15];
for ii = 1:length(nn)
gmax_comp(ii)   = max(randn(nn(ii),1));

end
gmax_theory = sqrt(2*log(nn));
figure(134634)
plot(gmax_theory,gmax_comp,gmax_theory,gmax_theory,'--')
title('Maximum of n dimensional noise')
xlabel('sqrt(2log(n))')
ylabel('Gaussian max')
```

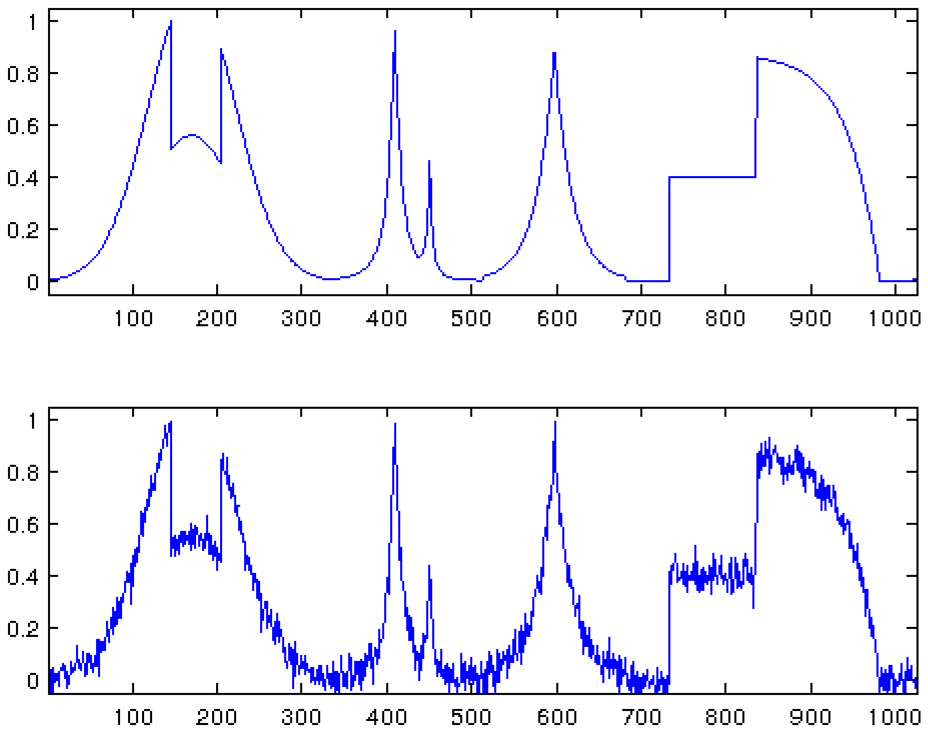


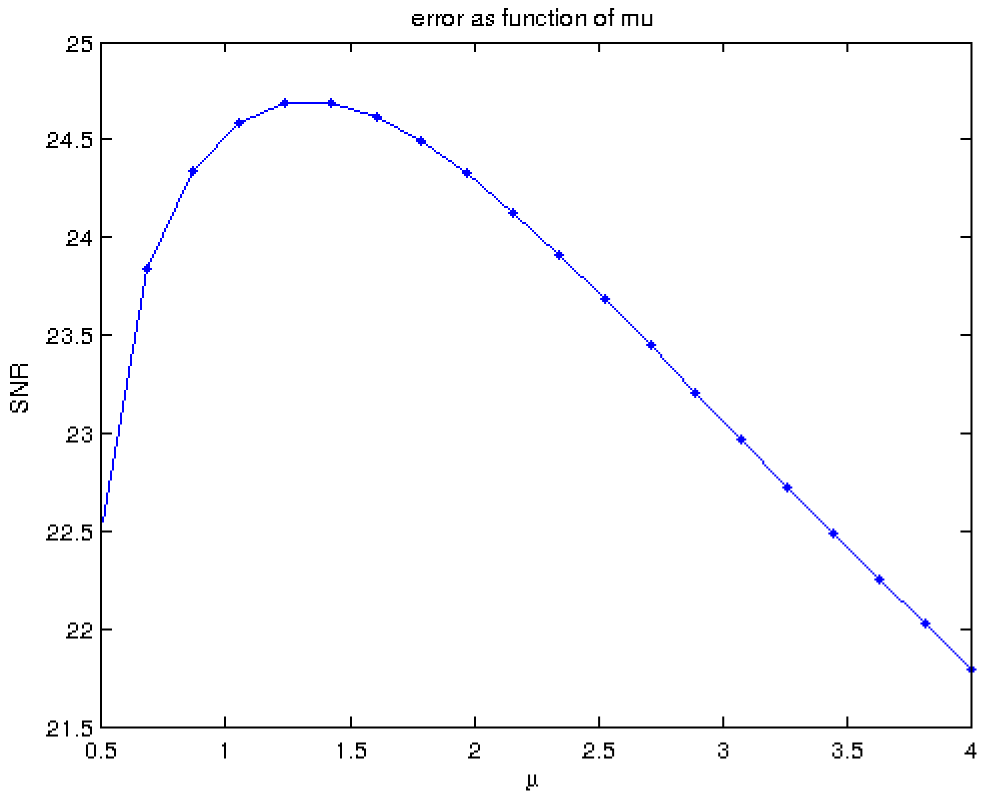
Exercise 1- Image Denoising with Linear Methods

```
name = 'piece-regular';
n = 1024;
x0 = load_signal(name,n);
x0 = rescale(x0);

% Add Noise
sigma = .04; % noise level
x = x0 + sigma*randn(size(x0));
figure(146)
subplot(2,1,1);
    plot(x0); axis([1 n -.05 1.05]);
subplot(2,1,2);
    plot(x); axis([1 n -.05 1.05]);

mulist = linspace(.5,4,20);
t = (-length(x)/2:length(x)/2-1)';


for ii=1:length(mulist)
    mu = mulist(ii);
    % compute the filter
    h = exp( -(t.^2)/(2*mu^2) );
    h = h/sum(h(:));
    % perform the blurring
    xh = real( ifft( fft(x) .* fft(fftshift(h)) ));
    SNR(ii) =  snr(x0,xh);
```

```
end

[maxsnr,ind] = max(SNR);
disp(sprintf('The optimal smoothing width is %f pixels, SNR= %f dB',...
    mulist(ind),maxsnr))
figure(1232)
plot(mulist,SNR,'.-')
title('error as function of mu')
xlabel('\mu')
ylabel('SNR')
```

```
The optimal smoothing width is 1.421053 pixels, SNR= 24.691303 dB
```



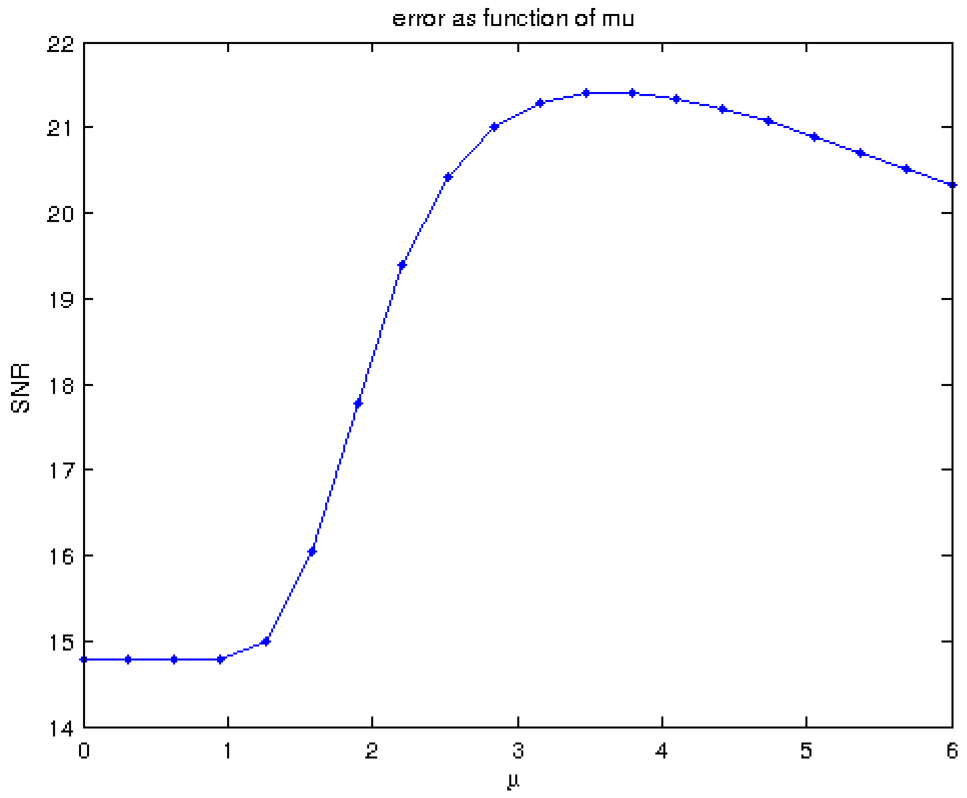


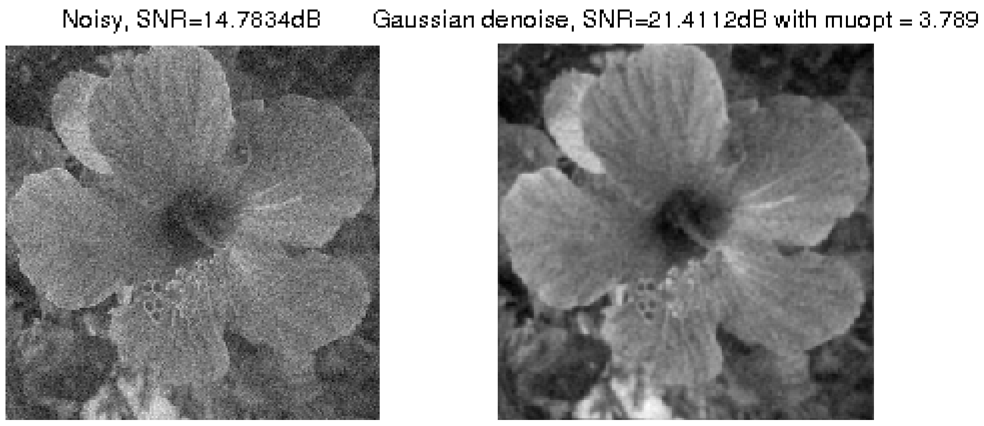
Exercise 2 - Image Denoising with Linear Methods

```
n = 256;
name = 'hibiscus';
M0 = load_image(name,n);
M0 = rescale( sum(M0,3) );

%Then we add some gaussian noise to it.
sigma = .08; % noise level
M = M0 + sigma*randn(size(M0));
% we use cyclic boundary condition since it is quite faster
options.bound = 'per';
% number of pixel of the filter
mu = 10;
Mh = perform_blurring(M,mu,options);
mulist = linspace(0,6,20);
figure(157)
for ii=1:length(mulist)
    mu = mulist(ii);
    Mh = perform_blurring(M,mu,options);
    SNR(ii) =  snr(M0,Mh);
end

[maxsnr,ind] = max(SNR);
disp(sprintf('The optimal smoothing width is %f pixels, SNR= %f dB',...
    mulist(ind),maxsnr))

figure(1357)
```

```
plot(mulist,SNR,'.-')
title('error as function of mu')
xlabel('\mu')
ylabel('SNR')

% optimal filter
Mgauss = perform_blurring(M,mulist(ind),options);
% display
figure(6834)
imageplot(M, strcat(['Noisy, SNR=' num2str(snr(M0,M)) 'dB']), 1,2,1);
imageplot(Mgauss, strcat(['Gaussian denoise, SNR=' num2str(snr(M0,Mgauss))...
    'dB with muopt = ' num2str(mulist(ind))]), 1,2,2);
```

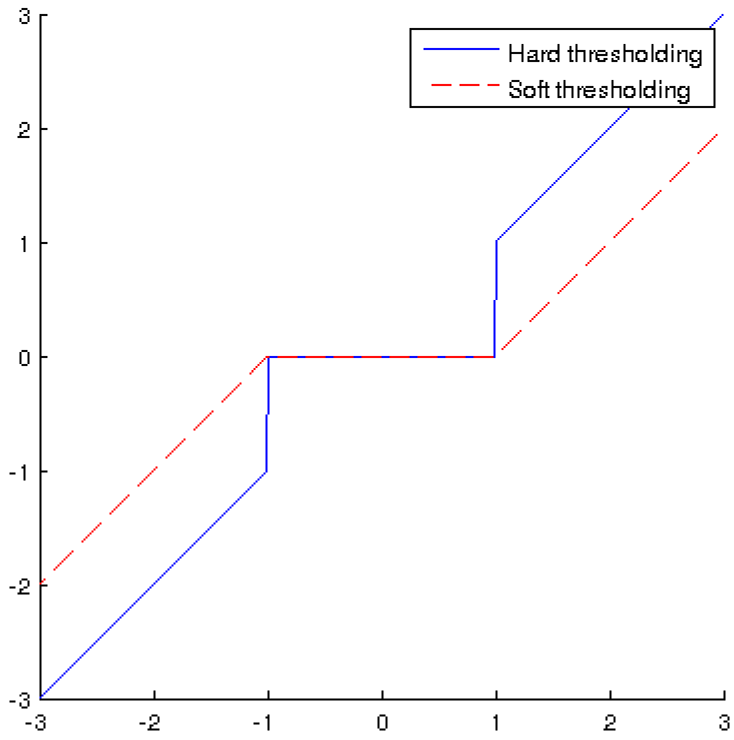
```
The optimal smoothing width is 3.789474 pixels, SNR= 21.411166 dB
```


error as function of mu
SNR
μ
22
21
20
19
18
17
16
15
14
0
1
2
3
4
5
6


Noisy, SNR=14.7834dB
Gaussian denoise, SNR=21.4112dB with muopt = 3.789

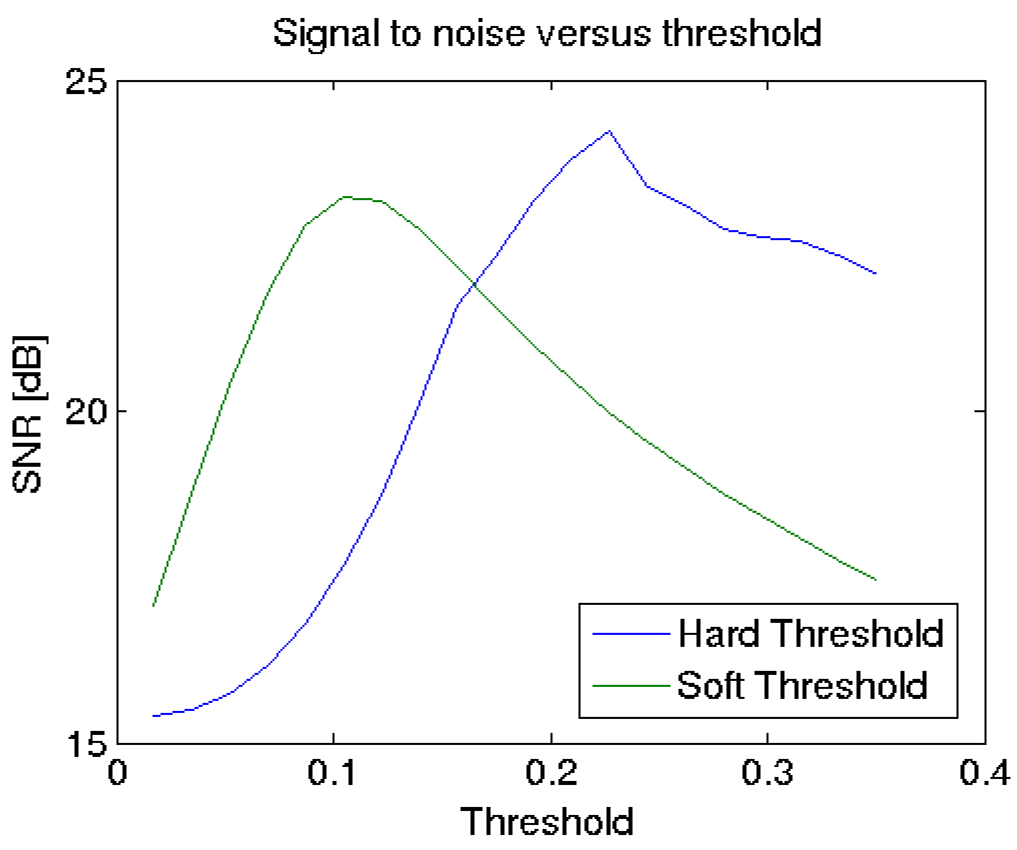
Exercise 1 - Signal Denoising with Wavelets

```
% size
n = 2048;
% clean signal
name = 'piece-regular';
x0 = load_signal(name, n);
x0 = rescale(x0,.05,.95);

% Add gaussian noise
sigma = 0.07;
x = x0 + randn(size(x0))*sigma;

% threshold value
T = 1;
v = -linspace(-3,3,2000);

% hard thresholding of the t values
v_hard = v.*(abs(v)>T);

% soft thresholding of the t values
v_soft = max(1-T./abs(v), 0).*v;

% display
figure(25831)
hold('on');
plot(v, v_hard);
plot(v, v_soft, 'r--');
axis('equal'); axis('tight');
legend('Hard thresholding', 'Soft thresholding');
hold('off');
```



Exercise 2 - Signal Denoising with Wavelets

```
options.ti = 0;
Jmin = 4;
xW = perform_wavelet_transf(x,Jmin,+1,options);


TT = sigma*[0.25:0.25:5];
for ii = 1:length(TT)
    T = TT(ii);
    % Hard Threshold
    xWT = perform_thresholding(xW,T,'hard');
    % re-inject the low frequencies
    xWT(1:2^Jmin) = xW(1:2^Jmin);
     % re-construct
    xhard = perform_wavelet_transf(xWT,Jmin,-1,options);
    SNRh(ii) = snr(x0,xhard);
    % Soft Threshold
    xWT = perform_thresholding(xW,T,'soft');
    % re-inject the low frequencies
    xWT(1:2^Jmin) = xW(1:2^Jmin);
     % re-construct
    xsoft = perform_wavelet_transf(xWT,Jmin,-1,options);
    SNRs(ii) = snr(x0,xsoft);
end
figure(2315)
    set_graphic_sizes([], 15);
    plot(TT,SNRh,TT,SNRs)
    title('Signal to noise versus threshold')
    xlabel('Threshold')
    ylabel('SNR [dB]')
    legend('Hard Threshold','Soft Threshold','Location','Best')
```

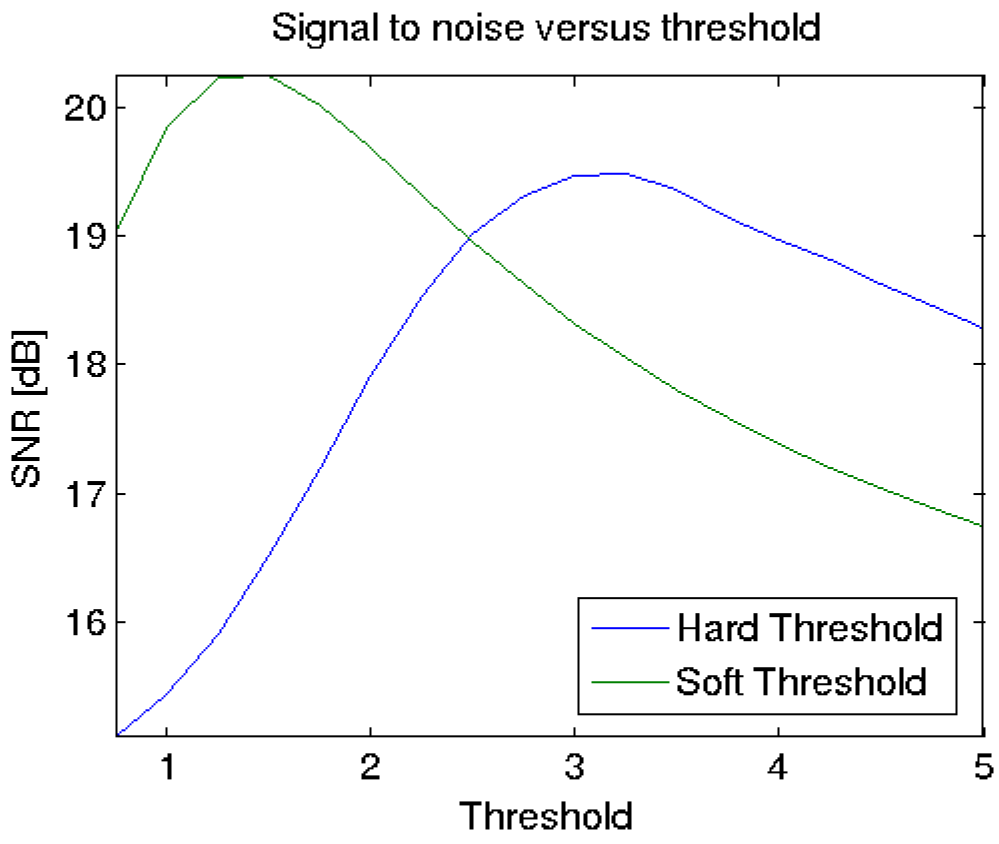


Commentary

```
% It can be seen that hard thresholding allows for more of the energy of
% the original image to be captured. Also more coefficients are required
% by the soft thresholding method, thus the results will be less smooth,
% and contain elements of what may be noise.
```

Exercise 1 - Image Denoising with Wavelets

```
n = 256;
name = 'hibiscus';
f0 = rescale( load_image(name,n) );
f0 = rescale( sum(f0,3) );

sigma = .08;
TT = sigma*[0.75:0.25:5];

f = f0 + sigma*randn(size(f0));

SNRh = 0; % Zero vectors
SNRs = 0;

options.ti = 0;
Jmin = 4;
a = perform_wavelet_transf(f,Jmin,+1,options);

for ii = 1:length(TT)
    T = TT(ii);
    aTh = perform_thresholding(a,T,'hard');
    aTh(1:2^Jmin,1:2^Jmin) = a(1:2^Jmin,1:2^Jmin);
    fHard = perform_wavelet_transf(aTh,Jmin,-1,options);
    SNRh(ii) = snr(f0,fHard);
    aTs = perform_thresholding(a,T,'soft');
    aTs(1:2^Jmin,1:2^Jmin) = a(1:2^Jmin,1:2^Jmin);
    fSoft = perform_wavelet_transf(aTs,Jmin,-1,options);
    SNRs(ii) = snr(f0,fSoft);
end

[maxsnr,ind] = max(SNRh);
aTh = perform_thresholding(a,TT(ind),'hard');
aTh(1:2^Jmin,1:2^Jmin) = a(1:2^Jmin,1:2^Jmin);
fHard = perform_wavelet_transf(aTh,Jmin,-1,options);

figure(2335)
    set_graphic_sizes([], 15);
    plot(TT./sigma,SNRh,TT./sigma,SNRs)
    axis tight
    title('Signal to noise versus threshold')
    xlabel('Threshold')
    ylabel('SNR [dB]')
    legend('Hard Threshold','Soft Threshold','Location','Best')
```

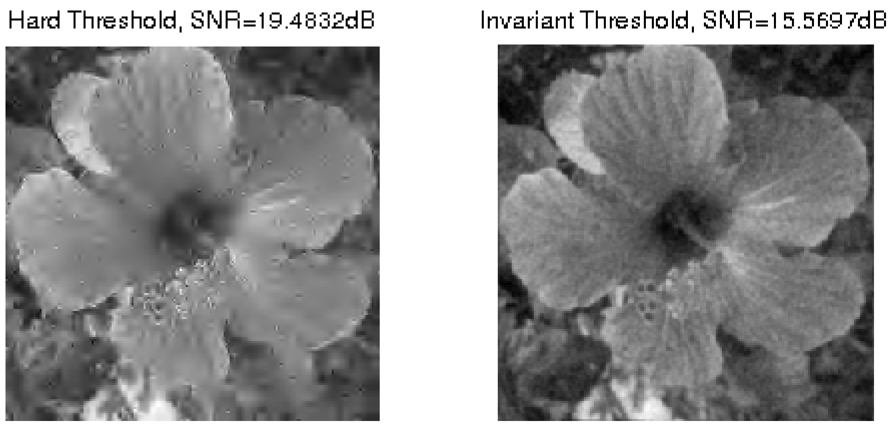


Commentary

The Soft thresholding has better performance when allowing for a larger number of coefficients. This is different than the 1D case.

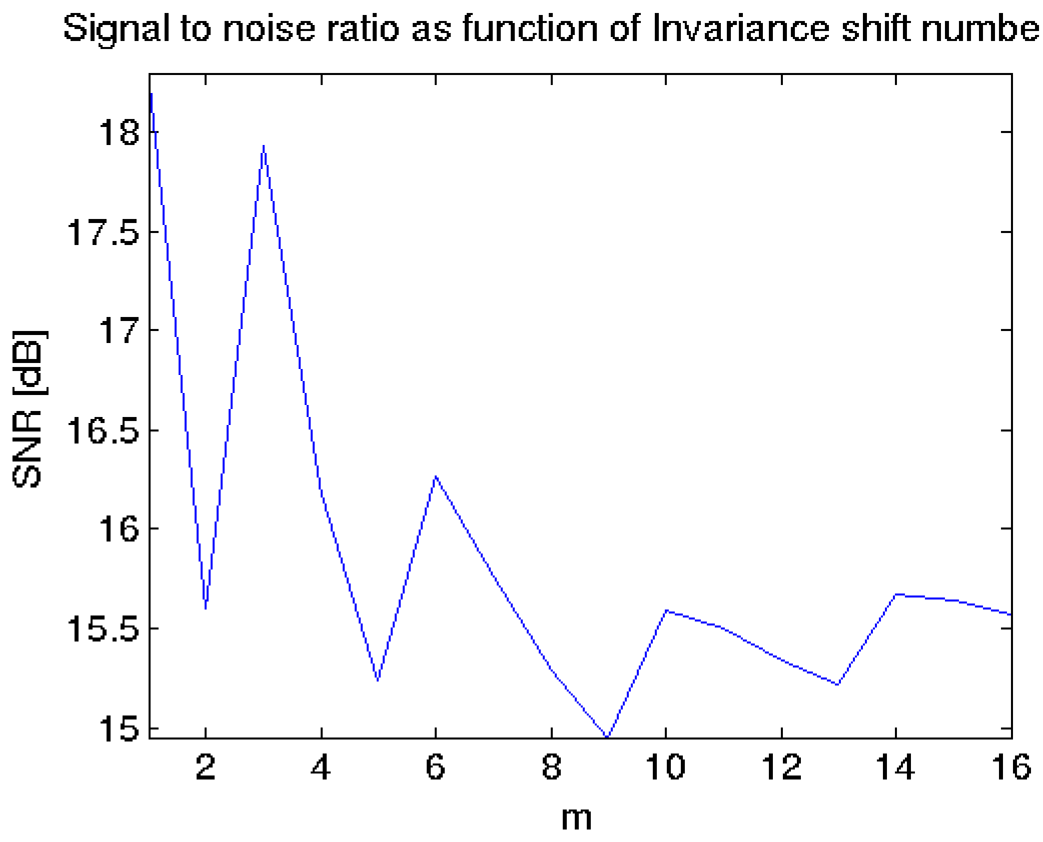
Exercise 2 - Image Denoising with Wavelets

```
m = 4;
% Generate Shifts
[dY,dX] = meshgrid(0:m-1,0:m-1);
delta = [dX(:) dY(:)]';
fTI = zeros(n,n);
for ii = 1:m^2;
    %Apply the shift, using circular boundary conditions.
    fS = circshift(f,delta(:,ii));

    %Apply here the denoising to fS.
    a = perform_wavelet_transf(fS,Jmin,1,options);
    T = perform_thresholding(a,T,'hard');
    fS = perform_wavelet_transf(T,Jmin,-1,options);

    %After denoising, do the inverse shift.
    fS = circshift(fS,-delta(:,ii));

    %Accumulate the result to obtain at the end the denoised image that
    % average the translated results.
    fTI = (ii-1)/ii*fTI + 1/ii*fS;

    % For studying the influence of number m on quality
    SNRi(ii) = snr(f0,fTI);
end


figure(3316)
imageplot(fHard, strcat(['Hard Threshold, SNR=' num2str(snr(f0,fHard))...
    'dB']), 1,2,1);
imageplot(Mgauss, strcat(['Invariant Threshold, SNR=' num2str(snr(f0,fTI))...
    'dB']), 1,2,2);
```


Hard Threshold, SNR=19.4832dB
Invariant Threshold, SNR=15.5697dB

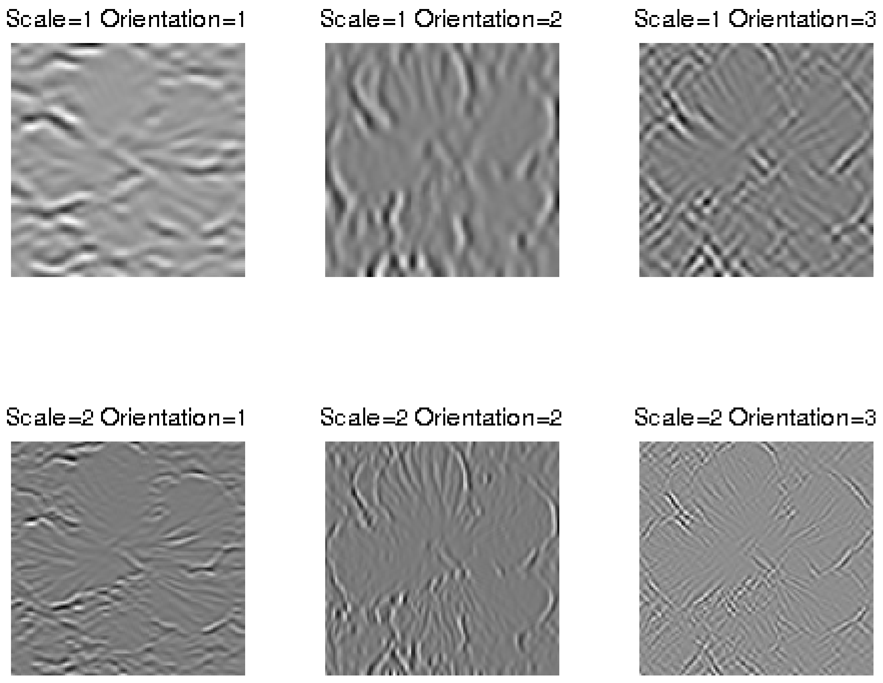
Exercise 2 - Image Denoising with Wavelets

```
figure(2311)
    set_graphic_sizes([], 15);
    plot(1:m^2,SNRi)
    axis tight
    title('Signal to noise ratio as function of Invariance shift number m')
    xlabel('m')
    ylabel('SNR [dB]')
```



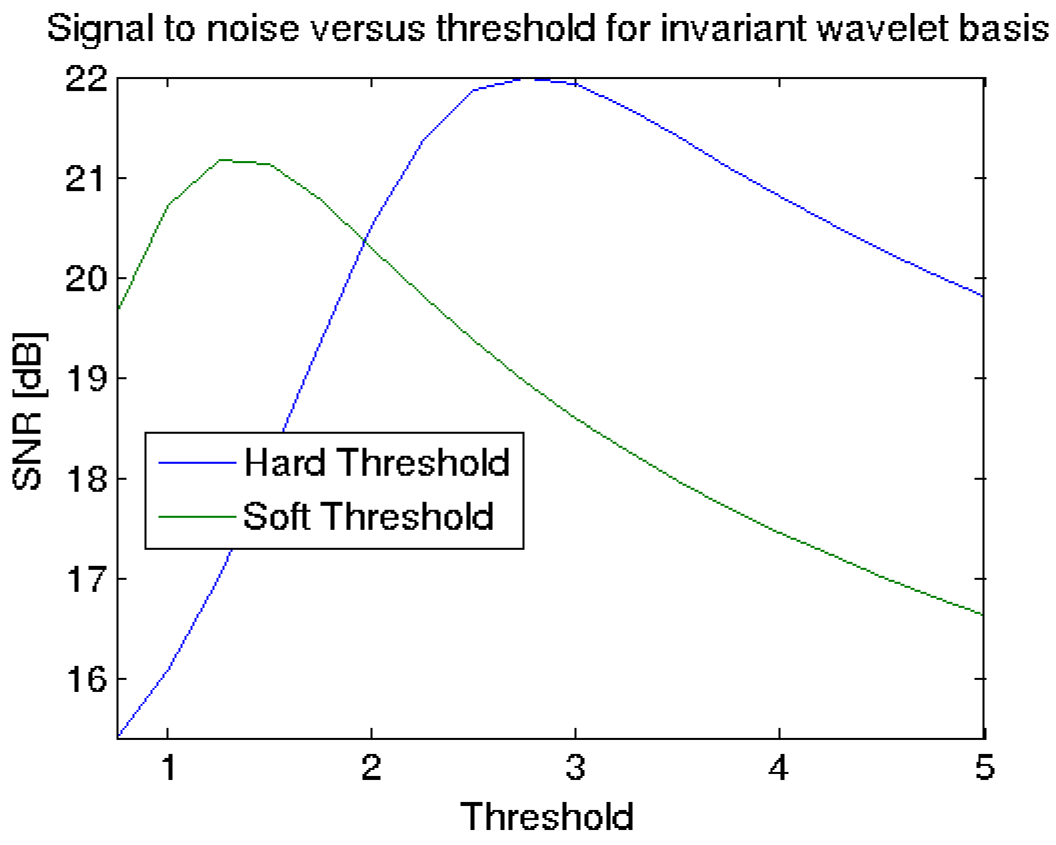
Exercise 3 - Image Denoising with Wavelets

```
options.ti = 1;
a = perform_wavelet_transf(f0,Jmin,+1,options);
figure(9932)
ii = 0;
for j=1:2
    for k=1:3
        ii = ii+1;
        imageplot(a(:,:,ii+1), strcat(['Scale=' num2str(j) ' Orientation=' num2str
    end
end
```



Exercise 4 - Image Denoising with Wavelets

```
options.ti = 1;
a = perform_wavelet_transf(f,Jmin,+1,options);

for ii = 1:length(TT)
    T = TT(ii);
    % Hard
    aTh = perform_thresholding(a,T,'hard');
    fHard = perform_wavelet_transf(aTh,Jmin,-1,options);
    SNRh(ii) = snr(f0,fHard);
    % Soft
    aTs = perform_thresholding(a,T,'soft');
    aTs(1:2^Jmin,1:2^Jmin) = a(1:2^Jmin,1:2^Jmin);
    fSoft = perform_wavelet_transf(aTs,Jmin,-1,options);
    SNRs(ii) = snr(f0,fSoft);
end

figure(2095)
    set_graphic_sizes([], 15);
    plot(TT./sigma,SNRh,TT./sigma,SNRs)
    axis tight
    title('Signal to noise versus threshold for invariant wavelet basis')
    xlabel('Threshold')
    ylabel('SNR [dB]')
    legend('Hard Threshold','Soft Threshold','Location','Best')
```



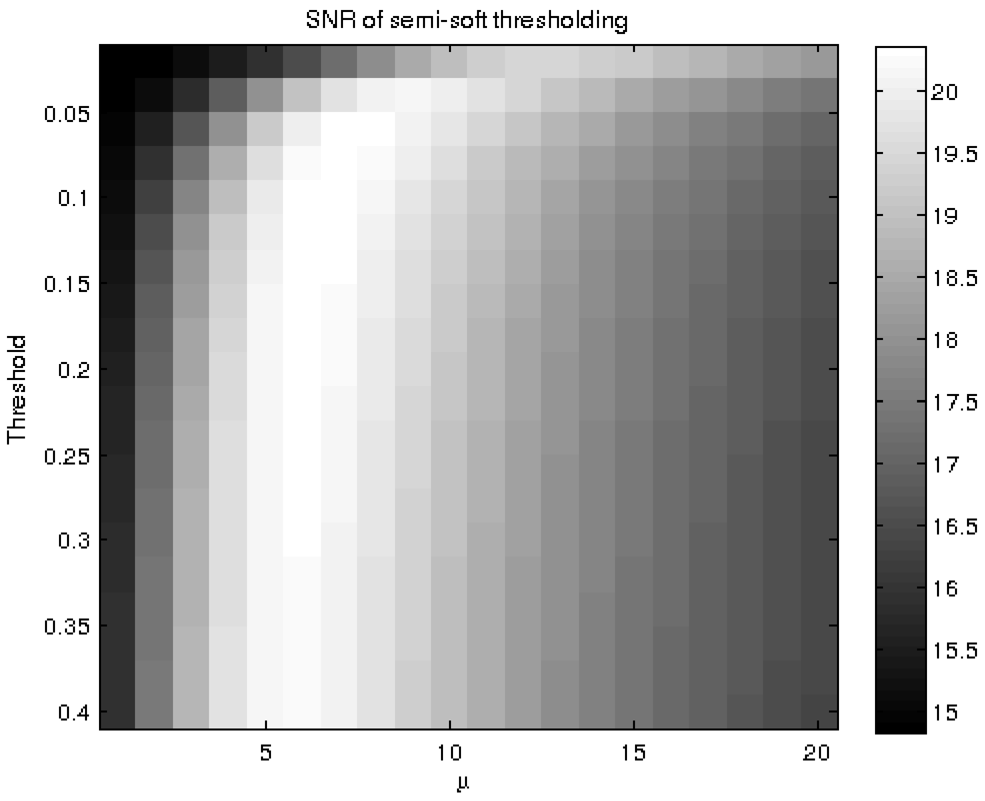
Commentary

The hard thresholding now provides a better resconstruction of the image with fewer coefficients than the soft thresholding case.

Exercise 1 - Advanced Wavelet Thresholding

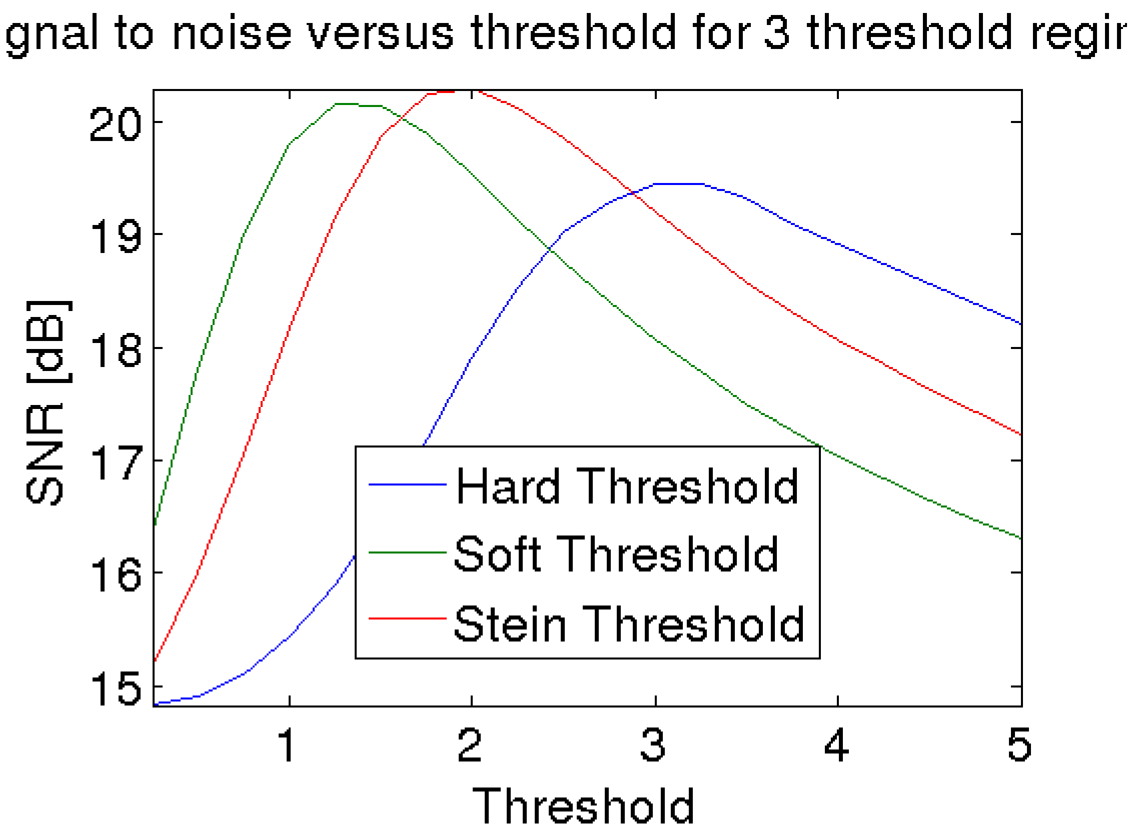
```
n = 256;
options.ti = 0;
name = 'hibiscus';
M0 = load_image(name,n);
M0 = rescale( sum(M0,3) );
%Noise level.
TT = sigma*linspace(0.25,5,20);
sigma = .08;
%Then we add some Gaussian noise to it.

M = M0 + sigma*randn(size(M0));
%Compute a 2D orthogonal wavelet transform.
SNR = 0;
Jmin = 3;
MW = perform_wavelet_transf(M,Jmin,+1);

for mu = 1:20
    for ii = 1:20
         MT = perform_thresholding(MW,[TT(ii) mu*TT(ii)],'semisoft');
         MT(1:2^Jmin,1:2^Jmin) = MW(1:2^Jmin,1:2^Jmin);
         Mf = perform_wavelet_transf(MT,Jmin,-1,options);
         SNR(mu,ii) = snr(M0,Mf);
    end
```

```
end

figure(3329)
imagesc(1:20,TT,SNR)
xlabel('\mu')
ylabel('Threshold')
title('SNR of semi-soft thresholding')
colormap Gray
colorbar
```



Exercise 2 - Advanced Wavelet Thresholding

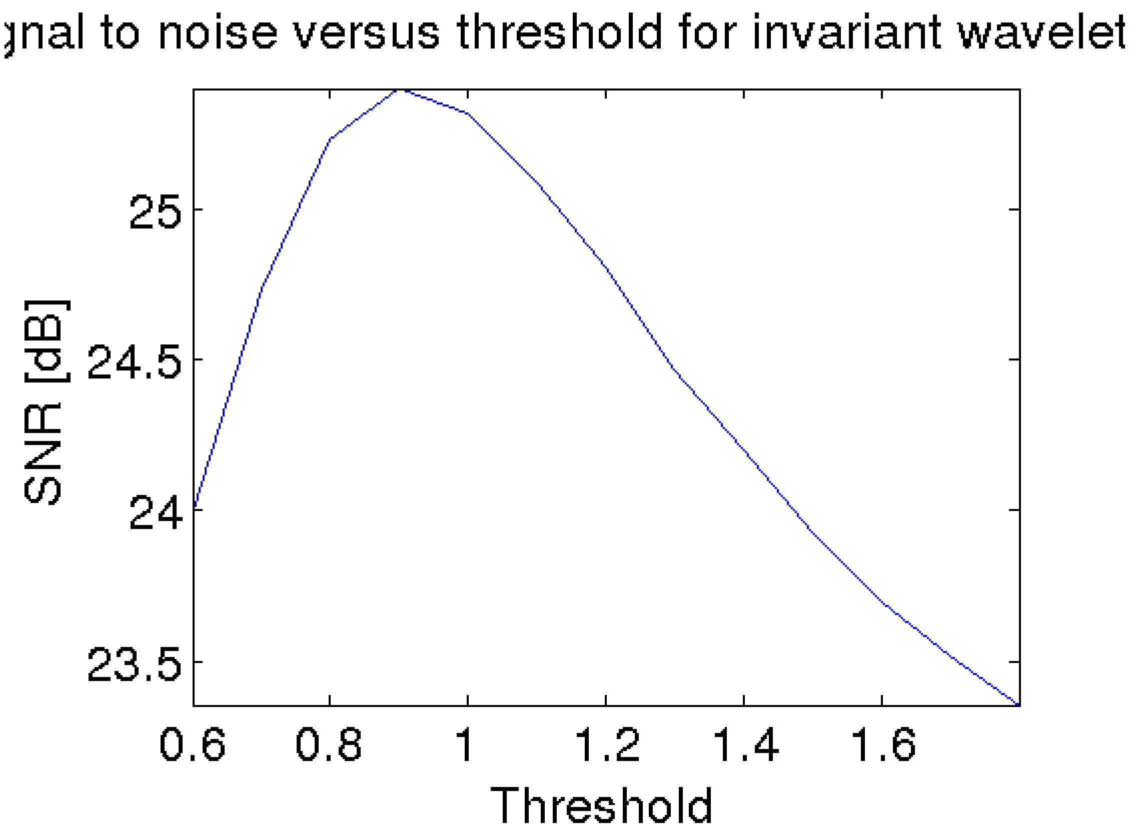
```
options.ti = 0;
a = perform_wavelet_transf(f,Jmin,+1,options);

for ii = 1:length(TT)
    T = TT(ii);
    % Hard
    aTh = perform_thresholding(a,T,'hard');
    fHard = perform_wavelet_transf(aTh,Jmin,-1,options);
    SNRh(ii) = snr(f0,fHard);
    % Soft
    aTs = perform_thresholding(a,T,'soft');
    aTs(1:2^Jmin,1:2^Jmin) = a(1:2^Jmin,1:2^Jmin);
    fSoft = perform_wavelet_transf(aTs,Jmin,-1,options);
    SNRs(ii) = snr(f0,fSoft);

    % Stein
    aTst = perform_thresholding(a,T,'stein');
    aTst(1:2^Jmin,1:2^Jmin) = a(1:2^Jmin,1:2^Jmin);
    fStein = perform_wavelet_transf(aTst,Jmin,-1,options);
    SNRst(ii) = snr(f0,fStein);
end

figure(22495)
    set_graphic_sizes([], 20);
    plot(TT./sigma,SNRh,TT./sigma,SNRs,TT./sigma,SNRst)
    axis tight
    title('Signal to noise versus threshold for 3 threshold regimes')
    xlabel('Threshold')
    ylabel('SNR [dB]')
    legend('Hard Threshold','Soft Threshold','Stein Threshold','Location','Best')
```

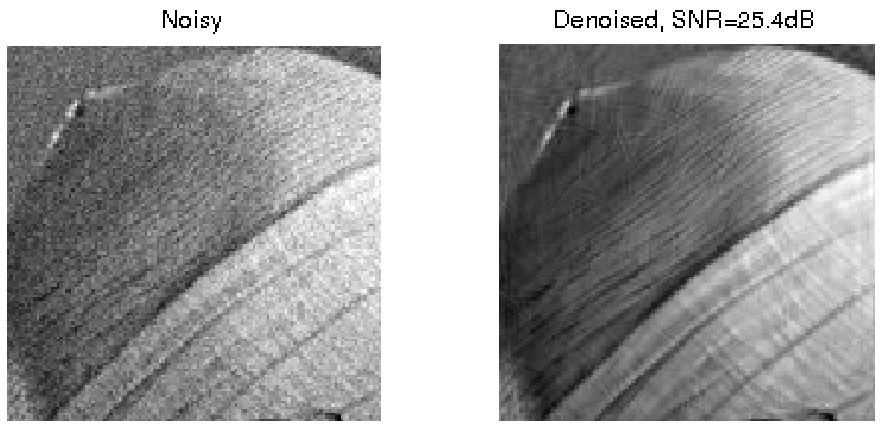


Exercise 1 - Curvelet Denoising

```
woptions = options;
name = 'lena';
n = 128;
M0 = rescale(crop(load_image(name),n, [108 200]));
options.null = 0;
options.finest = 1;
options.nbscales = 4;
options.nbangles_coarse = 16;
options.is_real = 1;
options.n = n;

SNR = 0 ;
%Add some noise.
sigma = .05;
M = M0 + sigma*randn(n);
```

```
TT = sigma*[0.6:0.1:1.8];
MW = perform_curvelet_transform(M, options);
for ii = 1:length(TT)
    T = TT(ii);
    MWT = perform_thresholding(MW, T, 'hard');
    M1 = perform_curvelet_transform(MWT, options);
    SNR(ii) = snr(M0,M1);
end
[maxsnr,ind] = max(SNR);
MWT = perform_thresholding(MW,TT(ind),'hard');
Mcurv = perform_curvelet_transform(MWT,options);

figure(1115)
    set_graphic_sizes([], 20);
    plot(TT./sigma,SNR)
    axis tight
    title('Signal to noise versus threshold for invariant wavelet basis')
    xlabel('Threshold')
    ylabel('SNR [dB]')

figure(332211)
imageplot(clamp(M), 'Noisy', 1,2,1);
imageplot(clamp(Mcurv), ['Denoised, SNR=' num2str(snr(M0,Mcurv),3) 'dB'], 1,2,2);
```

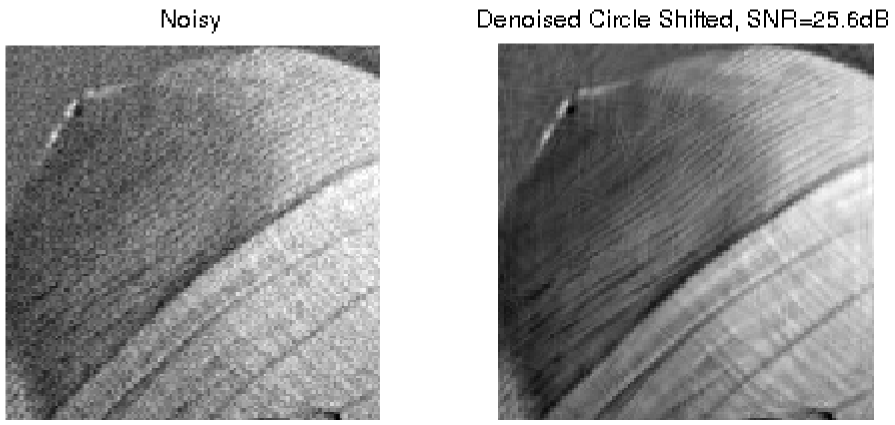




Exercise 2 - Curvelet Denoising

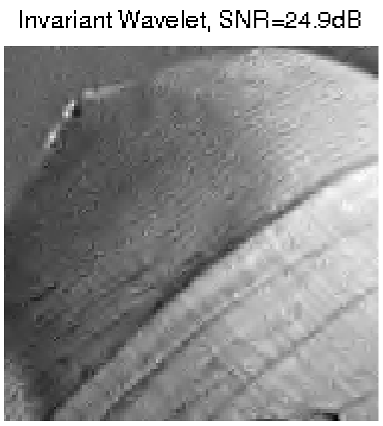
```
m = 4;
% Generate Shifts
[dY,dX] = meshgrid(0:m-1,0:m-1);
delta = [dX(:) dY(:)]';
MTI = zeros(n,n);

for ii = 1:m^2;
    %Apply the shift, using circular boundary conditions.
    MS = circshift(M,delta(:,ii));

    %Apply here the denoising to fS.
    MW = perform_curvelet_transform(MS,options);
    MWT = perform_thresholding(MW,TT(ind),'hard');
    MS = perform_curvelet_transform(MWT,options);

    %After denoising, do the inverse shift.
    MS = circshift(MS,-delta(:,ii));

    %Accumulate the result to obtain at the end the denoised image that
    % average the translated results.
    MTI = (ii-1)/ii*MTI + 1/ii*MS;

    % For studying the influence of number m on quality
    %SNRi(ii) = snr(M0,MTI);
end


figure(93111)
```

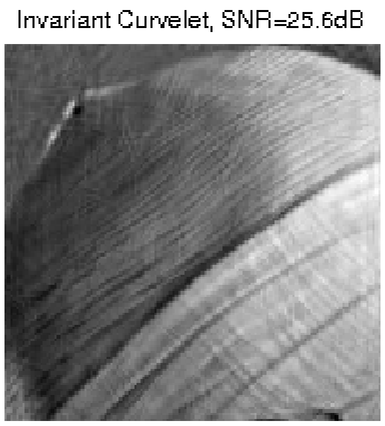
```
imageplot(clamp(M), 'Noisy', 1,2,1);
imageplot(clamp(MTI), ['Denoised Circle Shifted, SNR=' ...
    num2str(snr(M0,MTI),3) 'dB'], 1,2,2);
```



Exercise 3 - Curvelet Denoising

```
woptions.ti = 1;

% Hard


TT = sigma*[.8:0.2:4];

a = perform_wavelet_transf(M,3,+1,woptions);
for ii = 1:length(TT)
    T = TT(ii);
    aTh = perform_thresholding(a,T,'hard');
    fHard = perform_wavelet_transf(aTh,3,-1,woptions);
    SNR(ii) = snr(M0,fHard);
end
[maxsnr,ind] = max(SNR);
aTh = perform_thresholding(a,TT(ind),'hard');
fHard = perform_wavelet_transf(aTh,3,-1,woptions);

figure(9811)
imageplot(clamp(fHard), ['Invariant Wavelet, SNR=' ...
    num2str(snr(M0,fHard),3) 'dB'], 1,2,1);
imageplot(clamp(MTI), ['Invariant Curvelet, SNR=' ...
    num2str(snr(M0,MTI),3) 'dB'], 1,2,2);
```





Published with MATLAB® 7.11