```
% This exercise uses fake data from a natural movie to pracice obtaining an
% STRF using the analytical solution for linear regression, including ridge
% regression.
```

preliminary stuff: get the directory we're in

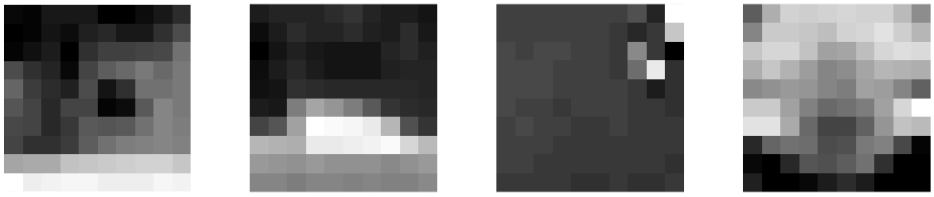
and add the proper subdirectories to the path

```
cpath = which('directfit_tutorial');
[rootDir, name, ext] = fileparts(cpath);
spath = fullfile(rootDir, 'strflab');
addpath(genpath(spath));
dfpath = fullfile(rootDir, 'direct_fit');
addpath(dfpath);
vpath = fullfile(rootDir, 'validation');
addpath(vpath);
ppath = fullfile(rootDir, 'preprocessing');
addpath(ppath);
dataDir = fullfile(rootDir, '..', 'data'); %contains stim/response pairs
stimsDir = fullfile(dataDir, 'all_stims'); %contains the .wav files
```

load a 20x20x20000 natural movie

```
load (strcat(dataDir,'/mov.mat'));
% let's only take a part of it
tlength = 15000;
rawStim = single(mov(1:10,1:10,1:tlength));  % single converts to single precision
meanStim = mean(mean(mean(rawStim)));
rawStim = rawStim - meanStim;

% Exercise 1. Using subplot plot the first 10 images.  Then plot images
% 10, 100, 1000, 10000.  Using these pictures comment on the temporal and spatial correlations
figure(1);
for ip=1:10
    subplot(1,10,ip);
    imagesc(rawStim(:,:,ip));
    colormap('gray');
    axis off;
    axis square;

end
```

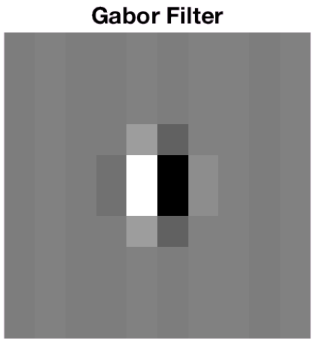
```
figure(2);
for ip=1:4
    subplot(1,4,ip);
    imagesc(rawStim(:,:,10^ip));
    colormap('gray');
    axis off;
    axis square;
end
```

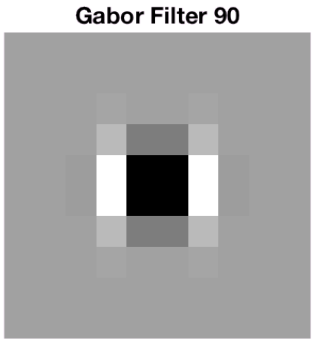


let's create some fake data for a simple cell V1 with a 2D Gabor filter

First we are going to make a Gabor filter

```
gparams = [.5 .5 0 5.5 0 .0909 .3 0]';
[gabor, gabor90] = make3dgabor([10 10 1], gparams);

% Exercise 2. Plot the Gabor filter.
figure(3);
subplot(2,1,1);
imagesc(gabor);
colormap('gray');
title('Gabor Filter');
axis off;
axis square;
subplot(2,1,2);
imagesc(gabor90);
colormap('gray');
title('Gabor Filter 90');
axis off;
axis square;
```





Convolve the Gabor filter with the stimulus and add Gaussian noise to get a response with an SNR of 1

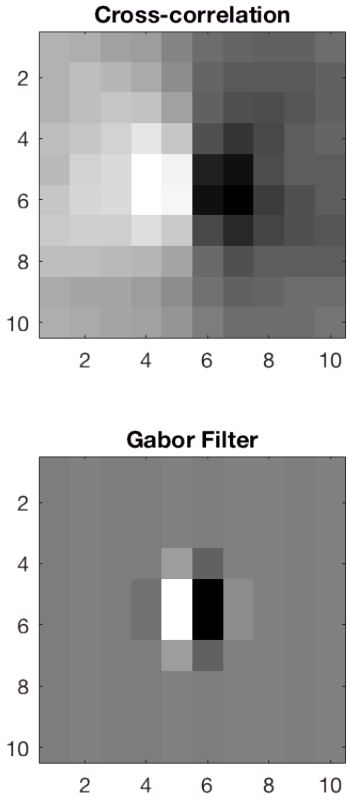
```
SNR = 0.5;
gabor_vector = reshape(gabor, [10*10 1]);
rawStim_vector = reshape(rawStim, [10*10 tlength]);
resp = dotdelay(gabor_vector, rawStim_vector);
resp_pow = var(resp);
resp = resp + sqrt(resp_pow/SNR)*randn(tlength,1);
```

now we're going to get estimate the gabor using the analytical solution. the stimulus and response

First cross-correlate the response and the stimulus. In this case there is no time component.

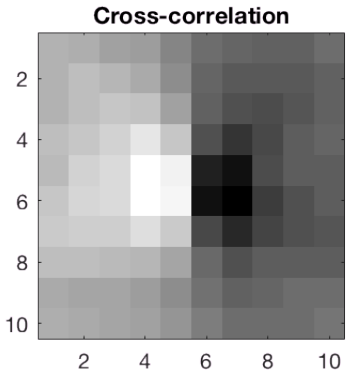
```
cross_stim_response = (rawStim_vector*resp)./tlength;  % This is a matrix multiply

% Plot the cross product and compare to the filter
figure(4);
subplot(2,1,1);
imagesc(reshape(cross_stim_response,[10 10]));
colormap('gray');
title('Cross-correlation');
axis image;
subplot(2,1,2);
imagesc(gabor);
colormap('gray');
```

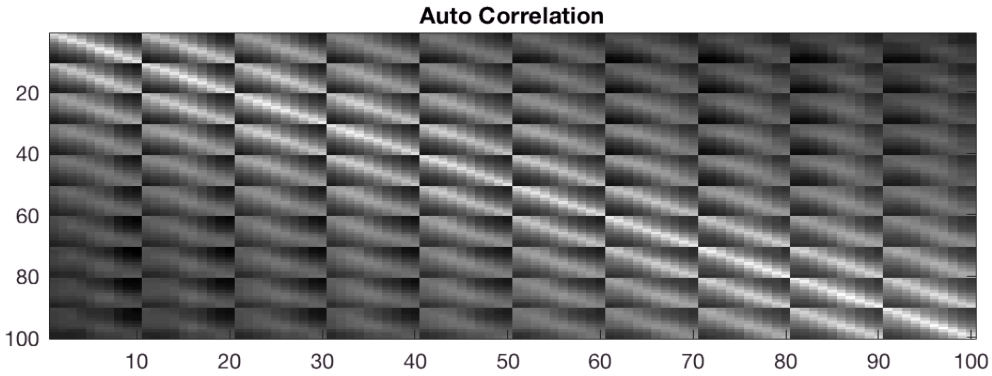
```
title('Gabor Filter');
axis image;
```



Now calculate the stimulus auto-correlation and image it.

```
mean_auto_corr = zeros(100, 100);
for it=1:tlength;
        auto_corr = rawStim_vector(:,it)*rawStim_vector(:,it)';
        mean_auto_corr = mean_auto_corr + auto_corr;
end
mean_auto_corr = mean_auto_corr./tlength;

figure(5);
imagesc(mean_auto_corr);
colormap('gray');
title('Auto Correlation');
```

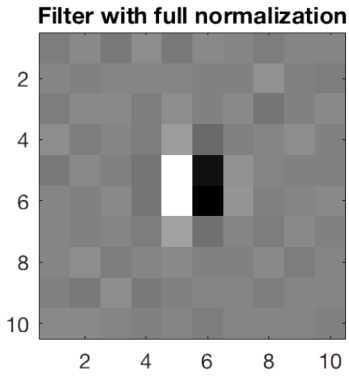


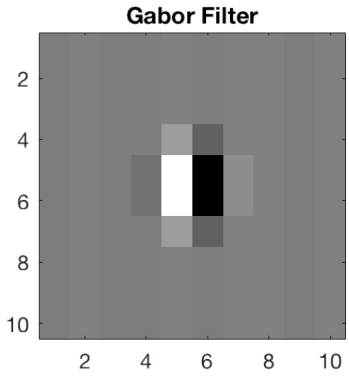


Now normalize the cross-correlation by the auto-correlation to recover the filter

Try first using the matrix division operator \

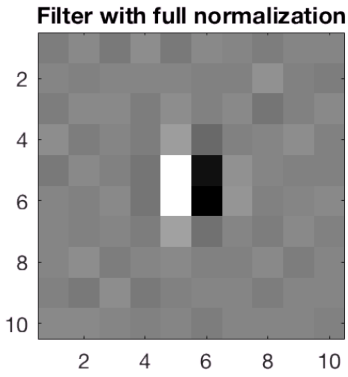
```
myfilter = mean_auto_corr\cross_stim_response;
figure(6);
subplot(2,1,1);
imagesc(reshape(myfilter,[10 10]));
colormap('gray');
title('Filter with full normalization');
axis image;
subplot(2,1,2);
imagesc(gabor);
colormap('gray');
title('Gabor Filter');
axis image;
```

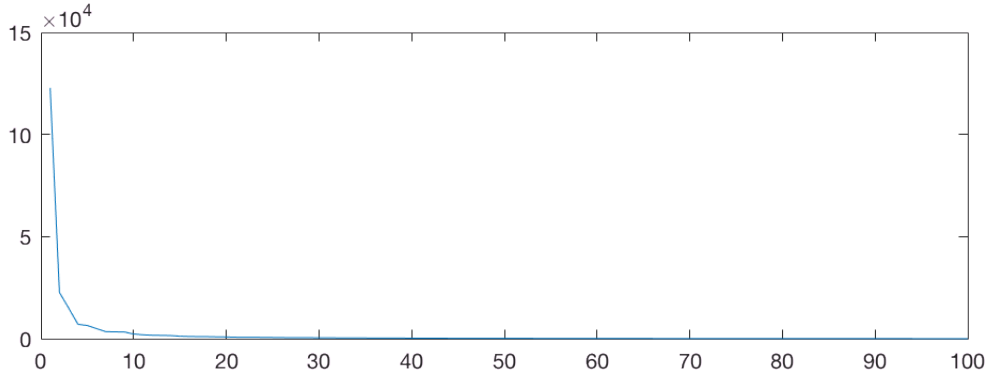




Then try using the PCA Regression

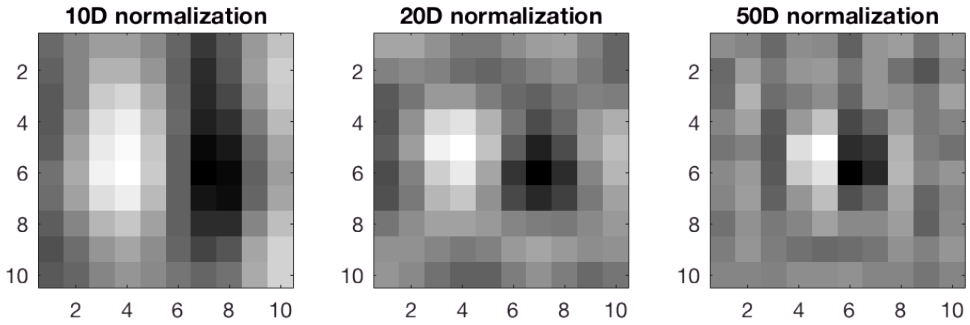
```
[u s v_svd] = svd(mean_auto_corr);
figure(7);
plot(diag(s));
```






```
% Let's try dividing in supspace of 10, 20, 50
sinv_10 = zeros(size(s));
for i=1:10;
    sinv_10(i,i) = 1./s(i,i);
end
myfilter_10 = v_svd*sinv_10*(u'*cross_stim_response);

sinv_20 = zeros(size(s));
for i=1:20;
    sinv_20(i,i) = 1./s(i,i);
end
myfilter_20 = v_svd*sinv_20*(u'*cross_stim_response);

sinv_50 = zeros(size(s));
for i=1:50;
    sinv_50(i,i) = 1./s(i,i);
end
myfilter_50 = v_svd*sinv_50*(u'*cross_stim_response);

figure(8);
subplot(1,3,1);
imagesc(reshape(myfilter_10, [10 10]));
colormap('gray');
title('10D normalization');
axis image;

subplot(1,3,2);
imagesc(reshape(myfilter_20, [10 10]));
colormap('gray');
title('20D normalization');
```

```
axis image;

subplot(1,3,3);
imagesc(reshape(myfilter_50, [10 10]));
colormap('gray');
title('50D normalization');
axis image;
```

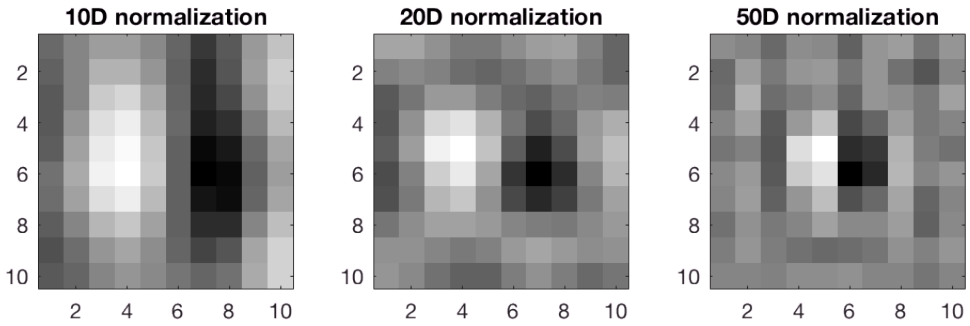


Repeat the previous section using eigen value decompostion. The eig command in matlab

does not necessarily sort the eigenvalues - use sort() to get then in descending order.

```
[v_eigen d] = eig(mean_auto_corr);
[ds, idx] = sort(diag(d), 'descend');
n_size = length(ds);
v_eigen_sorted = zeros(n_size, n_size); % Pre-allocate for speed
for i=1:n_size
    v_eigen_sorted(:,i) = v_eigen(:,idx(i));
end
d = diag(ds);   % Makes a new diagonal matrix


figure(9);
plot(diag(d));

% Let's try dividing in supspace of 10, 20, 50
dinv_10 = zeros(size(d));
for i=1:10;
```

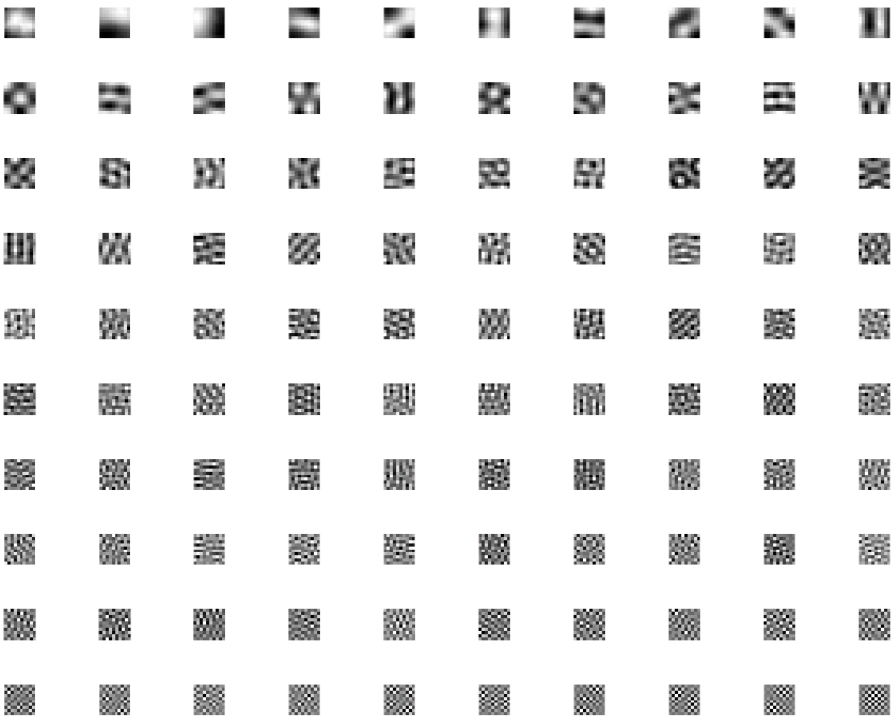
```
    dinv_10(i,i) = 1./d(i,i);
end
myfilter_10 = v_eigen_sorted*dinv_10*(v_eigen_sorted'*cross_stim_response);

dinv_20 = zeros(size(d));
for i=1:20;
    dinv_20(i,i) = 1./d(i,i);
end
myfilter_20 = v_eigen_sorted*dinv_20*(v_eigen_sorted'*cross_stim_response);

dinv_50 = zeros(size(s));
for i=1:50;
    dinv_50(i,i) = 1./s(i,i);
end
myfilter_50 = v_eigen_sorted*dinv_50*(v_eigen_sorted'*cross_stim_response);

figure(10);
subplot(1,3,1);
imagesc(reshape(myfilter_10, [10 10]));
colormap('gray');
title('10D normalization');
axis image;

subplot(1,3,2);
imagesc(reshape(myfilter_20, [10 10]));
colormap('gray');
title('20D normalization');
axis image;

subplot(1,3,3);
imagesc(reshape(myfilter_50, [10 10]));
colormap('gray');
title('50D normalization');
axis image;
```



Now examine all the eigenvectors of the auto correlation function

use a 10x10 subplot to display all the eigenvectors as 10x10 images. What do you see? The eigenvectors are found in the columns of V or U if you use svd and in the columns of v is you use eigen

```
figure(11);
for i=1:100
    subplot(10,10,i);
    imagesc(reshape(v_svd(:,i), 10, 10));
    axis off;
    axis image;
end

figure(12);
for i=1:100
    subplot(10,10,i);
    imagesc(reshape(v_eigen_sorted(:,i), 10, 10));
    axis off;
    axis image;
end
```

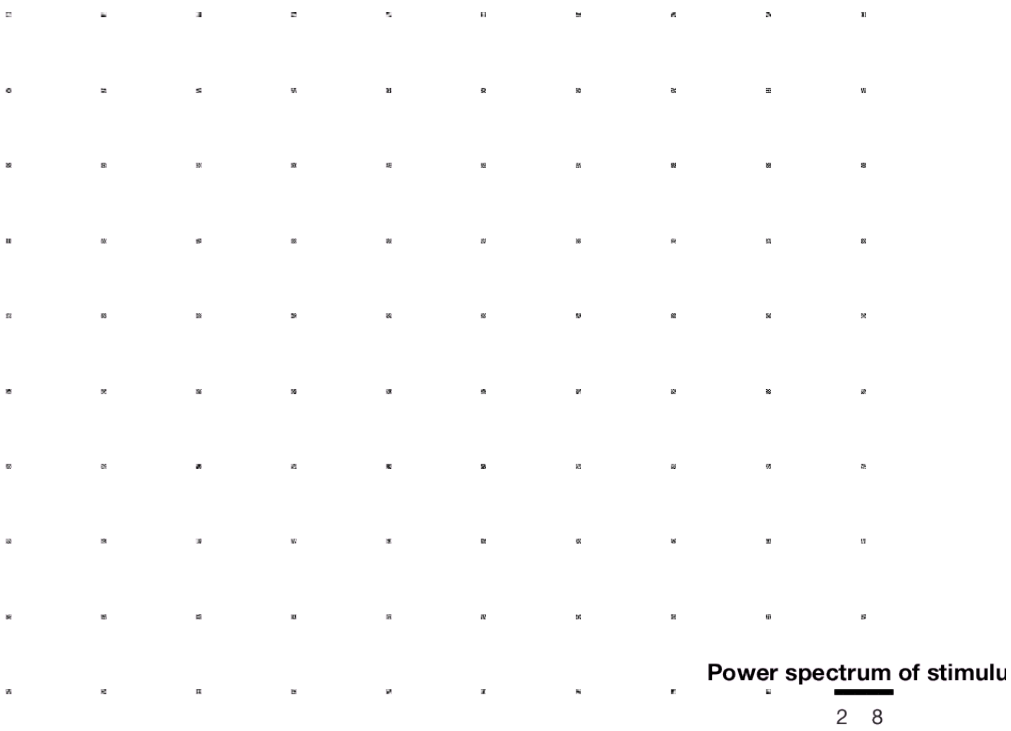


```
% note that the eigenvectors are similar to the fourier tranform. and that
% v_vsd and v_eigen are the same
```

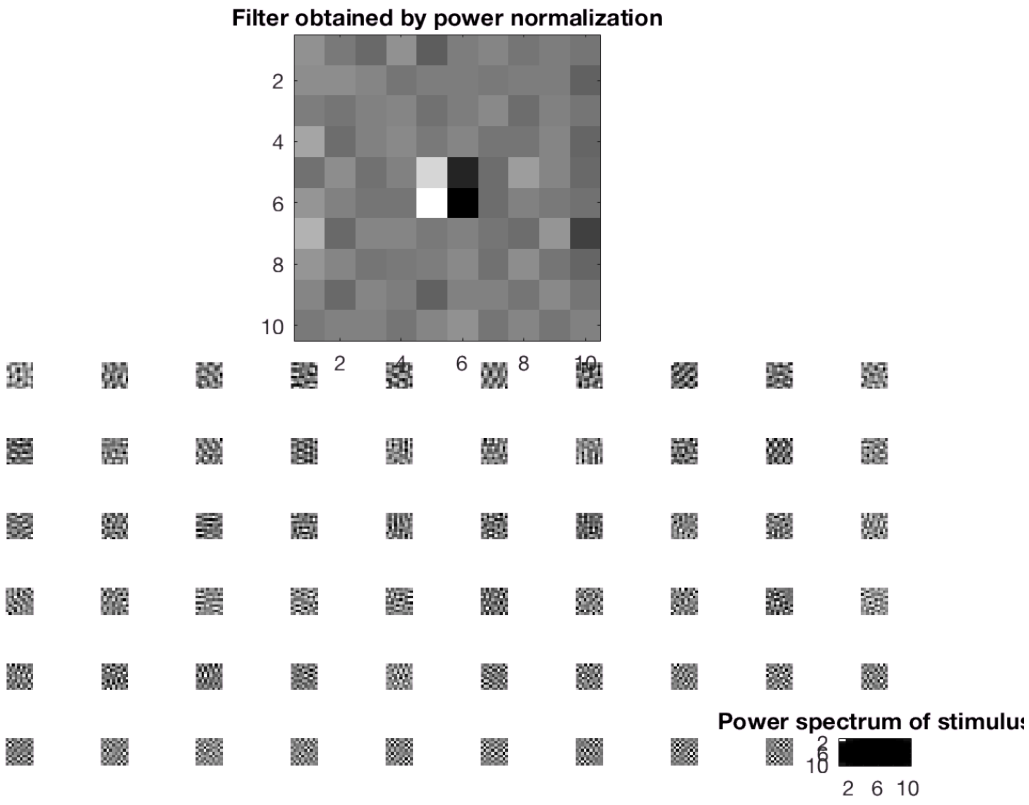
Finally, we are going to repeat the normalization in the Fourier domain.

First calculate the Fourier Transform of the cross-correlation and auto correlation function. You should do this calculation in the 10x10 space and for the auto correlation, you can either reshape and average the 100x100 auto-correlation function or recalculate the power spectrum of the image. You can then divide the cross-correlation by the auto-correlation in the Fourier domain. Finally take the inverse FFT to get back the filter. Did it give exactly the same result?

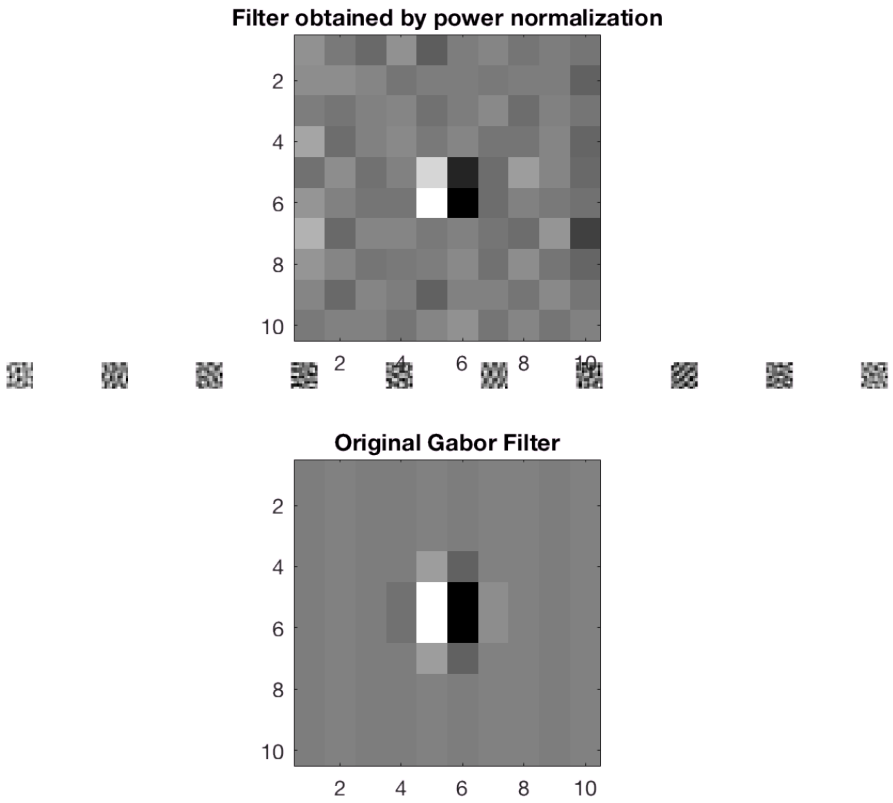
```
stim_pow = zeros(size(squeeze(rawStim(:,:,1))));

for it=1:tlength;
        stim_fft = fft2(rawStim(:, :, it));
        stim_pow = stim_pow + stim_fft.*conj(stim_fft);
end
stim_pow = stim_pow./tlength;

figure (13);
imagesc(stim_pow);
title('Power spectrum of stimulus');
```




```
% Take the fft of the cross-correlation
stim_response_fft = fft2(reshape(cross_stim_response,[10 10]));

% Normalize
filter_fft = stim_response_fft./stim_pow;
filter_from_fft = ifft2(filter_fft);

figure(14);
subplot(2,1,1);
imagesc(filter_from_fft);
colormap('gray');
title('Filter obtained by power normalization');
axis image;
```




```
subplot(2,1,2);
imagesc(gabor);
colormap('gray');
title('Original Gabor Filter');
axis image;
```

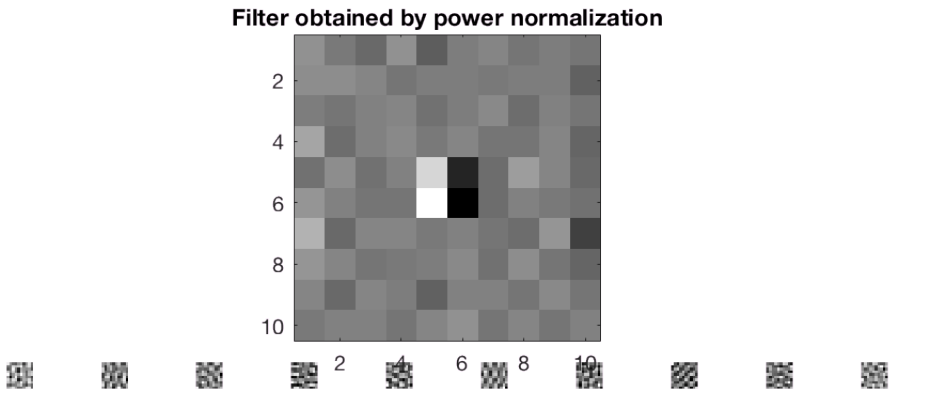


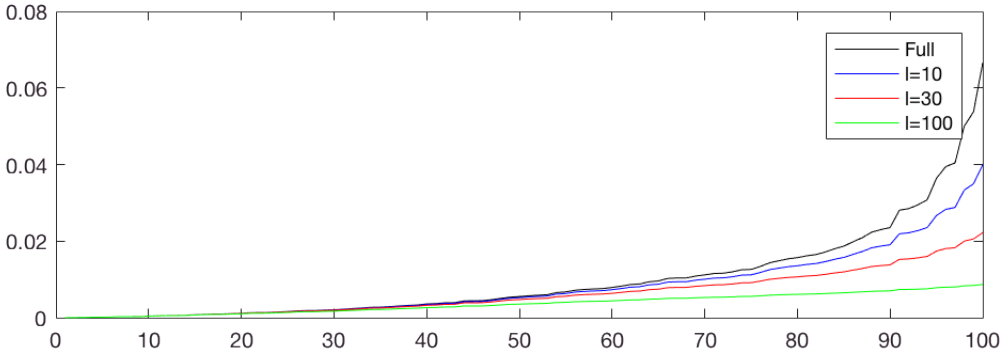

```
% The result is similar but not exactly the same
```

Now we are going to repeat the eigenvalue fit using a ridge parameter l=lambda

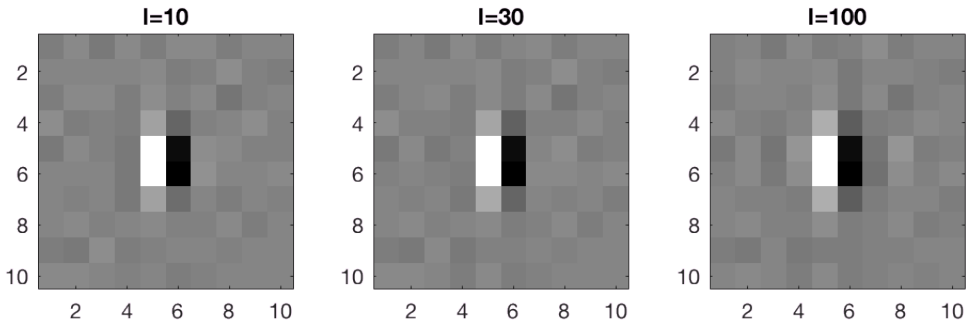
You can just copy the code from above and modify.

```
% Let's try l=10, 30 and 100
l = [10 30 100];

for il=1:length(l);
    dinv_l{il} = diag(1./(ds + l(il)));
    myfilter_l{il} = v_eigen_sorted*dinv_l{il}*(v_eigen_sorted'*cross_stim_response);
end

figure(15);
plot(1./ds, 'k');
hold on;
plot(diag(dinv_l{1}), 'b');
plot(diag(dinv_l{2}), 'r');
plot(diag(dinv_l{3}), 'g');
hold off;
legend('Full',sprintf('l=%d',l(1)), sprintf('l=%d',l(2)), sprintf('l=%d',l(3)));
```






```
figure(16);
for il=1:3
    subplot(1,3,il);
    imagesc(reshape(myfilter_l{il}, [10 10]));
    colormap('gray');
    title(sprintf('l=%d', l(il)));
    axis image;
end
```



We have calculated many linear filters. Which one is the best? Use the part of the

stimulus that we did not use, compare the predictions of the ridge regression filters and the full inversion filter.

```
testStim = single(mov(1:10,1:10,tlength+1:end));
testStim = testStim - meanStim;
testlength = size(testStim, 3);

testStim_vector = reshape(testStim, [10*10 testlength]);
testResp = dotdelay(gabor_vector, testStim_vector);
resp_pow = var(testResp);
testResp_noise = testResp + sqrt(resp_pow/SNR)*(rand(testlength,1)-0.5);

for il=1:length(l)
    predResp{il} = dotdelay(myfilter_l{il}, testStim_vector);
    rpred{il} = corrcoef(predResp{il}, testResp_noise);
end

predRespfull = dotdelay(myfilter, testStim_vector);
rpredfull = corrcoef(predRespfull, testResp_noise);

rpredmax = corrcoef(testResp, testResp_noise);

figure(17);
hb = bar([rpredfull(2)./rpredmax(2), rpred{1}(2)./rpredmax(2), rpred{2}(2)./rpredmax(2), rpred
ha = gca;
set(ha, 'XTickLabel', {'Full'; sprintf('l=%d', l(1)); sprintf('l=%d', l(2)); sprintf('l=%d', l
f17_axis = axis();
```

```
f17_axis(3) = 0.95;
f17_axis(4) = 1.0;
axis(f17_axis);
grid on;

% Diplay best prediction
figure(18);
plot(testResp, 'k');
hold on;
plot(predResp{2}, 'r');
legend('Response', 'Prediction');
hold off;
```