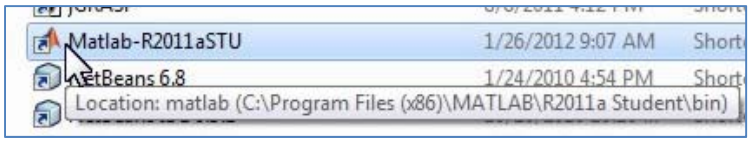
Matlab for CS6320 Beginners

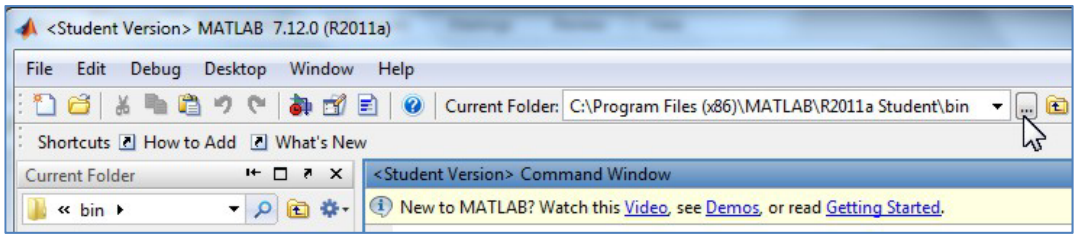
Basics:

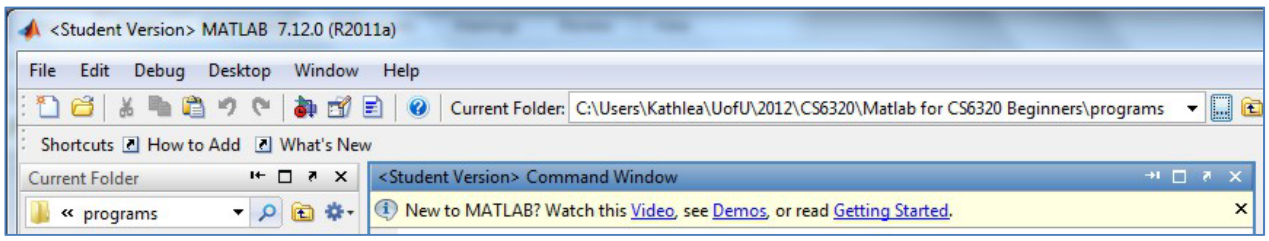
- Starting Matlab
  - CADE Lab remote access

  - Student version on your own computer

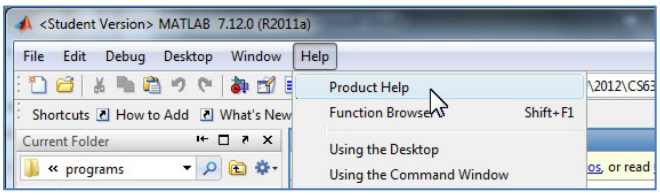


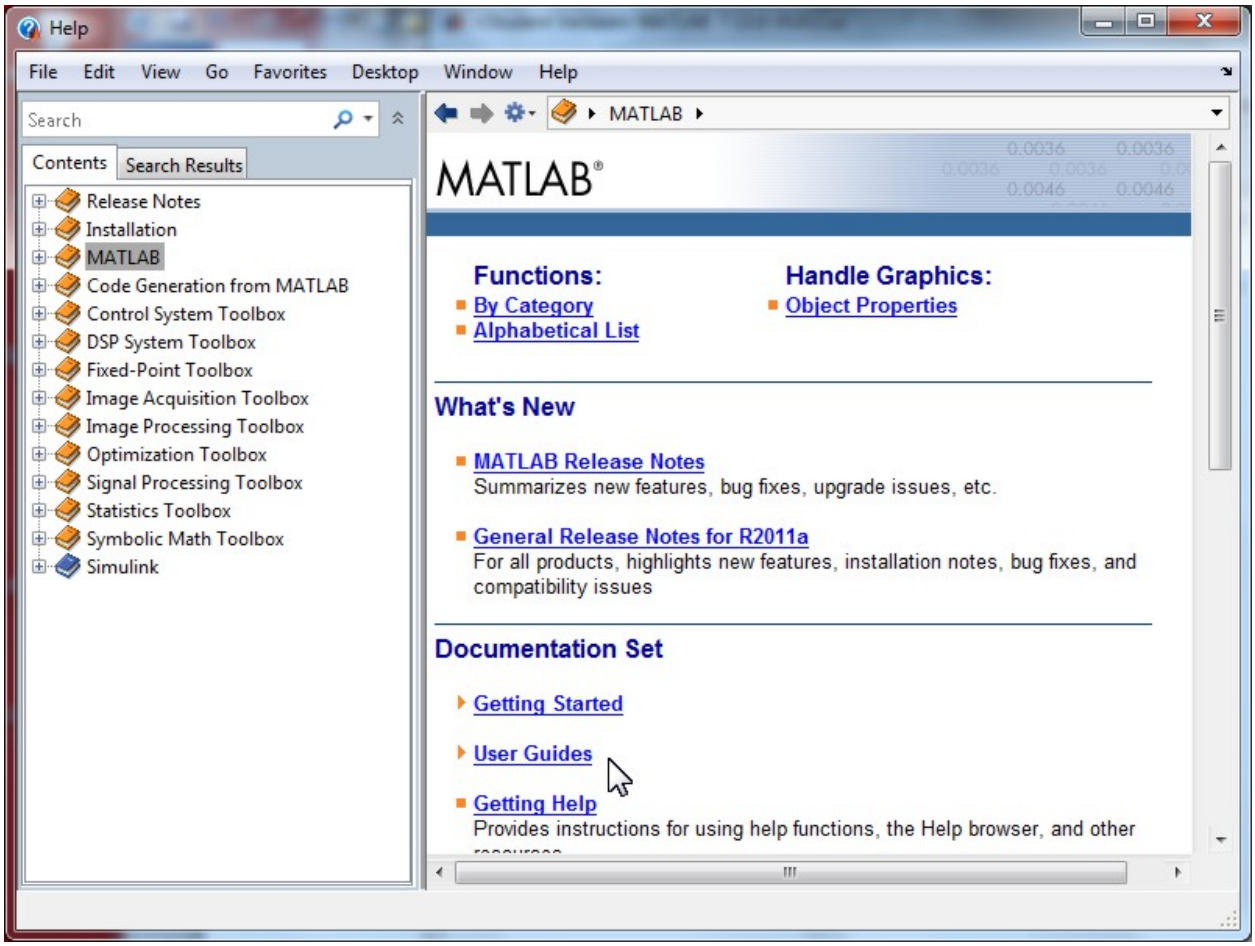
- Change the **Current Folder** to the directory where your programs, images, etc. will be stored





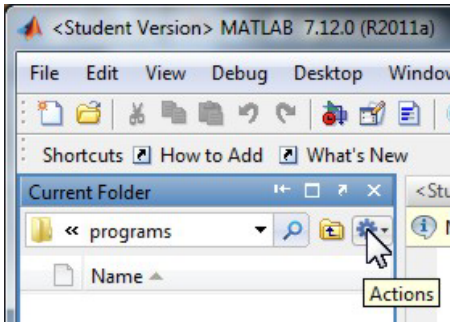
- Getting help from Matlab(primers, tutorials, online help)

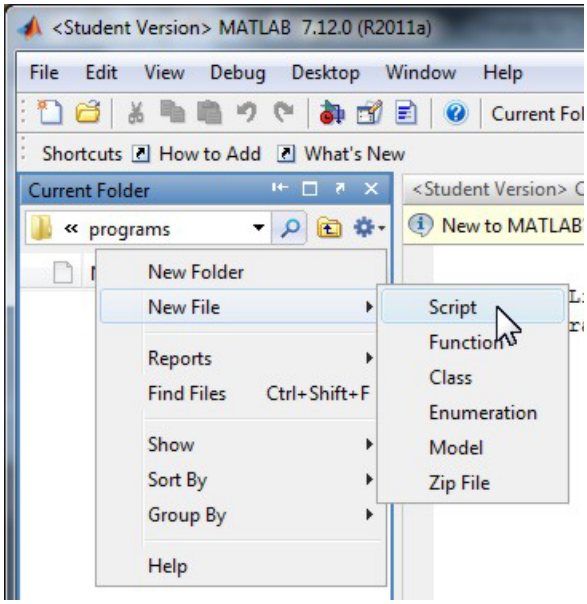




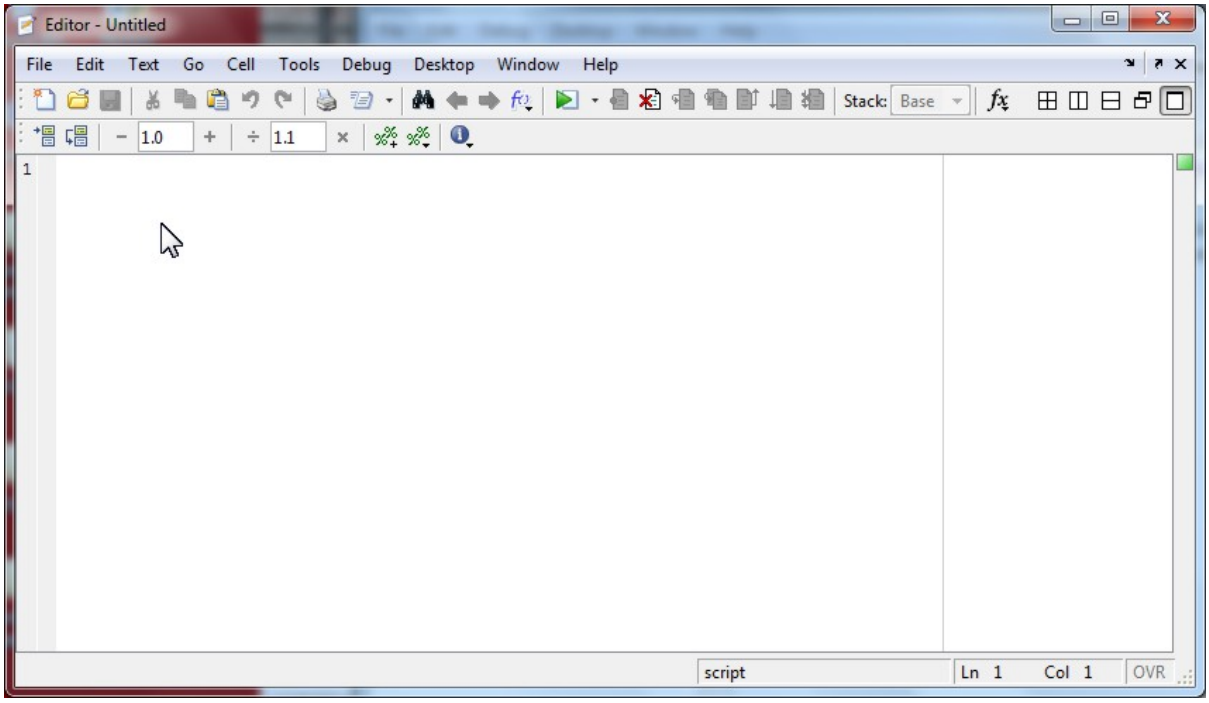
  - If you are a Matlab beginner you can click on **Getting Started** for general help with how Matlab works or click on **User's Guide | Programming fundamentals | Syntax Basics** for basic information such as how variables are created and initialized in Matlab.
  - An excellent description of Matlab expressions can be found in **Getting Started | Matrices and Arrays | Expressions**. It points out the fact that Matlab stands for "Matrix Laboratory". **Whenever possible use a matrix expression** instead of a `for` loop to make matrix calculations. These expressions will execute much faster than nested `for` loops, because Matlab is optimized for manipulating matrices.

- Creating, writing and running programs (m-files)
  - You can run commands in the **Command Window** to try out how they work
  - For automatically running commands, create an m-file. This is the "source code" for MatLab programs. m-files are interpreted programs, called scripts, and are not compiled before running. Remember: Matlab is a "Computational Program", not your usual programming language.
  - m-files can be created in two ways: type `edit` in the **Command window** or click on the Actions icon in the **Current Folder** window and select **New File | Script**:

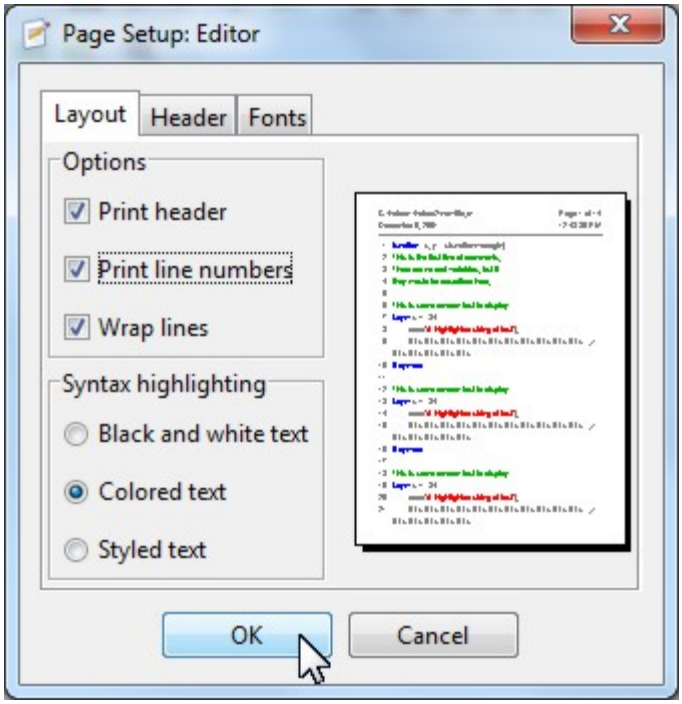




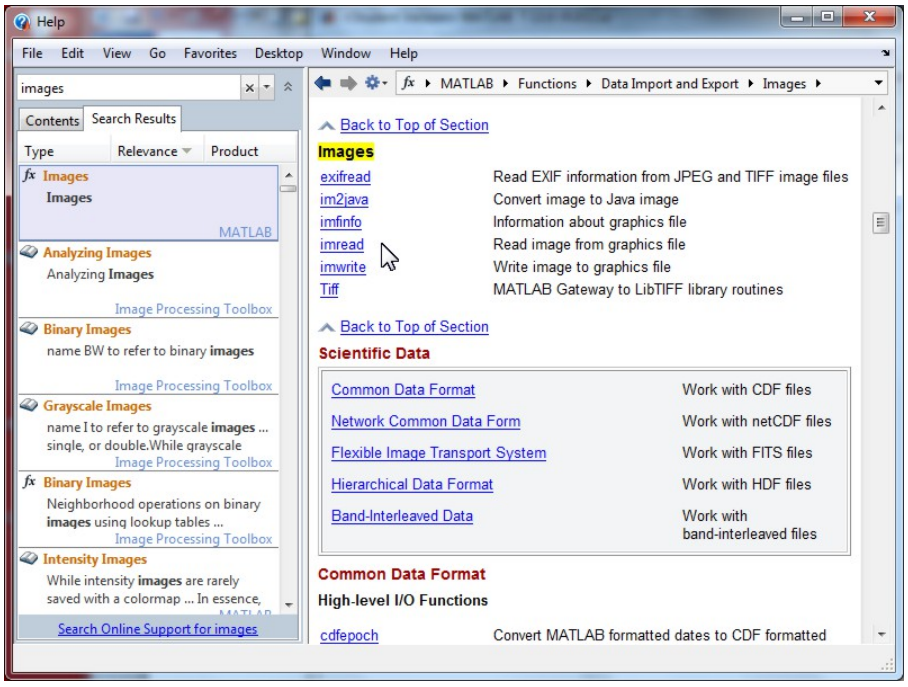
  - The **Editor - Untitled** window will pop up (when you save it you can give it a name):



- If you want to print your script, you can set the page layout to print line numbers for readability

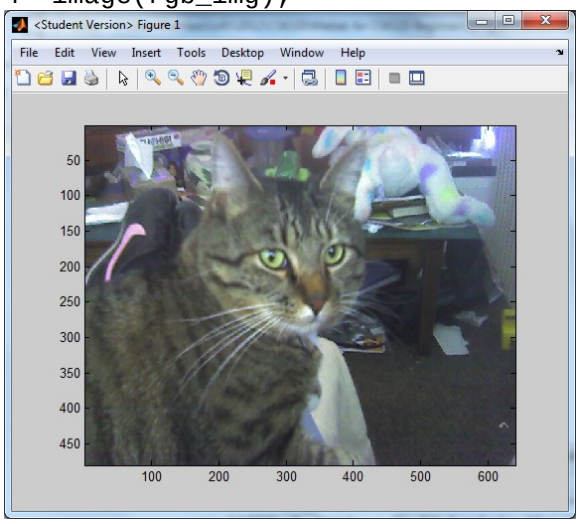


- Reading and writing images (matrices of pixel data)
  - type images in Search box and click on first result labeled as a basic MATLAB feature available without needing the Image Processing Toolbox



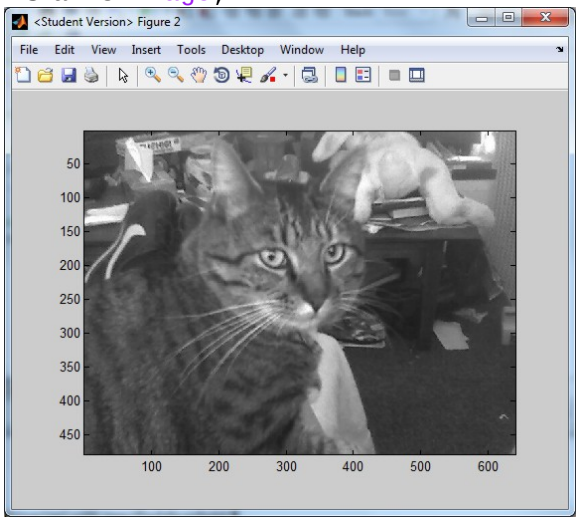
- reading images and displaying them (note the use of the semicolon to suppress command window echo). These are 3D matrices where each pixel is a 3-element vector:

```
clear all; close all; clc;
%load an image and display it
rgb_img = imread('Photo_062011_002.jpg');
image(rgb_img);
```



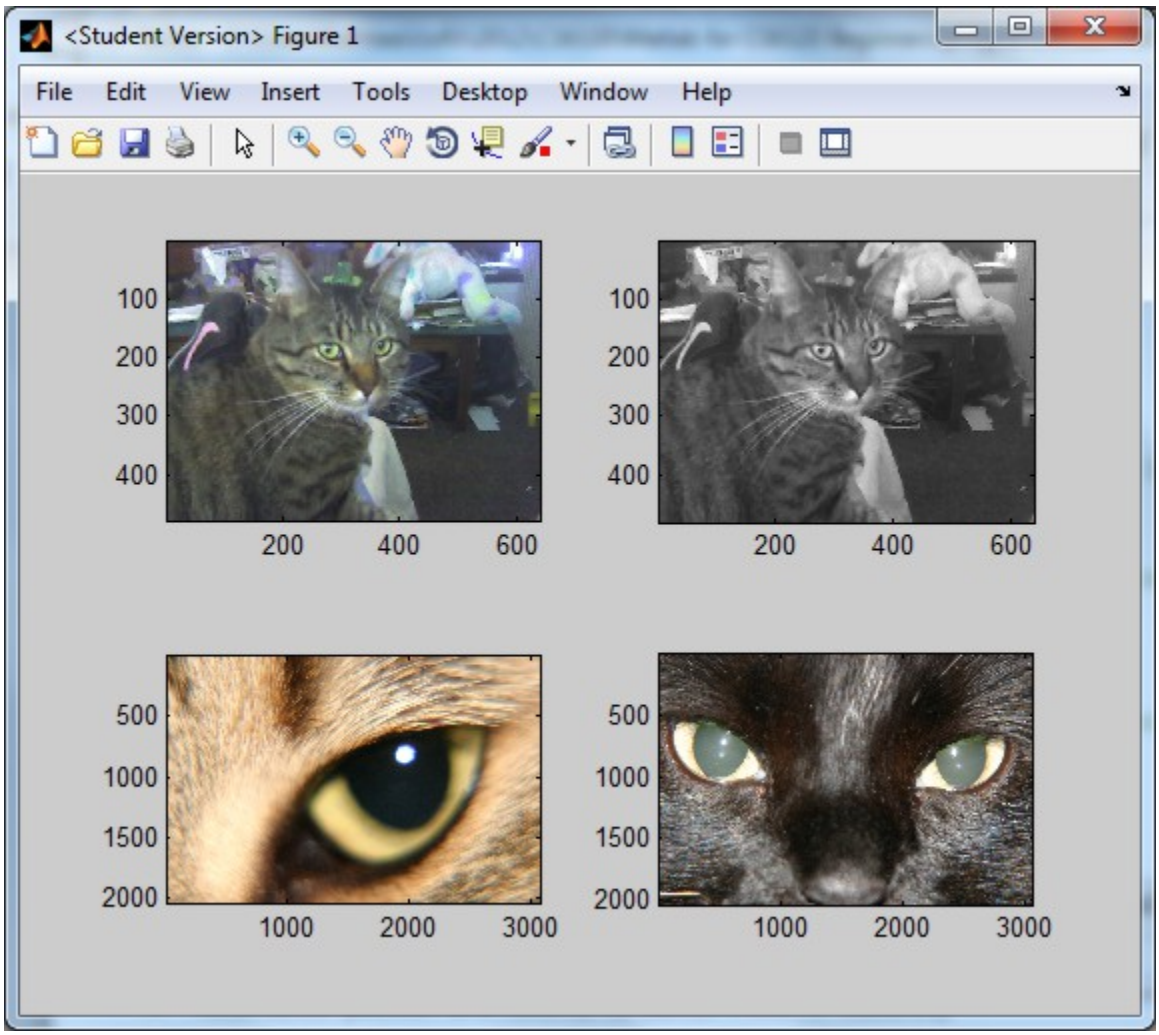
- Convert color to gray scale
  - Find help: User's Guide | Graphics | Displaying Bit-mapped Images | Working with 8-Bit and 16-Bit Images | Converting an 8-Bit RGB Image to Grayscale
  - Example:

```
clear all; close all; clc;
%load an image and display it in figure 1
rgb_img = imread('Photo_062011_002.jpg');
image(rgb_img);
%fit plot box tightly around the image data
axis image;
%Change image to grayscale 2D matrix
I = .2989*rgb_img(:,:,1)...
+.5870*rgb_img(:,:,2)...
+.1140*rgb_img(:,:,3);
%display grayscaled image in figure 2 with gray(256) colormap
figure; colormap(gray(256)); image(I);
axis image;
```



- Display images(multiple)
  - figure, image, and subplot

```
clear all; close all; clc;
%load an image and display it in first row, 1st column, figure 1
im1 = imread('Photo_062011_002.jpg');
subplot(2,2,1);image(im1);
%fit plot box tightly around the image data
axis image;
%Change image to grayscale 2D image
I = .2989*im1(:,:,1)...
+.5870*im1(:,:,2)...
+.1140*im1(:,:,3);
%display grayscaled image in 1st row, 2nd column, figure 1
subplot(2,2,2); colormap(gray(256)); image(I);
axis image;
%load another image and display it in second row, 1st column figure 1
im2 = imread('IMG_1766.jpg');
subplot(2,2,3); image(im2);
axis image;
%load 3rd color image and display it in second row, 2nd column figure 1
im3 = imread('IMG_1768.jpg');
subplot(2,2,4); image(im3);
axis image;
```



- writing images:

```
clear all; close all; clc;
%load an image and display it in figure 1
rgb_img = imread('Photo_062011_002.jpg');
image(rgb_img);
%fit plot box tightly around the image data
axis image;
%Change image to grayscale 2D matrix; note elipsis (...)
I = .2989*rgb_img(:,:,1)...
+.5870*rgb_img(:,:,2)...
+.1140*rgb_img(:,:,3);
%display grayscaled image in figure 2 with gray(256) colormap
figure; colormap(gray(256)); image(I);
axis image;
%write grayscaled image to new file
imwrite(I,gray(256),'grayStarbuck.jpg','jpg');
```

File: grayStarbuck.jpg

- Data types(discrete, conversion to float)
  - color images are 3D matrices, each pixel being a 3-element vector with integer data types (uint8 or uint16)
  - images converted to grayscale are 2D matrices, each pixel is an integer which indexes to a grayscale color map when it's displayed
  - conversion of a grayscale image to floating-point value (single or double):

```
clear all; close all; clc;
%load an image and display it in figure 1
rgb_img = imread('Photo_062011_002.jpg');
image(rgb_img);
%fit plot box tightly around the image data
axis image;
%Change image to grayscale 2D matrix; note elipsis (...)
I = .2989*rgb_img(:,:,1)...
 +.5870*rgb_img(:,:,2)...
 +.1140*rgb_img(:,:,3);
%display grayscaled image in figure 2 with gray(256) colormap
figure; colormap(gray(256)); image(I);
axis image;
whos I
%convert grayscale to single in [0,1) range
S = single(I)/255;
whos S

```

Command Window

```
  Name        Size                Bytes  Class    Attributes

  I         480x640             307200  uint8

  Name        Size                Bytes  Class    Attributes

  S         480x640            1228800  single
```

- convert from double in [0 1] range to uint8 or uint16

```
1  clear all; close all; clc;
2  %create random matrix with values in range [0 1]
3  D01 = rand(5)
4  whos D01
5  %convert doubles in range [0 1] to uint16
6  u16 = uint16(round(65535*D01))
7  whos u16
```

```
D01 =

    0.1622    0.6020    0.4505    0.8258    0.1067
    0.7943    0.2630    0.0838    0.5383    0.9619
    0.3112    0.6541    0.2290    0.9961    0.0046
    0.5285    0.6892    0.9133    0.0782    0.7749
    0.1656    0.7482    0.1524    0.4427    0.8173

  Name      Size            Bytes  Class     Attributes

  D01       5x5               200  double


u16 =

  10629  39451  29526  54120   6989
  52053  17234   5493  35280  63038
  20395  42865  15006  65282    304
  34637  45168  59856   5123  50784
  10856  49030   9986  29011  53562

  Name      Size            Bytes  Class     Attributes

  u16       5x5                50  uint16
```

```
1  clear all; close all; clc;
2  %create random matrix with values in range [0 1]
3  D01 = rand(5)
4  whos D01
5  %convert doubles in range [0 1] to uint16
6  u8 = uint8(round(255*D01))
7  whos u8
```

```
D01 =

    0.8687    0.4314    0.1361    0.8530    0.0760
    0.0844    0.9106    0.8693    0.6221    0.2399
    0.3998    0.1818    0.5797    0.3510    0.1233
    0.2599    0.2638    0.5499    0.5132    0.1839
    0.8001    0.1455    0.1450    0.4018    0.2400

  Name      Size            Bytes  Class     Attributes

  D01       5x5               200  double


u8 =

  222  110   35  218   19
   22  232  222  159   61
  102   46  148   89   31
   66   67  140  131   47
  204   37   37  102   61

  Name      Size            Bytes  Class    Attributes

  u8        5x5                25  uint8
```

- convert from uint8 or uint16 to double in [0 1] range

```
clear all; close all; clc;
%create random matrix with values in range [0 1]
D01 = rand(5);
%convert doubles in range [0 1] to uint16
u8 = uint8(round(255*D01))
whos u8
%convert uint16 to doubles in range [0 1]
D01 = double(u8)/255
whos D01
```

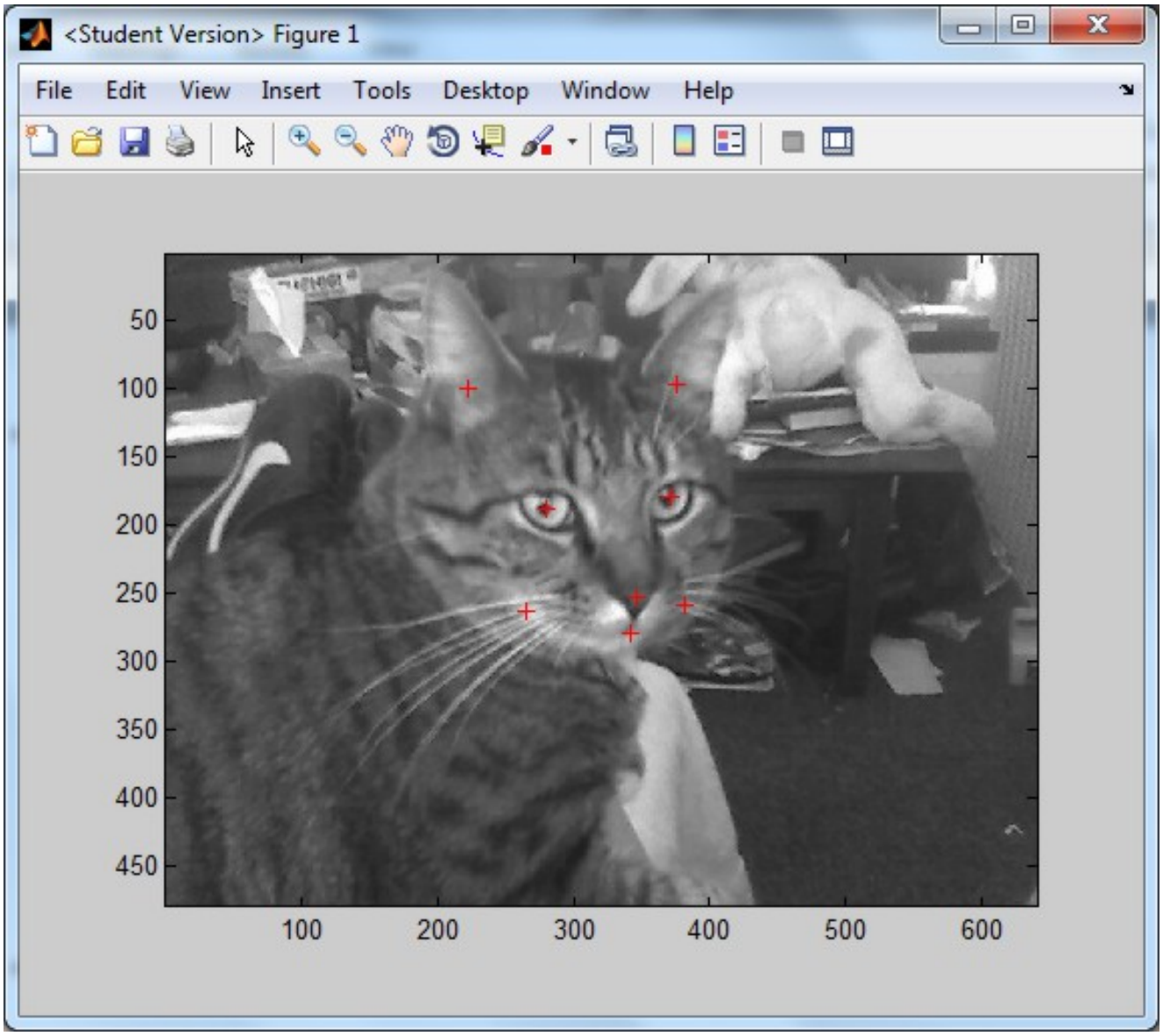
```
u8 =

  106  125  199   34   60
   13   86   99  240   90
  230  230   62  244  209
  241   94  103  147    4
  125   28   25   15   11

  Name      Size            Bytes  Class    Attributes

  u8        5x5                25  uint8


D01 =

    0.4157    0.4902    0.7804    0.1333    0.2353
    0.0510    0.3373    0.3882    0.9412    0.3529
    0.9020    0.9020    0.2431    0.9569    0.8196
    0.9451    0.3686    0.4039    0.5765    0.0157
    0.4902    0.1098    0.0980    0.0588    0.0431

  Name      Size            Bytes  Class     Attributes

  D01       5x5               200  double
```

```
clear all; close all; clc;
%create random matrix with values in range [0 1]
D01 = rand(5);
%convert doubles in range [0 1] to uint16
u16 = uint16(round(65535*D01))
whos u16
%convert uint16 to doubles in range [0 1]
D01 = double(u16)/65535
whos D01
```

```
u16 =

  11075  35848  12026  60907  20077
  42540  19419  24149  50836  33325
  47953  48803  41000  31902  33473
  42450  12383  51132  28564  53583
  29551  45008   5317  29280  52089

  Name      Size            Bytes  Class     Attributes

  u16       5x5                50  uint16


D01 =

    0.1690    0.5470    0.1835    0.9294    0.3064
    0.6491    0.2963    0.3685    0.7757    0.5085
    0.7317    0.7447    0.6256    0.4868    0.5108
    0.6477    0.1890    0.7802    0.4359    0.8176
    0.4509    0.6868    0.0811    0.4468    0.7948

  Name      Size            Bytes  Class     Attributes

  D01       5x5               200  double
```

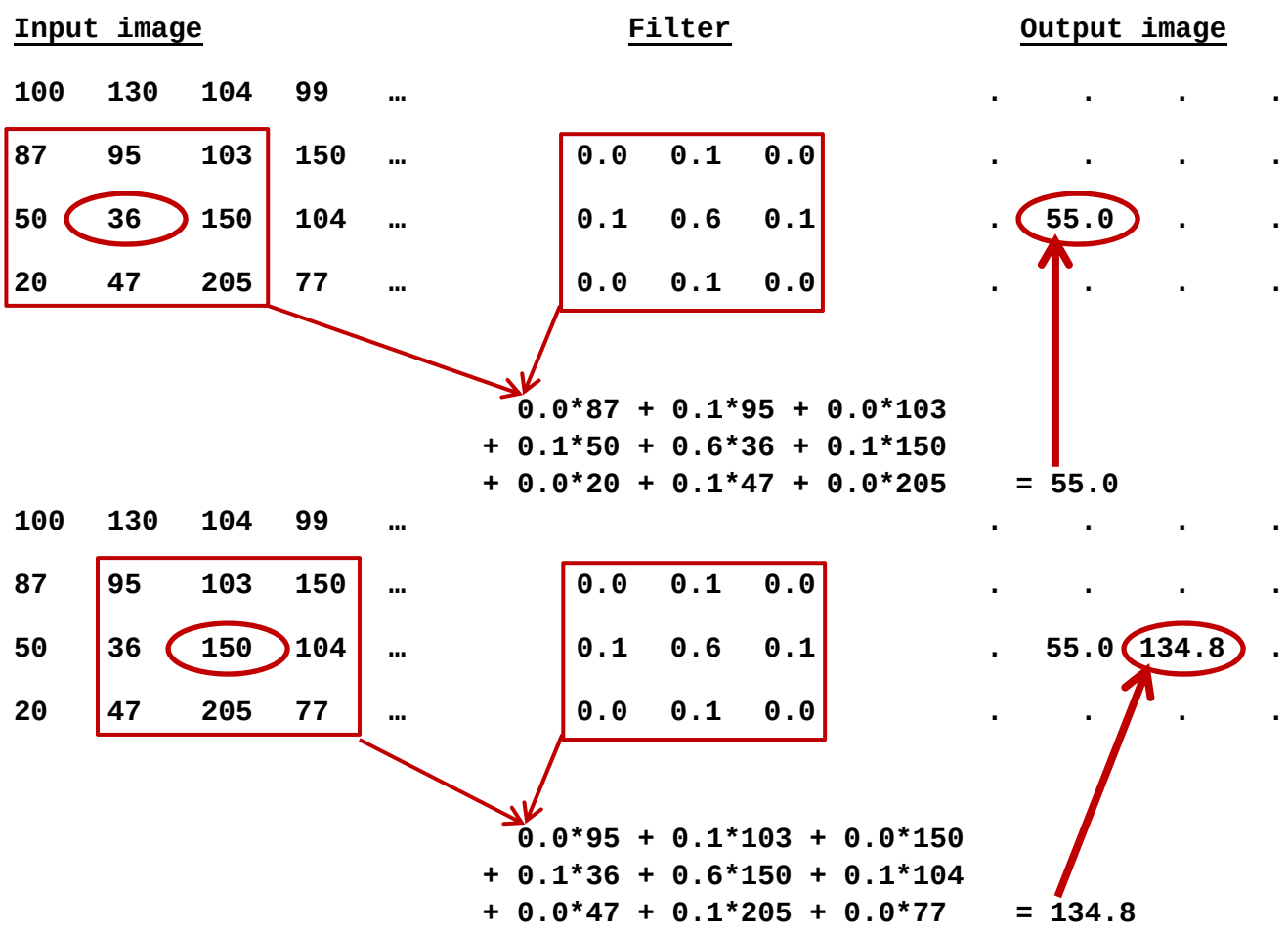
- read multiple cursor positions and overlay gray scale image with plotted cursor positions

```
clear all; close all; clc;
rgb_img = imread('Photo_062011_002.jpg');
I = .2989*rgb_img(:,:,1)...
+.5870*rgb_img(:,:,2)...
+.1140*rgb_img(:,:,3);
%display grayscaled image in figure 2 with gray(256) colormap
fig = figure; colormap(gray(256)); image(I);
axis image;
[x,y] = ginput;
xInt = uint16(round(x))';
yInt = uint16(round(y))';
hold 'on'; plot(xInt,yInt,'+r');
hold 'off';
```



Filtering:

- Neighborhood filter:
  - Convolution: create a mask, move the mask over an image calculating a new intensity for each pixel based on the intensity of its neighbor pixel intensities. The mask will center over the pixel

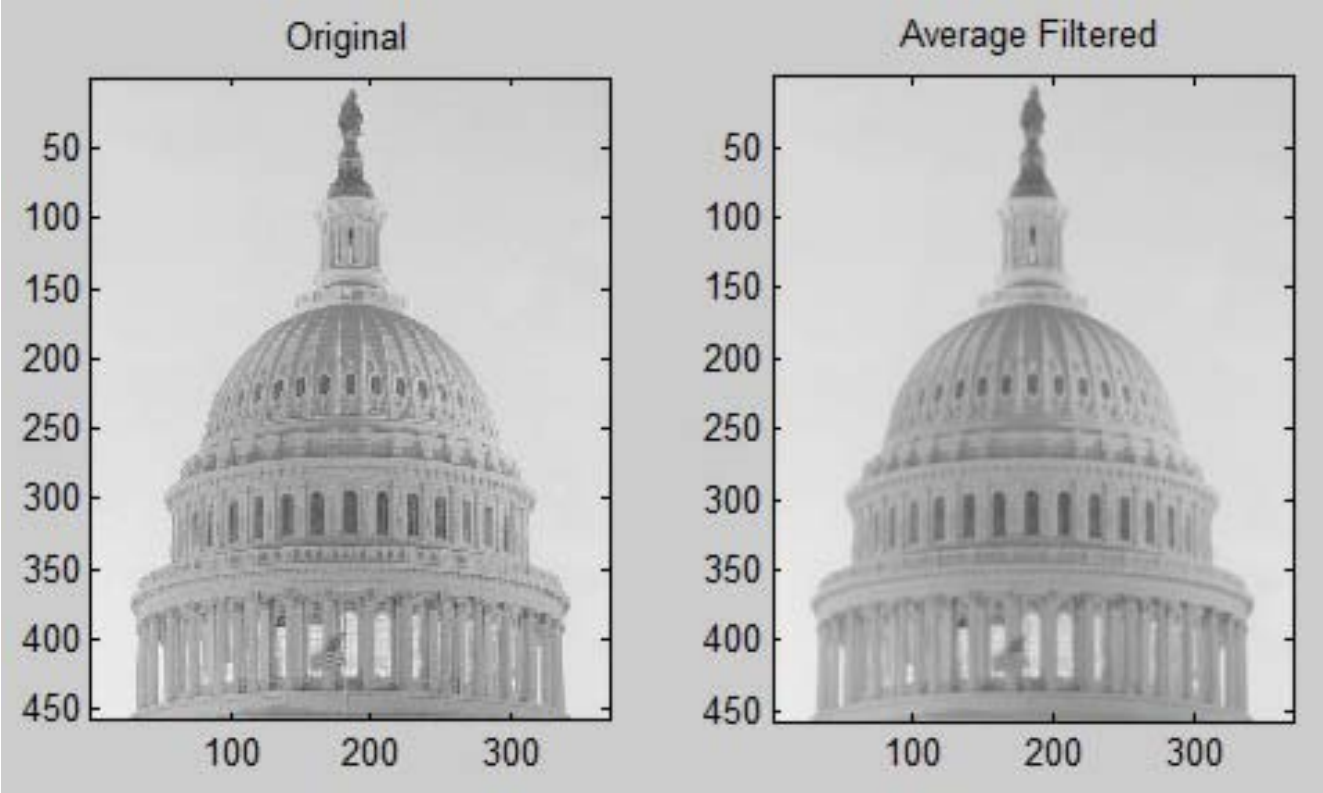


- A filter function that takes a grayscale image and mask and returns a uint8 grayscale image:

```
%Function: signedFilterImage
%parameters:    I = 2D array containing image data
%               maskSize = size of the Square mask applied
%               mask = Actual mask array of size maskSize x maskSize.
%               Account for the normalisation considering the division
%               factor.
%returns:   signedFilter = 2D array containing filtered image data. Please
%                          note that no border padding is done; hence the
%                          border pixels are left unfiltered
function [signedFilter] = signedFilterImage(I, maskSize, mask)

Idouble = double(I);
maskdouble = double(mask);

tempImage = double(I);

for i = ((maskSize-1)/2)+1 : (size(I,1)-((maskSize-1)/2))%row of image pixel
    for j = ((maskSize-1)/2)+1 : (size(I,2)-((maskSize-1)/2))%col of image pixel
        subImage = Idouble(i-((maskSize-1)/2) : i+((maskSize-1)/2), j-((maskSize-1)/2) : j+((maskSize-1)/2));
        %subImage is same size as mask; centered around current pixel
        filterResult = subImage .* maskdouble;
        %accumulate sum of products of subImage and mask
        pixelValue = double(0);%accumulator
        for p = 1:size(filterResult,1)%row of mask and subImage
            for q = 1:size(filterResult,2)%col of mask and subImage
                pixelValue = pixelValue + filterResult(p,q);
            end
        end
        %tempImage contains only non-negative values
        tempImage(i,j) = abs(pixelValue);
    end
end

signedFilter = uint8(tempImage);%return filtered image in uint8 format

end
```

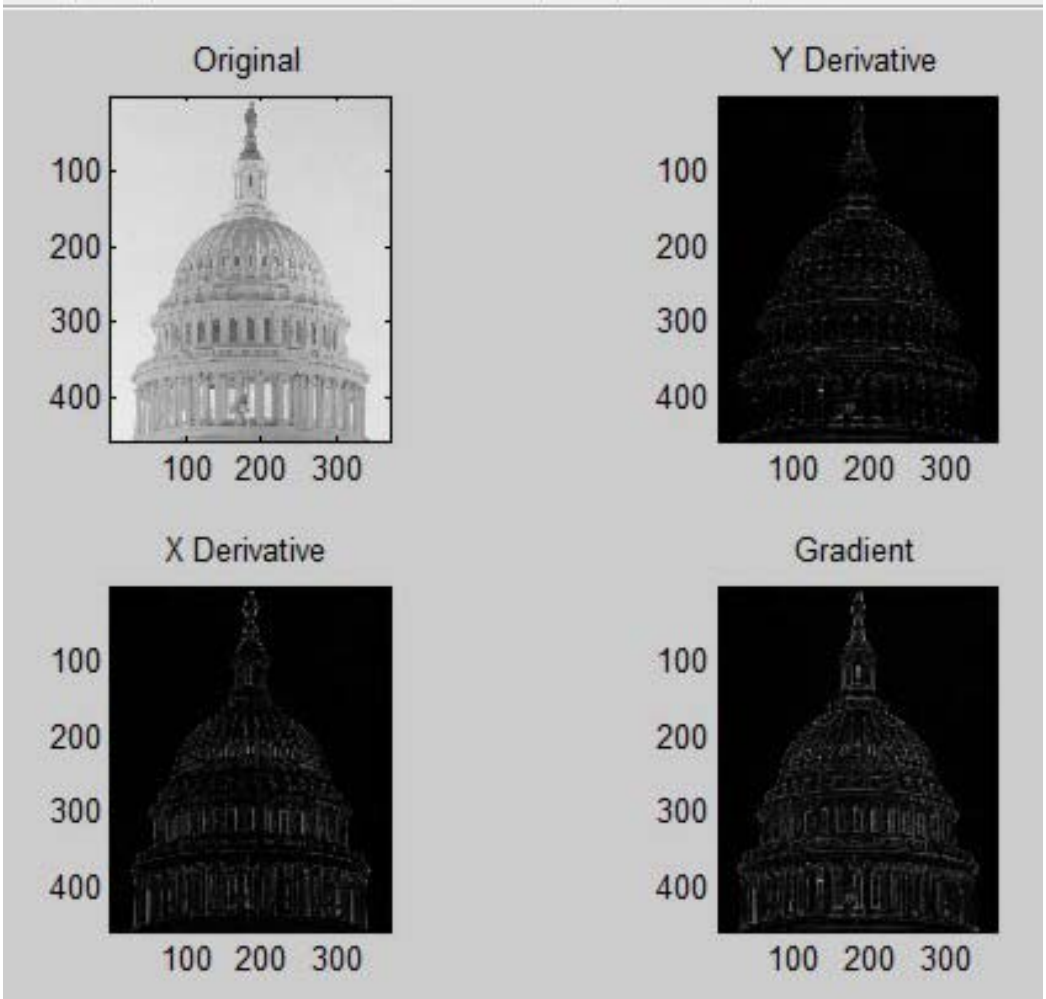
- Averaging (smoothing)

```
clear all; close all; clc;
%load an image and display it in first row, 1st column, figure 1
I = imread('capitol.jpg');
%display grayscaled image in 1st row, 2nd column, figure 1
subplot(1,2,1); colormap(gray(256)); image(I);
axis image; title('Original');
%averaging filter convolution with grayscale image
oneNinth = 1.0/9.0;
mask = [oneNinth,oneNinth,oneNinth;
oneNinth,oneNinth,oneNinth;
oneNinth,oneNinth,oneNinth];
av_filter_img = signedfilterImage(I,3,mask);
subplot(1,2,2); image(av_filter_img);
axis image; title('Average Filtered');
```



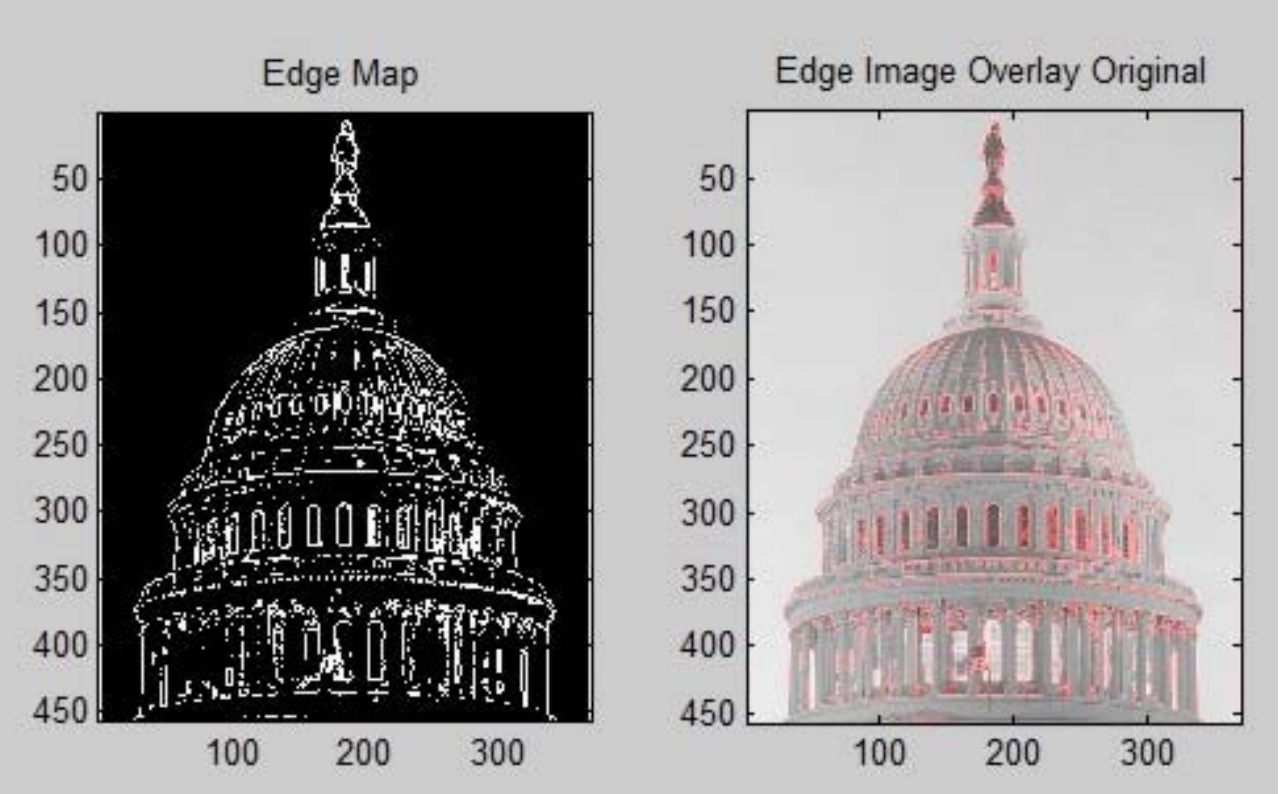
- Edge detection(x and y derivatives, magnitude -> edge image)

```
clear all; close all; clc;
%load an image and display it in first row, 1st column, figure 1
I = imread('capitol.jpg');
%display grayscaled image in 1st row, 2nd column, figure 1
subplot(2,2,1); colormap(gray(256)); image(I);
axis image; title('Original');
%y-derivative filter convolution with grayscale image
mask = [0,1,0;
0,0,0;
0,-1,0];
Iy = signedfilterImage(I,3,mask);
subplot(2,2,2); colormap(gray(256));  image(Iy);
axis image; title('Y Derivative');
%x-derivative filter convolution with grayscale image
mask = [0,0,0;
-1,0,1;
0,0,0];
Ix = signedfilterImage(I,3,mask);
subplot(2,2,3); colormap(gray(256));  image(Ix);
axis image; title('X Derivative');
%compute gradient = square root of Ix^2 + Iy^2
Ig = uint8(sqrt(double(Ix).^2 + double(Iy).^2));
subplot(2,2,4); colormap(gray(256));  image(Ig);
axis image; title('Gradient');
```



- Application: overlay binarized edge image with original

```
clear all; close all; clc;
%load an image and display it in first row, 1st column, figure 1
I = imread('capitol.jpg');
%display grayscaled image in 1st row, 2nd column, figure 1
%y-derivative filter convolution with grayscale image
mask = [0,1,0;
0,0,0;
0,-1,0];
Iy = signedfilterImage(I,3,mask);
%x-derivative filter convolution with grayscale image
mask = [0,0,0;
-1,0,1;
0,0,0];
Ix = signedfilterImage(I,3,mask);
%compute gradient = square root of Ix^2 + Iy^2
Ig = uint8(sqrt(double(Ix).^2 + double(Iy).^2));
%binarize gradient edge map
Ig(Ig>35) = 255;
Ig(Ig<=35) = 0;
subplot(1,2,1); colormap(gray(256)); image(Ig);
axis image; title('Edge Map');
imgOverlay(:,:,1) = I+Ig;
imgOverlay(:,:,2) = I;
imgOverlay(:,:,3) = I;
subplot(1,2,2); image(imgOverlay);
axis image; title('Edge Image Overlay Original');
```



- A filter function that takes a grayscale image and mask and returns a double-format image:

```
%Function: signedFilterImage
%parameters:    I = 2D array containing image data
%               maskSize = size of the Square mask applied
%               mask = Actual mask array of size maskSize x maskSize.
%               Account for the normalisation considering the division
%               factor.
%returns:   signedFilter = 2D array containing filtered image data. Please
%                          note that no border padding is done; hence the
%                          border pixels are left unfiltered
function [signedFilter] = signedFilterImage01(I, maskSize, mask)

Idouble = double(I);
maskdouble = double(mask);

tempImage = double(I);

for i = ((maskSize-1)/2)+1 : (size(I,1)-((maskSize-1)/2))%row of image pixel
    for j = ((maskSize-1)/2)+1 : (size(I,2)-((maskSize-1)/2))%col of image pixel
        subImage = Idouble(i-((maskSize-1)/2) : i+((maskSize-1)/2), j-((maskSize-1)/2) : j+((maskSize-1)/2));
        %subImage is same size as mask; centered around current pixel
        filterResult = subImage .* maskdouble;
        %accumulate sum of products of subImage and mask
        pixelValue = double(0);%accumulator
        for p = 1:size(filterResult,1)%row of mask and subImage
            for q = 1:size(filterResult,2)%col of mask and subImage
                pixelValue = pixelValue + filterResult(p,q);
            end
        end
        %tempImage contains only non-negative values
        tempImage(i,j) = abs(pixelValue);
    end
end

signedFilter = tempImage;%return filtered image in double format

end
```

Harris Corner Detection Algorithm

Hint: before making the following calculations, **convert the image to double in the range [0 1]** and use a version of signedFilterImage that takes a grayscale image and **returns the output image as a double**

1 calculate I_x , the x-derivative, a convolution
2 calculate I_y , the y-derivative, a convolution
3 calculate I_x^2 , a multiplication
4 calculate I_y^2 , a multiplication
5 calculate $I_x I_y$, a multiplication
6 a = smooth I_x^2 , a convolution
7 c = smooth I_y^2 , a convolution
8 b = smooth $I_x I_y$, a convolution
9 Response = $det(M) - \alpha\ trace(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$
10 threshold the Response image to get the corners
11 overlay the binarized Response image over the original

Response can be calculated without finding the eigenvalues, λ_1 and λ_2. The formula then becomes:

$Response = \det(M) - \alpha\ trace(M)^2 = (ac - b^2) - \alpha(a + c)^2$

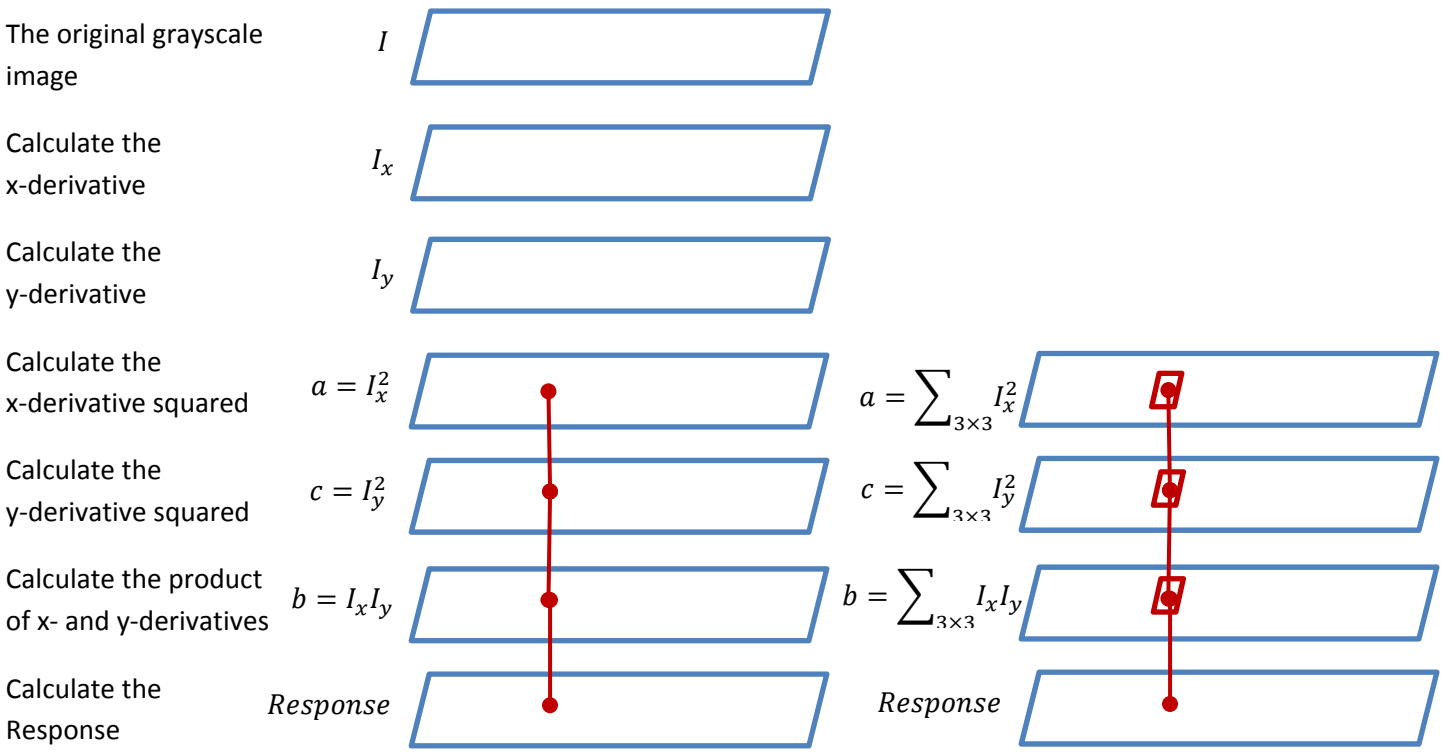
where

$a = I_x^2\ or\ \sum_{3\times3} I_x^2$ the latter being the smoothed version of the former

$b = I_x I_y\ or\ \sum_{3\times3} I_x I_y$ the latter being the smoothed version of the former

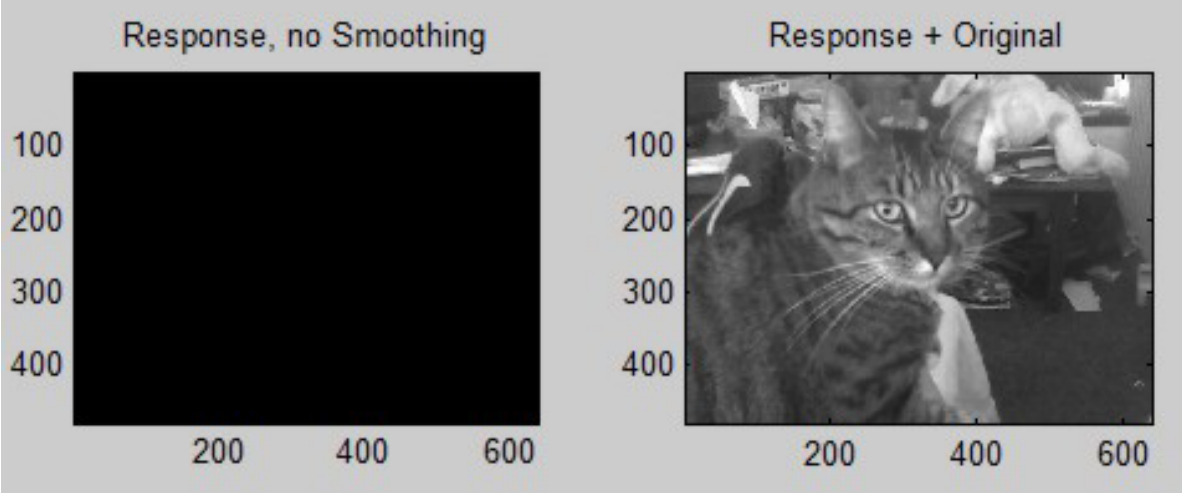
$c = I_y^2\ or\ \sum_{3\times3} I_y^2$ the latter being the smoothed version of the former

In diagramed form:

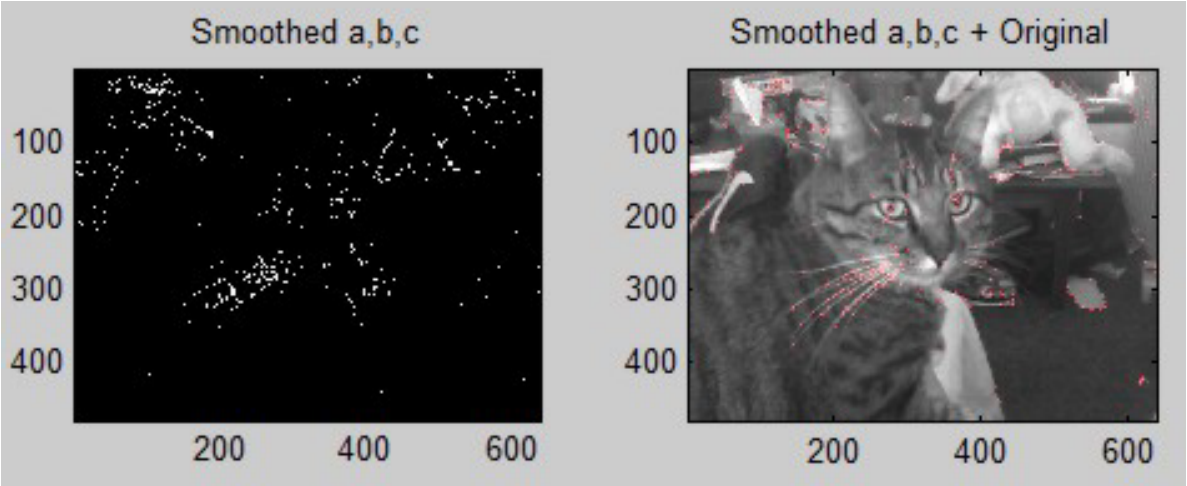


Sample runs of the Harris Corner Detection Algorithm

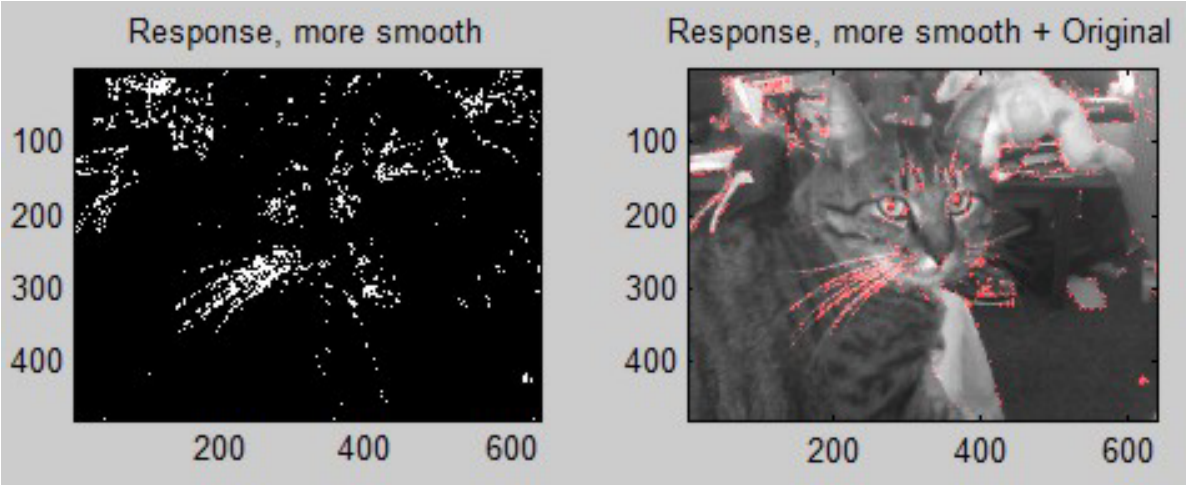
With no smoothing of a, b, and c (note that it does not work, since a*c-b*b = 0):



With smoothing of a, b, and c:



With smoothing of a, b, and c, followed by smoothing of the binarized Response image:



Here's the code that produced the sample runs above

```
clear all; close all; clc;
%load an image
rgb_img = imread('Photo_062011_002.jpg');
image(rgb_img);
%fit plot box tightly around the image data
axis image;
%Change image to grayscale 2D matrix; note elipsis (...)
I = .2989*rgb_img(:,:,1)...
 +.5870*rgb_img(:,:,2)...
 +.1140*rgb_img(:,:,3);
%convert to double range [0 1]
Idouble = double(I)/255;
%smooth I
mask = [0.0 0.1 0.0;
    0.1 0.6 0.1;
    0.0 0.1 0.0];
IdoubleS = signedFilterImage01(Idouble,3,mask);
%convert to grayscale smoothed image
IdoubleSImg = uint16(round(65535*IdoubleS));
%y-derivative filter convolution with image
mask = [0,1,0;
    0,0,0;
    0,-1,0];
Iy = signedfilterImage01(IdoubleS,3,mask);
%x-derivative filter convolution with image
mask = [0,0,0;
    -1,0,1;
    0,0,0];
Ix = signedfilterImage01(IdoubleS,3,mask);

%Harris Corner Detection Algorithm
%compute Ix.^2
Ixsqrd = Ix.^2;
IxsqrdImg = uint16(round(65535*Ixsqrd));
figure;
subplot(2,3,1); image(IxsqrdImg); colormap(gray);
axis image; title('Ix Squared');
%smooth Ix.^2
mask = [0.0 0.1 0.0;
    0.1 0.6 0.1;
    0.0 0.1 0.0];
Ixsqrdsmooth = signedFilterImage01(Ixsqrd,3,mask);
IxsqrdsmoothImg = uint16(round(Ixsqrdsmooth*65535));
subplot(2,3,4);  image(IxsqrdsmoothImg); colormap(gray);
axis image; title('Ix Squared Smoothed');

%compute Iy^2
Iysqrd = Iy.^2;
IysqrdImg = uint16(round(65535*Iysqrd));
subplot(2,3,2); image(IysqrdImg); colormap(gray);
axis image; title('Iy Squared');
%smooth Iy.^2
Iysqrdsmooth = signedFilterImage01(Iysqrd,3,mask);
IysqrdsmoothImg = uint16(round(Iysqrdsmooth*65535));
subplot(2,3,5);  image(IysqrdsmoothImg); colormap(gray);
axis image; title('Iy Squared Smoothed');

%compute Ix*Iy
IxIy = Ix.*Iy;
IxIyImg = uint16(round(65535*IxIy));
subplot(2,3,3); image(IxIyImg); colormap(gray);
axis image; title('IxIy');
%smooth IxIy
IxIysmooth = signedFilterImage01(IxIy,3,mask);
IxIysmoothImg = uint16(round(65535*IxIysmooth));
subplot(2,3,6);  image(IxIysmoothImg); colormap(gray);
axis image; title('IxIy Smoothed');

%M = [Ix.^2, Ix.*Iy; Ix.*Iy, Iy.^2] element-wise
a = Ixsqrd;
b = IxIy;
c = Iysqrd;
%Response = det[M] - alpha*(trace[M].^2) = (a.*c - b.^2)-0.05((a + c).^2)
Response1 = (a.*c - b.^2) - 0.04*((a + c).^2);
Response1(Response1>0)=1;
Response1(Response1<=0)=0;
```

```
Response1Img = uint16(round(65535*Response1));
figure;
subplot(1,2,1); image(Response1Img); colormap(gray);
axis image; title('Response, no Smoothing');
imgOverlay1(:,:,1) = IdoubleSImg+Response1Img;
imgOverlay1(:,:,2) = IdoubleSImg;
imgOverlay1(:,:,3) = IdoubleSImg;
subplot(1,2,2); image(imgOverlay1);
axis image; title('Response + Original');
%Response after smoothing Ix.^2, Iy.^2, and Ix.*Iy
asmth = Ixsqrdsmooth;
bsmth = IxIysmooth;
csmth = Iysqrdsmooth;
%Response2 = (asmth.*csmth - bsmth.^2) - 0.04*((asmth + csmth).^2);
Response2 = (asmth.*csmth - bsmth.^2) - 0.04*((asmth + csmth).^2);
Response2(Response2>0.000001)=1;
Response2(Response2<=0.000001)=0;
Response2Img = uint16(round(65535*Response2));
figure;
subplot(1,2,1); image(Response2Img); colormap(gray);
axis image; title('Smoothed a,b,c');
imgOverlay1(:,:,1) = IdoubleSImg+Response2Img;
imgOverlay1(:,:,2) = IdoubleSImg;
imgOverlay1(:,:,3) = IdoubleSImg;
subplot(1,2,2); image(imgOverlay1);
axis image; title('Smoothed a,b,c + Original');
%Response2s = smoothed Response2
Response2s = signedFilterImage01(Response2,3,mask);
Response2s(Response2s>0.000001)=1;
Response2s(Response2s<=0.000001)=0;
Response2sImg = uint16(round(65535*Response2s));
figure;
subplot(1,2,1); image(Response2sImg); colormap(gray);
axis image; title('Response, more smooth');
imgOverlay1(:,:,1) = IdoubleSImg+Response2sImg;
imgOverlay1(:,:,2) = IdoubleSImg;
imgOverlay1(:,:,3) = IdoubleSImg;
subplot(1,2,2); image(imgOverlay1);
axis image; title('Response, more smooth + Original');
```