1. Contours of maximum rate of change in an image can be found by locating directional maxima in the gradient magnitude of the image (or equivalently by finding zero crossings in the Laplacian of the image).
   a. Why it is considered important to convolve the image with a Gaussian before computing the Laplacian?
      This is important for noisy images as the Laplacian is sensitive to noise and texture, by smoothing the image before with a Gaussian you are able to reduce this noise and texture and get better edge detection.
   b. Convolving the image with a Gaussian and then taking the Laplacian, i.e. $\nabla^2(G \ast I)$, is equivalent to taking the Laplacian of the same Gaussian and then convolving that with the image, i.e. $(\nabla^2 G) \ast I$. Why does the second formulation lead to a more efficient implementation?
      The order of the operations matter as one is applying two filters onto a large image whereas the other is combining the two filters before applying it to the image. The latter is more effective as the most costly operations are those that are applied onto the image. Smoothing the image is needed (as stated above) however if you combine the smoothing filter with the laplacian you simply can apply one mask onto the image.
   c. The Laplacian of a Gaussian can be approximated by the difference of two Gaussian kernels with appropriately chosen standard deviations. As a result, show that the computation can be implemented as $(\nabla^2 G) \ast I = G_1 \ast I - G_2 \ast I$.
      The Laplacian of a Gaussian does equal the difference of two Gaussians rather it is an approximation. The shape of a Laplacian and the difference of two Gaussians is best when sigma of in the two Gaussians which we will denote by their subscript are $\sigma_1 = \sqrt{2}\sigma$ and $\sigma_2 = \frac{\sqrt{2}}{\sigma}$.
      \[
      \nabla^2 = \frac{d^2}{dx^2}
      \]
      \[
      g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}
      \]
      \[
      g'(x) = \frac{(\mu - x)}{\sqrt{2\pi}\sigma^3} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} = \frac{(\mu - x)}{\sigma^2} g(x)
      \]
      \[
      g''(x) = \frac{(\mu^2 - \sigma^2 + x^2 - 2\mu x)}{\sqrt{2\pi}\sigma^5} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} = \frac{1}{\sigma^4} g(x) = L o G
      \]
      \[
      G_1 = \frac{1}{\sqrt{2\pi}} \left( \frac{1}{\sqrt{\sigma}} \right) e^{-\frac{1}{2} \left( \frac{x-\mu}{\frac{1}{\sqrt{\sigma}}} \right)^2} = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2} \left( (x-\mu)^2 / 2\sigma \right)}
      \]
      \[
      G_2 = \frac{1}{\sqrt{2\pi}\sqrt{2\sigma}} e^{-\frac{1}{2} \left( \frac{x-\mu}{\sqrt{2\sigma}} \right)^2} = \frac{1}{\sqrt{2\pi}\sqrt{2\sigma}} e^{-\frac{1}{2} \left( (x-\mu)^2 + \sigma \right)}
      \]
X = -5:0.01:5;
sigma = 1;
mu = 0;
LoG = ((mu.^2-sigma.^2+X.^2-2*mu*X)./(sqrt(2*pi)*sigma.^5)).*exp(-0.5*(X-mu).^2/(sigma^2));
sigma1 = sqrt(2).*sigma;
G1 = 1./(sqrt(2*pi)*sigma1)*exp(-0.5*(X-mu).^2./(sigma1^2));
sigma2 = sigma./sqrt(2);
G2 = 1./(sqrt(2*pi)*sigma2)*exp(-0.5*(X-mu).^2./(sigma2^2));
plot(X, LoG,'r', 

X, G1-G2, 'b');

See it's not too bad of an approximation... however still an approximation.

d. Why is the difference of Gaussians implementation even more efficient than the $(\nabla^2 G)$ Laplacian of the Gaussian implementation?
Because a convolution is a more difficult operation to compute then a matrix subtraction which can be computed in linear time.

2.

a. Apply the following horizontal Sobel filter

\[
\begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\]
to the 5 x 5 image below

(compute the correlation at the center 9 pixels).

<table>
<thead>
<tr>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>10</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Ans=
\[
\begin{array}{ccc}
6 & 5 & 6 \\
-5 & -11 & -21 \\
16 & 11 & 12 \\
\end{array}
\]

b. Describe, how you would design a filter to detect boundaries, which uses multiple resolutions of the image, and makes use of intensity, texture and color.

By using smaller resolution images we lose some details and textures that you might find in larger images, this is similar to smoothing a large image (however some may rightfully argue more efficient as the filter runs over less pixels). Essentially what we are looking for is similar ratios between edges blobs or whatever we are looking for in comparison to the image size.

c. What is the Laplacian of a Gaussian (LoG) filter?

The Laplacian of the Gaussian filter is a filter that first applies a Gaussian to smooth the image then applies a Laplacian over this to detect edges. This is a combined filter that can be applied to images for edge detection.

d. Let the Gaussian used in the LoG filter for a given scale parameter \( t \) be expressed as to which structures would the LoG give strongest positive responses and to which structures would it give strongest negative responses?

\[
g(x, y, t) = \frac{1}{2\pi t^2} e^{-\frac{x^2 + y^2}{2t^2}}
\]

When \( x \) and \( y \) are very large will give the weakest response, and when \( x \) and \( y \) are close to zero will give the strongest.

3. Suppose that the 9 intensities of a 3X3 image neighborhood are perfectly fit by the planar model \( I[x,y] = px + qy + c \). Show that the simple LoG mask

\[
\begin{pmatrix}
0 & -1 & 0 \\
-1 & 4 & -1 \\
0 & -1 & 0
\end{pmatrix}
\]

\[
p \ast 0 + q \ast 0 + c  
=  
\begin{pmatrix}
p \ast 1 + q \ast 1 + c  
\end{pmatrix} 
=  
\begin{pmatrix}
p \ast 2 + q \ast 2 + c  
\end{pmatrix}
\]

\[
0 + (-p - q - c) + 0 + (-q - c) + (4p + 4q + 4c) + (-2p - q - c) + 0 + 
(-p - 2q - c) + 0 = -4p + 4p - 4q + 4q - 4c + 4c = 0
\]
4. Suppose that an image has all 0 pixels except for a single 1 pixel at the center of the image. What output image results from convolving the image with the 3X3 box filter (shown below)?

\[
\begin{array}{ccc}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9
\end{array}
\]

and surrounded/padded by zeroes if the matrix was larger than 3x3

5. Design a single mask to detect edge elements making an angle of 30 degrees with the Xaxis. The mask should not respond strongly to edge elements of other directions or to other patterns. Apply it to an image of your choice. Show your work.

I took a 5x5 horizontal sobel filter and multiplied it by \(\cos(30)\) and added it to a 5x5 vertical sobel which I multiplied by \(\sin(30)\).

\[
\begin{align*}
sobel_x &= \begin{bmatrix}
-1 & -4 & -6 & -4 & -1 \\
-2 & -8 & -12 & -8 & -2
\end{bmatrix} \\
\text{sobel}_y &= \begin{bmatrix}
2 & 8 & 12 & 8 & 2 \\
1 & 4 & 6 & 4 & 1
\end{bmatrix}
\end{align*}
\]

\[
\text{Sobel}_{30} = sobel_x \ast \cos \theta + sobel_y \ast \sin \theta
\]

\[
\begin{align*}
&-1.37 & -4.46 & -5.20 & -2.46 & -0.37 \\
&-3.73 & -10.93 & -10.39 & -2.93 & 0.2679 \\
&-2.679 & 2.93 & 10.39 & 10.93 & 3.73 \\
&0.3660 & 2.46 & 5.20 & 4.46 & 1.37
\end{align*}
\]

By breaking up the sobel filter into an x and y component and multiplying by \(\cos(30)\) and \(\sin(30)\) respectively you are able to generate a new sobel filter that responds best to line at angles of 30 degrees. This can obviously work with any angle which can be trivially seen with the common horizontal (0deg) and vertical (90deg) filters.

I then wrote a program to compare simply rotating the filter to the breaking up method I created.

I found that the breaking up a 5x5 sobel filter was the most effective method by visual inspection of a square rotated 0, 30, and 60 deg. See image bellow of my filter on this square. Or run the code yourself

```matlab
close all;
clear;
clc;
I = imread('tri-60.jpg')>200;
```
prewittx = [0 0 0 0; -1 -1 -1 -1; 0 0 0 0; 1 1 1 1; 0 0 0 0]

prewitty = imrotate(prewittx, 90);
prewitt302 = imrotate(prewittx, 30)
prewitt303 = prewitty* sind(30) + prewittx* cosd(30)

sobelx = [-1 -4 -6 -4 -1; -2 -8 -12 -8 -2; 0 0 0 0 0; 2 8 12 8 2; 1 4 6 4 1];
sobely = imrotate(sobelx, 90);
sobel302 = imrotate(sobelx, 30)
sobel303 = sobely* sind(30) + sobelx* cosd(30)

b=filter2(prewitt302, I);
b=mat2gray(abs(b));
c=filter2(prewitt303, I);
c=mat2gray(abs(c));

e=filter2(sobel302, I);
e=mat2gray(abs(e));
f=filter2(sobel302, I);
f=mat2gray(abs(f));

figure;
imshow(b);
figure;
imshow(c);

figure;
imshow(e);
figure;
imshow(f);
6. Design a set of four 5x5 masks to detect the corners of any rectangle aligned with the image axis. The rectangle can be brighter or darker than the background.
7. Use the corner detection procedure of the previous problem, to implement a program that finds rectangles in an image. (Rectangle sides are assumed to align with the sides of the image). The first step should detect candidate rectangle corners. A second step should extract subsets of four candidate corners that form a proper rectangle according to geometric constraints. An optional third step might further check the four corners to make sure that the intensity within the candidate rectangle is uniform and contrasting with the background. Test your program on a noisy checker board image and the image of a building with rectangular windows. (you will need to create a "noisy checkerboard image).
tr = [0 0 0 0;1 1 2 0;-1 -1 -4 2;0 0 -1 1];
bl = [0 1 -1 0;0 1 -1 0;2 -4 -1 -2;0 0 2 1];
br = [0 0 -1 1;0 0 -1 1;-1 -4 2 0;1 1 2 0];

TLF=abs(filter2(tl,c))>7;
TRF=abs(filter2(tr,c))>7;
BLF=abs(filter2(bl,c))>7;
BRF=abs(filter2(br,c))>7;
figure;
imshow(TLF+TRF+BLF+BRF);
figure;
imshow(c);
newI = zeros(size(c));
[y,x]=size(c);
countOfRectangles = 0;
topCornerIndexs = find(TLF==1);
for topCornerIndex=1:size(topCornerIndexs)
    [TLy,TLx]=ind2sub([y,x],topCornerIndexs(topCornerIndex));
    TRx = find(TRF(TLy, (TLx+1:x)) == 1)+TLx;
    BLy = rot90(fliplr(find(BLF((TLy+1:y),TLx) == 1)+TLy));
    found = 0;
    for i = 1:numel(BLy)
        for j = 1:numel(TRx)
            BRy = BLy(i);
            BRx = TRx(j);
            if (BRF(BRy,BRx)==1)
                newI(TLy:BLy,TLx:TRx)=1;
                disp(['R#: ' num2str(countOfRectangles)]);
                disp(['topLeft: (',num2str(TLy),',',num2str(TLx),')']);
                disp(['bottomLeft: (',num2str(TLy),',',num2str(BRx),')']);
                disp(['topRight: (',num2str(BRx),',',num2str(TLy),')']);
                disp(['bottomRight: (',num2str(BRx),',',num2str(BRy),')]')
                found=1;
                countOfRectangles=countOfRectangles+1;
                break
            end
        end
    end
    if found==1
        break
    end
end
end

imshow(newI);

output (with 2x2 checkerboard each square of 5 pixels):
R#: 0
  topLeft: (1,6)
  bottomLeft: (1,10)
  topRight: (5,6)
  bottomRight: (5,10)
Diff output with no noise and 20% noise it mistook two out of the 10 rectangles.  
https://www.diffchecker.com/qa8rvxa6

Diff output with no noise and 20% noise worst case found it mistook all but two rectangles.  
https://www.diffchecker.com/6a944j5s

**Visual representation of checkerboard:**

Original:

![Original Checkerboard]

Found Corners:

![Found Corners]
Found rectangles (all in the middle half on edge):

Test on half gray and black half white and black checkerboard.
Original Image:

Corners Found:

Rectangles Found (found all but some of the edges same as other checkerboard):
Visual Representation with Noise:

Original Image:

Corners Found:

Rectangles Found (most of the rectangles are properly sized however some are using corners from other rectangles and are therefore twice or 4 times as big):
Test on building with thresholding:
Input image (high contrast cropped version of original):

After Thresholding:

All Corners Found:
All Rectangles Found:

Test on building without thresholding:
All Corners Found:

All Rectangle found (it's a mess):
Test on randomly sized rectangles:

Original Image:

Corners found (they are difficult to see as they are singular white pixels but there):
8. Suppose a function $h(x)$ takes value 1 for $-\frac{1}{2} \leq x \leq \frac{1}{2}$ and is zero elsewhere, and suppose that function $g(x)$ takes the value 1 for $10 \leq x \leq 20$ and zero elsewhere. Sketch the two functions $h$ and $g$. Compute and sketch the convolution of $g$ and $h$: $f = g * h$. Finally, compute and sketch $h*h$.

$$f = \begin{cases} 
  t - 9.5 & 9.5 < T \leq 10.5 \\
  1 & 10.5 < T \leq 19.5 \\
  20.5 - T & 19.5 < T \leq 20.5 \\
  0 & \text{otherwise}
\end{cases}$$
9. Write a program that performs histogram equalization. Apply it to an image of your choice. Show your work.

```matlab
close all;
clear;
clc;
i = imread('planet.jpg');
[y,x]=size(i);
his=tabulate(i(:));
his = his(:,2);
cumsum=cumsum(his);
i2=cumsum(i)/(x*y)*256;
figure;
imshow(mat2gray(i2));
```

![Image of planet before and after histogram equalization]

10. Stretch an image: Begin with an image I(x,y) of your choice and “change” the pixels (x,y) to (x’,y’) such that x’ = ax + by + c and y’ = dx + ey + f, for given a,b,c,d,e and f. Show your result.

```matlab
close all;
clear;
clc;
I = imread('yan.jpg');
imshow(I);
a=3;
b=1;
c=1;
d=1;
e=3;
```
f=1;
%these will be replaced with larger values in the for loop
maxx = 0;
maxy = 0;
size=size(I);
numb =numel(I);
points = NaN([5,numel(I)]);
for ind = 1:numb
    [i,j]=ind2sub(size,ind);
    newx= floor(a * i + b * j + c);
    newy= floor(d * i + e * j + f);
    points(1,ind)=i;
    points(2,ind)=j;
    points(3,ind)=I(i,j);
    points(4,ind)=newx;
    points(5,ind)=newy;
end

i2=uint8(NaN(max(points(4)),max(points(5))));
for count=1:numb
    oldx=points(1,count);
    oldy=points(2,count);
    val=points(3,count);
    newx=points(4,count);
    newy=points(5,count);
    coe = sub2ind(size,min(size(1),oldx+1),min(size(2),oldy+1));
    i2(newx:points(4,coe)-1,newy:points(5,coe)-1)=val;
end
figure;
imshow(i2);

->