Name: Per Arne Kjelsvik email adres: p.a.kjelsvik@student.utwente.nl

Student id: s2049201

Educational Program: EEMCS

Exercise 3:
measurements of the diameter of arteries in angiograms

We develop and evaluate image processing for measuring the diameters of a blood vessel in coronary angiographic images such as the one shown in Figure 3. The exercise consists of two parts:

part I: edge detection and morphological operation
Hand-on training for multi-scale edge detection, such as Canny and Marr-Hildreth.

part II: assessment of the accuracy of the diameter measurements
Edge detection is useful for delineation of objects. But it has also limitations, which we will examine.

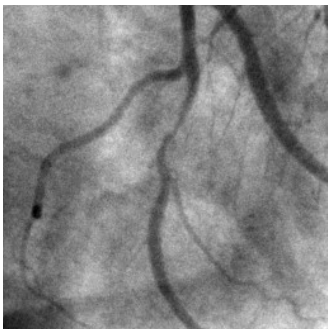


Figure 1.An angiogram showing the image of artery

Part I: Edge based image segmentation

1. The scale space

A 2D Gaussian point spread function can be created and visualized with:

```
sigma = 5;                    % set the width of the Gaussian
L = 2*ceil(sigma*..)+1;       % fill in a constant to define the matrix size
xymax = (L-1)/2;              % the maximum of the x and the y coordinate
xrange = -xymax:xymax;        % the range of x values
yrange = xrange;              % the range of y values
% create the PSF matrix
h = fspecial('gaussian', L, sigma);

%% visualize as a 3D plot:

% create a RGB matrix to define the colour of the surface plot
C = cat(3,  ones(size(h)),ones(size(h)),zeros(size(h)));

% create the surface plot of the gaussian
figure(1)
hd =surf(xrange,yrange,h,C,'FaceColor','interp','Facelight','phong');
camlight right                % add a light at the right side of the scene
xlim([…]);                    % set appropriate axis limits
ylim([…]);
xlabel('x');                  % add axis labels
ylabel('y');
zlabel('h(x,y)');
print -r150 -dpng ex3_1.png   % print the result to file
```

The function `fspecial` is used to create a Gaussian PSF matrix `h` with the deviation (width) of the Gaussian set to σ (matlab: `sigma`). The size of `h` should be such that the truncation error of the tails of the Gaussian is negligible. On the other hand, a too large size leads to a waste of computations. The function `surf` creates a surface plot of the PSF. The 3D array `C` defines the colour of the surface. The other functions, `camlight`, `xlabel`, etc, are needed to refine your graph[1].

a) Initiate a new matlab script. That is, start with `%% ex 3 - name`, then clear the work space and close all figure windows, then add a line “ `%% question 1`. Next copy and paste the code given above[2]. Complete the unfinished lines.

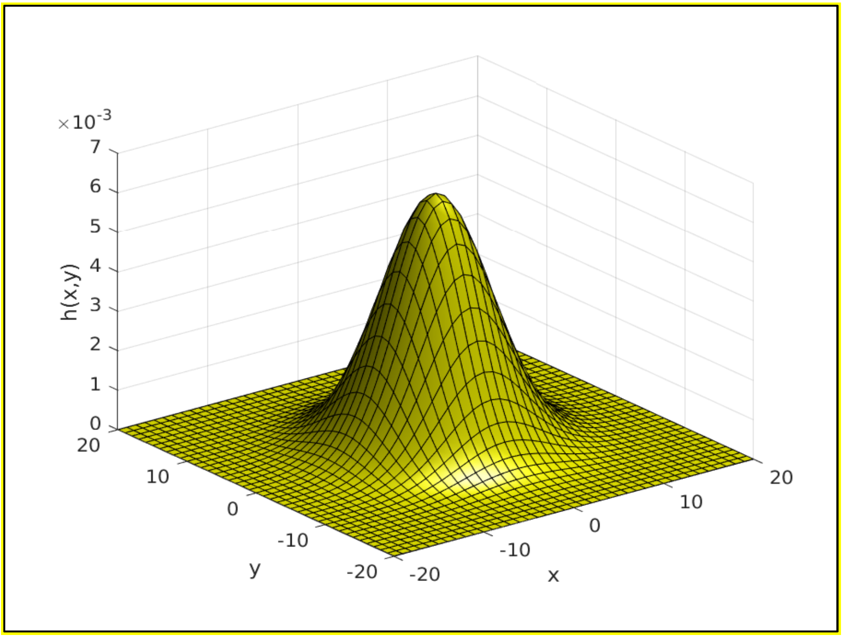
Give the completed Matlab code of the 2nd line above:

```
L = 2*ceil(sigma*4)+1; % fill in a constant to define the matrix size
```

[1] **Note**: in the sequel, omission of axis labels in any graph (images excluded) in your report will be considered as a shortcoming of your report, and will have a negative impact on the grade.

[2] Maybe you have to retype part of the text as the adobe pdf may use other fonts than that the Matlab editor expects.

Insert the graph:



b) Apply the filter to the test image stored in `ang2.png`. Apply this with four different values of σ . That is: $\sigma = 1$, 10, 20 and 35. You can use the following template for your code:

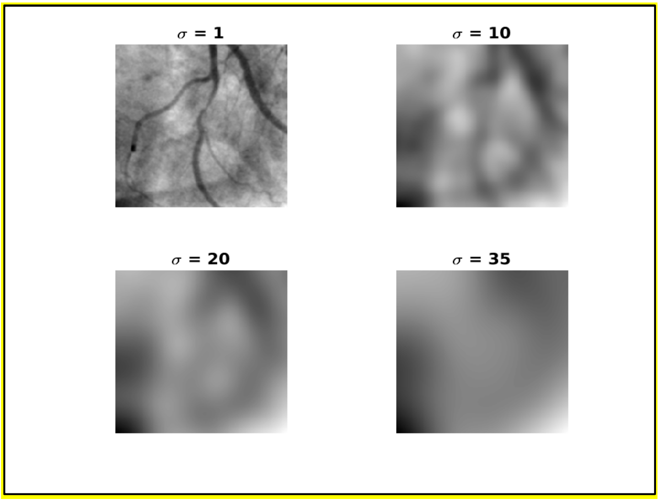
```
im = ...                  % read the image from file
sigma = [1 2 4 8];        % fill array with sigma values
for i=1:4                 % do 4 times:
    h = fspecial(... ;                      % create the PSF
    imfiltered = imfilter(... ;             % apply the filter
    subplot(2,2,i);                         % define a subplot
    imshow(imfiltered,[]);                  % show the image
    title(['\sigma = ' num2str(sigma(i))]);% include a plot title
end
```

Select the options of `imfilter` such that the output image has the same size as the input image, and the boundary option is such that boundary effects on the border of the image are visually minimized.

Give the completed Matlab code of the lines that contain the `fspecial` function and `imfilter` function:

```
h = fspecial('gaussian', L, sigma(i)); % create the PSF
imfiltered = imfilter(im,h,'replicate', 'same'); % apply the filter
```

Insert the resulting images:



What are the scales of these four images?

```
[1 100 400 1225] % Sigma^2 defines the scale of the images
```

2. Derivatives

The first derivative of a continuous image $f(x,y)$ is denoted by $f_x(x,y) = \partial f(x,y)/\partial x$. In digital image processing, only digital representations $f(n\Delta, m\Delta)$ and $f_x(n\Delta, m\Delta)$ are available. The derivative $f_x(n\Delta, m\Delta)$ is only an approximation. In the discrete case, differentiation is necessarily extended to a finite neighbourhood. It can be approximated by a convolution that implements the differentiation combined with some sort of low-pass filtering.

In the continuous domain this combination is nicely accomplished by:

$$\frac{\partial}{\partial x}\big(f(x,y) * h(x,y)\big) = f(x,y) * \left(\frac{\partial}{\partial x} h(x,y)\right)$$

With that, differentiation in the digital domain is accomplished by:

$$f_x(n\Delta, m\Delta) \cong f(n\Delta, m\Delta) * h_x(n\Delta, m\Delta)$$

In the scale space theory it has been argued that the best low-pass filter needed for differentiation is the Gaussian:

$$h(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{1}$$

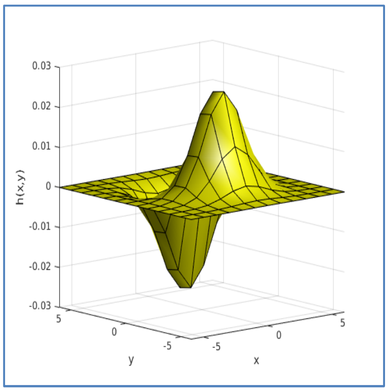
Code to create the PSF matrix `h` with a size of `LxL` (as an alternative to using the function `fspecial`) is:

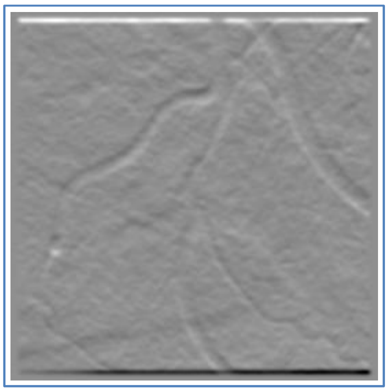
```
N = ceil((L-1)/2);          % get the size of half of the full range
[x,y]=meshgrid(-N:N,-N:N); % create the coordinates of a 2D orthogonal grid
h = exp(-(x.^2 + y.^2)/(2*sigma^2))/(2*pi*sigma^2);
```

a) Give the analytical expression for $h_y(x,y)$. That is, differentiate expression (1) analytically with respect to y. Insert the found expression in Matlab syntax such as in the sample code for $h(x,y)$:

```
hy  = -(exp(-(x.^2 + y.^2)/(2*sigma^2)).*y)/(2*pi*sigma^4);
```

b) Create a PSF matrix `hy` that contains a sampled version $h_y(n\Delta, m\Delta)$ of $h_y(x,y)$. Use $\sigma = 1.5\Delta$ (which is equivalent to $\sigma = 1.5$ and $\Delta = 1$). Visualize the PSF as in question 1, and insert the figure. Apply the convolution with `hy` to the artery image, insert the resulting image. Use the function `imwrite` and `mat2gray`, rather than a copy of a figure window)





c) Explain the response of the operator to:

- Areas without much contrast.
- Horizontal step transitions such as at the boundaries of vertical blood vessels.
- Vertical step transitions such as at the boundaries of horizontal blood vessels.

The areas where there is little contrast becomes "uniform", a smooth gray surface. So the information in these areas become less important, due to being subdued.
The horizontal step transitions also become similar to the areas without much contrast, more washed out uniformly gray. This makes sense since we're differentiating with respect to y, and the real part of the operator hy will be zero for transitions in the horizontal direction.
For the vertical step transitions at the boundaries of horizontal blood vessels are much more visible. Here we can see almost like a depression in a stone wall, more contrast. You can see the vertical parts being smoothed, and then slowly coming more and more in contrast as the arteries swing horizontally. This also makes sense since the derivative is a harmonic function for these transitions. Also, there's a dark edge above and a lighter edge below for the transitions of the high-contrast areas.
If we differentiate for x instead we get the same result, just opposite (vertical blood vessel visible, horizontal smoothed out).

INTERMEZZO
The usage of the matlab function `mat2gray`:

Remember:

- If the number representation of an image is of type **`uint8`**:
  - The range of pixel is: 0, 1, 2, …, 255. So, the resolution of the grey scale is 1.
  - 0 → black and 255 → white
- If the number representation is **`double`**:
  - The range of pixels is between 0 and 1, and the resolution of the grey scale is very fine (10^{-16})
  - 0 → black and 1 → white

When writing an image **`im`** to file (jpg or tif), then:

- If the number representation of **`im`** is **`uint8`**, the pixels are written to file without processing. So, the bytes of the image are literally written to the file (apart from a possible compression).
- If the number representation of **`im`** is **`double`**, then the pixels are rescaled by a factor of 255, that is: **`im`** becomes **`im*255`**, then converted to **`uint8`**, and finally written to file.

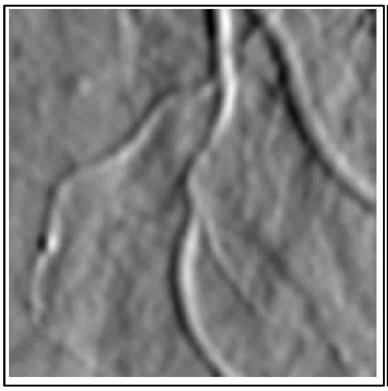
In the latter case, information may be lost. First of all, there will be round-off errors. Secondly, if the grey level of a pixel was larger than 1 (whiter than white), or smaller then 0 (blacker than black), then in the conversion from **`double`** to **`uint8`**, the pixel will be first truncated to 1 or to 0, respectively.

This truncation can be prevented by the function **`mat2gray`**. This function maps the greyscale of an image linearly (**`im`** becomes **`a*im+b`**) such that the maximum of **`im`** becomes 1 and the minimum becomes 0. This guarantees that no truncation takes place when writing to a file, but it also may enlarge the round-off error.

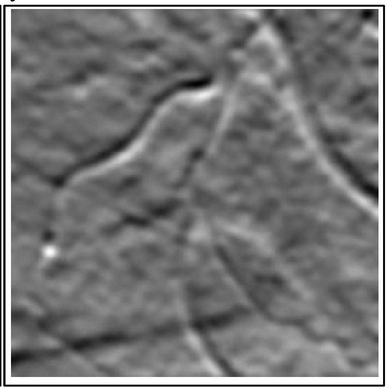
FROM NOW IT IS UNDERSTOOD THAT YOU USE MAT2GRAY WHENEVER NEEDED !
FAILING TO DO SO HAS A NEGATIVE IMPACT ON THE GRADE

3. The function `ut_gauss` is an easy-to-use implementation[3] to calculate the derivatives $f_x(n,m)$, $f_y(n,m)$, $f_{xx}(n,m)$, $f_{yy}(n,m)$, and $f_{xy}(n,m)$. Apply these operations to the test image. Use $\sigma = 3$.

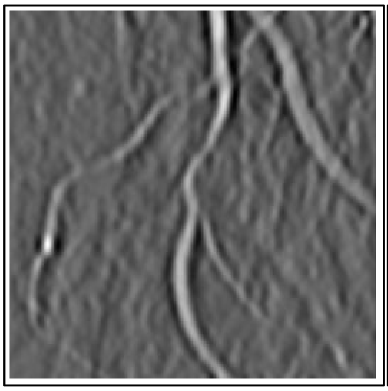
fx:



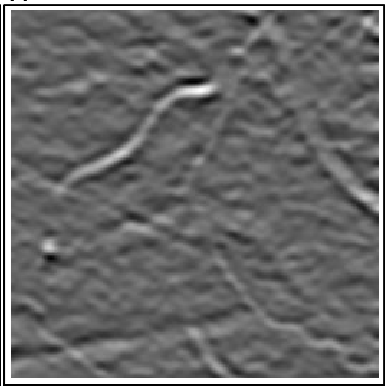
fy:



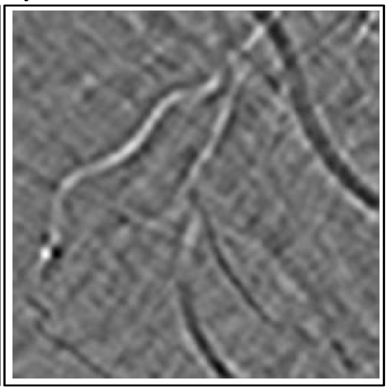
fxx:



fyy:



fxy:



[3] We silently assume that $\Delta = 1$ from now on.

Consider a vertical artery. Which of these images convey information about this artery, and what kind of information is that? Which of these images do not convey information in this particular case?

Both fx and fxx convey information about vertical arteries, or rather, about the edges developing in the horizontal plane. We can see a strong contrast in the the images where the arteries are, and in fx we can see a light edge on the right side and a dark edge on the left side, like there's a light shining down from the sky to the left towards the arteries (and then the left edge of the artery being like a canyon edge). In fxx the transitions are emphasized more, with a black border on both sides of the arteries. The information we can deduce is which direction the arteries are, and we could also say something about the diameter of the artery. The information conveyed is the vertical (and partly horizontal) transitions' amplitude, which are both shown in fx and fxx.

fy and fyy don't give us information of the vertical artery, except small transitions visible due to the horizontal transitions not being completely horizontal.

fxy is interesting, because it tells us about diagonally aligned arteries, so arteries like the s-swing which are more inbetween vertical and horizontal (diagonal transitions). So this also tells us partly about vertical arteries, but fx and fxx is what we're mostly interested in this case (horizontal transitions).

4. Gradient magnitude and Laplacian

The gradient magnitude of an image is defined as:

$$\left|\nabla f(n,m)\right| = \sqrt{f_x^2(n,m) + f_y^2(n,m)}$$

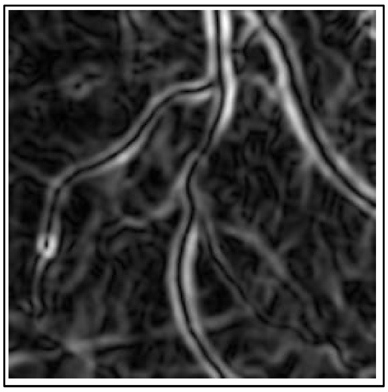
or, in short, $\sqrt{f_x^2 + f_y^2}$. The Laplacian is defined as:

$$\Delta f(n,m) = f_{xx}(n,m) + f_{yy}(n,m)$$

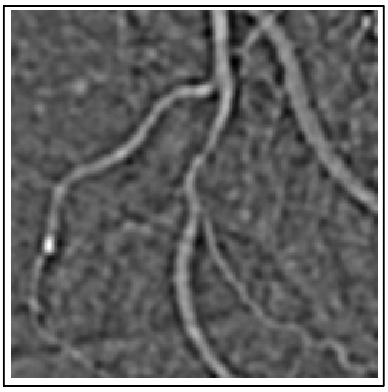
or shortly $f_{xx} + f_{yy}$.

Using the results from question 3, calculate the gradient magnitude and the Laplacian of the test image. **Don't use for-loops.**

gradient magnitude



Laplacian:



How can these images help to find edge segments?

Well, we can clearly see in the resulting images that the edges of the arteries are more pronounced than in the original image. This can be translated to finding zero crossings in the laplacian image, or the local maxima in the gradient magnitude. For local maxima, we need to distinguish between noise and real edges with some type of threshold; If the threshold is too high, real edges are lost, and if it's too low, fake edges will be recognized as real. With our eyes we can see the white lines in the gradient magnitude, and the black lines in the laplacian (zerocrossing), and use this to find the edges in the image. This, incidentally, we can also do with code.

5. Marr-Hildreth's zero crossings

In the continuous domain the equation

$$f_{xx}(x,y) + f_{yy}(x,y) = 0$$

defines, under mild conditions, curves in the x, y-plane. These curves are called the *zero crossings* of the Laplacian.

The goal of the Marr-Hildreth operation is to find an edge map. The edge map is a binary image in which all found edge elements are marked by `1` where all other pixels are `0`. Marr and Hildreth mark each pixel as an edge element if the Laplacian exhibits a zero crossing near that pixel position.

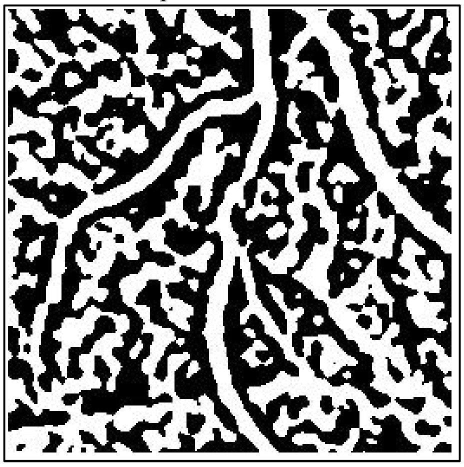
A cheap zero crossing detection is accomplished by first finding areas with a positive Laplacian, and then to find the boundary of these areas. We apply this procedure to the Laplacian of the test image obtained in question 4:

a) Create a binary image in which a pixel is set to 1 if the corresponding Laplacian for that pixel is positive. **(Don't use for-loops!)**

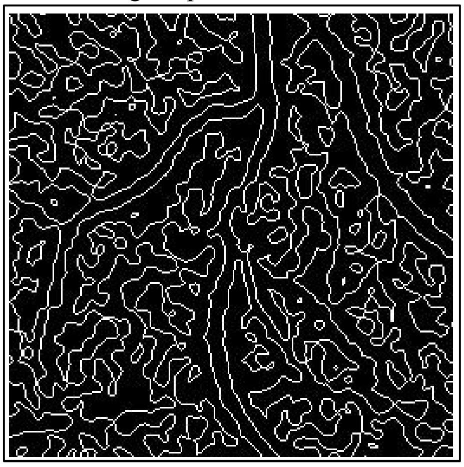
b) Determine the boundary pixels of the binary image, and display it on the screen. For that purpose:
   - Create a structuring element consisting of a 4-neighbourhood (use `strel` with option `diamond`.)
   - Erode the binary image by this structuring element, so that only the interior of the foreground is left.
   - Subtract this interior from the original binary image.

   Note: `bwmorph` with option `'remove'` does almost the same job but is still less useful as it operates differently on the border of the image).

thresholded Laplacian:



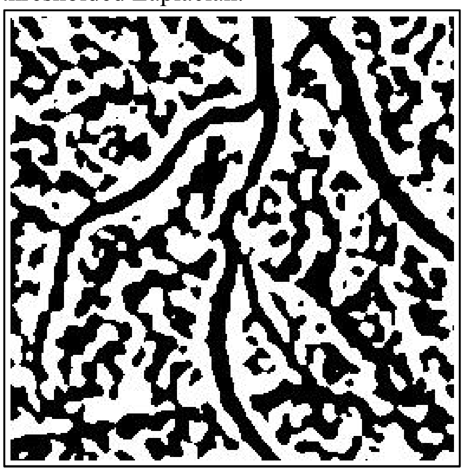
zero crossings Laplacian:



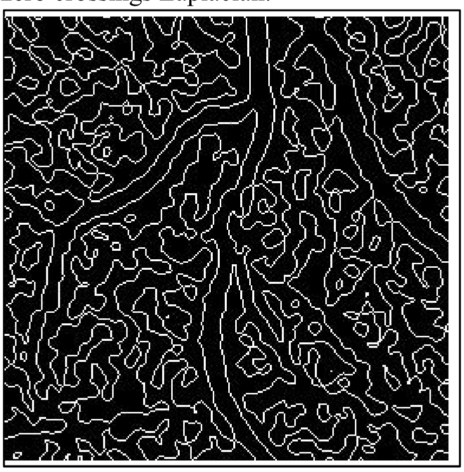
c) Form the resulting zero crossings 8-connected paths or 4-connected paths? `8-connected path`

d) Instead of marking areas with positive Laplacian values, areas with negative Laplacians can be marked. Repeat a) and b) with these negative areas.

thresholded Laplacian:



zero crossings Laplacian:



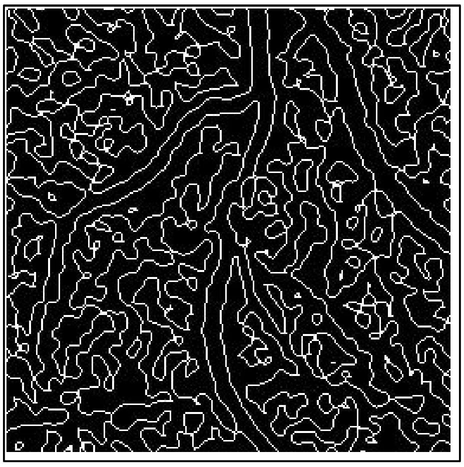
e) If you compare the results of question 5b with the one of 5~~e~~, which of the two will underestimate the diameter of an artery, and which of the two will overestimate it?

By comparing, we can see that the negative zero crossing (5d) overestimates the diameter of an artery, while the positive zero crossing (5b) underestimates it.

f) To prevent the under- or overestimation, mentioned in question e, the two results will be combined by means of a logical or operation. In matlab, this is the operation |. Calculate this, and show the results below. The next step is to skeletonize the found structure to assure paths with a thickness of 1 pixel.

Combined zeros crossings:

Skeletonized:



6. Hysteresis thresholding the gradient magnitude

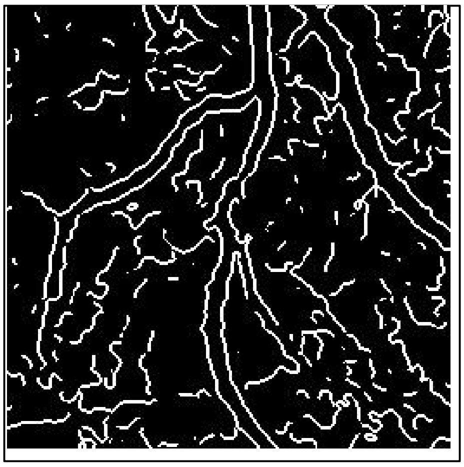
The zero crossings of the Marr-Hildreth operator produces many candidate edge elements, but many of them will be false. The reason for this is that the Laplacian may exhibit a zero crossing at positions for which the gradient magnitude is near zero. So, the condition for a zero crossing to be an acceptable edge element is that the gradient magnitude exceeds some suitable threshold.

a) Determine the gradient magnitude as in question 4. Mask this image by the Marr-Hildreth zero crossings obtained in question 5. That is, all gradient magnitude pixels that are not a zero crossing should be made zero, while all other pixels are retained.

b) Compare the masked gradient magnitude image against a suitable threshold `T`; and show an edge map consisting of zero crossings for which the gradient magnitude is larger than `T`. Choose the threshold such that the narrowing in the centre is just visible in the edge map. Show the result on the screen. You can interactively select the threshold by visually evaluating the edge map.

Found threshold: `0.18`

Insert found edge map:



In hysteresis thresholding, the gradient magnitude is first masked by the zero crossings (so as to exclude all pixels which are not a zero crossing, as before). Next, the result is thresholded twice resulting in two edge maps. The first map, called the `marker`, is created with a large threshold. It may miss many weak edges, but there will be only a few false edges. The second map, called the `mask`, is created with a lower threshold. It contains both weak and strong edges, but also many false edges. The strategy is to accept an edge segment in the second image only if such an edge segment contains at least one edge element from the first edge map. Thus, the `marker` will be propagated into the `mask` to yield a more reliable edge map.

c) Apply a threshold operation with a threshold that is somewhat larger than the previously found threshold `T`. Name the resulting image `marker`.
d) Apply a threshold operation with a threshold `T`. Name the resulting image `mask`.
e) Propagate the `marker` image into the `mask` image, and interpret the result. **Hint**: use the function `imreconstruct`.

Resulting edge map:

Describe the (potential) advantages of hysteresis thresholding:

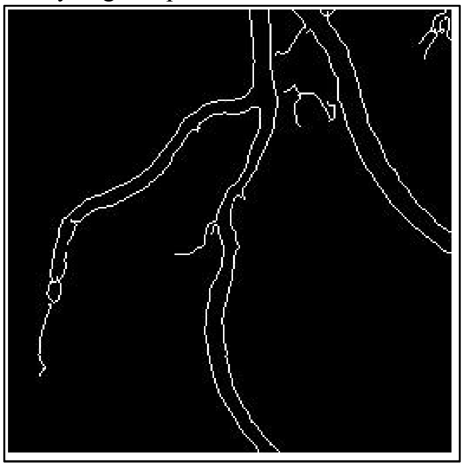
Our result is potentially better since we use two thresholds, where the result is that we only keep the small edges that are connected with our large edges. This way we can filter out a lot of the fake edges, and we get a more connected result in turn. To further improve on the edge map we get might be hard with code, but we can see visually that there is small squiggly lines here and there which indicates that this is not actual arteries, or at least not as significant as the major arteries we can see in the center. Perhaps another combinations of thresholds than what I chose could have removed these lines, or we could perhaps apply some extra constraints, like a constraint that only accepts segments of a certain length, but this would in turn not be an equally good general solution.

7. Edge detection

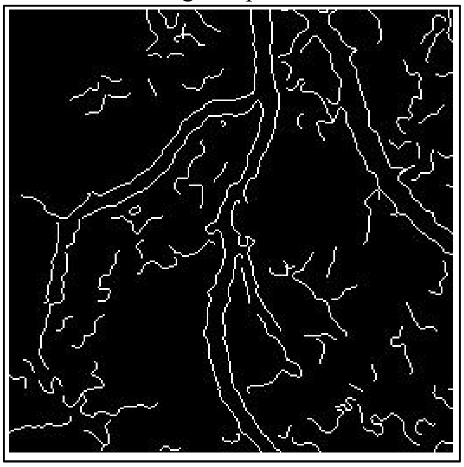
The Marr-Hildreth, including hysteresis thresholding, is implemented in the function `ut_edge`. This function uses the function `ut_levelx` to find the zero crossings. `ut_levelx` implements a more accurate method to find the zero crossings than the one outlined in question 5. `ut_edge` also contains an implementation of the Canny edge detector.

b) Apply `ut_edge` to the test image in order to get an edge map in which the edge elements form two non-fragmented edge segments. Select the options of `ut_edge` as follows:
   - Set the option '`c`' That is, apply Canny. Later we will also use '`m`' (Marr-Hildreth)
   - Use option '`s`' to set the scale. You can try it first with $\sigma = 4$. But his may not be the best choice.
   - Use the option '`h`' (hysteresis thresholding). The thresholds are specified as two additional parameters. Read the help of the function to understand how these parameters are defined. Select these parameters and the scale interactively such that the narrowing is well delineated, but without too much spurious edges in the remaining part of the map. Perhaps, you need some iterations to establish the right parameters and scale.
   - Repeat with the Marr-Hildreth operator.

Canny edge map:



Marr Hildreth edge map:



c) Describe in one sentence the main difference between the localization criterion of the Marr-Hildreth edge detector and the one of the Canny edge detector. Hint: read the powerpoint sheets.

> The main difference is that Marr-Hildreth uses the that "Laplacian must be 0", while Canny uses the second derivative, taken in the direction of the gradient, which must also be 0.

d) What is the theoretical advantage of the Canny's criterion above Marr-Hildreth crieterion?

> Marr-Hildreth generates responses that doesn't correspond to edges (false edges), and the localization error might be severe at curved edges.
>
> Canny uses non maximum suppresion, meaning edge candidates which are not dominant in their neighborhood arent' considered to be edges (reduces false edges). It also detects the gradient change in curved edges better because it's using the second derivative

Part II: Accuracy of edge-based diameter measurements

In this part, an experiment will be conducted to quantitatively assess the quality of diameter estimation from X-ray images using edge detection. Software is available that simulates the imaging of an artery with a given diameter. An example of a test image is provided in Figure 2.

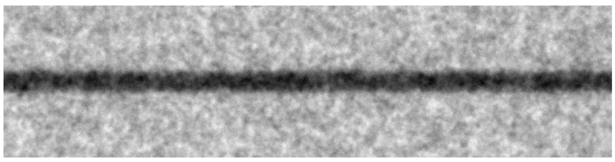


Figure 2. Test image based on a simulation of X-ray imaging of an artery

The test image shows a horizontally aligned artery. Prior knowledge of this alignment facilitates the determination of the diameters because only two horizontal edge segments needs to be localized. The assumption of horizontally aligned arteries is not very unrealistic as a suitable geometric transform can be applied that ensures this. See Figure 3 for an example.

The simulation of the test image is based on an accurate model of the imaging device. It includes:

- A circular model of the cross section of the blood vessel.
- The pixel size is $\Delta = 40\,\mu m$. Pixels are square. A diameter of 1 mm corresponds to 25 pixels.
- The maximum exposure is equivalent to an average of 12500 quant/mm^2, which is 20 quant/pixel.
- The OTF of the imaging device, a so-called image intensifier, is given by:

$$H(\rho) = \exp\left(-\frac{\rho}{\rho_0}\right) \text{ with } \rho_0 = 2\ lp/\text{mm}$$

The Matlab function that accomplishes this simulation is `im_bloodvessel`. This function produces a 250x1000 image. The diameter of the simulated artery is defined in units of mm, and should be passed to the function as an input argument of the function. In each call of the function, a new random realisation of the noise is generated.

a) b)

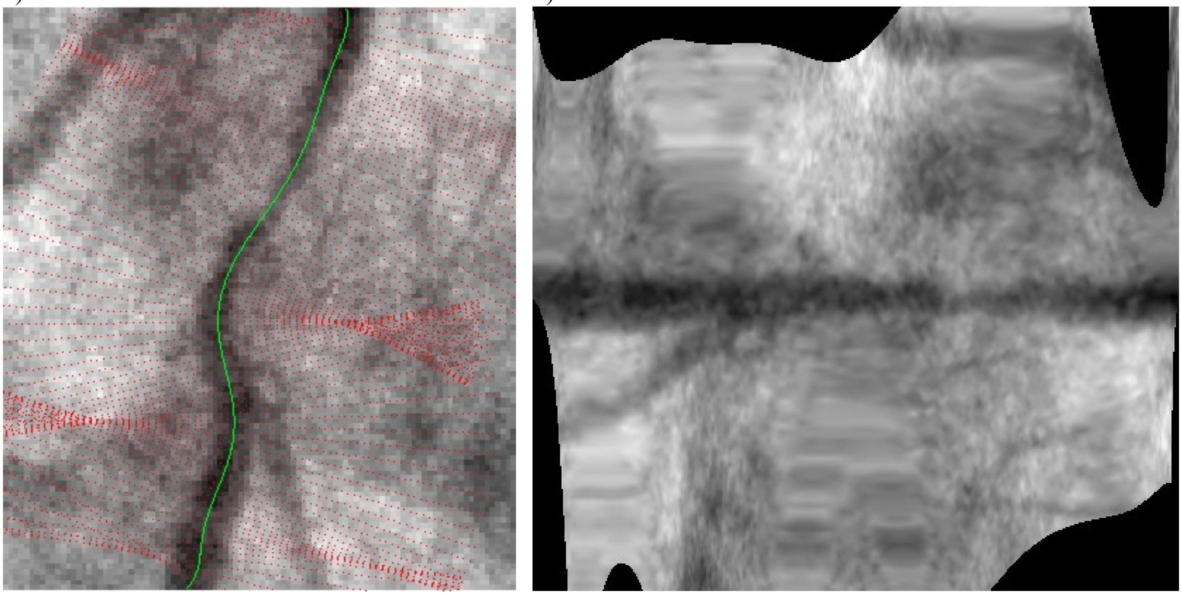


Figure 3 De-skewing a coronary angiogram
a) Original X-ray image with superimposed the centre line (green) of a blood vessel, and with a grid (red) of which the rows are orthogonally aligned across this centre line. The columns are aligned along the centre line.
b) The de-skewed image. The grid in a) is rearranged in an orthogonal grid. This aligns the artery horizontally in the centre of the image.

A second function, `get_diameters`, is available that calculates for each column the diameter, expressed in pixels. The help of this function is as follows:

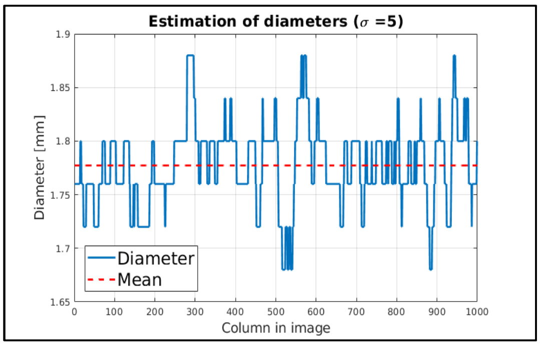
```
[diameters,success]=get_diameters(im,sigma,op)

Estimation of the diameter of a horizontally aligned blood vessel
INPUT:
   im:          input intensity image
   sigma:       gaussian width of the applied filters in the edge detector
   op:          edge operator. 'm' : marrhildreth, |'c'| : canny
OUTPUT:
   diameters    array with for each column in the image, the estimated diameter
   success:     flag to indicate successful estimation. 'true' (1): success

Example:
  im = im_bloodvessel(2);                 % simulation of an artery with 2 mm diameters
  [diam,suc] = get_diameters(im,3,'m');   % sigma=3 performed with Marr-Hildreth operator
```

If the flag `success` is false (0), the estimation was not successful, and the resulting diameters are not valid.

8. Generate a new test image with a diameter of 2 mm, and estimate with `get_diameters` the diameters using the canny operator and with $\sigma = 1$. Inspect and describe what happens with the success flag and the resulting diameters. Explain how these results should be interpreted in this particular case..

> The success flag is set to 0, and the resulting diameters is []. That means that the function did not succeed in estimating the diameters of the columns of the image. If we look into the function ut_gauss, we can find a comment saying that the elements of the PSF are obtained by area sampling of the continuous function, rather than impulse sampling. That area sampling is equivalent to impulse sampling preceded by a prefilter, which prevents large aliasing errors that would occur if SIGMA is too small (say, smaller than 1.5). So in our cause this is where results break down, not being able to estimate properly. Should be interpreted like data we want to discard.

9. Estimate the diameters again, but now with $\sigma = 5$. Check the value of the success flag. Show the found diameters by plotting them in a graph. Don't forget the axis labels. Insert the graph:



The quality of the edge-based diameter estimation has four aspects:

- Robustness: the ability to produce results that make sense. Robustness can be expressed in the *success rate*: the probability of not having an outlying result caused by some internal fatal error.
- Precision: the ability to reproduce the same result if the measurement is repeated under the same conditions. This can be quantified by means of the *standard deviation* of the results.
- Accuracy: the ability of having a low systematic error. A systematic error is an error that replicates each time the experiment is repeated under the same conditions. This can be quantified by the *bias* of the results.
- Detailedness: the ability to resolve the curvedness of the boundaries of the artery.

This exercise will only address the first three aspects.

10. Calculate the standard deviation (function **`std`**) and the average of the diameters (function **`mean`**) of the estimated diameters in question 9.

Standard deviation of the estimated diameters: `0.0439`

Average diameter: `1.7638`

What is in this case the real diameter if expressed in pixels? `50 %2*25`

The bias of **`get_estimators`** can be roughly assessed by the difference between the average diameter and the real diameter. What is this difference (in pixels)? `5.905 %6 pixels`

Why is this only a rough assessment?

We only ran the estimation once, meaning that we haven't analyzed a larger set of data, making the assessment rough in statistical terms.

11. A failure of the estimator is internally detected inside **`get_diameters`**, and indicated by the flag **`success`**. To determine the success rate, a single image does not suffice. To assess the robustness, i.e. to estimate the probability of success, a number of images with different noise realizations are needed.

a) Suppose that we have N images with different noise realizations, and suppose that from these N images **`get_diameters`** reports n images with successful results, give the expression for estimating the success rate (use Matlab syntax):

```
success = n/N
```

b) If it is required, that the uncertainty of the estimated success rate is about 10%, how many images with different noise realizations would be approximately enough to achieve this uncertainty?

```
N = n/0.1
```

c) Write Matlab code, in which the N images are generated, and in which consecutively the success rate is estimated. Do this with the following parameters: real diameter = 1.5 mm, canny operator, $\sigma = 4$. What is the resulting success rate?

```
0.8 % 80% success
```

12. In question 10, you calculated the standard deviation and bias of the estimator based on just one image. In question 11, a couple of images were generated to estimate the success rate. Now that you have these

images, the estimation of the standard deviation and bias can also be based on these multiple images, so as to improve the uncertainties of these estimates. Expand the code in question 11 to implement this.

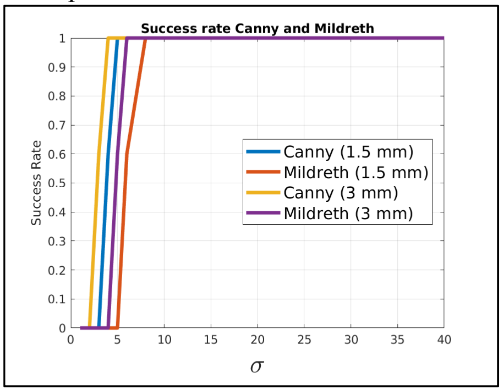
Tips: you can do so by first collecting the results of **`get_diameters`** in N dimensional arrays, and then processing these arrays to get the estimates of the standard deviation and bias. Note that when **`get_diameters`** indicates a failure by means of the **`success`** flag, the values of the diameters are not valid.

13. The estimated success rate, standard deviation, and bias depends on:
   - the real diameter D, expressed in mm.
   - the edge operator: canny **`'c'`**, or marr-hildreth **`'m'`**.
   - the width of the edge operator σ.

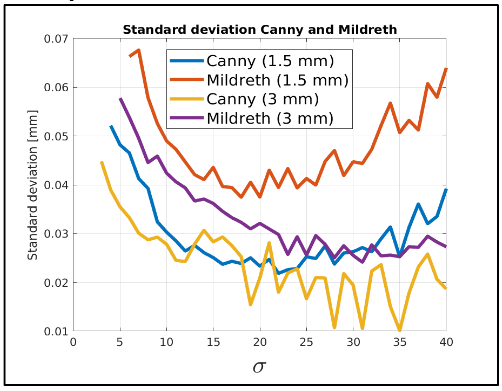
   a) Set up an experiment, in which the success rate, standard deviation, and bias are estimated with varying width: $\sigma = 1, 2, \cdots, 40$. Do this with $D = 1.5$ mm, and the canny operator. Repeat this with the marr-hildreth operator. Put the results of both operators in three graphs: 'success rate', 'standard deviation', and 'bias'. Don't forget the axes labels.

   b) Repeat this with $D = 3$ mm. Add the results to the 3 graphs. Don't forget to add a legend (**`legend`**).

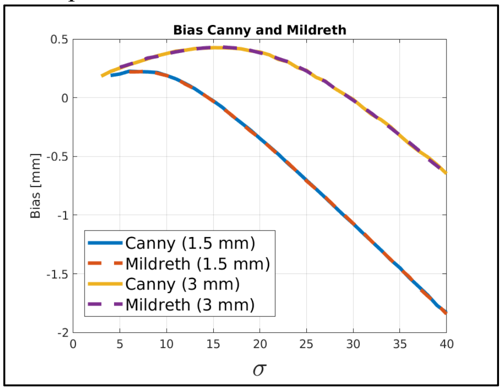
   Insert plot success rate :



   Insert plot standard deviation:



   Insert plot bias:



14. Observe typical and a typical behaviour in the resulting graphs. If possible, explain (put question mark if you can't explain):
    a) Success rate:
    Behaviour for small σ :

We have 0 success rate until sigma around 5, which means that the function fails. If we increase the number of trials, we might succeed more when close to 5, but in general for small sigmas, the function will fail

    Behaviour for large σ :

After 5 we see that the success rate stays a perfect 100% for all sigmas above, meaning that the functions doesn't fail and that it does not depend on the diameter or operator used.

    Explanation behaviour with respect to σ :

When we have low sigmas, the noise will affect the edge detector more. Higher sigmas, and the noise doesn't affect the result as much. Sigma is representing the width of the lines analyzed by the edge detector, so it also depends on the diameter of the blood vessel.

    Behaviour comparing canny and marr-hildreth::

We can see that Canny is better than Marr-Mildreth, with successful estimation in Canny actually appearing before Mildreth for both diameters. So Canny is better equipped to deal with a smaller sigma than what Mildreth is.

    Explanation behaviour with respect to canny and marr-hildreth:

Canny were successful earlier since it's a better method when it comes to how to deal with noise in the edge detection (threshold method). Canny uses nonmaxima suppression as mentioned before, so it can improve the signal as we can get one pixel wide ridges as output.

    Behaviour comparing $D = 1.5$ mm and $D = 3$ mm

When we have D=1.5, we need a slightly higher sigma than what we need for D=3. So a larger diameter allows us to have a smaller sigma and still successfully estimate the diameters

    Explanation behaviour with respect to diameters:

Sigma represents width of the lines being analyzed, so a blood vessel with D=1.55, we can expect it to work with sigmas higher than 1.5 (for example sigma higher than 2). The same with D=3. Here it will be easier to see edges. So, the larger the sigma, the larger the weight of the tail pixels will be. We need a small amount of pixels to have a threshold for D=3 since the difference between inside and outside of vessel is more noticeable.

    b) Standard deviation:
    Behaviour for small σ :

We can see higher standard devaiation for the small sigmas than the rest of the sigmas.

    Behaviour for large σ :

For D=1.5, the standard deviation seems to increase when sigma increases above 20. This is however not the case for D=3, where it looks more to become standing oscillating, though the same could maybe happen if we were to increase sigma even more

Explanation behaviour with respect to σ :

This could be due to the fact that sigma will scale the image so much that the noise from the edges of the image will impact the edge detectors in such a way that they will vary more from column to column that what would be intuitively expected. This means that small deflections in noise/edge would increase so that the standard deviation between each column would become much more varied.

Behaviour comparing canny and marr-hildreth::

We can see that Canny again performs better, with a lower standard deviation for both canny cases.

Explanation behaviour with respect to canny and marr-hildreth:

For Mildreth, we combine Gaussian filtering with Laplacian to break down our agniograms where intensity varies to detect edges. We find the edges and test a wider are around the pixels. Canny operator check local maxima of the gradient to find edges, and this method uses two thresholds to detect the strong and weak edges and only keep those that are connected to each other (making the result better, se hysteresis threshold from before).

Behaviour comparing $D = 1.5$ mm and $D = 3$ mm

We see that with D=1.5, the standard deviation is higher, and that it also increases as sigma increase above 20.

Explanation behaviour with respect to diameters:

The 3 mm standard deviations might be lower because of noise. A smaller diameter means that the noise will make a bigger presence, and will affect the scale of the numbers more, so as we increase sigma, we approach the size of the diameter itself, and then we struggle to distinguish the edges properly

c) Bias:

Behaviour for small σ :

Bias increases for small sigmas, then decreases shortly after. It is however very close to 0

Behaviour for large σ :

For large sigmas, the bias is increasing with a linear rate

Explanation behaviour with respect to σ :

As we increase sigma, we also increase the standard devation, and the mean of the diameter will also be increased more and more as we start to oscillate the actual estimation of the diameters. This means that the bias will also increase

Behaviour comparing canny and marr-hildreth::

Very similar, just that Canny starts slightly before

Explanation behaviour with respect to canny and marr-hildreth:

Here the advantages of Canny isn't that important, since we have a very simple horizontal image where the information from the second derivative doesn't play that much of a role. This explains why Mildreth is able to perform as well as Canny in terms of the bias, and that Canny isn't necessarily that much better in this statistic of the performance of the operators.

Behaviour comparing $D = 1.5$ mm and $D = 3$ mm

D=1.55, bias is larger than for D=3 for every sigma

Explanation behaviour with respect to diameters:

We reach the real diameter with a lower sigma, and as the real diameter stays the same, the mean increase, and it's expected to be as above. Sigma represents the width of the edge detector, so the weight of the tail pixels will increase and than the edge detector finds the edge before the real one. So with larger diameters it's not as affected by increasing sigma as it is with lower diameter.

15. What is/are the disadvantage(s) of using edge detection in this application?

Edge detection doesn't always have a continuous boundary, so the amount of parameters needed will make us tweak a lot for a slightly better result. It's still needed to connect edges to extract complete edges that seem obvious for human inspection. Also due to Gaussian smoothing, location might be off. Hard to detect corners and junctions, the corn pixels looks in the wrong direction for their neighbors, so we get open ended edges.

Insert m-code (note a copy-and- paste of this code to Matlab's native editor, and a single 'run' should suffice to run the whole code):

```
%% Per Arne Kjelsvik s2049201 - Exercise 3
% This exercise uses several given functions and a given picture. Figures
% from this script are saved to a subfolder 'figures'. The given functions
% used in this script are:
%
% get_diameters  - Returns diameter of a horizontally aligned bloodvessel
% im_bloodvessel - Generates simulated X-ray image of a bloodvessel
% ut_edge   - Marr-Hildreth and Canny edge detectors
% ut_gauss  - Apply 2D FIR filter with approx. gaussian PSF
% ut_levelx - Used by ut_edge for finding level crossings

%% PART I
% Init and define variables
close all
clc
clear variables

l = 4; % Constant for calculating L

%% Question 1 - a)
sigma = 5;              % set the width of the Gaussian
L = 2*ceil(sigma*l)+1;  % fill in a constant to define the matrix size
xymax = (L-1)/2;        % the maximum of the x and the y coordinate
xrange = -xymax:xymax;  % the range of x values
yrange = xrange;        % the range of y values
h = fspecial('gaussian', L, sigma); % create the PSF matrix

% visualize as a 3D plot:
% create a RGB matrix to define the colour of the surface plot
C = cat(3, ones(size(h)),ones(size(h)),zeros(size(h)));
% create the surface plot of the gaussian
figure;
surf(xrange,yrange,h,C,'FaceColor','interp','Facelight','phong');
camlight right % add a light at the right side of the scene
xlim([-xymax xymax]); % set appropriate axis limits
ylim([-xymax xymax]);
xlabel('x'); % add axis labels
ylabel('y');
zlabel('h(x,y)');
print -r150 -dpng figures/ex3_1a.png % print the result to file

%% Question 1 - b)
im = imread('ang2.png');    % read the image from file
sigma = [1 10 20 35];       % fill array with sigma values
figure;                     % figure to place the subplots

for i=1:4 % do 4 times:
    L = 2*ceil(sigma(i)*l)+1; % Calculate L for each sigma
    h = fspecial('gaussian', L, sigma(i)); % create the PSF
    imfiltered = imfilter(im,h,'replicate', 'same'); % apply the filter
    subplot(2,2,i); % define a subplot
    imshow(imfiltered,[]); % show the image
    title(['\sigma = ' num2str(sigma(i))]);% include a plot title
end

print -r150 -dpng figures/ex3_1b.png % print the result to file

%% Question 2 - ab)
sigma   = 1.5;                  % set the width of the Gaussian
L       = 2*ceil(sigma*l)+1;    % calculate appropiate L
N       = ceil((L-1)/2);        % get the size of half of the full range
[x,y]   = meshgrid(-N:N,-N:N);  % get coordinates of a 2D orthogonal grid
xymax   = (L-1)/2;              % the maximum of the x and the y coordinate
xrange  = -xymax:xymax;         % the range of x values
yrange  = xrange;               % the range of y values

% Give expressions for the Gaussian and it's y-derivative
h   = exp(-(x.^2 + y.^2)/(2*sigma^2))/(2*pi*sigma^2);
hy  = -(exp(-(x.^2 + y.^2)/(2*sigma^2)).*y)/(2*pi*sigma^4);

% visualize as a 3D plot:
% create a RGB matrix to define the colour of the surface plot
```