a)

b)

c)

d)

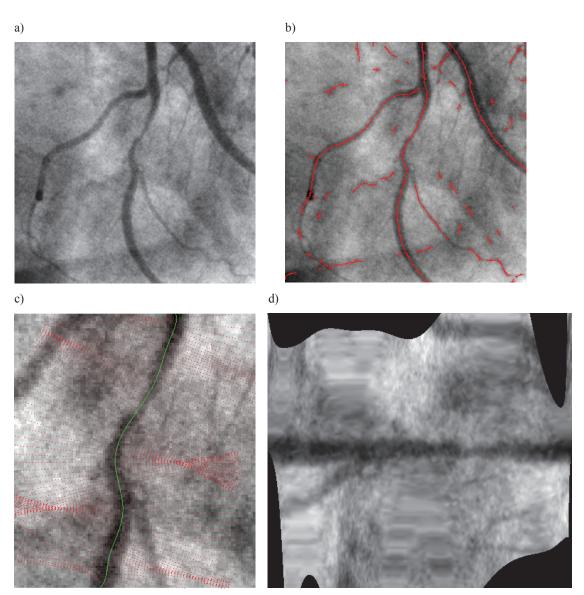**Figure 1 A coronary angiogram**

a) Original.
c) ROI with medial axis (green dots)
   and a grid that is aligned along
   and across the medial axis.

b) Original with lines detected.
d) ROI aligned along the medial axis.

Exercise 3: edge detection and morphological operations

In the following two exercises, we develop and evaluate software for an image processing system that is able to measure the diameters of a blood vessel in coronary angiographic images such as the one shown in Figure 1a. These measurements are useful for the quantitatively assessment of the degree of stenosis (narrowing) to diagnose the cardiac disease. The full image analysis system consists of the following steps:

1. Detection and localization of the blood vessels using a line detector (Figure 1b).
2. Interactive definition of a ROI (region-of-interest) containing the vessel with stenosis (Figure 1c).
3. Alignment of the image along the detected line,
   i.e. the medial axis of the blood vessel, by application of a geometrical transform that is adapted to this medial axis (Figure 1c and d).
4. Edge detection along the blood vessel.
5. Measurement of the nominal diameter.
6. Measurement of the diameter at the stenosis.

Part I: Edge based image segmentation

The purpose of this part is to develop an image processing system that is able to find the contours of the vessel.

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
   figure(1)
   h = fspecial('gaussian', L, sigma);  % create the PSF matrix
   C = cat(3,  ones(size(h)),ones(size(h)),zeros(size(h)));
         % create a RGB matrix to define the colour of the surface plot
   hd =surf(xrange,yrange,h,C,'FaceColor','interp','Facelight','phong');
                                 % create the surface plot of the gaussian
   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
   ```

   Here, 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].

---

[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.

a) Initiate a new Matlab script. That starts with `%% 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. Complete the highlighted lines.

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

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: $\sigma = 1, 5, 10$ and 20. You can use the following template for your code:

```
im = ...                    % read the image from file
sigma = [1 5 10 20];   % 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 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:

Insert the resulting images:

What are the scales (σ^2) of these four images?

2. Derivatives

Consider a continuous image $f(x, y)$ of which only a sampled representation $f(n\Delta, m\Delta)$ is available. Suppose that we are interested in the first derivative $f_x(x, y) = \partial f(x, y)/\partial x$. The purpose is to get a sampled version $f_x(n\Delta, m\Delta)$ of this derivative by processing the digital image $f(n\Delta, m\Delta)$. However, because only a digital representation is available (and not the continuous one) such a derivative can only be approximated. In the continuous case, differentiation is defined within an infinitesimally small neighbourhood of a point. But in the discrete case, this neighbourhood is necessarily extended to a finite neighbourhood. The differentiation is necessarily approximated by a convolution that implements the differentiation combined with some sort of low-pass filtering.

In the continuous domain the combination of differentiation and low-pass filtering is nicely accomplished by:

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

In other words, differentiation of a low-pass filtered image is equivalent to convolving the image with the derivative of the PSF of the low-pass filter. This opens the door to differentiation in the discrete (digital) domain: If $h_x(x, y) = \partial h(x, y)/\partial x$, then differentiation combined with low-pass filtering in the discrete domain is accomplished by a PSF matrix $h_x(n\Delta, m\Delta)$:

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

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

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

Sample code to create the PSF matrix h with a size of L*L (as an alternative to using the function fspecial) is:

```
N = (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 $h(x, y)$:

b) Create a PSF matrix hy that contains a sampled version $h_y(n\Delta, m\Delta)$ of $h_y(x, y)$. Visualize the PSF as in question 1. Implement this with $\sigma = 3\Delta$ (which is equivalent to $\sigma = 3$ and $\Delta = 1$).

c) Use psf2otf to calculate the OTF matrix HY associated to hy. You have to add an appropriate argument to this function such that the size of HY matches the size of the image ang2.png.
    Inspect manually (in the command window of Matlab) whether the real part of HY is zero, or that the imaginary part of HY is zero, or that none are zero. Hint use, for instance, max(real(HY(:))) and max(imag(HY(:))) to check the maximum of the real and imaginary part of the 2D array.

d) Visualize the OTF as a surface plot. If in question 2d you have found that the real part of the OTF is zero, you visualize only the imaginary part. If the imaginary was zero, you visualize only the real part. If both were non-zero, you visualize both in two graphs. **Hints**:
    *   Add the option 'edgecolor','none' to the function surf to suppress the black mesh lines on the surface (the edges).
    *   For visualization, use fftshift so that the origin is in the centre of the graph.
    *   Use xlim and ylim to set appropriate limits to the x- and y axes.

Insert the graph of question 2b:

Insert the graph(s) of question 2d:

What is your findings in question 2c:
- The real part of the OTF is zero: (yes/no)
- The imaginary part of the OTF is zero: (yes/no)

e) Explain the findings of question 2c. **Hint**: find out how the operator will responds to a harmonic function $A\exp(2\pi j(ux+vy))$:

f) Describe and explain the observation of the plotted OTF by answering the following questions: Hints: - Consider the imaginary parts of Hy. - You can substitute by the values of u and v in the Hy equation to check the behaviour.

- How does `Hy` behave at frequencies $v = 0$?

- What is the explanation for that?

- How does `Hy` behave at low frequencies of v, i.e. $v \approx 0$?

- What is the explanation for that?

- How does `Hy` behave at high frequencies of v, i.e. $v \gg 1/\sigma$?

- What is the explanation for that?

g) Apply the convolution with `hy` to the test image.

Implement this convolution in the frequency domain using the OTF given in the matrix `HY`. Insert the resulting image on the right side of this text (use the function `imwrite` and `mat2gray`, rather than a copy of a figure window):

Explain the response of the operator to:
- Areas without much contrast.
- Positive step transitions such as at one of the boundary of the blood vessel.
- Negative step transitions such as at the other boundary of the blood vessel
- What is the influence of σ on these responses?

> ## 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[2] 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 $\sigma = 4$

fx: fy:

fxx: fyy: fxy:

---

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

Knowing that the edge segments are vertically aligned, which of these images convey information about these segments, and what kind of information is that? Which of them do not convey information in this particular case?

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:

    $$\nabla^2 f(n,m) = f_{xx}(n,m) + f_{yy}(n,m)$$

    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.** Use $\sigma = 4$.

    gradient magnitude   Laplacian:

    How can the gradient magnitude on one hand and the Laplacian images on the other hand, be used to locate the edges?

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).

        5a   Thresholded Laplacian: 5b zero crossings Laplacian:

    c) This method for finding the zero crossings introduces a bias (a systematic error). Explain why.

6. Hysteresis thresholding the gradient magnitude

    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. Show the result on the screen. You can interactively select the threshold by visually evaluating the edge map. A useful tool is provided by `imcontrast`.

Found threshold: [   ]

Insert found edge map:

Hysteresis thresholding is an improvement of this simple threshold operation. 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. In the first threshold operation, the threshold is larger than in the second threshold operation. The first map, called the `marker`, may miss many weak edges, but there will be no (or at least only a few) false edges. The second map, called the `mask`, 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 threshold T. Name the resulting image `marker`.

d)   Apply a threshold operation with a threshold that is somewhat smaller than the 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:

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 4. `ut_edge` also contains an implementation of the Canny edge detector.

   a) 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. For now use $\sigma = 3$.
   - 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 them interactively such that two non-broken edge segments arise. Perhaps, you need some iterations to establish the right thresholds.

   Canny edge map: Marr Hildreth edge map:

   Describe the main difference between Marr-Hildreth and Canny edge detectors considering the following aspects: Localization criteria, sensitivity to noise and edge connectedness.

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):