Signal and image processing
Assignment 6

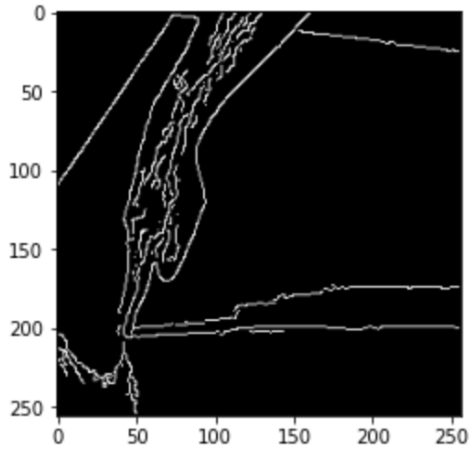
Marcus Hansen - tkz347
Casper Bresdahl - Whs715

March 23, 2021

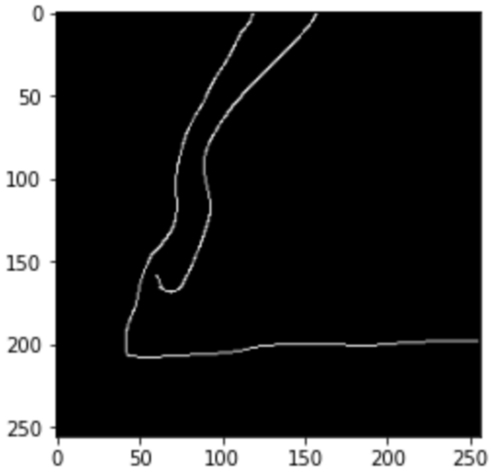
Exercise 1

1.1

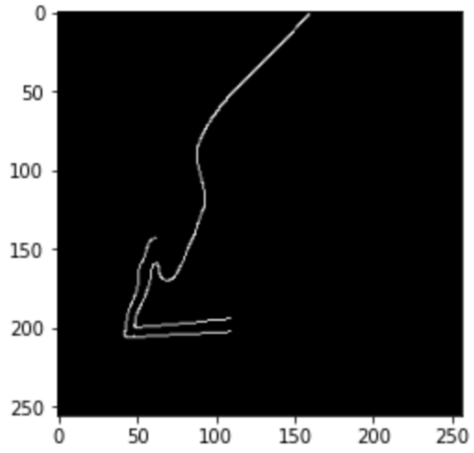
The high threshold parameter determines which pixel intensities we definitely consider edges, as we always keep pixels above this threshold. The low threshold then determines which pixel we want to consider to be edges, as only if a pixel intensity is above the low threshold and connected to / a part of a connection to a pixel above the high threshold it will be considered an edge. Thus, we can see the high threshold as a 'this is definitely an edge' threshold, and the low threshold as a 'candidate' edge threshold. Sigma determines the smoothing of the image. This makes edges disappear if too high, but allows for a more 'coarse' view of the image, i.e. we get the overall structure of the image, rather than all of the details.
In Figure 1 we see the result of using canny on the hand image with different parameters. We note how low blurring in Figure 1a gives a lot of edges compared to high blurring in Figure 1b giving few due to the edges being smoothed away. We also note how few 'definitely edges' and few 'candidate edges' in Figure 1c gives us a lot fewer edges than when we lower the thresholds in Figure 1d.



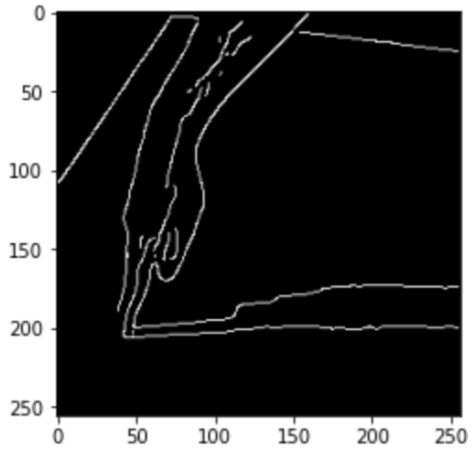
(a) $\sigma = 0.5$, $t_l = 1$, $t_h = 50$.



(b) $\sigma = 5$, $t_l = 1$, $t_h = 50$.



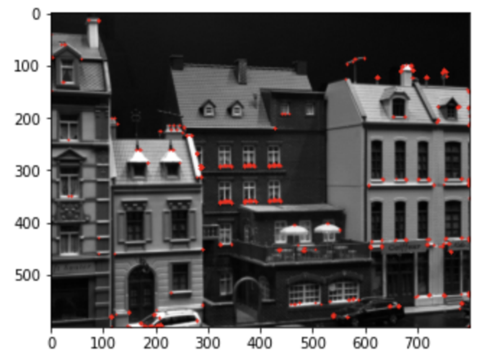
(c) $\sigma = 1.5$, $t_l = 50$, $t_h = 200$.



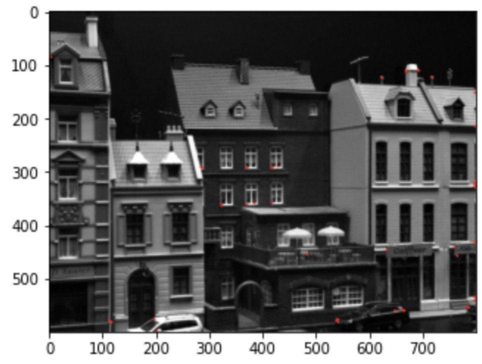
(d) $\sigma = 1.5$, $t_l = 1$, $t_h = 50$.

Figure 1: Different canny parameter settings.

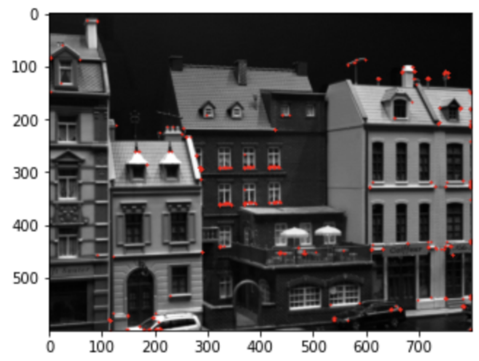
1.2



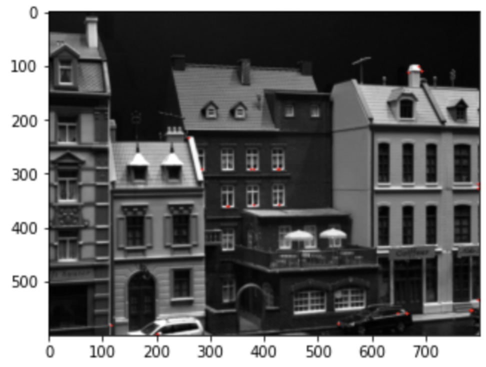
(a) $k = 0.0001$, $\sigma = 0.5$.



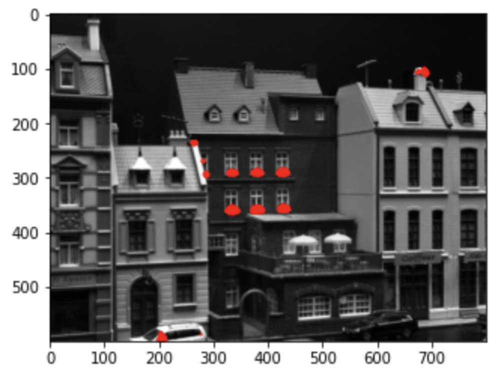
(b) $k = 0.1$, $\sigma = 0.5$.



(c) $\epsilon = 0.01$, $\sigma = 0.5$.



(d) $\epsilon = 5$, $\sigma = 0.5$.



(e) $k = 0.0001$, $\sigma = 5$.

Figure 2: Different parameter settings for the Harris corner detector.

In Figure 2 we see the result of applying the Harris corner detector with different parameters on the houses image. As we decrease k we get more results, however, these results becomes 'worse', i.e. we begin to detect edges. This can be seen in Figure 2a vs Figure 2b where we see a lot more 'detected corners', but some become edges rather than corners. The same is the case for ϵ except we get worse results with higher epsilon values. This can be verified in Figure 2c vs Figure 2d. From the documentation we find that the k and ϵ methods are two different methods to compute the corners, where we can see k as a sensitivity factor to separate edges from corners, and ϵ as a normalization factor. Sigma is the smoothing factor, and if too high it will remove some edges. We can see this when comparing Figure 2a and Figure 2e. However, we also see that we get a lot more hits on top of each other around the windows.

1.3

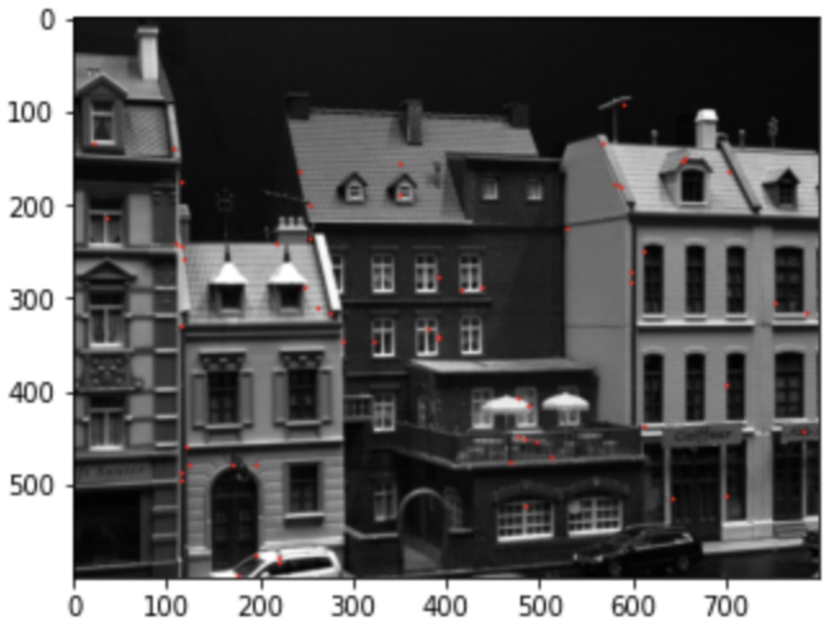
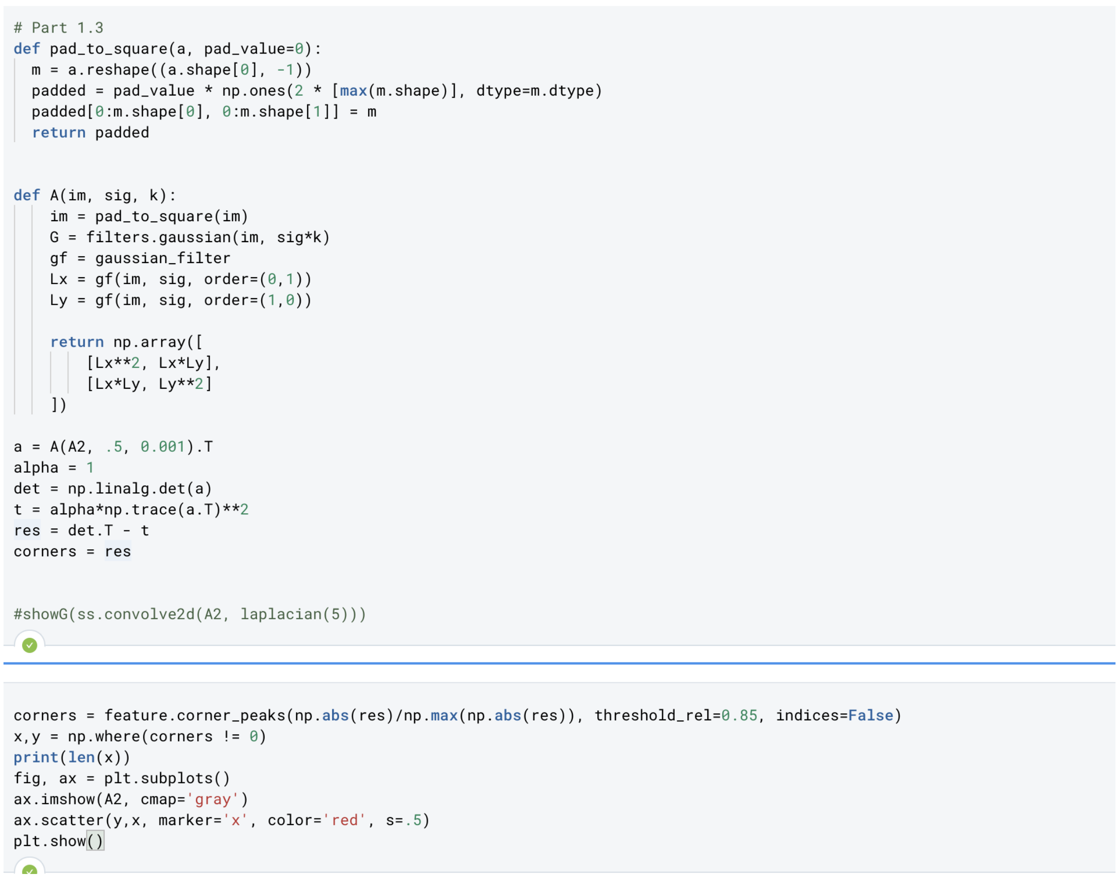


Figure 3: Result of our implementation with $k = 0.001$, $\sigma = 0.5$ and $\alpha = 1$.

In Figure 3 we see the result of our implementation and in Figure 4 we see our code for this exercise. For our implementation we construct the A tensor from the lecture slides with $\sigma = 0.5$ and the $k = 0.001$. We then find the determinant of A and the trace of A and subtract the determinant from the trace multiplied by our $\alpha = 1$. We then use the *skimage.features.corner_peaks* function to find local maxima. This gave us 61 local maxima which can be seen in Figure 3. Most of these seem to be actual corners, however we do get some edges.



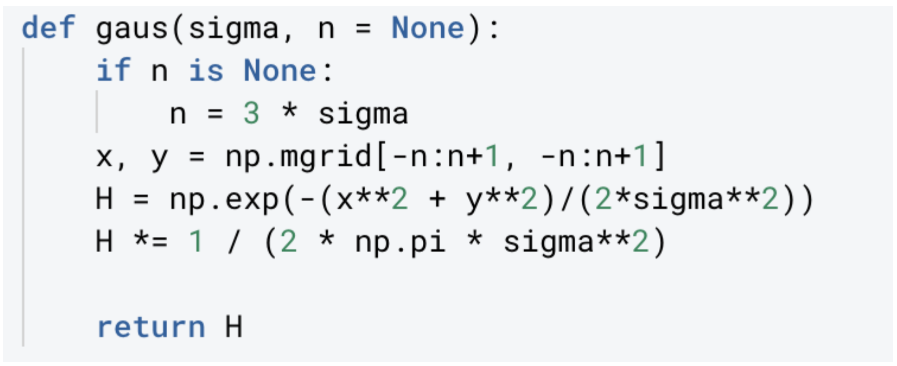
```python
# Part 1.3
def pad_to_square(a, pad_value=0):
    m = a.reshape((a.shape[0], -1))
    padded = pad_value * np.ones(2 * [max(m.shape)], dtype=m.dtype)
    padded[0:m.shape[0], 0:m.shape[1]] = m
    return padded


def A(im, sig, k):
        im = pad_to_square(im)
        G = filters.gaussian(im, sig*k)
        gf = gaussian_filter
        Lx = gf(im, sig, order=(0,1))
        Ly = gf(im, sig, order=(1,0))

        return np.array([
            [Lx**2, Lx*Ly],
            [Lx*Ly, Ly**2]
        ])

a = A(A2, .5, 0.001).T
alpha = 1
det = np.linalg.det(a)
t = alpha*np.trace(a.T)**2
res = det.T - t
corners = res


#showG(ss.convolve2d(A2, laplacian(5)))
```

```python
corners = feature.corner_peaks(np.abs(res)/np.max(np.abs(res)), threshold_rel=0.85, indices=False)
x,y = np.where(corners != 0)
print(len(x))
fig, ax = plt.subplots()
ax.imshow(A2, cmap='gray')
ax.scatter(y,x, marker='x', color='red', s=.5)
plt.show()
```

Figure 4: The code used for this exercise.

Exercise 2

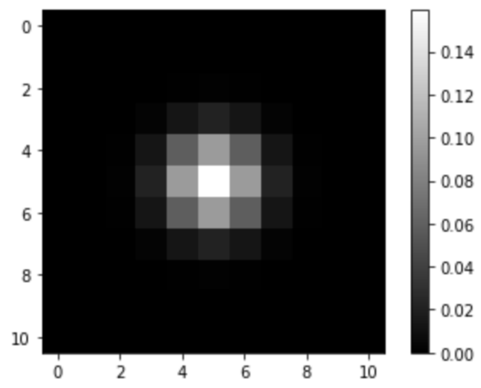
2.1

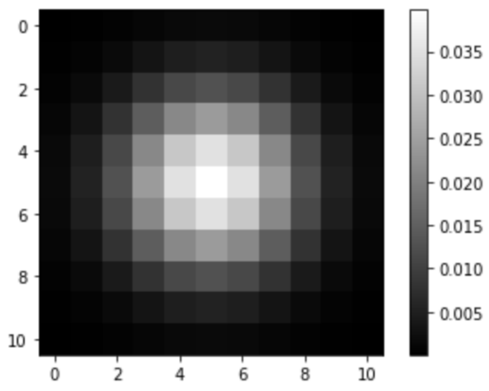


```python
def gaus(sigma, n = None):
    if n is None:
        n = 3 * sigma
    x, y = np.mgrid[-n:n+1, -n:n+1]
    H = np.exp(-(x**2 + y**2)/(2*sigma**2))
    H *= 1 / (2 * np.pi * sigma**2)

    return H
```

Figure 5: Code for exercise 2.1

In Figure 5 we see the code we use to construct Gaussians. We creating a grid over which the Gaussian gets defined, and we then compute the value of the Gaussian for each coordinate in the grid.


(a) Gaussian with $\sigma = 1$.


(b) Gaussian with $\tau = 2$.

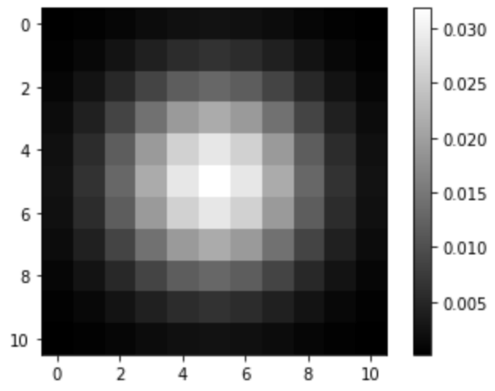
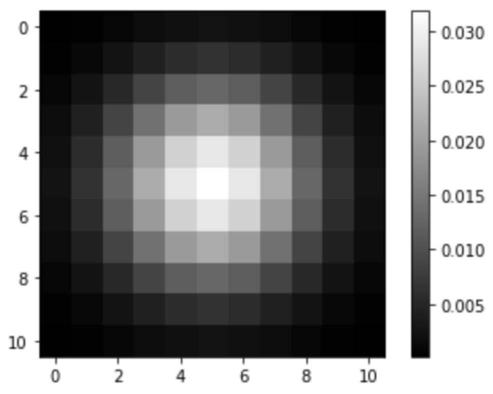
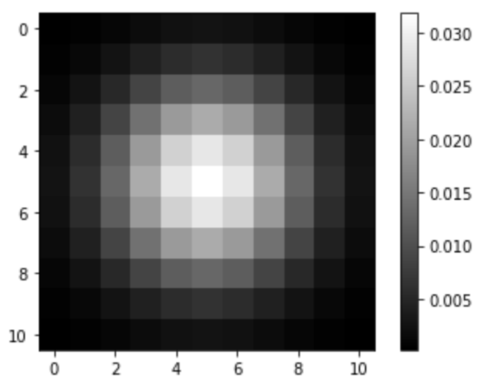

(c) Result of convolving the two Gaussians.

Figure 6: Convolving a Gaussian with a Gaussian results in a Gaussian.

As we can see in Figure 6 when we convolve a Gaussian with another Gaussian we get a new Gaussian. We note however, that the pixel intensities have changed. When comparing the Gaussian with $\tau = 2$ and the Gaussian after convolution we see it has become more flat.


(a) Result of convolving the two Gaussians.


(b) Gaussian made from eq.2.

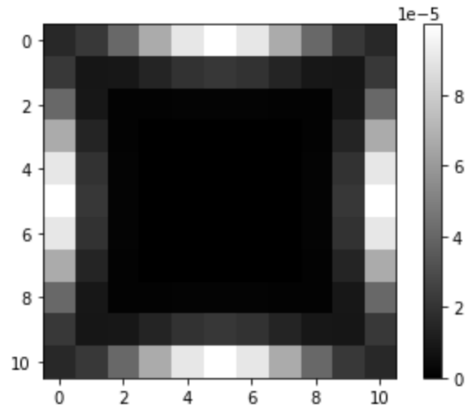

(c) Difference image of the two Gaussians.

Figure 7: The difference between the result of convolving two Gaussians and constructing a Gaussian from eq. 2.

In Figure 7 we see that the result of convolving two Gaussians and constructing a Gaussian from eq. 2 is not the same. At a grid size of 10 we see that the two Gaussians differ around the edges, but are the same in the middle. We also note that these differences are very small (numerically). However, as we increase the grid size (without changing σ and τ) we begin to see differences closer and closer to the center. This might however be due to the discrete representation and how x and y then gets sampled (everything gets very compact).

2.2

Given a Gaussian defined as:

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}}$$

We can now show:

$$\begin{aligned}
G(x,y,\sigma) * G(x,y,\tau) &= \mathcal{F}^{-1}(\mathcal{F}(G(x,y,\sigma))\mathcal{F}(G(x,y,\tau))) \qquad \textit{Convolution theorem} \\
&= \mathcal{F}^{-1}\left(exp\left[-jw(x+y)\right] exp\left[\frac{-\sigma^2 w^2}{2}\right] exp\left[-jw(x+y)\right] exp\left[\frac{-\tau^2 w^2}{2}\right]\right) \\
&= \mathcal{F}^{-1}\left(exp\left[-jw(x+y) - jw(x+y)\right] exp\left[\frac{-\sigma^2 w^2}{2} + \frac{-\tau^2 w^2}{2}\right]\right) \\
&= \mathcal{F}^{-1}\left(exp\left[-jw((x+y)+(x+y))\right] exp\left[\frac{-\left(\sigma^2+\tau^2\right) w^2}{2}\right]\right) \\
&= \mathcal{F}^{-1}\left(exp\left[-jw2x - jw2y\right] exp\left[\frac{-\left(\sigma^2+\tau^2\right) w^2}{2}\right]\right) \\
&= \mathcal{F}^{-1}\left(\sqrt{exp\left[-jw2x - jw2y\right] exp\left[\frac{-\left(\sigma^2+\tau^2\right) w^2}{2}\right]}\right) \\
&= \mathcal{F}^{-1}\left(exp\left[\frac{-jw2x - jw2y}{2}\right] exp\left[\frac{\frac{-\left(\sigma^2+\tau^2\right) w^2}{2}}{2}\right]\right) \\
&= \mathcal{F}^{-1}\left(exp\left[-jw(x+y)\right] exp\left[\frac{\frac{-\left(\sigma^2+\tau^2\right) w^2}{2}}{2}\right]\right) \\
&= G(x,y,\sqrt{\sigma^2+\tau^2})
\end{aligned}$$

2.3

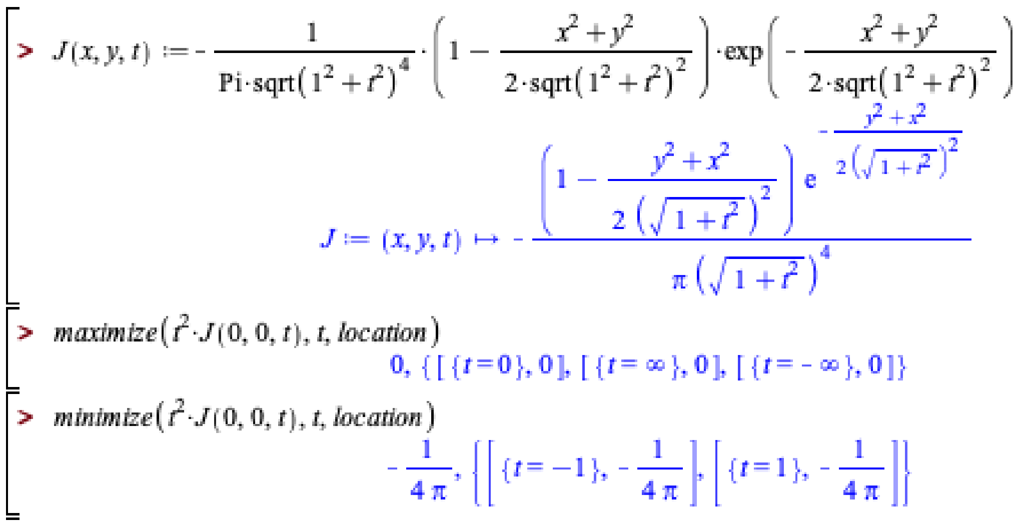
a

Finding a closed form solution for H we can show:

$$\begin{aligned}
H(x,y,\tau) &= \tau^2 \frac{\partial^2 I(x,y,t)}{\partial x^2} + \tau^2 \frac{\partial^2 I(x,y,t)}{\partial y^2} && (1) \\
&= \tau^2 \frac{\partial^2 B(x,y) * G(x,y,\tau)}{\partial x^2} + \tau^2 \frac{\partial^2 B(x,y) * G(x,y,\tau)}{\partial y^2} && (2) \\
&= \tau^2 \frac{\partial^2 G(x,y,\sqrt{\sigma^2+\tau^2})}{\partial x^2} + \tau^2 \frac{\partial^2 G(x,y,\sqrt{\sigma^2+\tau^2})}{\partial y^2} && (3) \\
&= \tau^2 \frac{\partial^4 G(x,y,\sqrt{\sigma^2+\tau^2})}{\partial x^2 \partial y^2} && (4) \\
&= \tau^2 \left(-\frac{1}{\pi \sqrt{1^2+\tau^2}^4}\left(1 - \frac{x^2}{\sqrt{1^2+\tau^2}^2}\right) e^{\left(-\frac{x^2}{\sqrt{1^2+\tau^2}^2}\right)}\right) && (5)
\end{aligned}$$

Where we end up having τ^2 multiplied with the Laplacian of Gaussian.

b



$$> J(x,y,t) := -\frac{1}{\mathrm{Pi}\cdot\mathrm{sqrt}\left(1^2+t^2\right)^4}\cdot\left(1-\frac{x^2+y^2}{2\cdot\mathrm{sqrt}\left(1^2+t^2\right)^2}\right)\cdot\exp\left(-\frac{x^2+y^2}{2\cdot\mathrm{sqrt}\left(1^2+t^2\right)^2}\right)$$

$$J := (x,y,t) \mapsto -\frac{\left(1-\frac{y^2+x^2}{2\left(\sqrt{1+t^2}\right)^2}\right)\mathrm{e}^{-\frac{y^2+x^2}{2\left(\sqrt{1+t^2}\right)^2}}}{\pi\left(\sqrt{1+t^2}\right)^4}$$

$$> \mathit{maximize}\left(t^2\cdot J(0,0,t), t, \mathit{location}\right)$$

$$0, \{[\{t=0\},0], [\{t=\infty\},0], [\{t=-\infty\},0]\}$$

$$> \mathit{minimize}\left(t^2\cdot J(0,0,t), t, \mathit{location}\right)$$

$$-\frac{1}{4\pi}, \left\{\left[\{t=-1\}, -\frac{1}{4\pi}\right], \left[\{t=1\}, -\frac{1}{4\pi}\right]\right\}$$

Figure 8: Deriving the optimal τ for which $H(0,0,\tau)$ is extremal when $\sigma = 1, -1, 0$ with Maple.

Using Maple we found that when $\tau = 0$ we would get a maximum with function value 0, and when $\tau = 1$ or $\tau = -1$ we find a minimum with function value $-\frac{1}{4\pi}$.

c

Plotting $H(0,0,\tau)$ in python we get the results seen in Figure 8.

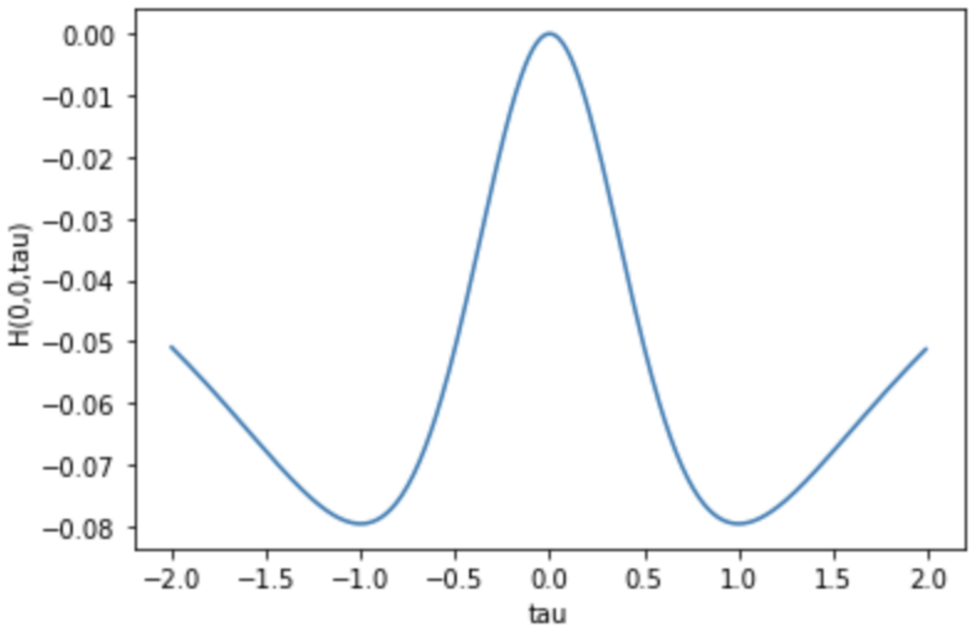


Figure 9: Plot of the value of $H(0,0,\tau)$ over different values of τ, it is easy to see that H is extremal at $\tau = 0$ and $\tau = \pm 1$

We see the results match what we found in the previous exercise, and we have 2 minima at $\tau = \pm 1$ and a maxima at $\tau = 0$.

2.4

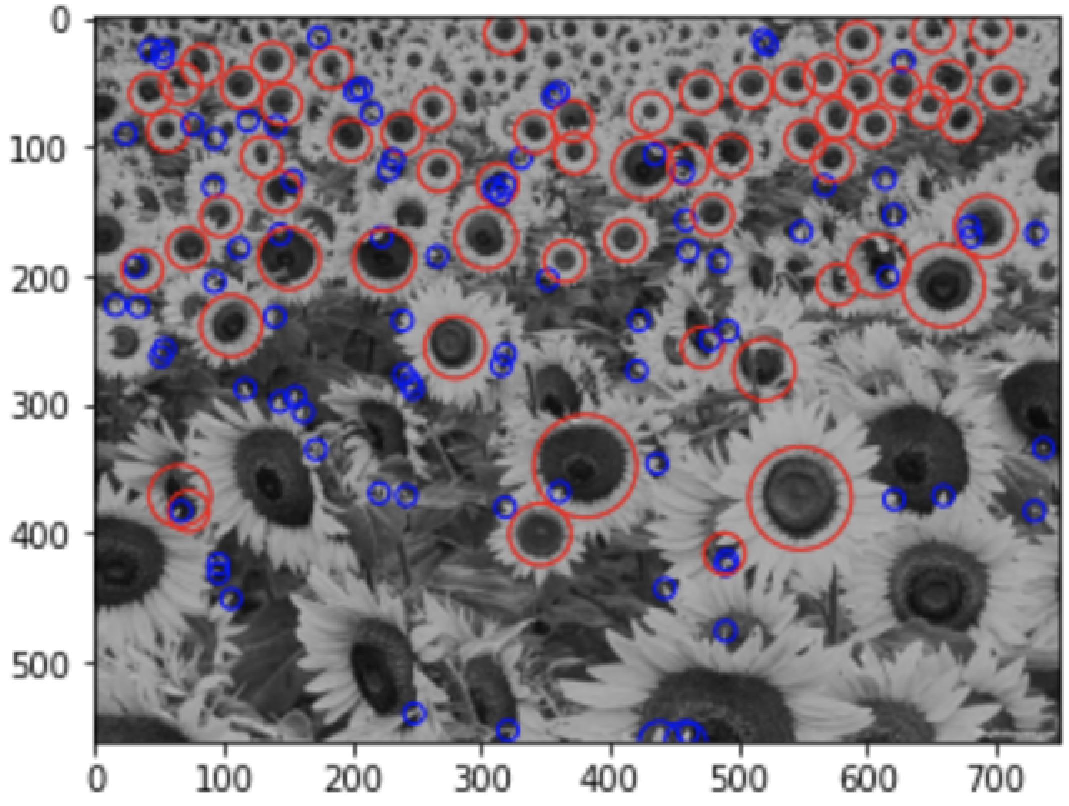


Figure 10: Sunflower image overlaid with blue circles indicating minima and red circles indicating maxima.

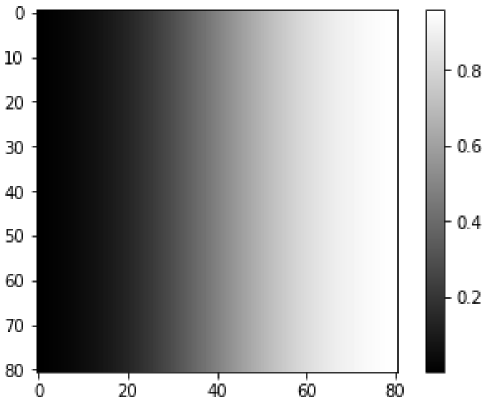
In Figure 10 we see the result of our blob detection. We see the minimas circled with red finding the sunflower heads, and the maxima circled by blue, finding what seems to be petal tips.
In Figure 11 we see the code used for this exercise. We first construct several scalespaces by convolving the sunflower image with a scale normalized Laplacaian of Gaussian (not seen in the image). We then stack all the scalespaces on top of eachother, constructing a 3D structure. Then, for each pixel in this structure, we check the 26-neighbourhood to see if the pixel we are at is a local minimum / maximum. If it is, we save this coordinate along its scale, if not, we simply move to the next pixel. Afterwards we sort the extrema points by absolute value, and we keep the 150 which have the highest. Lastly we simply overlay the sunflower image with circles scaled based on the scale the extrema was found.

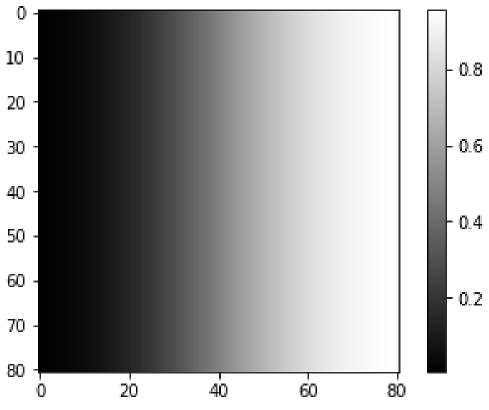
```
extrema = []
copy = np.array(res)
z,y,x = copy.shape
zero = np.zeros((y,x))
print(copy.shape)
stack = np.pad(copy, 1, mode='constant', constant_values=(np.mean(A3/255)))
print(stack.shape)
for i in range(1,z+1):
    for j in range(1,y+1):
        for k in range(1,x+1):
            vs = np.delete(stack[i-1:i+2,j-1:j+2,k-1:k+2], [13])
            if(stack[i,j,k] > np.max(vs)):
                extrema.append([i,j,k, stack[i,j,k]])
            elif(stack[i,j,k] < np.min(vs)):
                extrema.append([i,j,k, stack[i,j,k]])
```

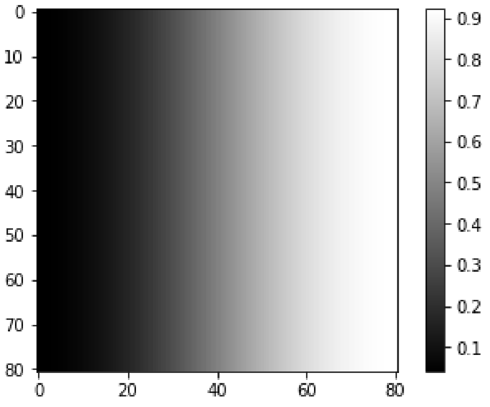
Figure 11: Code used for blob detection.

Exercise 3

3.1


(a) Original image.


(b) $\tau = 1$.


(c) $\tau = 10$.

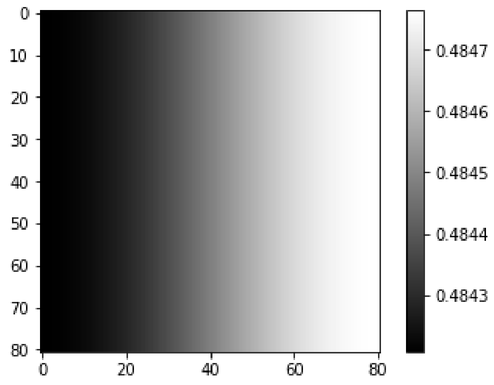
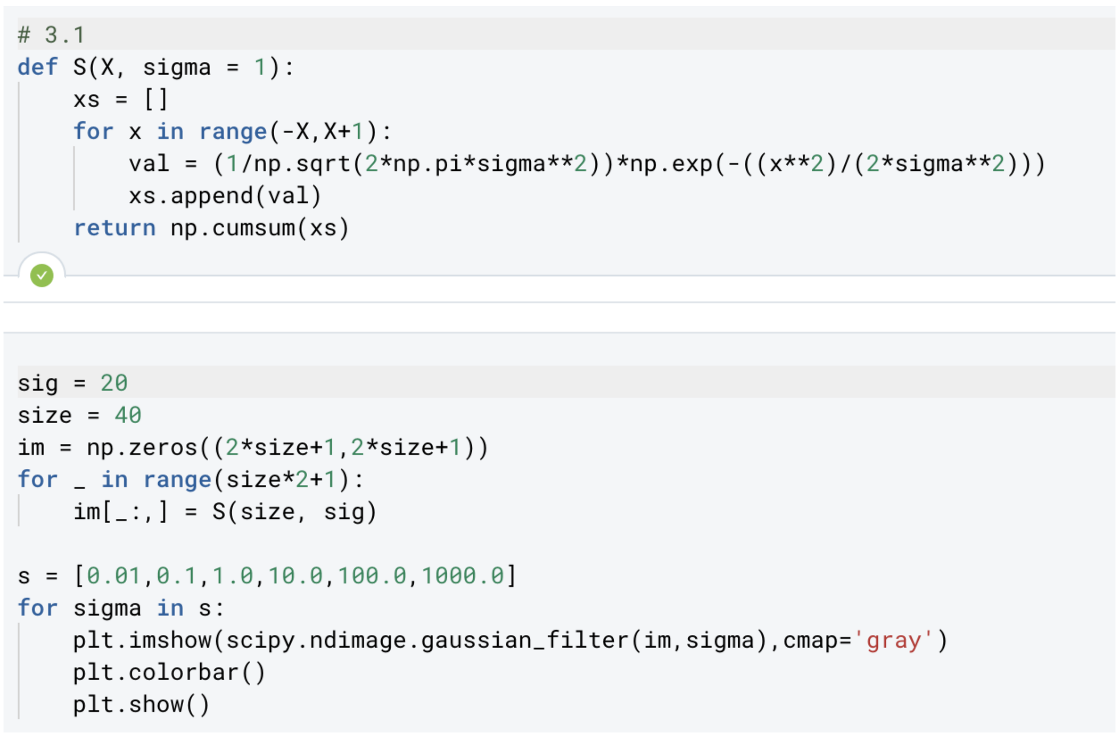

(d) $\tau = 100$.

Figure 12: Images of the S function with $\sigma = 20$ convolved with a Gaussian using different values of τ.

In Figure 13 we see the code used for this exercise. Convolving the S function with Gaussians with different τ values yields the results seen in Figure 12. We note although the images looks similar, that the maximal values gets smaller as τ increases.



```python
# 3.1
def S(X, sigma = 1):
    xs = []
    for x in range(-X,X+1):
        val = (1/np.sqrt(2*np.pi*sigma**2))*np.exp(-((x**2)/(2*sigma**2)))
        xs.append(val)
    return np.cumsum(xs)
```

```python
sig = 20
size = 40
im = np.zeros((2*size+1,2*size+1))
for _ in range(size*2+1):
    im[_:,] = S(size, sig)

s = [0.01,0.1,1.0,10.0,100.0,1000.0]
for sigma in s:
    plt.imshow(scipy.ndimage.gaussian_filter(im,sigma),cmap='gray')
    plt.colorbar()
    plt.show()
```

Figure 13: Code for calculating S and the code for convolving S with a Gaussian and creating the images above

3.2

a

We define

$$h(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Where σ is some constant. We can now show:

$$\tau^{2\cdot\frac{1}{2}} \left(\frac{\partial J(x,y,\tau)}{\partial x}\right)^2 + \tau^{2\cdot\frac{1}{2}} \left(\frac{\partial J(x,y,\tau)}{\partial y}\right)^2 \tag{6}$$

$$= \tau \left(\frac{\partial S(x,y) * G(x,y,\tau)}{\partial x}\right)^2 + \tau \left(\frac{\partial S(x,y) * G(x,y,\tau)}{\partial y}\right)^2 \tag{7}$$

$$= \tau \left(\frac{\partial S(x,y)}{\partial x} * G(x,y,\tau)\right)^2 + \tau \left(\frac{\partial S(x,y)}{\partial x} * G(x,y,\tau)\right)^2 \tag{8}$$

$$= \tau \left(\frac{\partial}{\partial x}\left(\int_{-\infty}^{x} h(x,y)dx\right) * G(x,y,\tau)\right)^2 + \tau \left(\frac{\partial}{\partial y}\left(\int_{-\infty}^{x} h(x,y)dx\right) * G(x,y,\tau)\right)^2 \tag{9}$$

$$= \tau \left(\left(\int_{-\infty}^{x} \frac{\partial}{\partial x} h(x,y)dx + h(x,x)\cdot\frac{\partial}{\partial x}x - h(x,-\infty)\cdot\frac{\partial}{\partial x} - \infty\right) * G(x,y,\tau)\right)^2 + \tag{10}$$

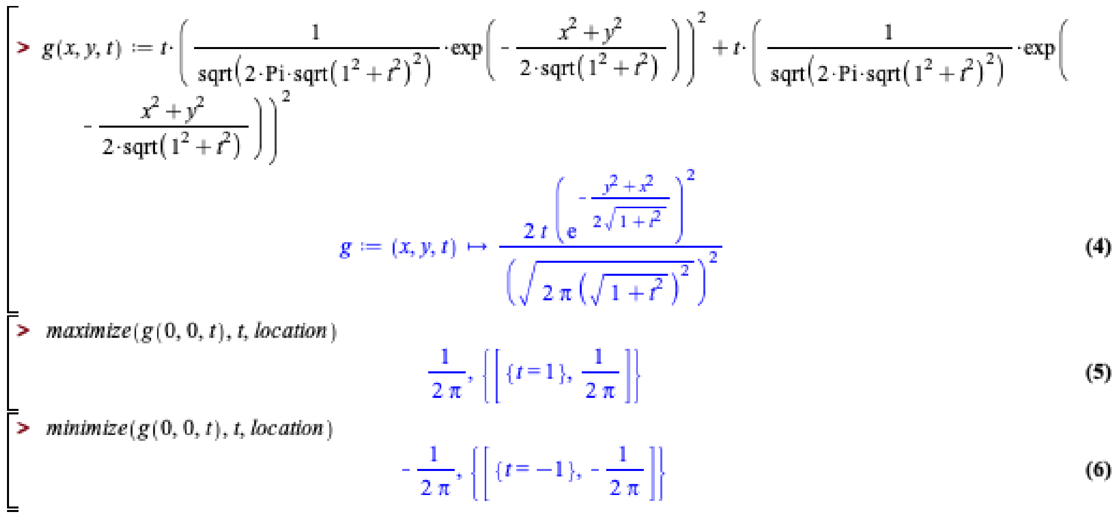
$$\tau \left(\left(\int_{-\infty}^{x} \frac{\partial}{\partial y} h(x,y)dx + h(x,x)\cdot\frac{\partial}{\partial x}x - h(x,-\infty)\cdot\frac{\partial}{\partial x} - \infty\right) * G(x,y,\tau)\right)^2 \tag{11}$$

$$= \tau \left(h(x,x) * G(x,y,\tau)\right)^2 + \tau \left(h(x,x) * G(x,y,\tau)\right)^2 \tag{12}$$

$$= 2\tau \left(G(x,y,\sqrt{\sigma^2+\tau^2})\right)^2 \tag{13}$$

In step (8) we know we can move the partial derivative to S by taking the Fourier transform of the convolution and rearrange the terms. In (10) we make use of the fundamental theorem of calculus, and see all terms become zero except $h(x,x)\cdot\frac{\partial}{\partial x}$, and thus we get (11). We note h does not depend on its second argument, and thus $h(x,x) = h(x,y)$.

b



```
> g(x, y, t) := t·(1/sqrt(2·Pi·sqrt(1^2 + t^2)^2)·exp(-(x^2 + y^2)/(2·sqrt(1^2 + t^2))))^2 + t·(1/sqrt(2·Pi·sqrt(1^2 + t^2)^2)·exp(-(x^2 + y^2)/(2·sqrt(1^2 + t^2))))^2
```

$$g := (x, y, t) \mapsto \frac{2\, t \left(e^{-\frac{y^2+x^2}{2\sqrt{1+t^2}}}\right)^2}{\left(\sqrt{2\,\pi \left(\sqrt{1+t^2}\right)^2}\right)^2} \qquad \textbf{(4)}$$

```
> maximize(g(0, 0, t), t, location)
```

$$\frac{1}{2\,\pi}, \left\{\left[\{t = 1\}, \frac{1}{2\,\pi}\right]\right\} \qquad \textbf{(5)}$$

```
> minimize(g(0, 0, t), t, location)
```

$$-\frac{1}{2\,\pi}, \left\{\left[\{t = -1\}, -\frac{1}{2\,\pi}\right]\right\} \qquad \textbf{(6)}$$

Figure 14: Deriving the optimal τ for which $H(0,0,\tau)$ is maximized when $\sigma = 1$ and minimized at $\sigma = -1$ with Maple.

With Maple we found that when $\tau = 1$ we would get a maxima, and when $\tau = -1$ we get a minima with function values $\pm\frac{1}{2\pi}$. When τ goes to $\pm\infty$ we get function values going towards 0, thus $\tau = 1$ is a global maximum in scale-space.

c

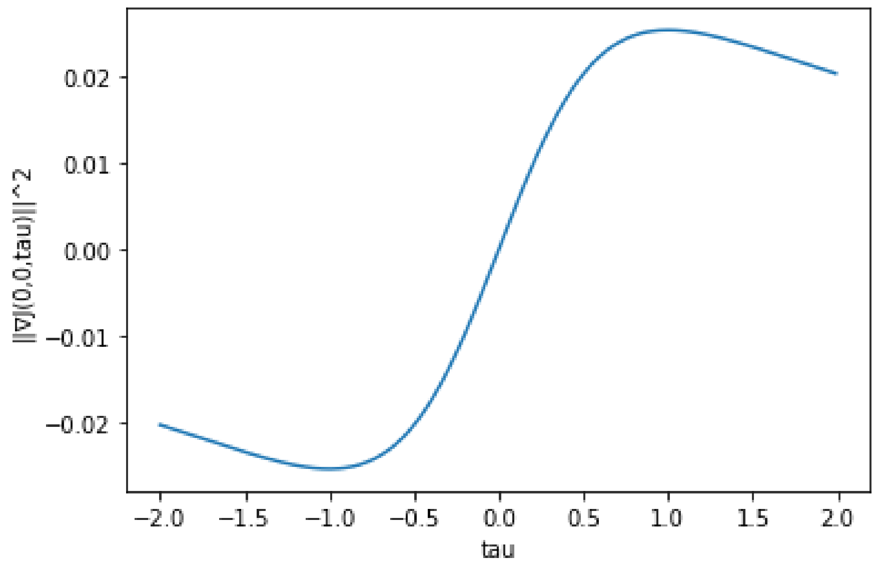


Figure 15: Plot of the value of $\nabla J(0,0,\tau)$ over different values of τ, it is easy to see that J is extremal at $\tau = \pm 1$

When plotting in python we can confirm our findings from the previous exercise seen in Figure 15.

3.3

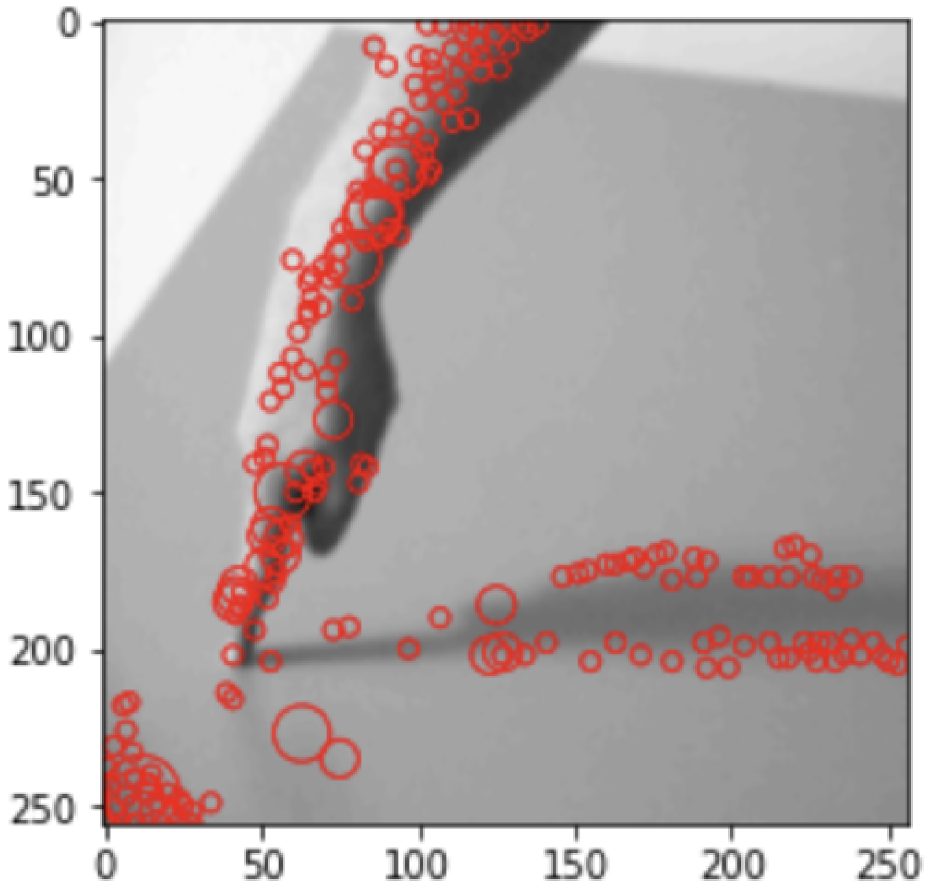
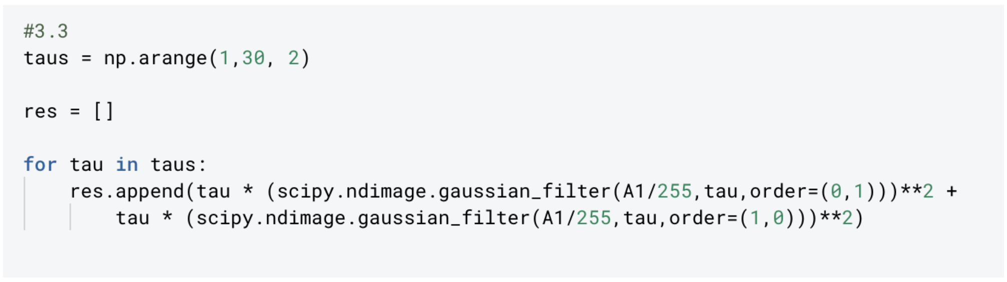


Figure 16: The 200 maxima with the largest value plotted as circles with radiuses depending on their detected scale.

In Figure 16 we see the result of plotting the 200 maxima with the greatest value as circles with radiuses depending on their detected scale. The code used is the same as in exercise 2.4 and can be seen in Figure 11 expect we convolve the image with the first derivative of the Gaussian and square the result as can be seen in Figure 17.



```
#3.3
taus = np.arange(1,30, 2)

res = []

for tau in taus:
    res.append(tau * (scipy.ndimage.gaussian_filter(A1/255,tau,order=(0,1)))**2 +
        tau * (scipy.ndimage.gaussian_filter(A1/255,tau,order=(1,0)))**2)
```

Figure 17: The code used to convolve the hand image.