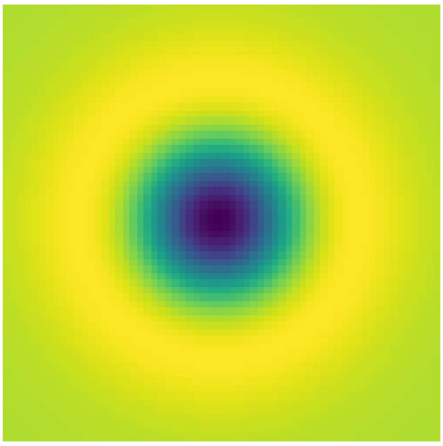
Index number : 190026T

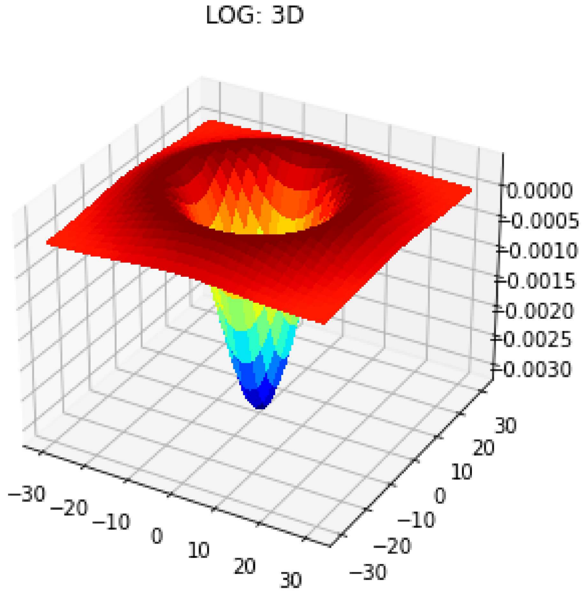
Name : AHAMED M.I.I

In [ ]:

```
#blobs

#1)

import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

sigma = 10
delta = 1
X, Y = np.meshgrid(np.arange(-30, 31 + delta, delta), np.arange(-30, 31 + delta, del

gaussian = np.exp(-(X**2 + Y**2)/(2*sigma**2))

LOG = gaussian*(X**2/sigma**2 + Y**2/sigma**2 - 2)/(2*np.pi*sigma**4)
LOG = LOG*sigma**2 #scale normalize LOG

fig, ax = plt.subplots()
ax.imshow(LOG)
ax.title.set_text('LOG: 2D')
ax.axis('off')
ax.xaxis.tick_top()

fig1, ax = plt.subplots(subplot_kw={"projection": "3d"}, figsize=(5,5))
ax.title.set_text('LOG: 3D')
surf = ax.plot_surface(X, Y, LOG, cmap=cm.jet,
                       linewidth=0, antialiased=False)

del gaussian; del LOG; del X; del Y
```

LOG: 2D



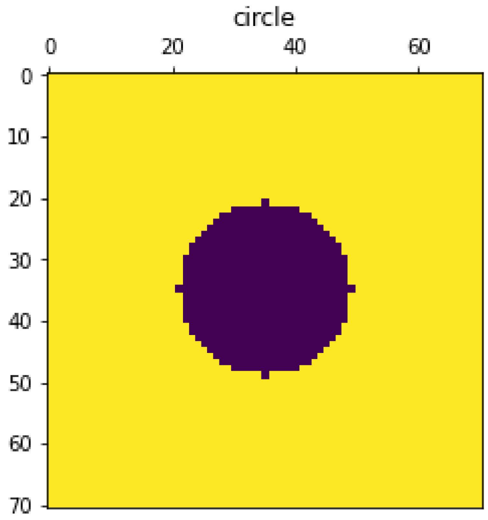


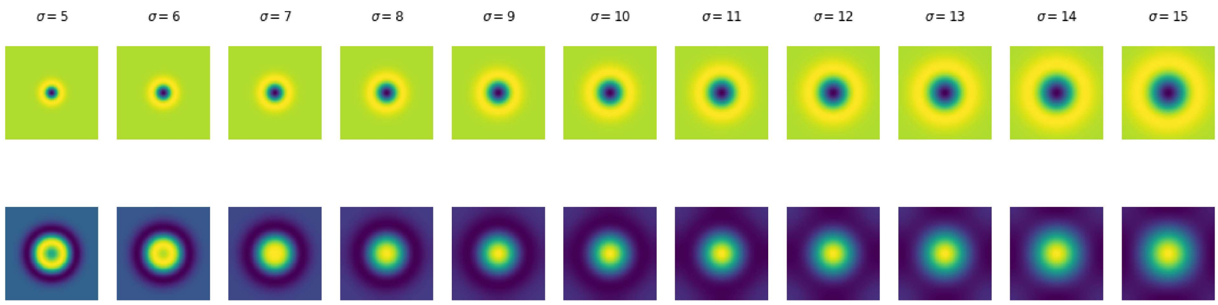
In [ ]:

```
#2)
width = 71
height = 71

hw = width//2
hh = height//2

im = np.ones((height, width), dtype=np.float32)*255
delta = 1
X, Y = np.meshgrid(np.arange(-hh, hh + delta, delta), np.arange(-hw, hw + delta, del

radius = width//5

im *= X**2 + Y**2 > radius**2

fig, ax = plt.subplots()
ax.imshow(im)
ax.title.set_text('circle')
ax.xaxis.tick_top()

scale = 11
scale_space = np.empty((height, width, scale), dtype=np.float32)

fig, ax = plt.subplots(2, scale, figsize=(20,5))
sigmas = np.arange(5,16,1)
for i, sigma in enumerate(sigmas):
    log_hw = 3*np.max(sigmas)

    delta = 1
    X, Y = np.meshgrid(np.arange(-log_hw, log_hw + delta, delta), np.arange(-log_hw,
    gaussian = np.exp(-(X**2 + Y**2)/(2*sigma**2))
    LOG = gaussian*(X**2/sigma**2 + Y**2/sigma**2 - 2)/(2*np.pi*sigma**4)
    LOG = LOG*sigma**2 #scale normalize LOG

    im_log = cv.filter2D(im, -1, LOG)
    scale_space[:, :, i] = im_log

    ax[0, i].imshow(LOG)
    ax[0, i].title.set_text(r'$\sigma = $'+str(sigma))
    ax[0, i].axis('off')
    ax[0, i].xaxis.tick_top()

    ax[1, i].imshow(im_log)
```

```
    ax[1, i].axis('off')
    ax[1, i].xaxis.tick_top()



indices = np.unravel_index(np.argmax(scale_space, axis=None), scale_space.shape)
# print(indices)
print('scale space extremum = ',sigmas[indices[2]])

del sigmas; del scale_space; del im; del X; del Y
del LOG; del im_log; del log_hw; del gaussian
```

```
scale space extremum =  10
```





scale space extremum = 10, that is the maximum response occure when $\sigma = 10$.

This also goes along with,

$$\sigma = \frac{r}{\sqrt{2}}$$

$(r = 14)$

$$\frac{r}{\sqrt{2}} = \frac{14}{\sqrt{2}} = 9.89 \approx 10 = \text{scale space extremum}$$

In [ ]:

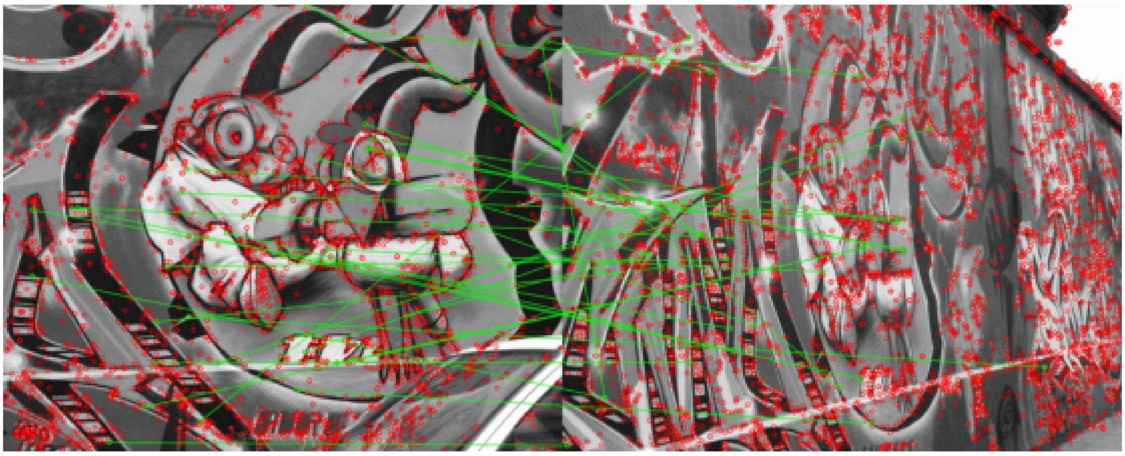
```
#3
img1 = cv.imread(r'E:\Aca\aca sem 4\Image Processing & Machine vision\exercises\exer
assert img1 is not None

img2 = cv.imread(r'E:\Aca\aca sem 4\Image Processing & Machine vision\exercises\exer
assert img2 is not None

fig, ax = plt.subplots(2, 1, figsize=(5,10))
ax[0].imshow(img1, cmap = 'gray', vmin =0, vmax=255)
ax[0].title.set_text('image 1')
```

```
ax[0].axis('off')
ax[0].xaxis.tick_top()

ax[1].imshow(img2, cmap = 'gray', vmin =0, vmax=255)
ax[1].title.set_text('image 2')
ax[1].axis('off')
ax[1].xaxis.tick_top()

# Initiate SIFT detector
sift = cv.SIFT_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)   # or pass empty dictionary
flann = cv.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in range(len(matches))]

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0),
                   matchesMask = matchesMask,
                   flags = cv.DrawMatchesFlags_DEFAULT)

img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)

fig, ax = plt.subplots(figsize=(10,10))
ax.imshow(img3, cmap = 'gray', vmin =0, vmax=255)
ax.title.set_text('features matched')
ax.axis('off')
ax.xaxis.tick_top()
```

image 1

image 2

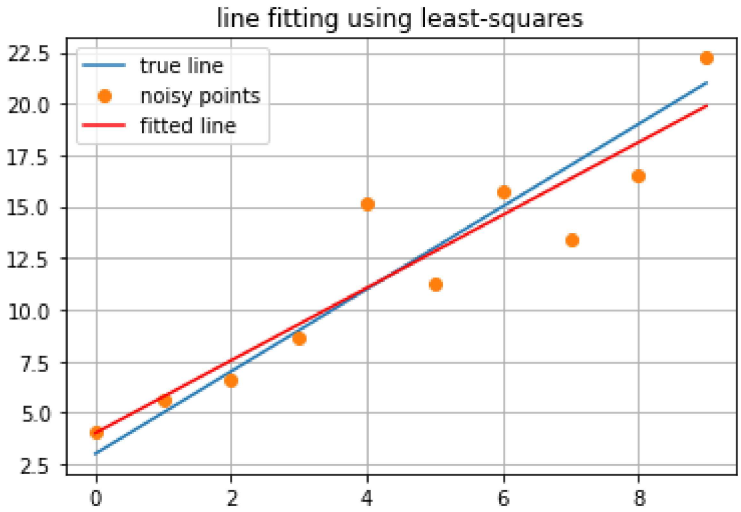
features matched



In [ ]:

```
#4)
# Lineequation : y = m*x + c , m is the slope, c is the intercept
m = 2
c = 1
x = np.arange(1 ,11 , 1)
n = 2.*np.random.randn(len(x))
o = np.zeros(x.shape)
# o [=1] = 20
y = m*x + c + n + o
```

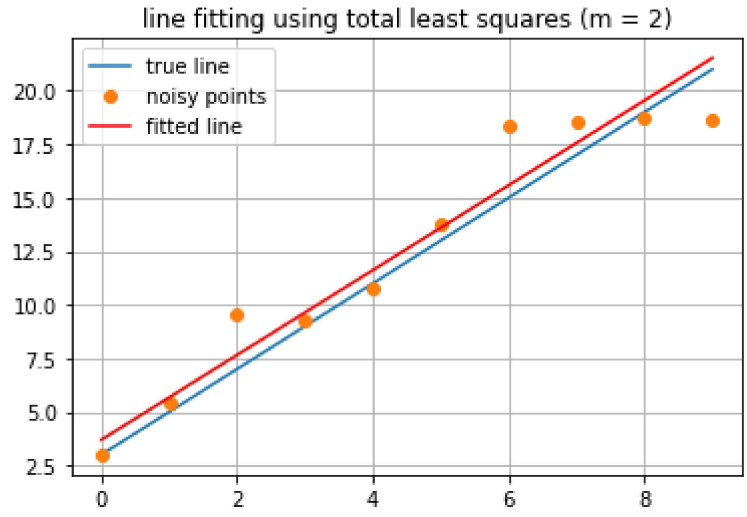
```
fig, ax = plt.subplots()
ax.plot(m*x + c, label='true line')
ax.plot(y, 'o', label='noisy points')
ax.grid(True)
ax.title.set_text('line fitting using least-squares')

X = np.concatenate([x.reshape(10,1), np.ones((10, 1))], axis=1)
Y = y.reshape(10, 1)
B = np.linalg.pinv(np.transpose(X) @ X) @ np.transpose(X) @ y

m_fit = B[0]
c_fit = B[1]
y_fit = x*m_fit + c_fit

ax.plot(y_fit, color='r', label='fitted line')
ax.legend();
```

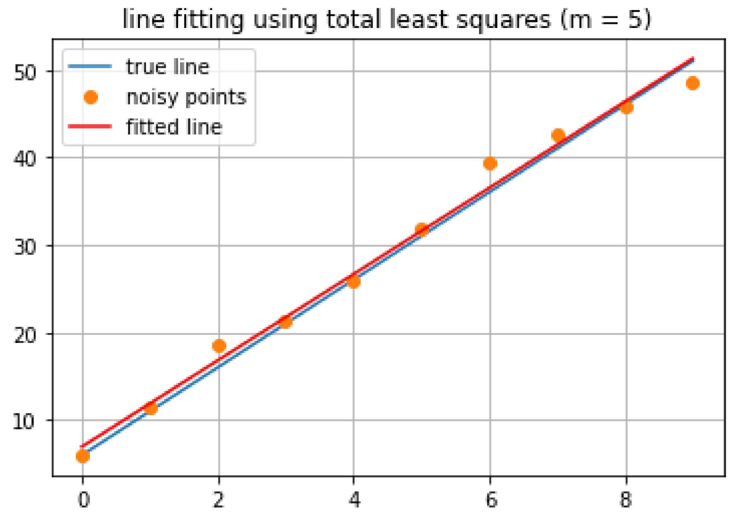


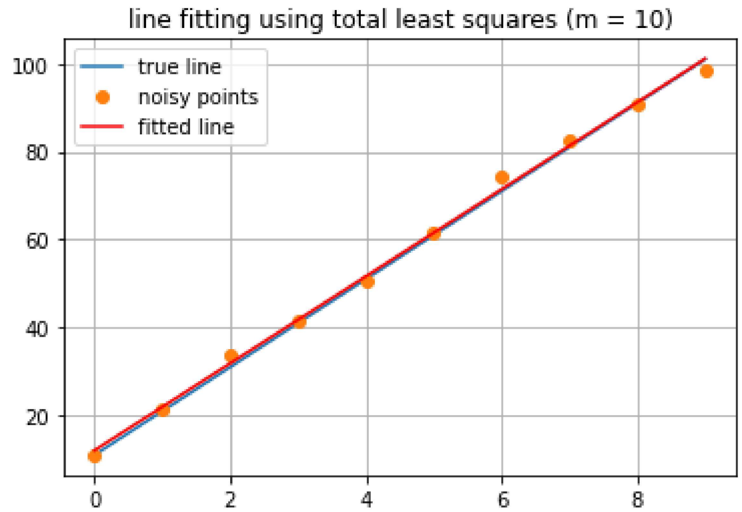
In [ ]:

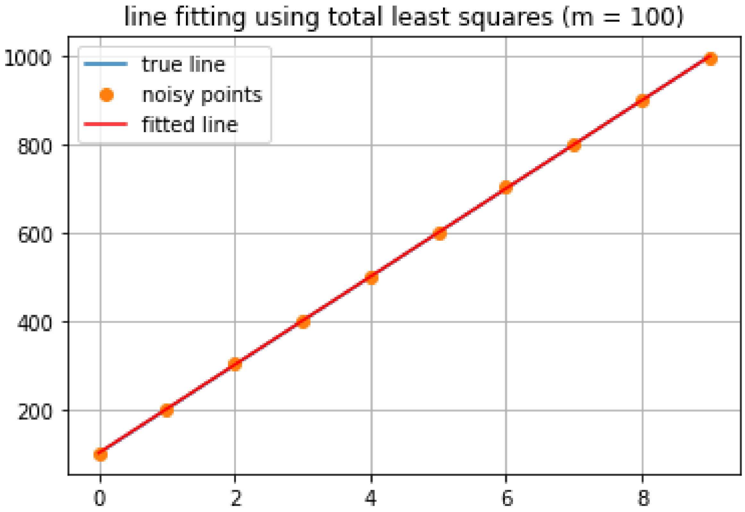
```
#5)
# Lineequation : y = m*x + c , m is the slope, c is the intercept
M = [2, 5, 10, 100]
c = 1
x = np.arange(1 ,11 , 1)
n = 2.*np.random.randn(len(x))
o = np.zeros(x.shape)
# o [=1] = 20
for m in M:
    y = m*x + c + n + o

    fig, ax = plt.subplots()
    ax.plot(m*x + c, label='true line')
    ax.plot(y, 'o', label='noisy points')
    ax.grid(True)
    ax.title.set_text('line fitting using total least squares (m = '+str(m)+')')

    u11 = np.sum((x - np.mean(x))**2)
    u12 = np.sum((x - np.mean(x))*(y - np.mean(y)))
    u21 = u12
    u22 = np.sum((y - np.mean(y))**2)

    U = np.array([[u11, u12],
                  [u21, u22]])
    W, V = np.linalg.eig(U)

    ev_correspoding_to_smallest_ev = V[:, np.argmin(W)]

    a = ev_correspoding_to_smallest_ev[0]
```

```
    b = ev_correspoding_to_smallest_ev[1]
    d = a*np.mean(x) + b*np.mean(y)

    m_fit = -a/b
    c_fit = d/b
    y_fit = x*m_fit + c_fit

    ax.plot(y_fit, color='r', label='fitted line')
    ax.legend();
```








line fitting using total least squares (m = 100)
true line
noisy points
fitted line
1000
800
600
400
200
0
2
4
6
8