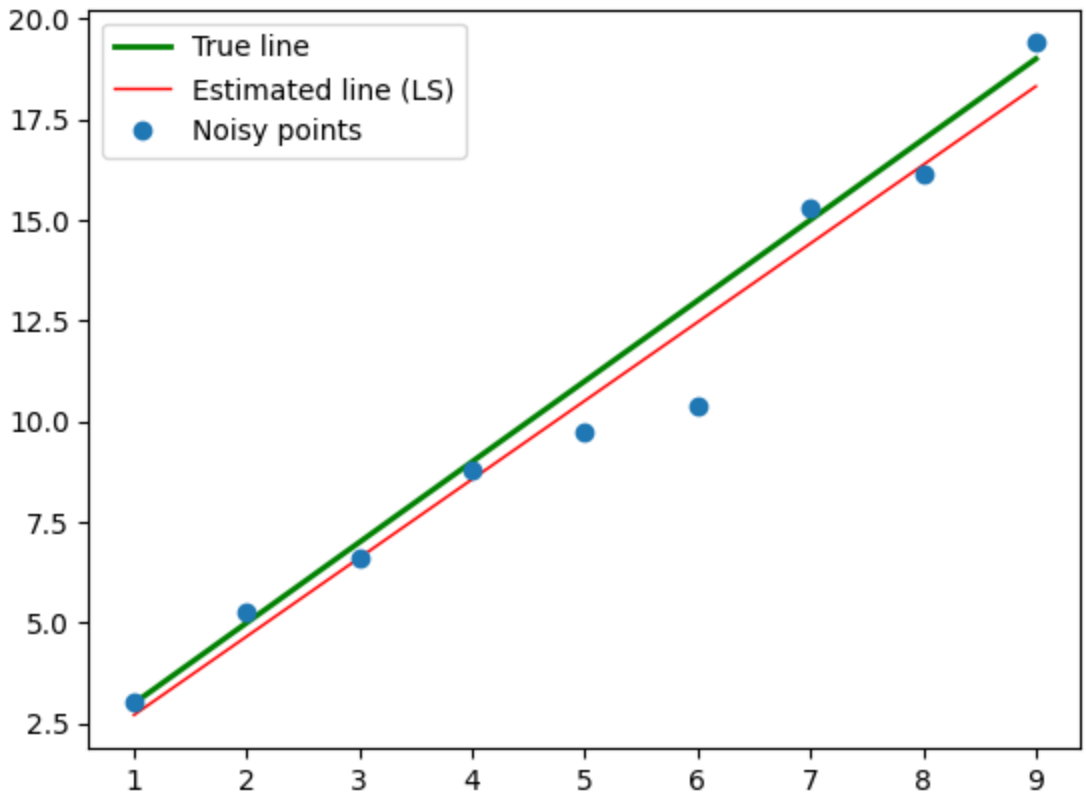
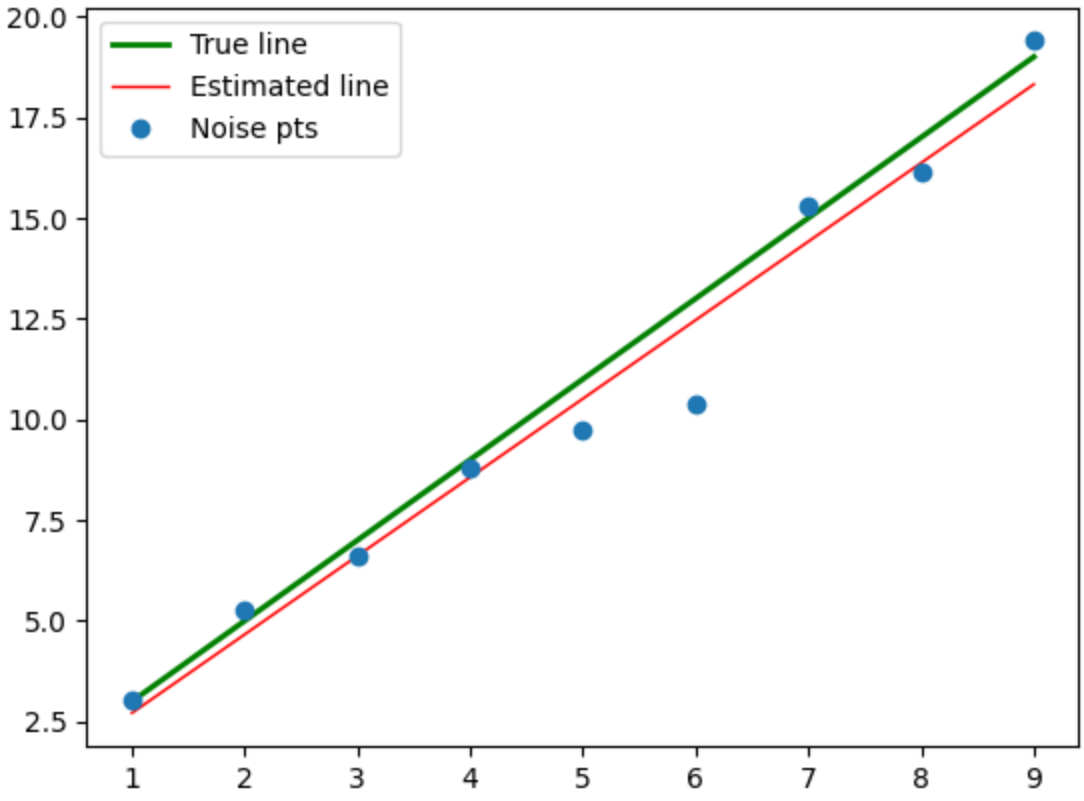
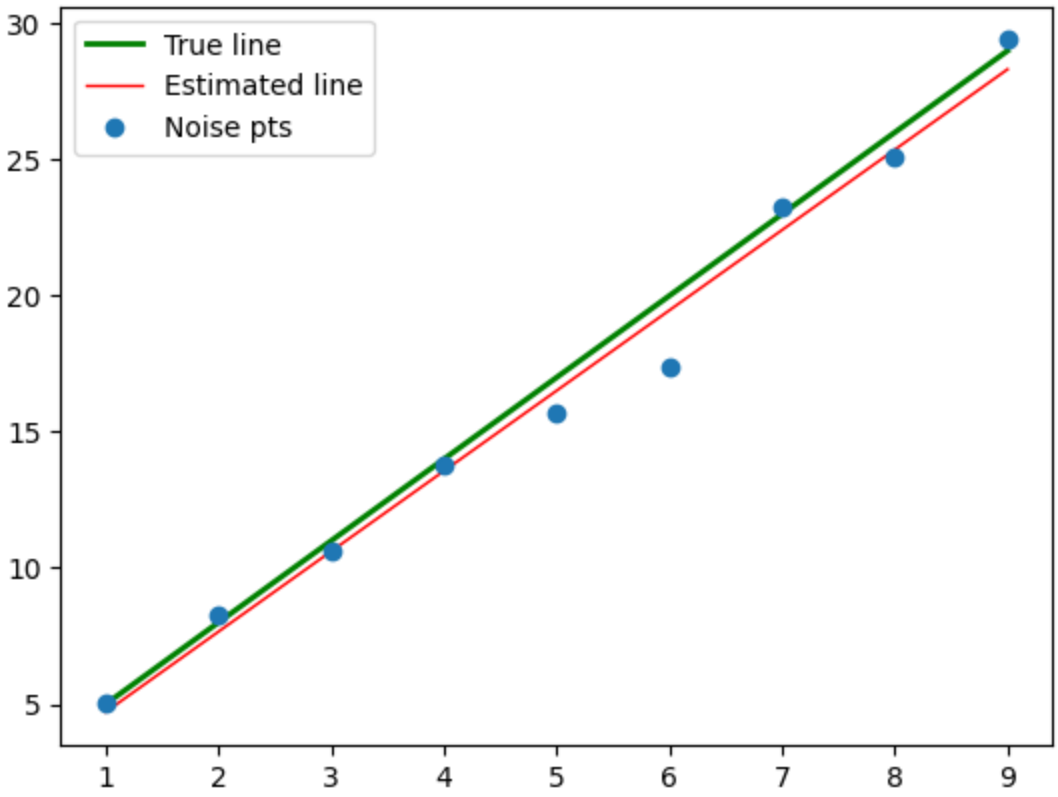
In [ ]:
```
# Question 01
# Least-squares line fitting
import numpy as np
import matplotlib.pyplot as plt

# Genrating the true line y = m*x + c
m = 2 # gradient
c = 1 # intercept
x = np.arange(1,10, 1) # creating a array 1-9 with stepsize of 1
np.random.seed(45) # The noise is added to the y-values using numpy.random.randn()
no = np.random.randn(len(x)) #Noise
o = np.zeros(x.shape) #o is initialized to an array of zeros with the same shape as
#o[1] = 20 #outliers
y = m*x + c + no + o #This line calculates the noisy y-values by adding the true li
n = len(x)
X = np.concatenate([x.reshape(n,1), np.ones((n, 1))], axis=1) #The numpy.concatenat
B = np.linalg.pinv(X.T @ X) @ X.T @ y
mstar = B[0] #we obtain the coefficients B of the estimated line in the form of a c
cstar = B[1]
#The numpy.linalg.pinv() function calculates the pseudo-inverse of the matrix X.T @
#Finally, the estimated slope and intercept are extracted from B.
plt.plot([x[0], x[-1]], [m*x[0] + c, m*x[-1] + c], color='g', linewidth=2, label=r'
plt.plot([x[0], x[-1]], [mstar*x[0] + cstar, mstar*x[-1] + cstar], color='r', linew
plt.plot(x,y, 'o', label='Noisy points')
plt.legend(loc='best')
plt.show()
```

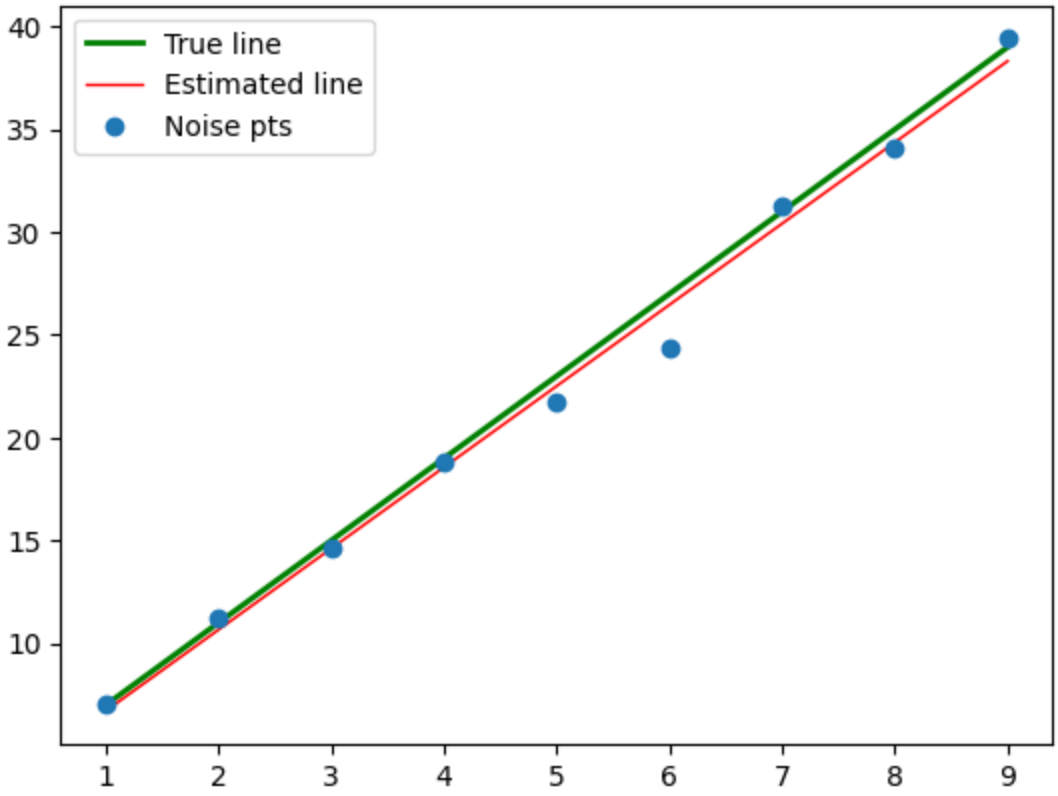


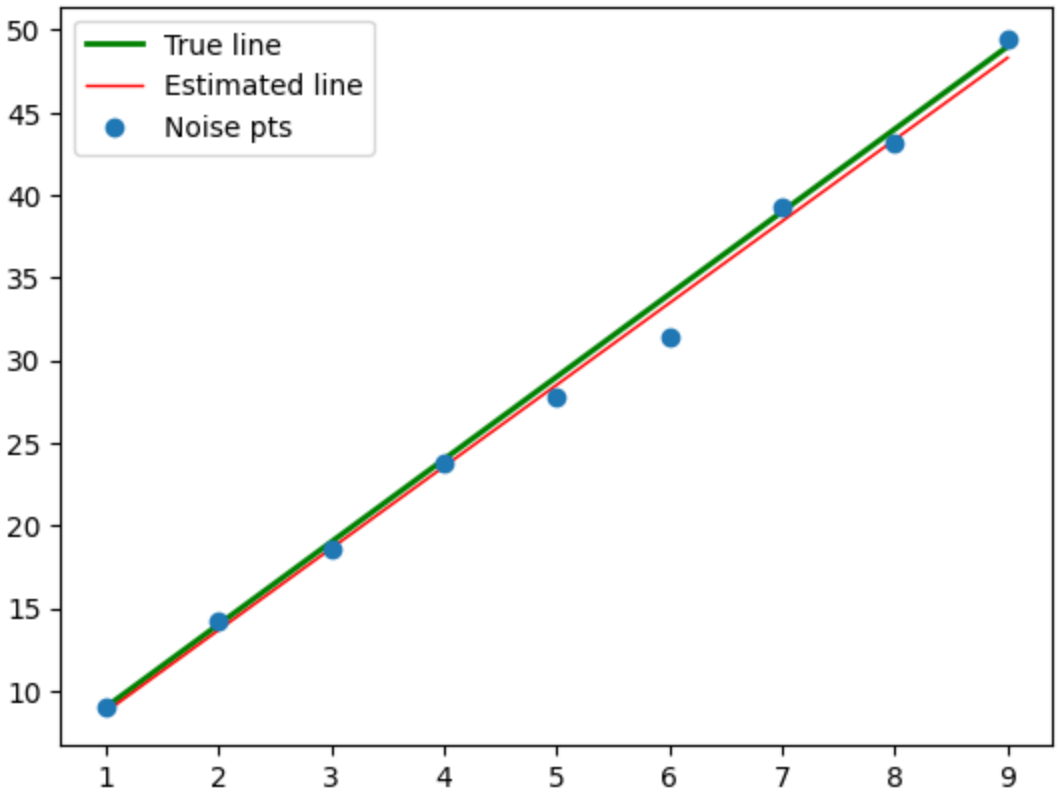
In [ ]:
```
# Question 02
## Least-squares line fitting
```

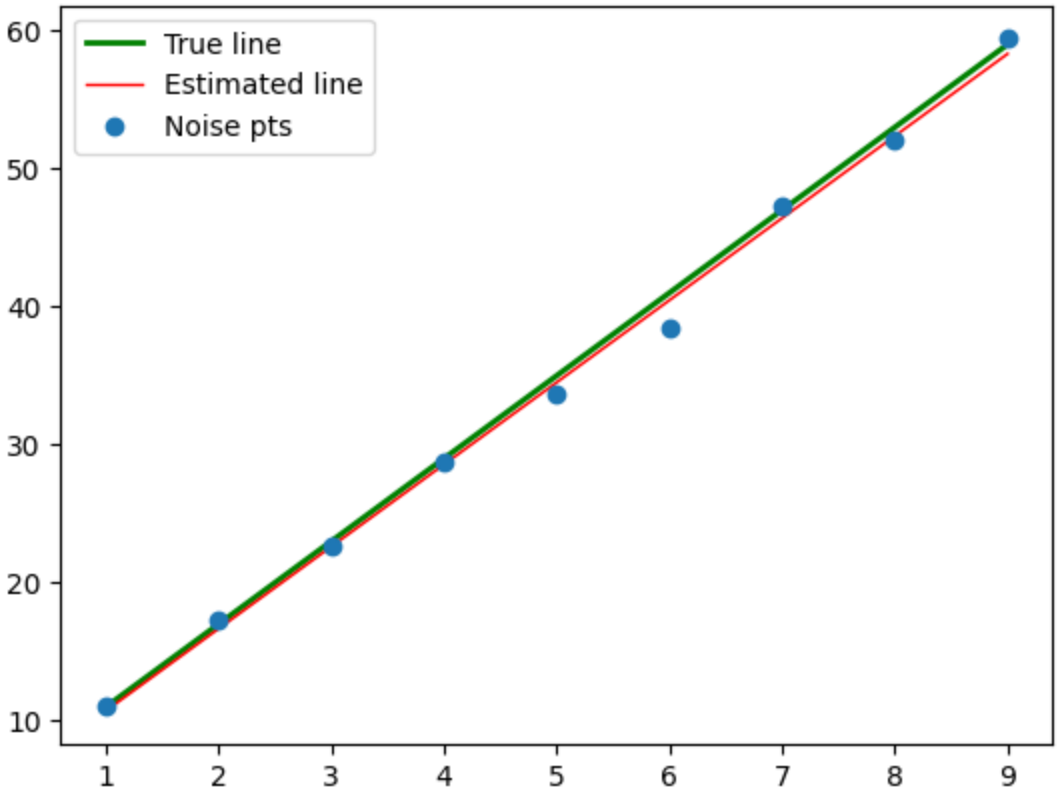
```
import numpy as np
import matplotlib.pyplot as plt
i = 0
## using a while loop to generate 10 different lines and use least squares line fit
while i < 10:
    # Genrating the true line y = m*x + c
    m = 2+i # gradient
    c = 1+i # intercept
    x = np.arange(1,10, 1)
    np.random.seed(45)
    no = np.random.randn(len(x)) #Noise
    o = np.zeros(x.shape)
    #o[1] = 20 # outliers
    y = m*x + c + no + o
    n = len(x)
    X = np.concatenate([x.reshape(n,1), np.ones((n, 1))], axis=1)
    B = np.linalg.pinv(X.T @ X) @ X.T @ y
    mstar = B[0]
    cstar = B[1]
    plt.plot([x[0], x[-1]], [m*x[0] + c, m*x[-1] + c], color='g', linewidth=2, labe
    plt.plot([x[0], x[-1]], [mstar*x[0] + cstar, mstar*x[-1] + cstar], color='r', l
    plt.plot(x,y, 'o', label='Noise pts')
    plt.legend(loc='best')
    plt.show()
    i = i + 1
```

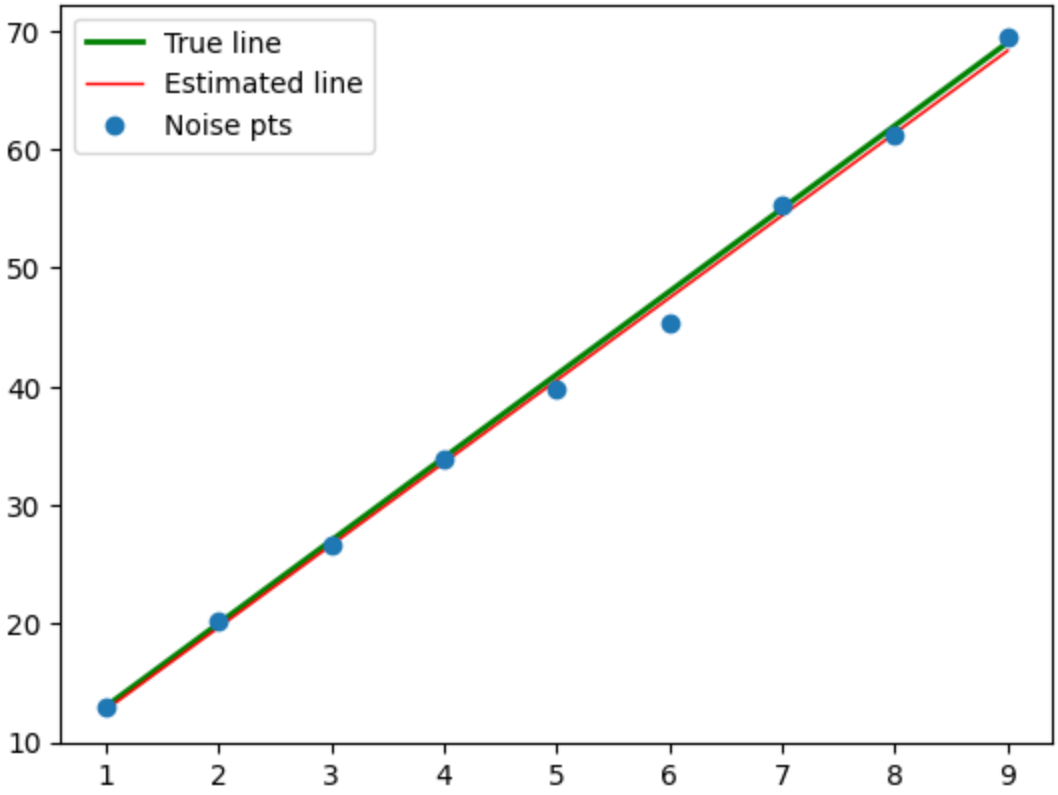


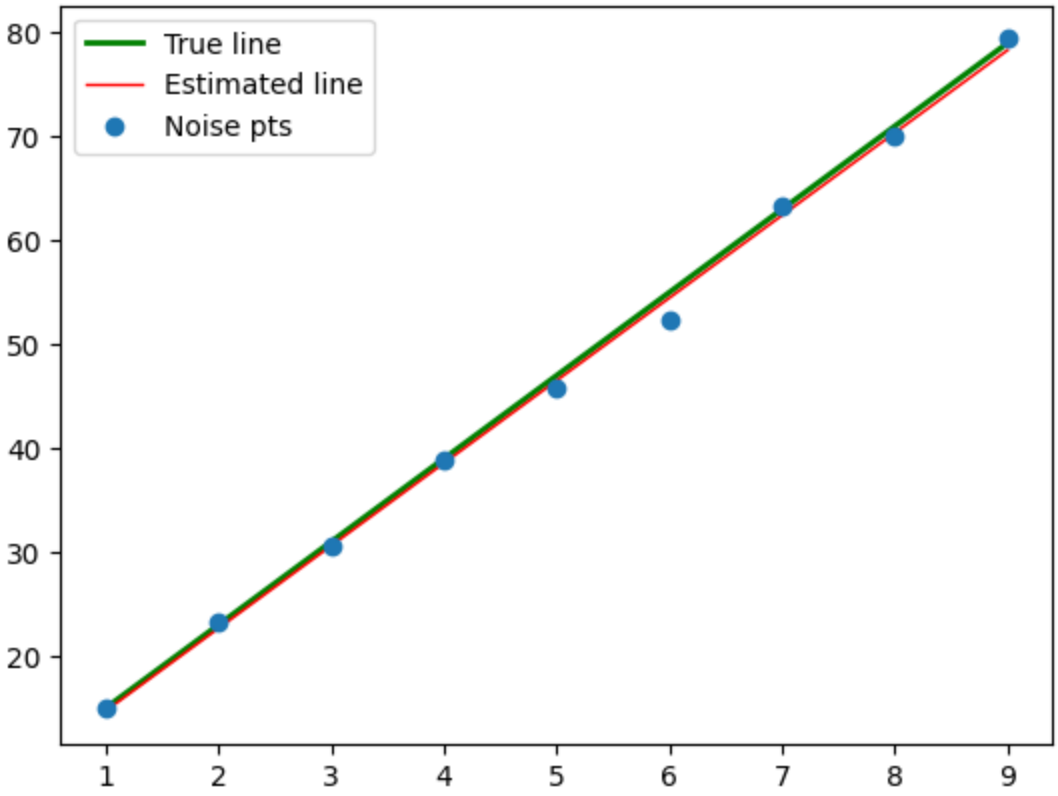

True line
Estimated line
Noise pts
30
25
20
15
10
5
1
2
3
4
5
6
7
8
9

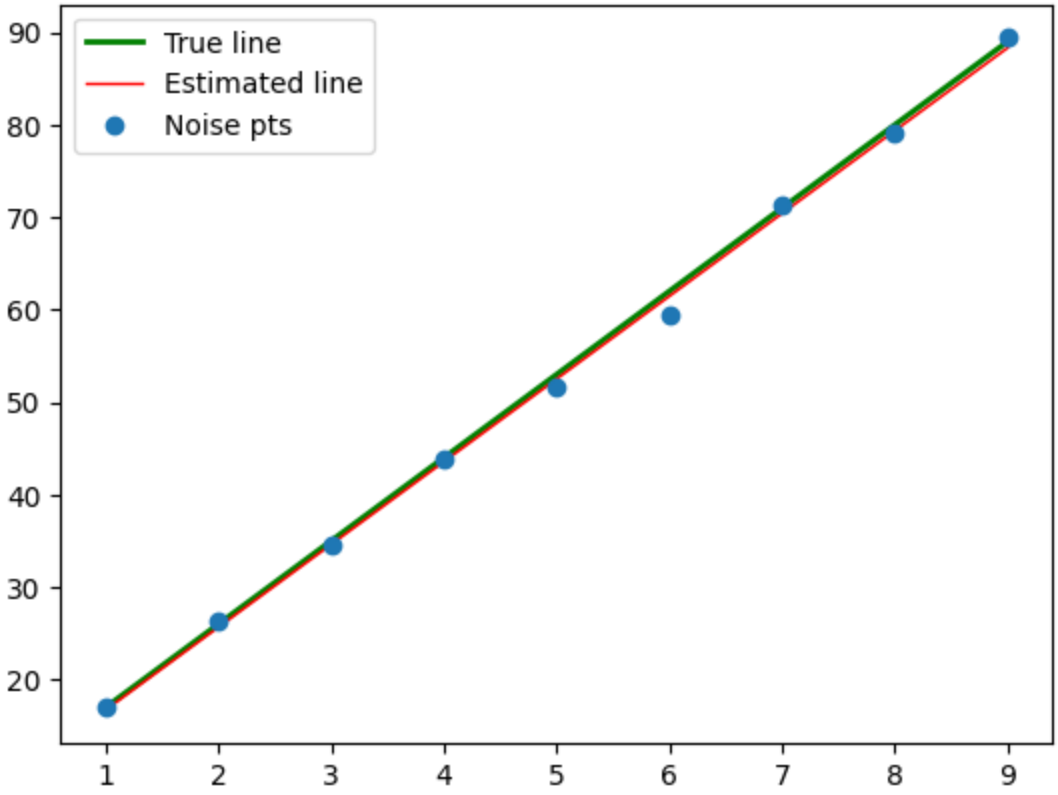

True line
Estimated line
Noise pts
40
35
30
25
20
15
10
1
2
3
4
5
6
7
8
9

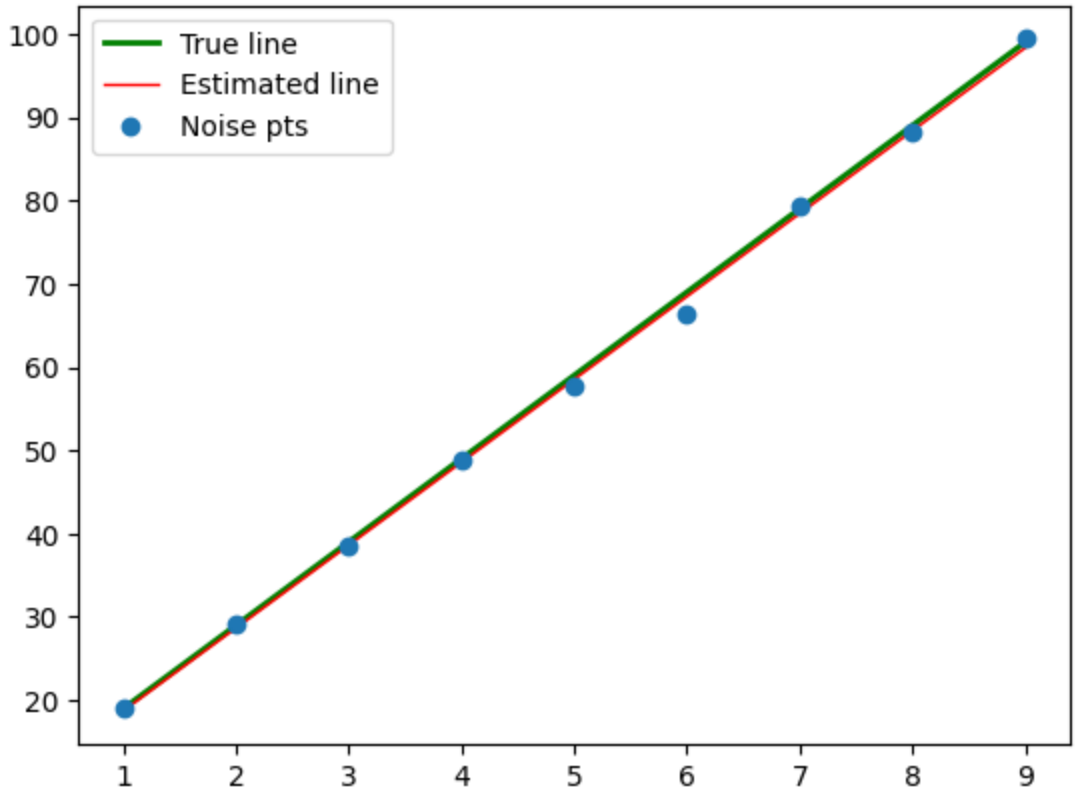

True line
Estimated line
Noise pts
50
45
40
35
30
25
20
15
10
1
2
3
4
5
6
7
8
9

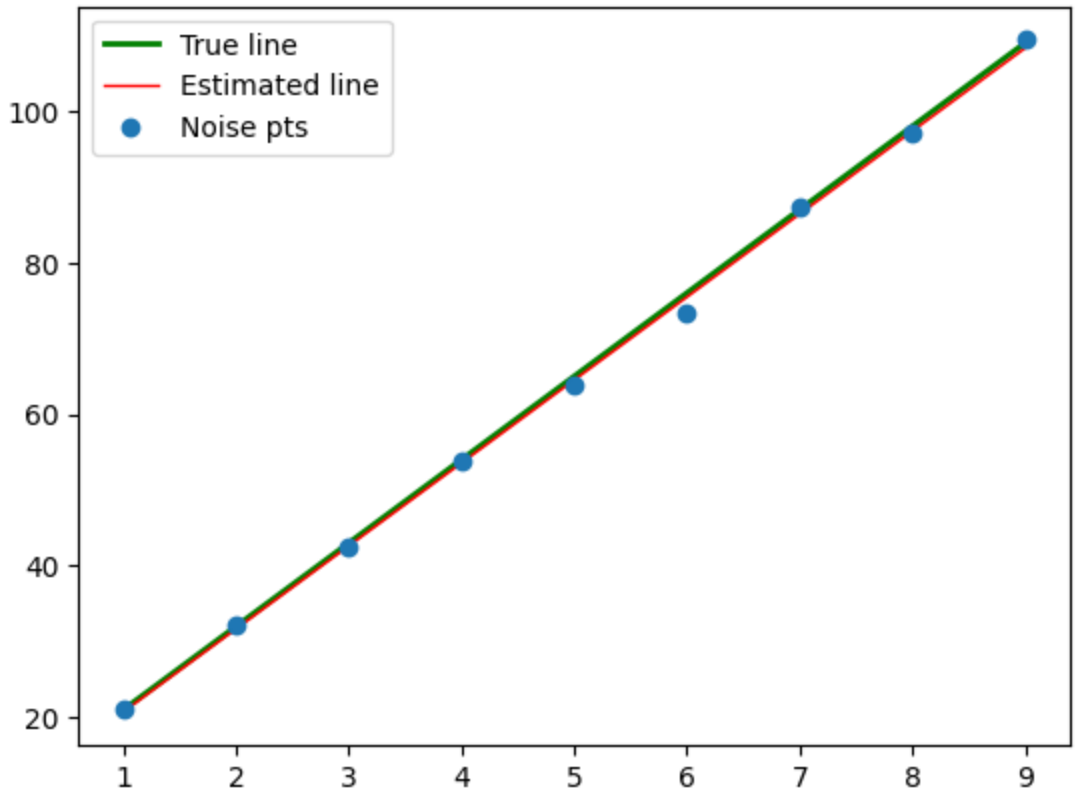

True line
Estimated line
Noise pts
60
50
40
30
20
10
1
2
3
4
5
6
7
8
9


True line
Estimated line
Noise pts
70
60
50
40
30
20
10
1
2
3
4
5
6
7
8
9

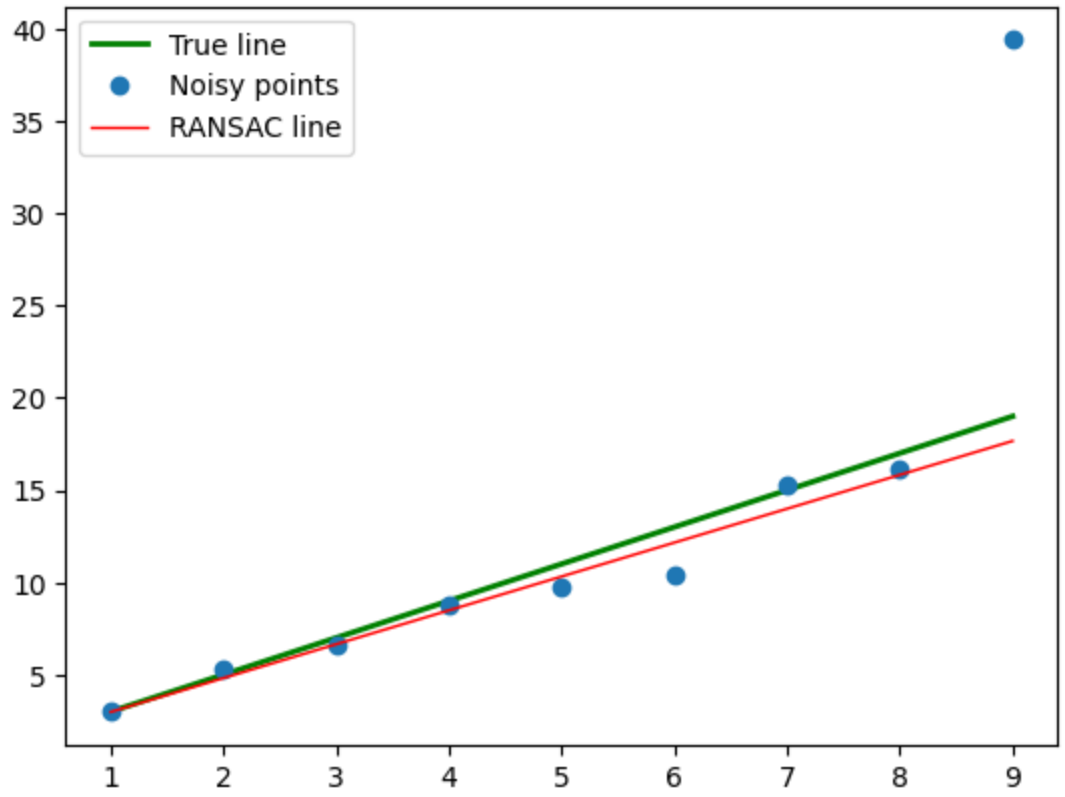

True line
Estimated line
Noise pts
80
70
60
50
40
30
20
1
2
3
4
5
6
7
8
9


True line
Estimated line
Noise pts
90
80
70
60
50
40
30
20
1
2
3
4
5
6
7
8
9

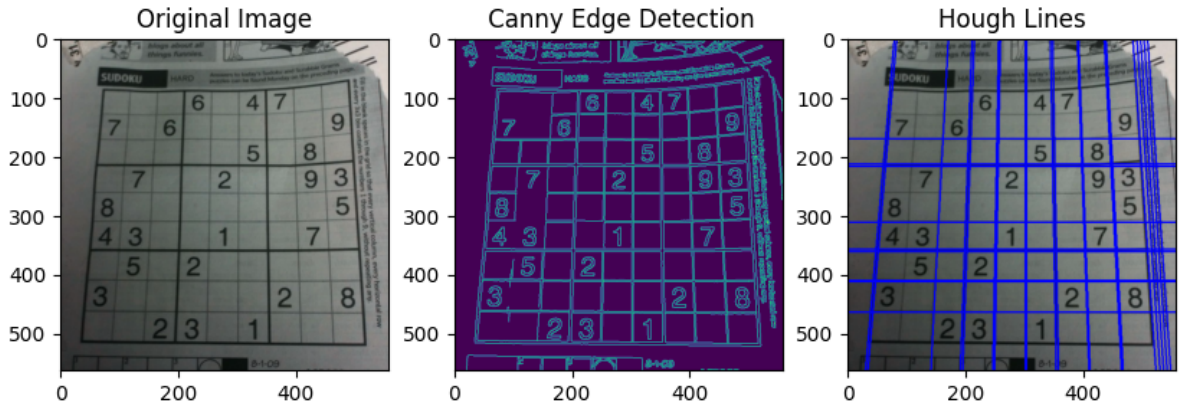

True line
Estimated line
Noise pts
100
90
80
70
60
50
40
30
20
1
2
3
4
5
6
7
8
9



In [ ]:
```
#Question 03
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import RANSACRegressor
## The RANSQC Method ignores the outlier
m = 2 # gradient
c = 1 # intercept
x = np.arange(1,10, 1)
np.random.seed(45)
no = np.random.randn(len(x)) #Noise
o = np.zeros(x.shape)
o[-1] = 20 #outlier
y = m*x + c + no + o
X = np.concatenate([x.reshape(-1, 1), np.ones((len(x), 1))], axis=1)
# Fitting a line using RANSAC Alogorithm imported
ransac = RANSACRegressor()
ransac.fit(X, y)
# Extracting the slope and intercept of the best-fit line
mstar = ransac.estimator_.coef_[0]
cstar = ransac.estimator_.intercept_
# Plotting the data points, true line, and best-fit line
plt.plot([x[0], x[-1]], [m*x[0] + c, m*x[-1] + c], color='g', linewidth=2, label=r'
plt.plot(x, y, 'o', label='Noisy points')
plt.plot([x[0], x[-1]], [mstar*x[0] + cstar, mstar*x[-1] + cstar], color='r', linew
plt.legend(loc='best')
plt.show()
```

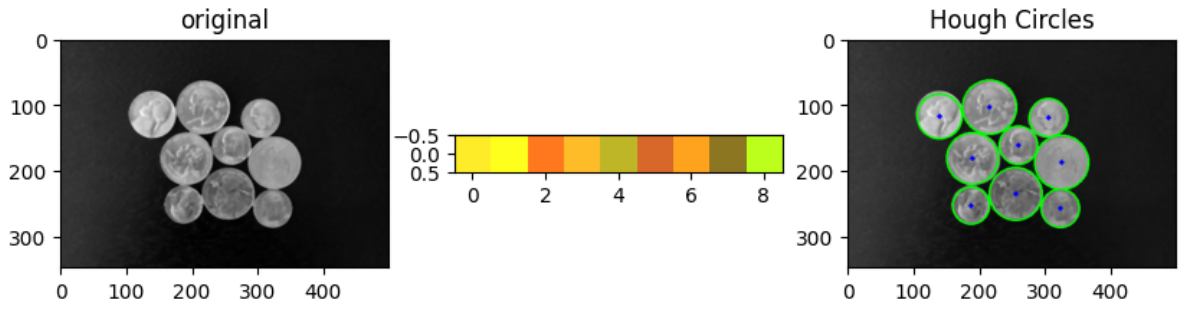


In [ ]:
```python
# Question 04
## Used the code given in the class as a reference When answering this Question
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
im = cv.imread('/examples/sudoku.png', cv.IMREAD_COLOR)
assert im is not None
gray = cv.cvtColor(im, cv.COLOR_BGR2BGRA)
# Apply edge detection
edges = cv.Canny(gray, 50, 150, apertureSize = 3)
lines = cv.HoughLines(edges, 1, np.pi/180, 200)
for line in lines:
    rho, theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv.line(im, (x1,y1), (x2,y2), (0,0,255), 2)
fig, ax= plt.subplots(1,3, figsize=(10,20))
ax[0].imshow(gray)
ax[0].set_title("Original Image")
ax[1].imshow(edges)
ax[1].set_title(" Canny Edge Detection")
ax[2].imshow(im)
ax[2].set_title("Hough Lines")
plt.show()
```



In [ ]:
```
# Question 05
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread('/examples/coins.jpg', cv.IMREAD_GRAYSCALE)
img = cv.medianBlur(img,5)
cimg = cv.cvtColor(img,cv.COLOR_GRAY2BGR)
circles = cv.HoughCircles(img,cv.HOUGH_GRADIENT,1,20,
                          param1=180,param2=50,minRadius=0,maxRadius=0)
circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

fig, ax = plt.subplots(1,3, figsize=(10,20))
ax[0].imshow(img, cmap='gray')
ax[0].set_title("original")
ax[1].imshow(circles)
ax[2].imshow(cimg)
ax[2].set_title("Hough Circles")

plt.show()
```

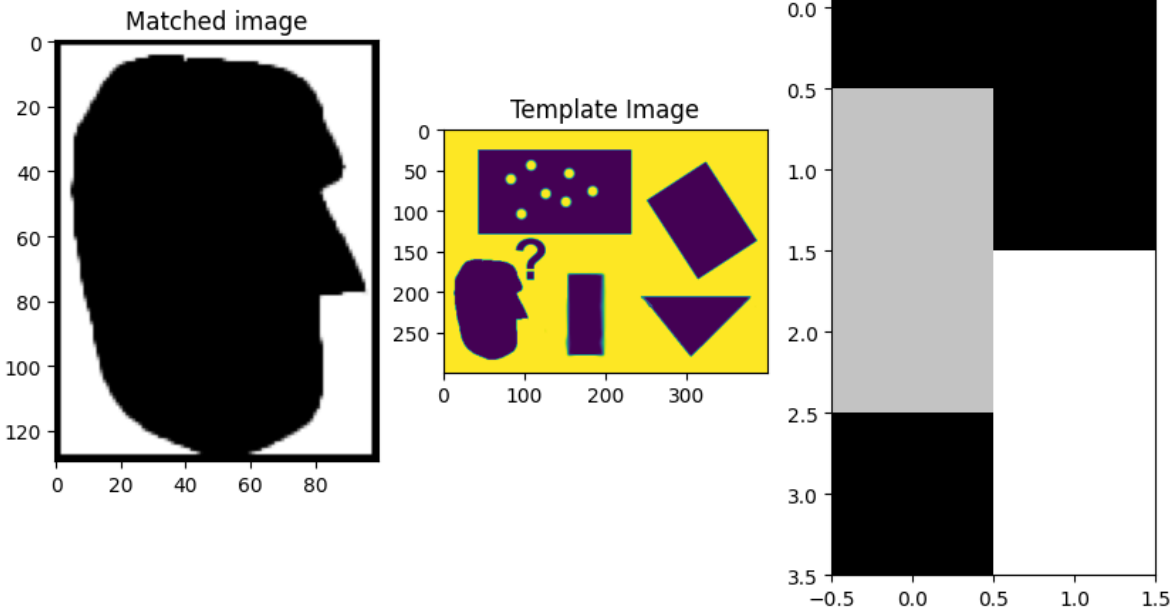
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



In [ ]:
```
# Question 06
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
# Loading the input image and the template
```

```
img = cv.imread('/examples/templ.png', cv.IMREAD_GRAYSCALE)
template = cv.imread('/examples/pic1.png', cv.IMREAD_GRAYSCALE)
_, thresh_img = cv.threshold(img, 0, 255, cv.THRESH_BINARY)
_, thresh_template = cv.threshold(template, 0, 255, cv.THRESH_BINARY)
# Finding the contours of the template and the input image
contours_template, _ = cv.findContours(thresh_template, cv.RETR_EXTERNAL, cv.CHAIN_
contours_img, _ = cv.findContours(thresh_img, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIM
# Match the contours
best_match = None
min_score = np.inf
for c1 in contours_template:
    for c2 in contours_img:
        score = cv.matchShapes(c1, c2, cv.CONTOURS_MATCH_I2, 0)
        if score < min_score:
            min_score = score
            best_match = c2
# using a if loop to confirm that they are matching images
if best_match is not None:
    rect = cv.minAreaRect(best_match)
    box = cv.boxPoints(rect)
    box = np.int0(box)
    cv.drawContours(img, [box], 0, (0, 0, 255), 2)
    fig, ax = plt.subplots(1,3, figsize=(10,20))
    ax[0].imshow(img, cmap='gray')
    ax[0].set_title("Matched image")
    ax[1].imshow(template)
    ax[1].set_title('Template Image')
    ax[2].imshow(box.squeeze(), cmap='gray')
    ax[2].set_title('Best Match')
    plt.show()
else:
    print('No match found')

# first thresholding the input and template images, finding their contours,
# and then using the cv.matchShapes function to find the best match between contour
# Then we use cv.minAreaRect to find the bounding rectangle and draws it on the inp
```

```
C:\Users\Sureka Siriwardana\AppData\Local\Temp\ipykernel_18296\3669658781.py:26: D
eprecationWarning: `np.int0` is a deprecated alias for `np.intp`.  (Deprecated Num
Py 1.24)
  box = np.int0(box)
```


Matched image
Template Image
Best Match