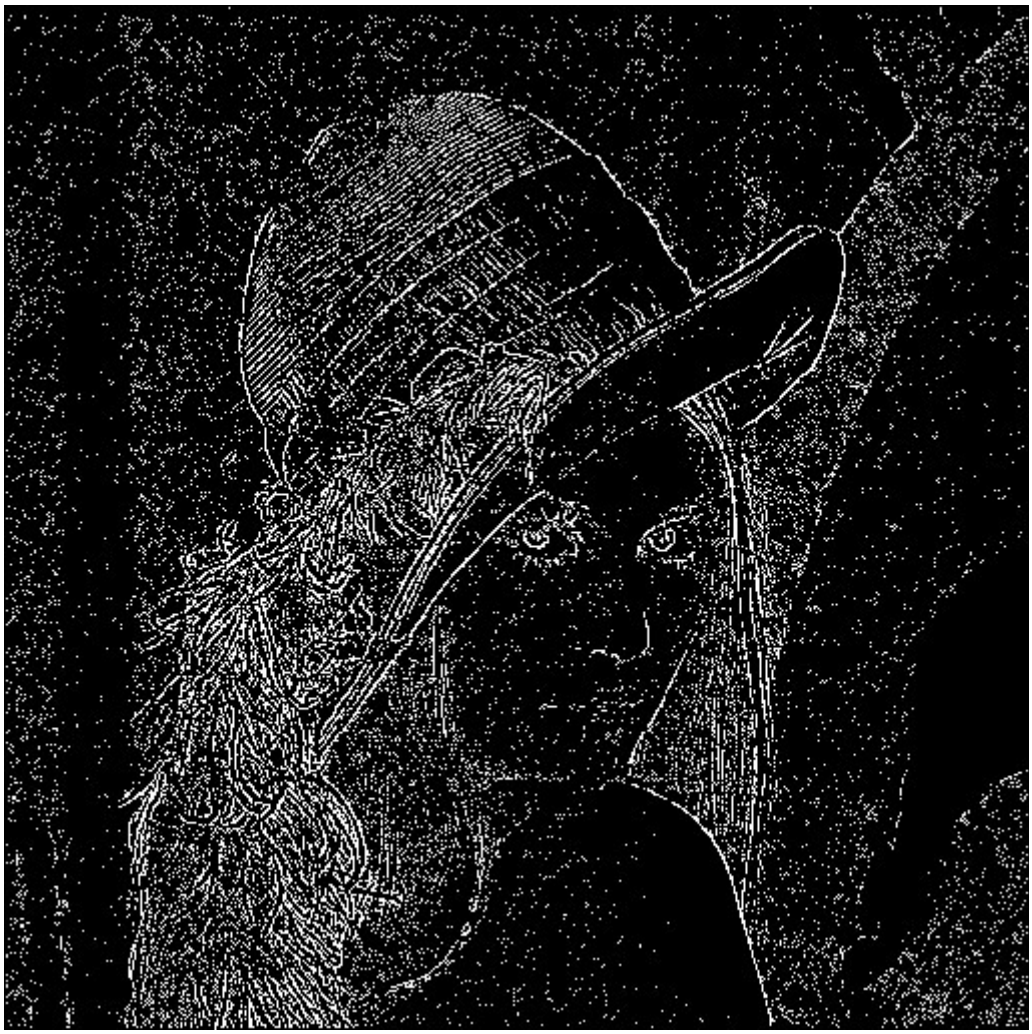
Computer Vision Hw10 Report

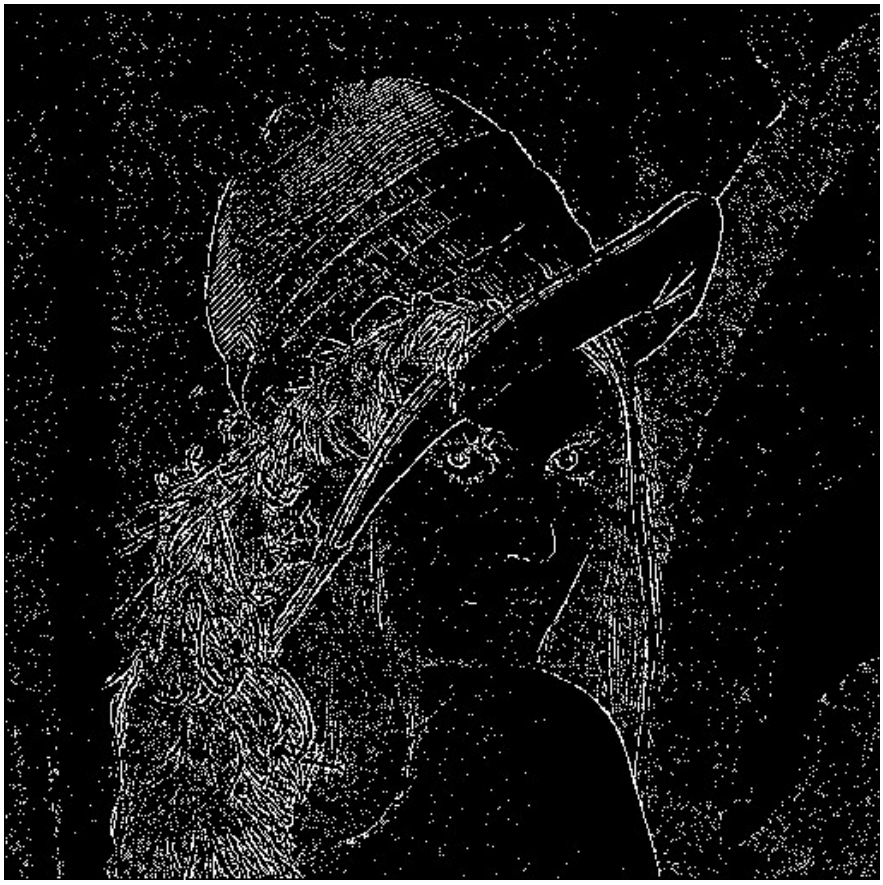
R07921010 余奇安

1.Laplace mask1
[[0,1,0],
  [1,-4,1],
  [0,1,0]]

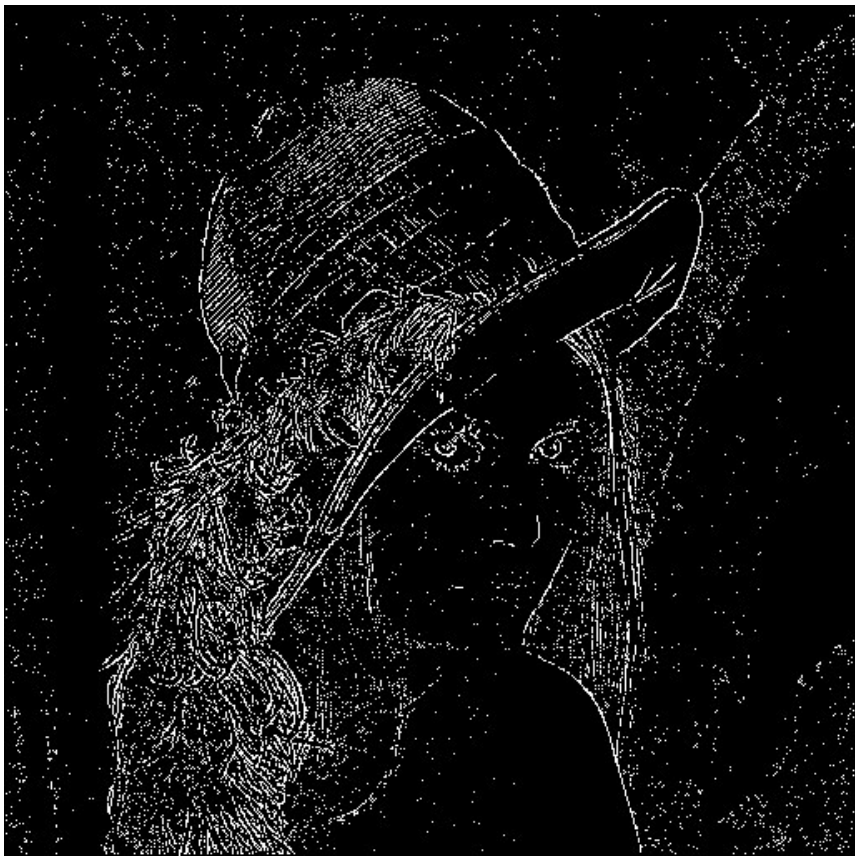
threshold at 15



threshold at 18

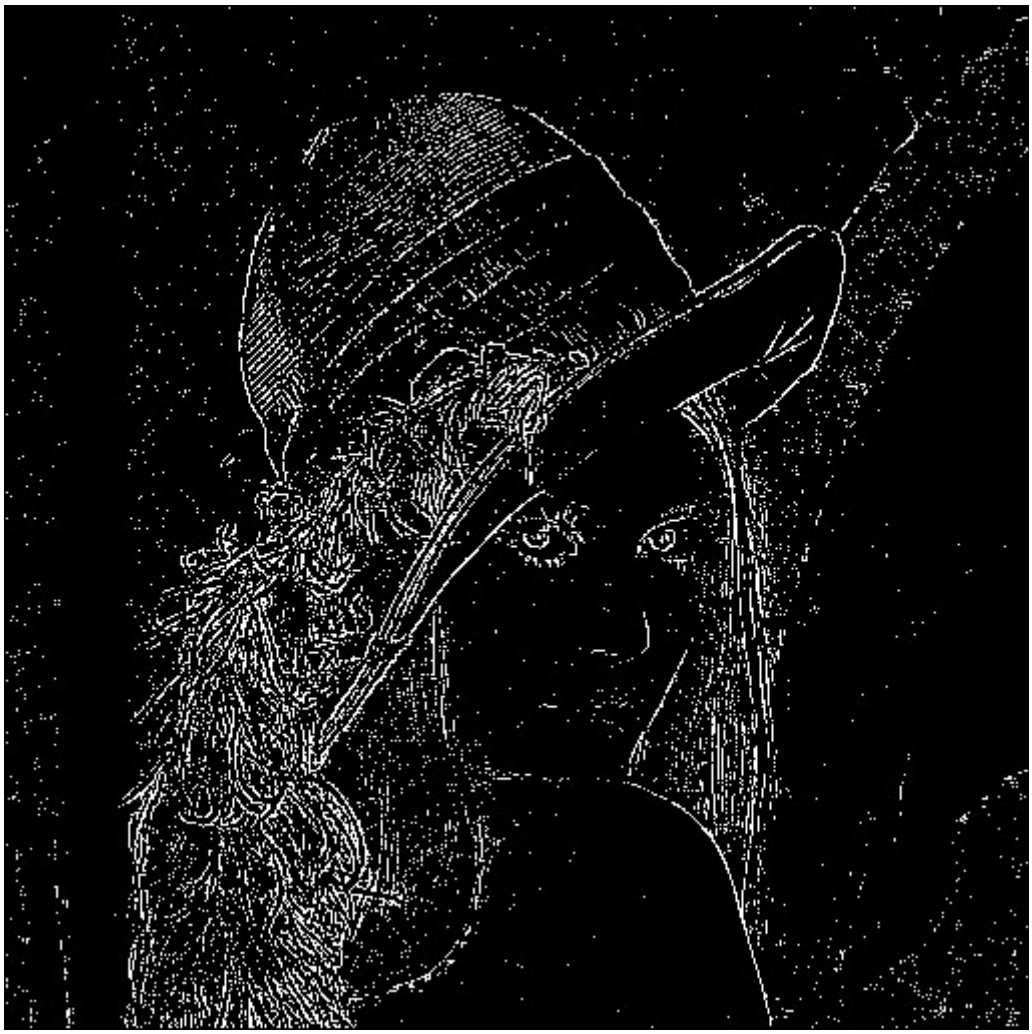


threshold at 20

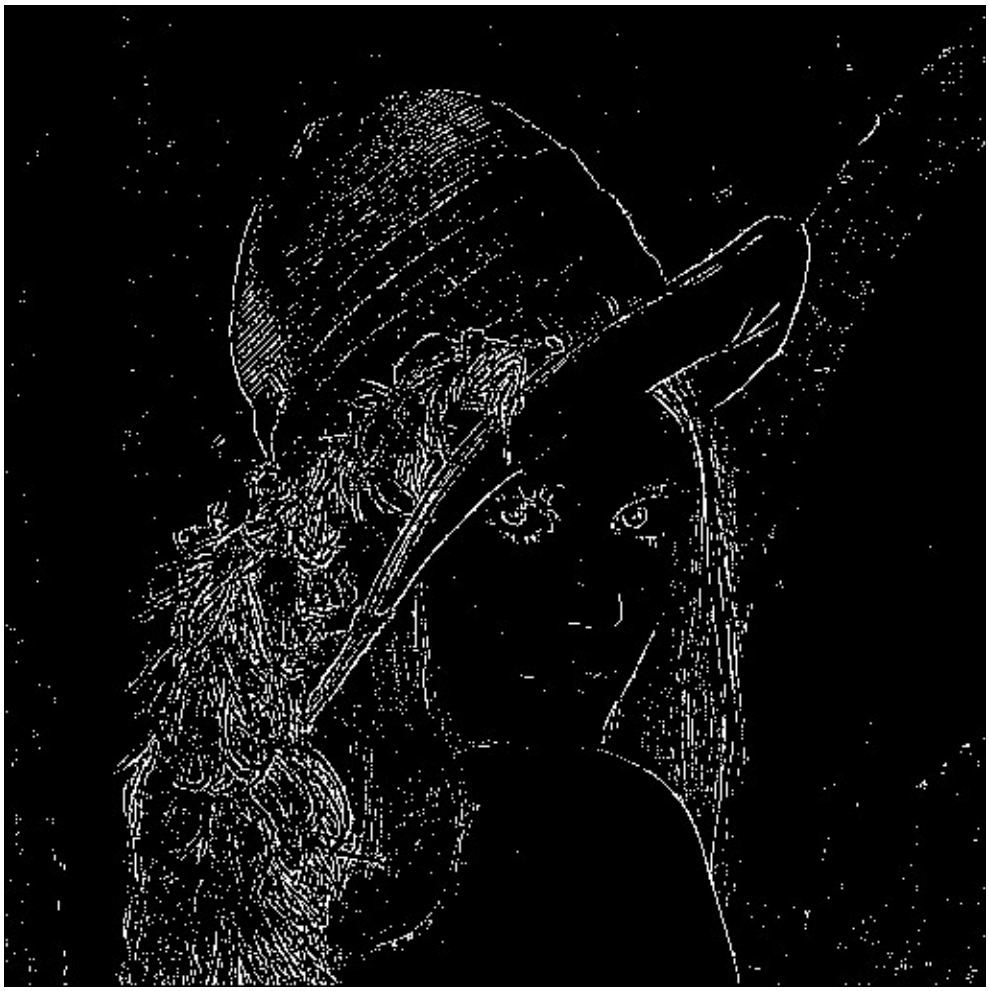


2.Laplace mask 2
mask=[[1.0/3,1.0/3,1.0/3],[1.0/3,-8.0/3,1.0/3],[1.0/3,1.0/3,1.0/3]]

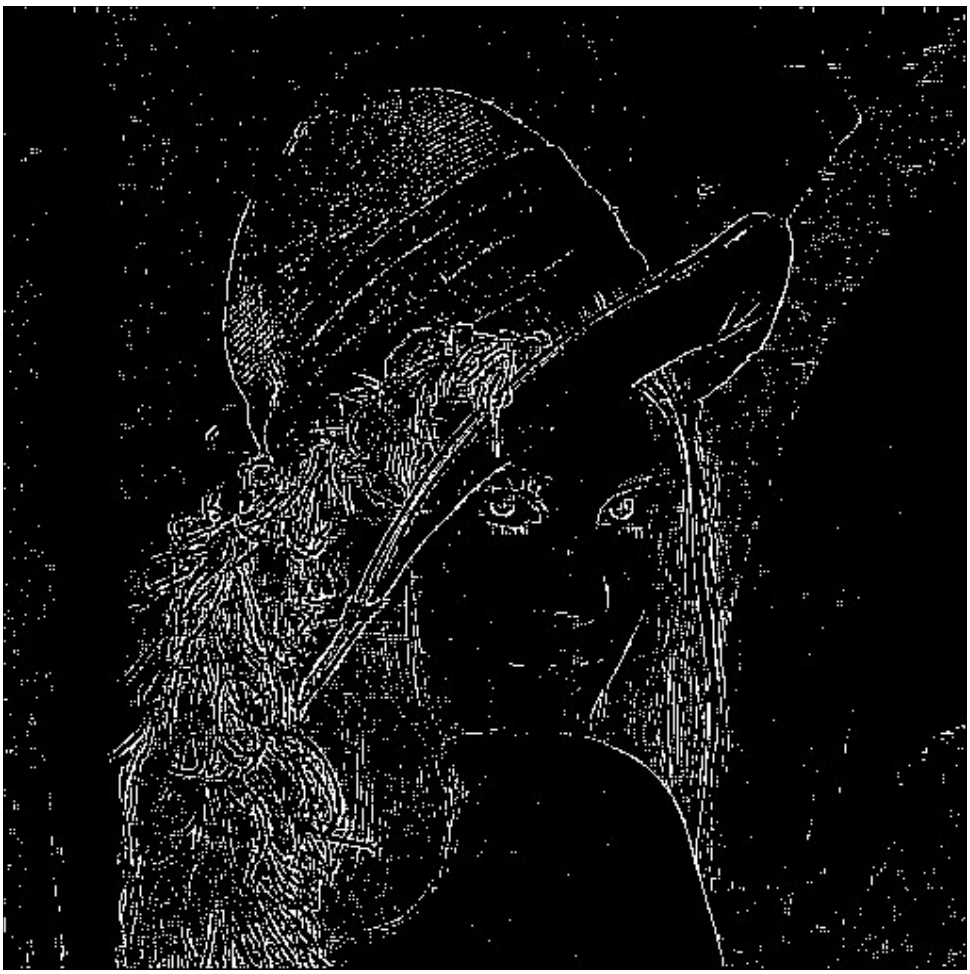
threshold at 15



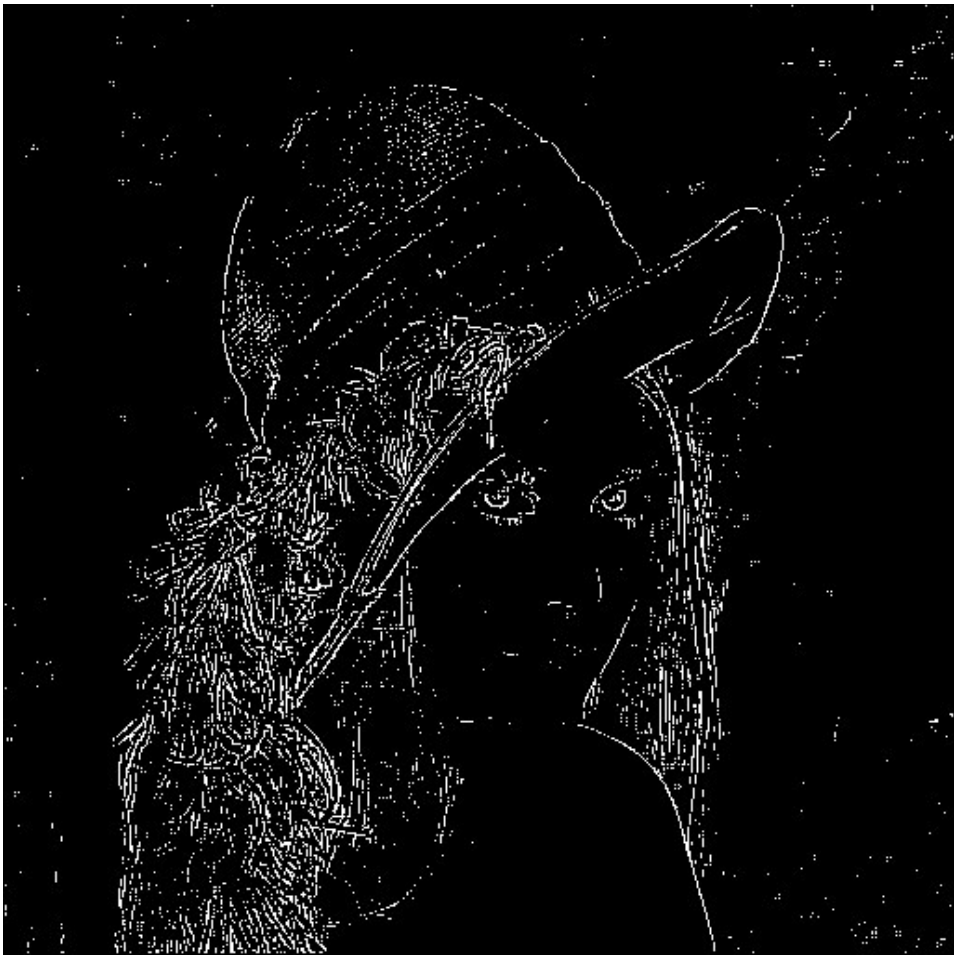
threshold at 18
good for noise removed and clear contour



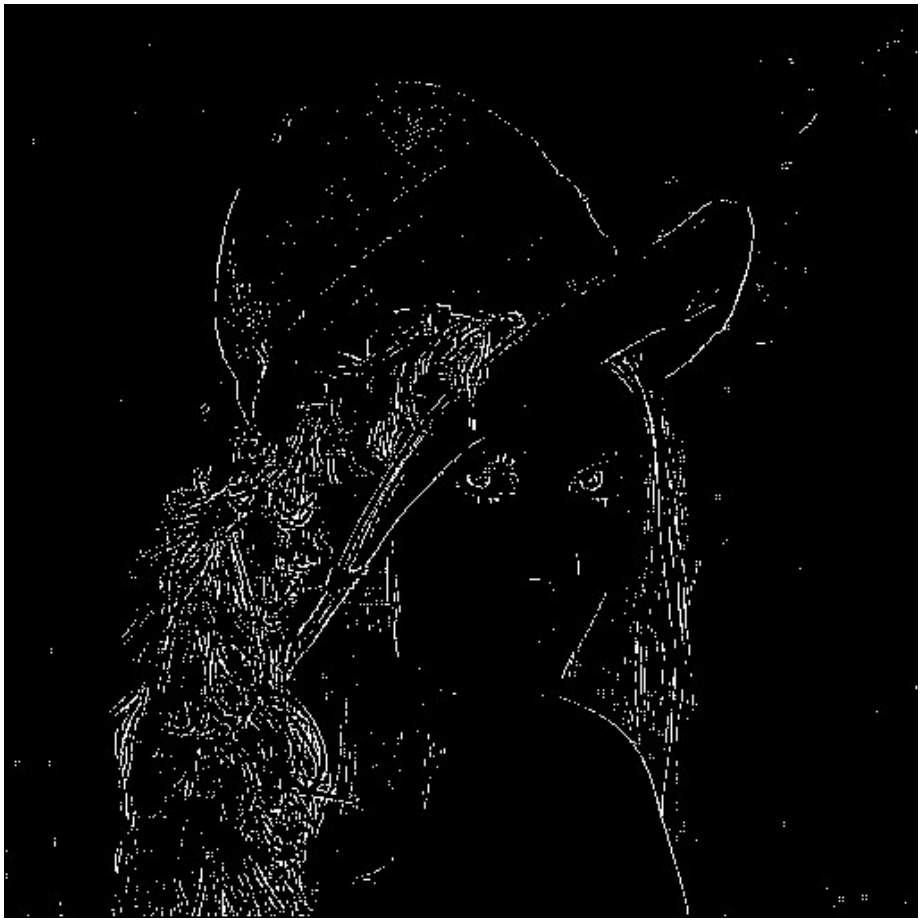
3.mini variance
mask=[[2.0/3,-1.0/3,2.0/3],[-1.0/3,-4.0/3,-1.0/3],[2.0/3,-1.0/3,2.0/3]]
threshold at 12



threshold at 15



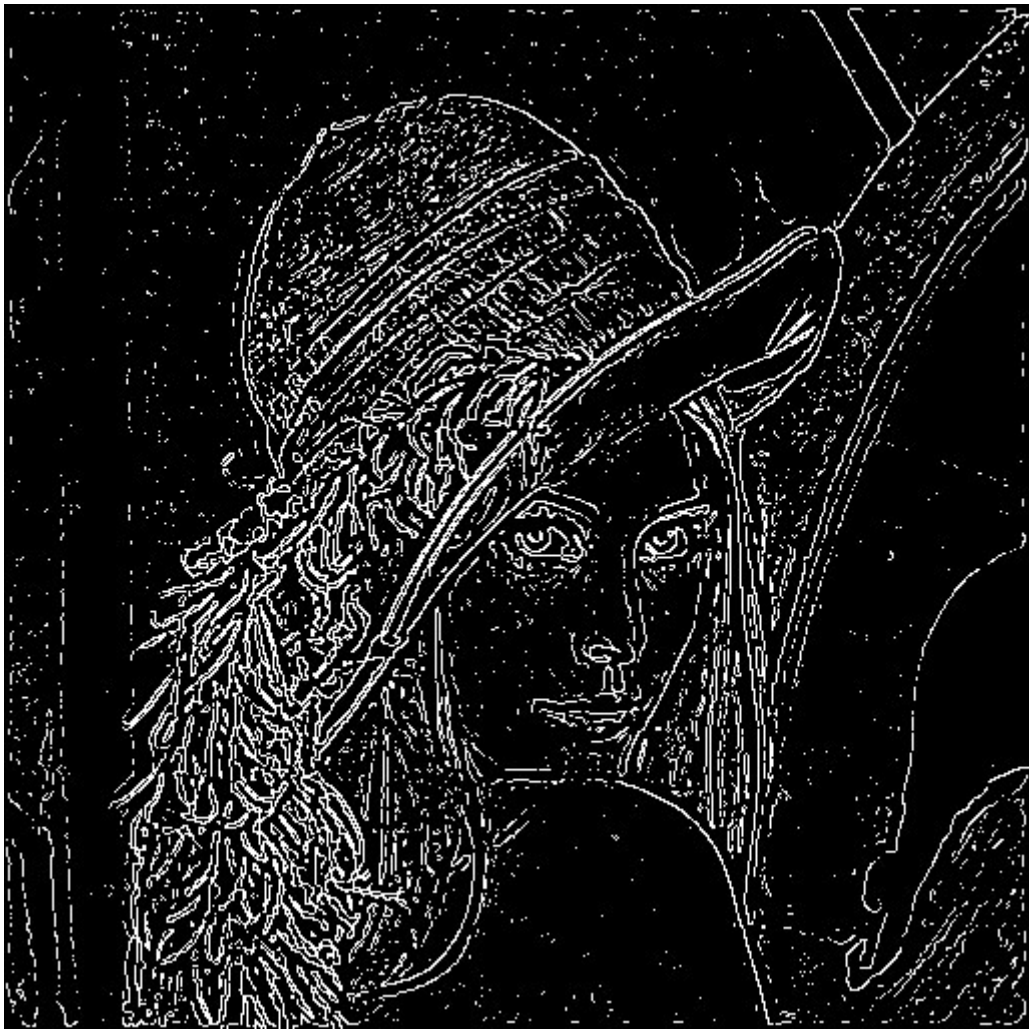
threshold at 20



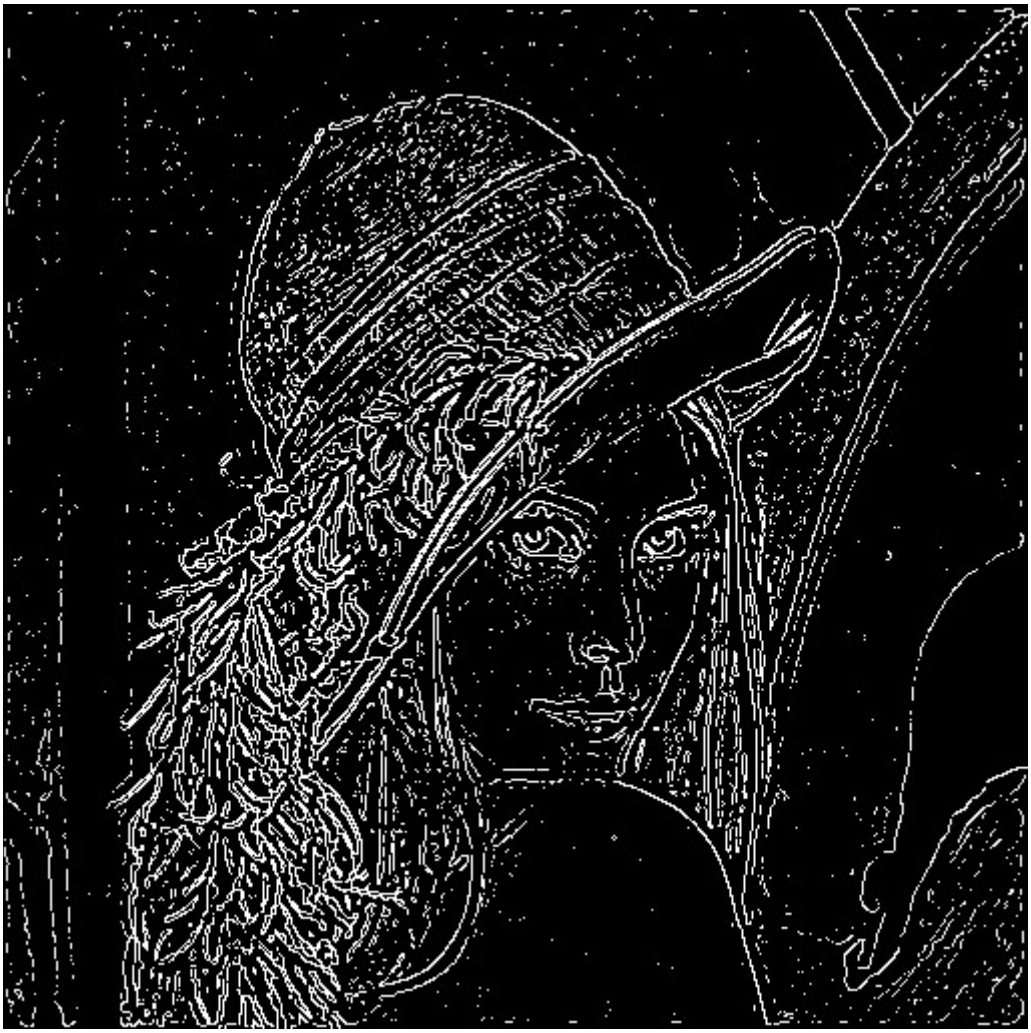
4. Laplace of Gaussion

```
mask=[[0,0,0,-1,-1,-2,-1,-1,0,0,0],
      [0,0,-2,-4,-8,-9,-8,-4,-2,0,0],
      [0,-2,-7,-15,-22,-23,-22,-15,-7,-2,0],
      [-1,-4,-15,-24,-14,-1,-14,-24,-15,-4,-1],
      [-1,-8,-22,-14,52,103,52,-14,-22,-8,-1],
      [-2,-9,-23,-1,103,178,103,-1,-23,-9,-2],
      [-1,-8,-22,-14,52,103,52,-14,-22,-8,-1],
      [-1,-4,-15,-24,-14,-1,-14,-24,-15,-4,-1],
      [0,-2,-7,-15,-22,-23,-22,-15,-7,-2,0],
      [0,0,-2,-4,-8,-9,-8,-4,-2,0,0],
      [0,0,0,-1,-1,-2,-1,-1,0,0,0]]
```

threshold at 1200



threshold at 1300



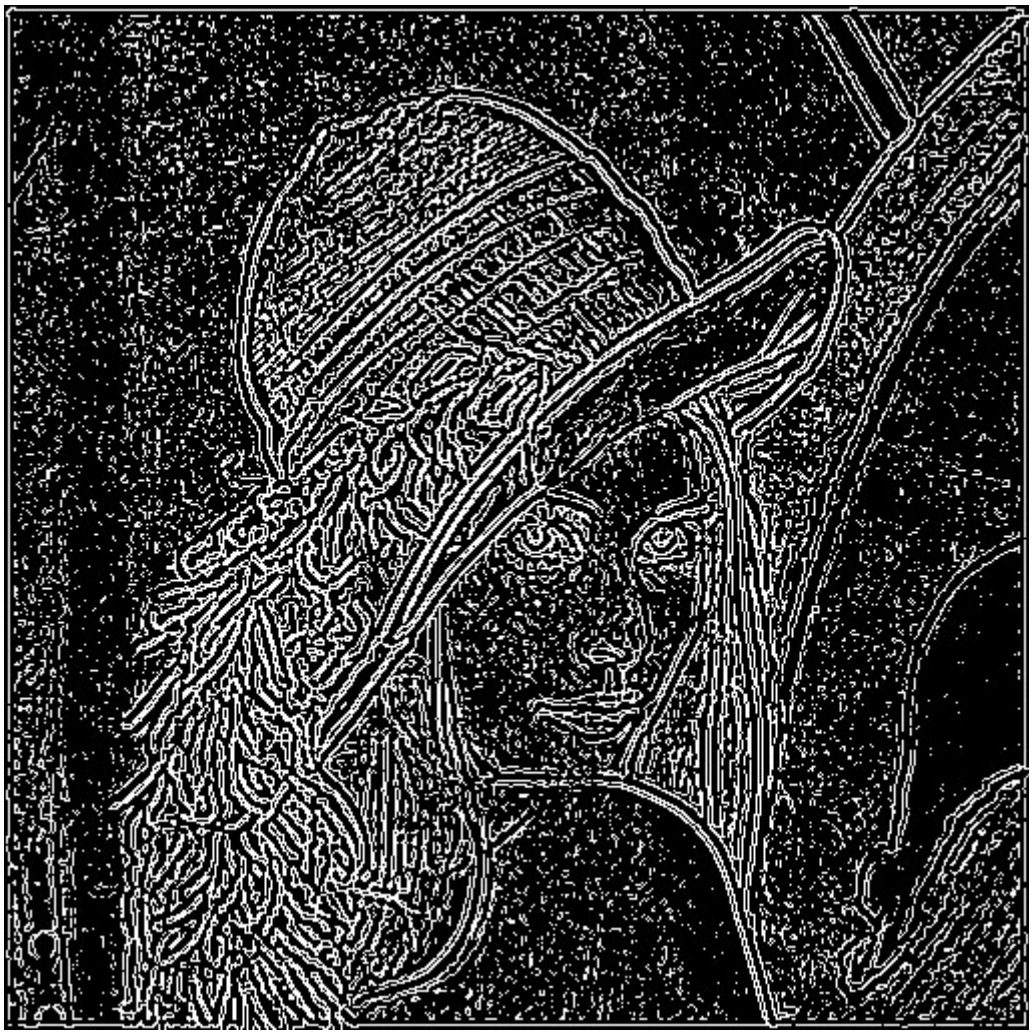
5. Difference Gaussian

I generate difference Gaussian mask by G(0,1)-G(0,3) in 11*11 mask

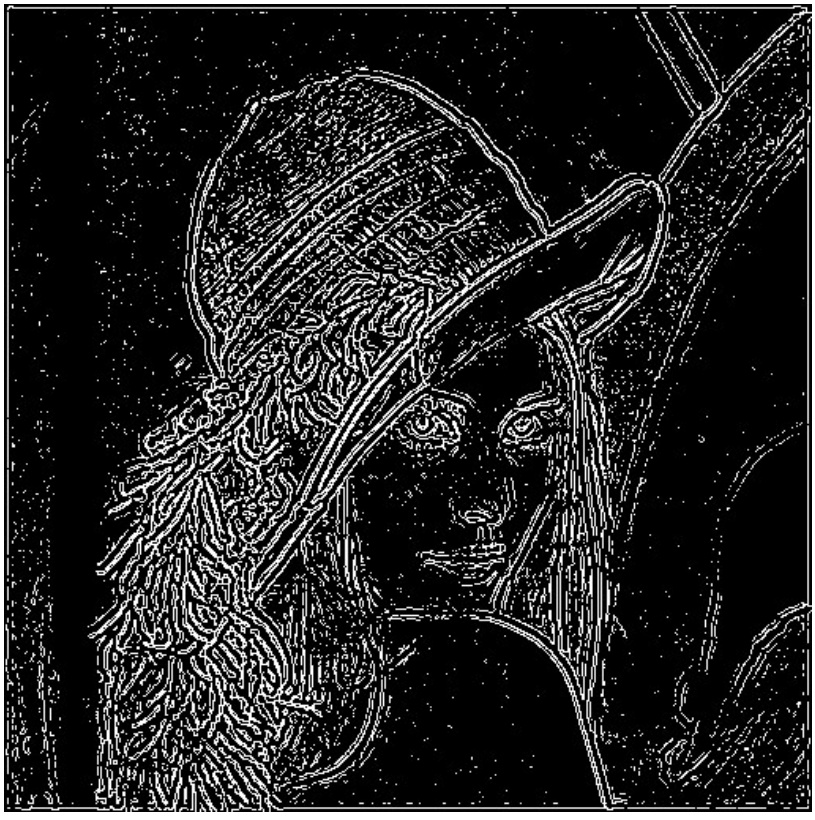
code:

```
mask=np.zeros(shape=(11,11))
sigma1=1
sigma2=3
mean=0
for i in range(-5,6):
    for j in range(-5,6):
        a=math.exp( -(i**2+j**2)/(2.0*sigma1*sigma1) )/ (math.sqrt(2*math.pi)*sigma1)
        b=math.exp( -(i**2+j**2)/(2.0*sigma2*sigma2) )/ (math.sqrt(2*math.pi)*sigma2)
        mask[i+5][j+5]=a-b
        mean+=a-b
mean/=11*11
for i in range(11):
    for j in range(11):
        mask[i][j]-=mean
```

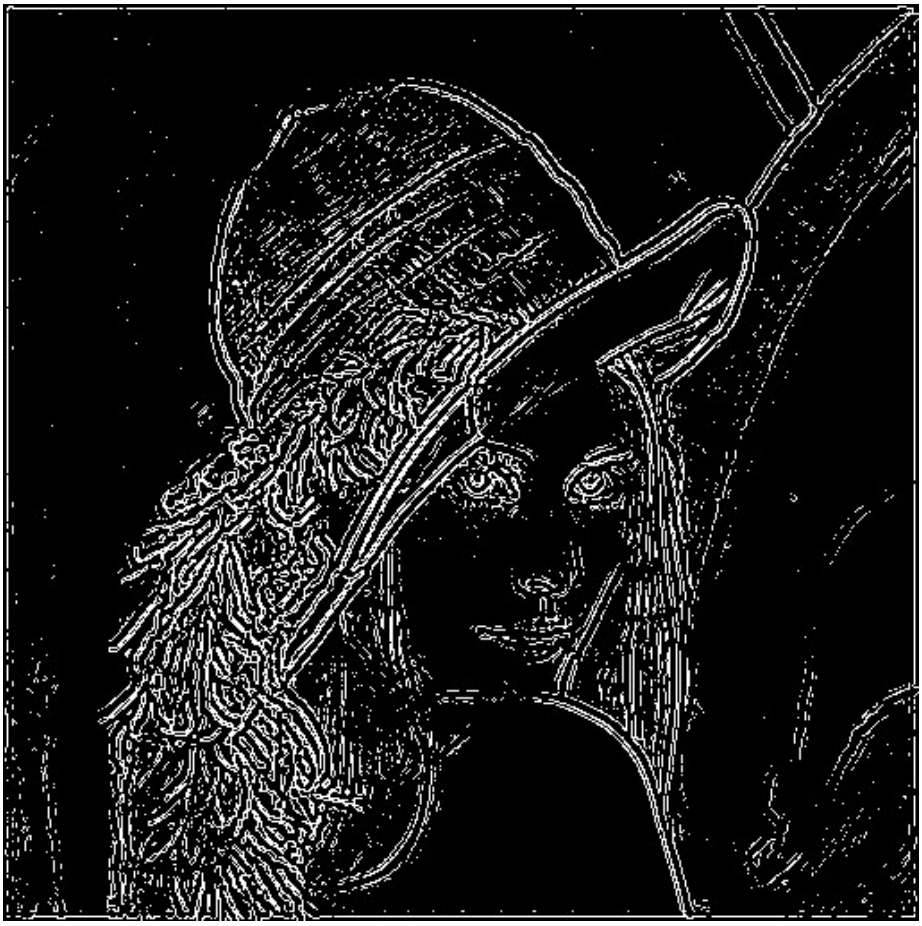
threshold at 1



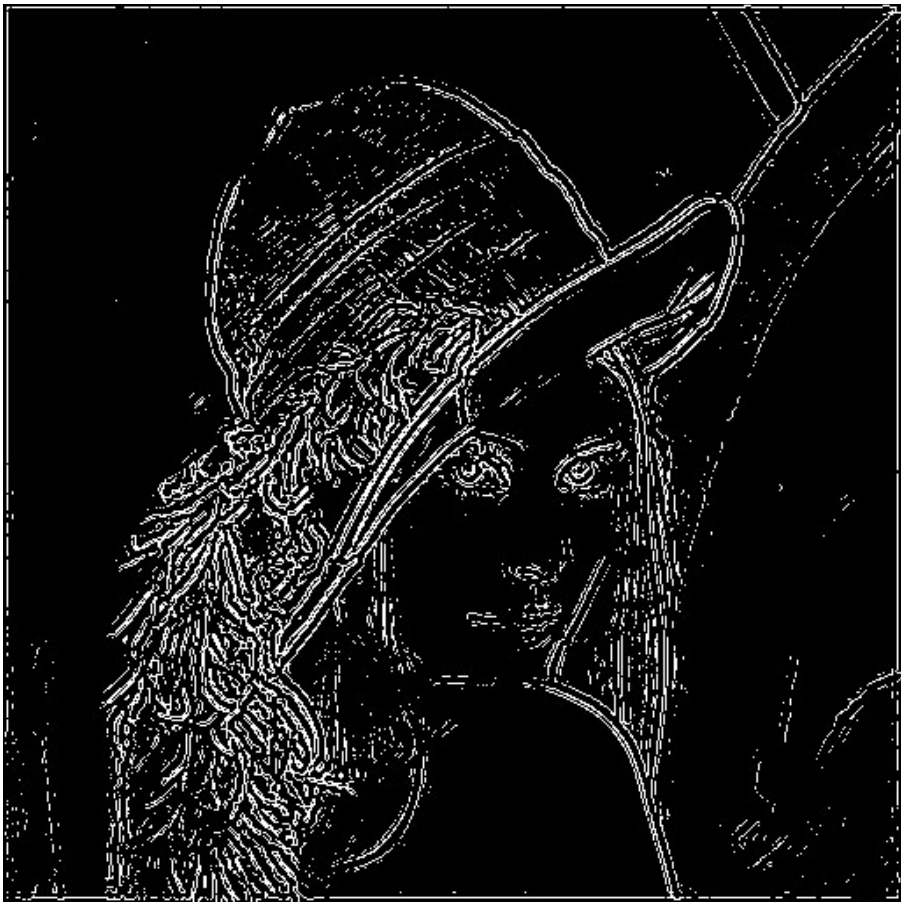
threshold at 2



threshold at 3



threshold at 4



code:

```
def Laplacian1(img, threshold):
    row,col=img.shape
    new=np.zeros(shape=(row,col))
    res=np.zeros(shape=(row,col))
    mask=[[0,1,0],[1,-4,1],[0,1,0]]
    for i in range(row):
        for j in range(col):
            s1=0
            for X in range(-1,2):
                for Y in range(-1,2):
                    if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                        s1+=mask[X+1][Y+1]*img[i+X][j+Y]
            new[i][j]=s1
    for i in range(row):
        for j in range(col):
            if new[i][j]>threshold:
                for X in range(-1,2):
                    for Y in range(-1,2):
                        if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                            if new[i+X][j+Y]<-1*threshold:
                                res[i][j]=255

            #if s1>threshold:
            #   res[i][j]=255
    return res
```

```
def Laplacian2(img,threshold):
    row,col=img.shape
    new=np.zeros(shape=(row,col))
    res=np.zeros(shape=(row,col))
    mask=[[1.0/3,1.0/3,1.0/3],[1.0/3,-8.0/3,1.0/3],[1.0/3,1.0/3,1.0/3]]
    for i in range(row):
        for j in range(col):
            s1=0.0
            for X in range(-1,2):
                for Y in range(-1,2):
                    if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                        s1+=mask[X+1][Y+1]*img[i+X][j+Y]
            new[i][j]=s1
    for i in range(row):
        for j in range(col):
            if new[i][j]>threshold:
                for X in range(-1,2):
                    for Y in range(-1,2):
                        if new[i+X][j+Y]<-1*threshold:
                            res[i][j]=255
    return res
```

```python
def mini_variance(img,threshold):
    row,col=img.shape
    new=np.zeros(shape=(row,col))
    res=np.zeros(shape=(row,col))
    mask=[[2.0/3,-1.0/3,2.0/3],[-1.0/3,-4.0/3,-1.0/3],[2.0/3,-1.0/3,2.0/3]]
    for i in range(row):
        for j in range(col):
            s1=0.0
            for X in range(-1,2):
                for Y in range(-1,2):
                    if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                        s1+=mask[X+1][Y+1]*img[i+X][j+Y]
            new[i][j]=s1
    for i in range(row):
        for j in range(col):
            if new[i][j]>threshold:
                for X in range(-1,2):
                    for Y in range(-1,2):
                        if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                            if new[i+X][j+Y]<-1*threshold:
                                res[i][j]=255
    return res
```

```python
def Laplace_Gaussion(img,threshold):
    row,col=img.shape
    new=np.zeros(shape=(row,col))
    res=np.zeros(shape=(row,col))
    mask=[[0,0,0,-1,-1,-2,-1,-1,0,0,0],
          [0,0,-2,-4,-8,-9,-8,-4,-2,0,0],
          [0,-2,-7,-15,-22,-23,-22,-15,-7,-2,0],
          [-1,-4,-15,-24,-14,-1,-14,-24,-15,-4,-1],
          [-1,-8,-22,-14,52,103,52,-14,-22,-8,-1],
          [-2,-9,-23,-1,103,178,103,-1,-23,-9,-2],
          [-1,-8,-22,-14,52,103,52,-14,-22,-8,-1],
          [-1,-4,-15,-24,-14,-1,-14,-24,-15,-4,-1],
          [0,-2,-7,-15,-22,-23,-22,-15,-7,-2,0],
          [0,0,-2,-4,-8,-9,-8,-4,-2,0,0],
          [0,0,0,-1,-1,-2,-1,-1,0,0,0]]
    for i in range(row):
        for j in range(col):
            s1=0.0
            for X in range(-5,6):
                for Y in range(-5,6):
                    if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                        s1+=mask[X+5][Y+5]*img[i+X][j+Y]
            new[i][j]=s1
    for i in range(row):
        for j in range(col):
            if new[i][j]>threshold:
                for X in range(-1,2):
                    for Y in range(-1,2):
                        if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                            if new[i+X][j+Y]<-1*threshold:
                                res[i][j]=255
    return res
```

```
def Difference_Gaussion(img,threshold):
    row,col=img.shape
    new=np.zeros(shape=(row,col))
    res=np.zeros(shape=(row,col))
    mask=np.zeros(shape=(11,11))
    sigma1=1
    sigma2=3
    mean=0
    for i in range(-5,6):
        for j in range(-5,6):
            a=math.exp( -(i**2+j**2)/(2.0*sigma1*sigma1) )/ (math.sqrt(2*math.pi)*sigma1)
            b=math.exp( -(i**2+j**2)/(2.0*sigma2*sigma2) )/ (math.sqrt(2*math.pi)*sigma2)
            mask[i+5][j+5]=a-b
            mean+=a-b
    mean/=11*11
    for i in range(11):
        for j in range(11):
            mask[i][j]-=mean

    for i in range(row):
        for j in range(col):
            s1=0.0
            for X in range(-5,6):
                for Y in range(-5,6):
                    if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                        s1+=mask[X+5][Y+5]*img[i+X][j+Y]
                        #print(X,Y,i,j)
            new[i][j]=s1
    for i in range(row):
        for j in range(col):
            if new[i][j]>threshold:
                for X in range(-1,2):
                    for Y in range(-1,2):
                        if X+i>=0 and X+i<row and j+Y>=0 and j+Y<col:
                            if new[i+X][j+Y]<-1*threshold:
                                res[i][j]=255
    return res
```

Discussion:
we can find that the bigger kernel has function not only on finding contour but also remove the noise. However, it’s also need more time to process the image.
I regard Laplace of Gaussian as the best performance.
I think this is because zero crossing check more constrains such as whether it’s neighbor is less than threshold, which increases the correctness of finding contour. Besides, Gaussian distribution also a good model to describe the image pixels distribution.