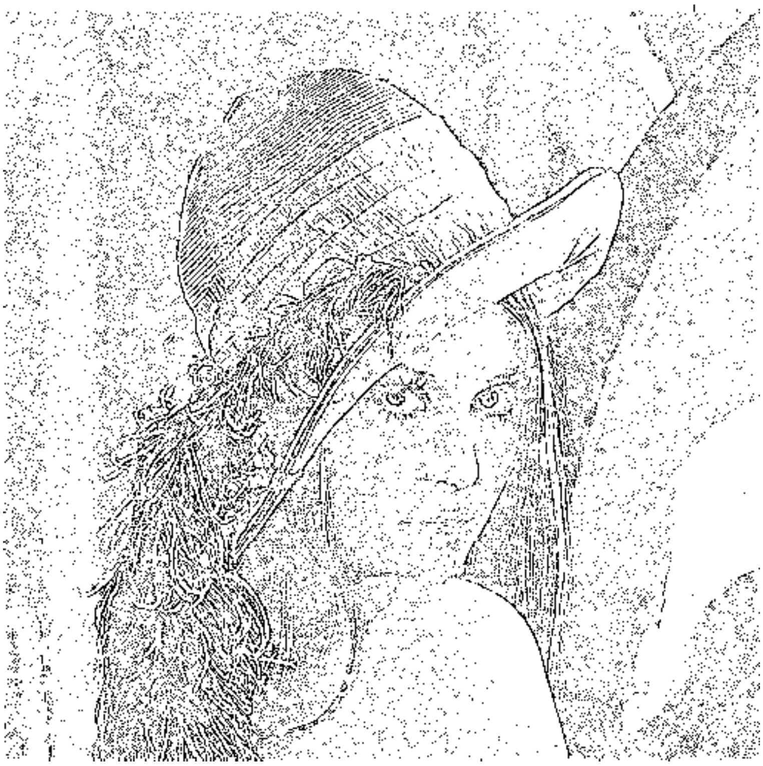
Homework 10

R13525009 羅筠笙

a. Laplace Mask1 (0, 1, 0, 1, -4, 1, 0, 1, 0): 15

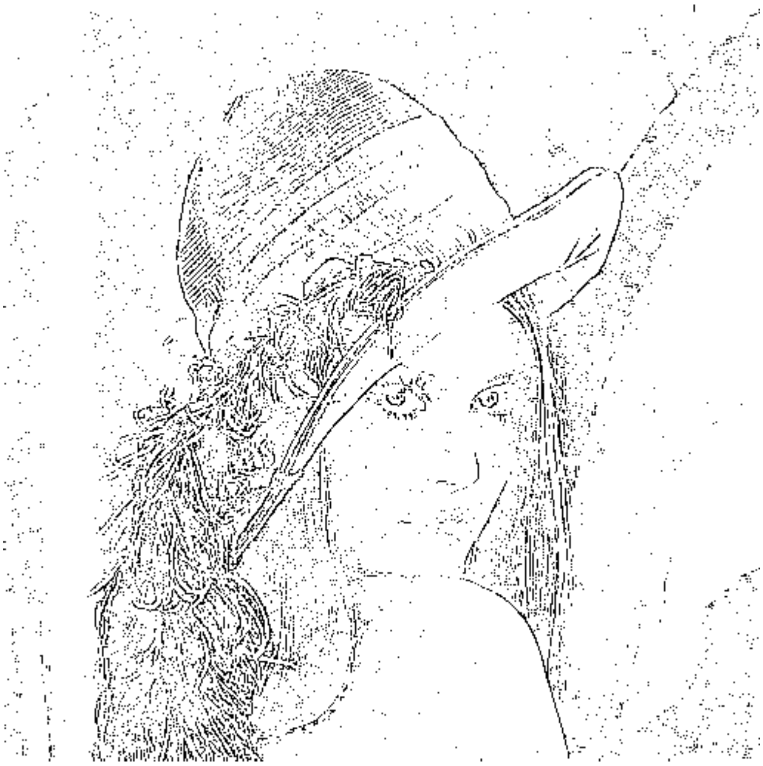
Description: Applies a simple Laplace operator for edge detection and further filters the edge points based on neighborhood pixel characteristics, producing the final edge-detected image.



```
def laplace1(img_arr, threshold):
    mask = np.array([[0, 1, 0],
                     [1, -4, 1],
                     [0, 1, 0]])

    # Expand the image
    img_arr = expand_with_replicate(img_arr, 1)

    # Using convolution
    res1 = my_conv(img_arr, mask, threshold)

    res1 = expand_with_replicate(res1, 1)
    res2 = np.ones((img_size0, img_size1))

    # Neighbor checking
    for i in range(img_size0):
        for j in range(img_size1):
            if res1[i + 1, j + 1] == 1:
                tmp = False
                for k in range(3):
                    for l in range(3):
                        if res1[i + k, j + l] == -1:
                            tmp = True
                            break
                    if tmp:
                        break
                if tmp:
                    res2[i, j] = 0

    return res2 * 255
```

b. Laplace Mask2 (1, 1, 1, 1, -8, 1, 1, 1, 1): 15

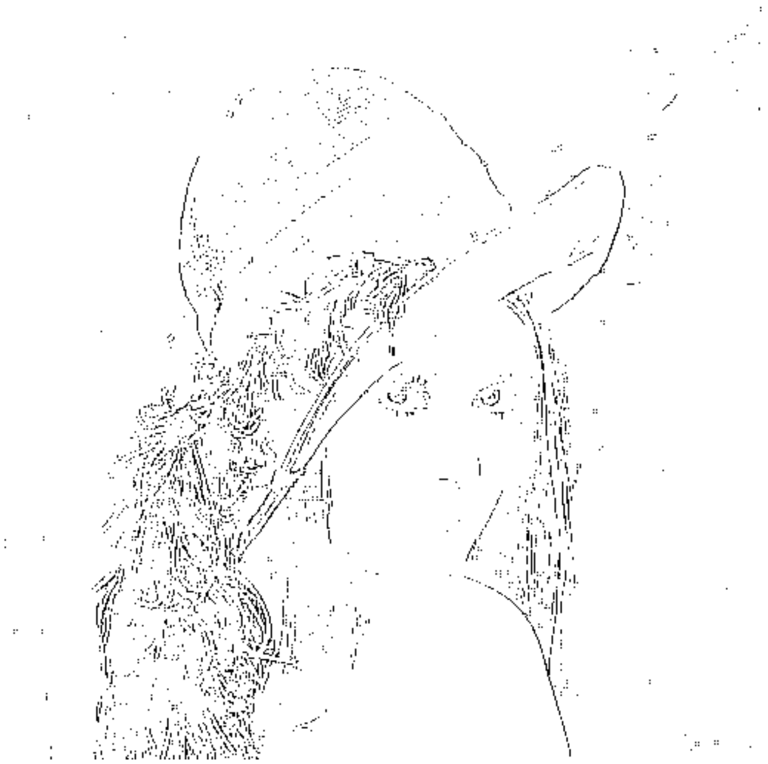
Description: Applies another Laplace operator for edge detection. Similar to question a, it filters edge points based on neighborhood characteristics, resulting in smoother edge outcomes.



```
def laplace2(img_arr, threshold):
    mask = np.array([[1, 1, 1],
                     [1, -8, 1],
                     [1, 1, 1]], dtype=np.float64)
    mask /= 3

    # Expand input for padding using numpy
    img_arr = expand_with_replicate(img_arr, 1)

    # Perform convolution using numpy array
    res1 = my_conv(img_arr, mask, threshold)

    # Expand result for neighbor checking using numpy
    res1 = expand_with_replicate(res1, 1)
    res2 = np.ones((img_size0, img_size1))  # Initialize with ones

    # Neighbor checking
    for i in range(img_size0):
        for j in range(img_size1):
            if res1[i + 1, j + 1] == 1:
                tmp = False
                for k in range(3):
                    for l in range(3):
                        if res1[i + k, j + l] == -1:
                            tmp = True
                            break
                    if tmp:
                        break
                if tmp:
                    res2[i, j] = 0

    return res2 * 255
```

c. Minimum variance Laplacian: 20

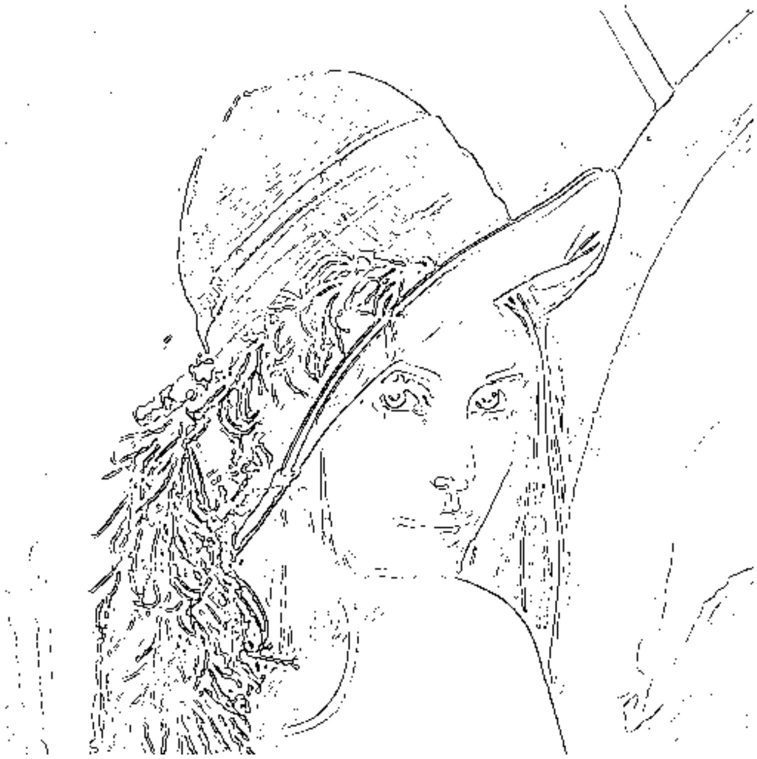
Description: Applies the minimum variance Laplace operator to detect edges, focusing on accurately representing edge features while minimizing noise, producing more stable detection results.



```python
def minimum_variance_laplacian(img_arr, threshold):
    mask = np.array([[2, -1, 2],
                     [-1, -4, -1],
                     [2, -1, 2]], dtype=np.float64)
    mask /= 3

    # Expand input for padding using numpy
    img_arr = expand_with_replicate(img_arr, 1)

    # Perform convolution using numpy array
    res1 = my_conv(img_arr, mask, threshold)

    # Expand result for neighbor checking using numpy
    res1 = expand_with_replicate(res1, 1)
    res2 = np.ones((img_size0, img_size1))  # Initialize with ones

    # Neighbor checking
    for i in range(img_size0):
        for j in range(img_size1):
            if res1[i + 1, j + 1] == 1:
                tmp = False
                for k in range(3):
                    for l in range(3):
                        if res1[i + k, j + l] == -1:
                            tmp = True
                            break
                    if tmp:
                        break
                if tmp:
                    res2[i, j] = 0

    return res2 * 255
```

d. Laplace of Gaussian: 3000

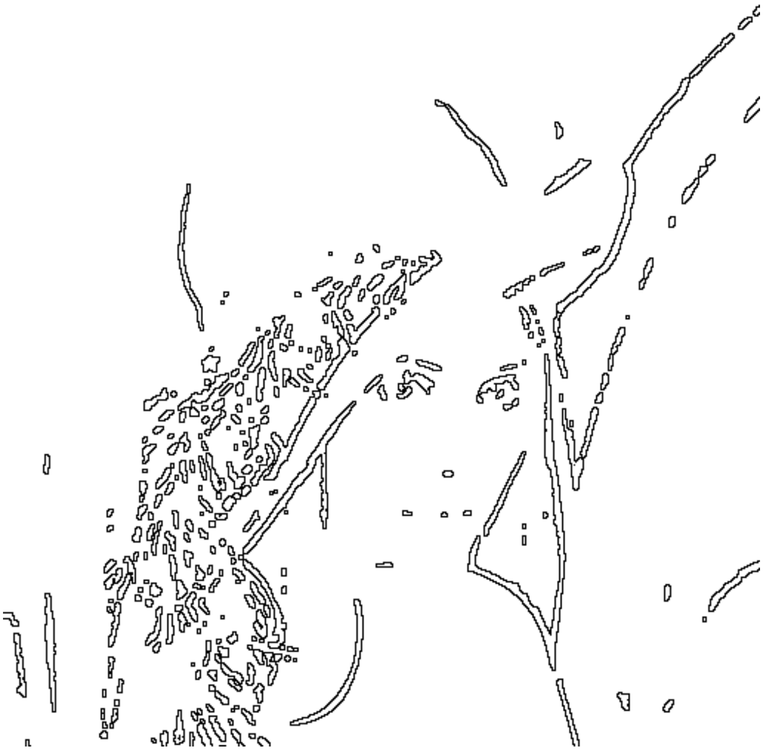
Description: Combines Gaussian smoothing with the Laplace operator. It removes high-frequency noise before edge detection, making it particularly suitable for noisy images.



```
def laplace_gauss(img_arr, threshold):
    mask = np.array([[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]])

    # Expand input for padding using numpy
    img_arr = expand_with_replicate(img_arr, 5)

    # Perform convolution using numpy array
    res1 = my_conv(img_arr, mask, threshold)

    # Expand result for neighbor checking using numpy
    res1 = expand_with_replicate(res1, 5)
    res2 = np.ones((img_size0, img_size1))  # Initialize with ones

    # Neighbor checking
    for i in range(img_size0):
        for j in range(img_size1):
            if res1[i + 1, j + 1] == 1:
                tmp = False
                for k in range(3):
                    for l in range(3):
                        if res1[i + k, j + l] == -1:
                            tmp = True
                            break
                    if tmp:
                        break
                if tmp:
                    res2[i, j] = 0

    return res2 * 255
```

e. Difference of Gaussian: 1

Description: Performs edge detection using the Difference of Gaussian method. It computes edges by applying two Gaussian filters with different standard deviations, emphasizing fine details and edge features in the image.



```
def difference_gauss(img_arr, threshold):
    mask = np.array([[-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1],
                     [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3],
                     [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
                     [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
                     [-7, -13, -17, 0, 85, 160, 85, 0, -17, -13, -7],
                     [-8, -13, -17, 15, 160, 283, 160, 15, -17, -13, -8],
                     [-7, -13, -17, 0, 85, 160, 85, 0, -17, -13, -7],
                     [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
                     [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
                     [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3],
                     [-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1]])

    # Expand input for padding using numpy
    img_arr = expand_with_replicate(img_arr, 5)

    # Perform convolution using numpy array
    res1 = my_conv(img_arr, mask, threshold)

    # Expand result for neighbor checking using numpy
    res1 = expand_with_replicate(res1, 5)
    res2 = np.ones((img_size0, img_size1))  # Initialize with ones
```

```python

    # Neighbor checking
    for i in range(img_size0):
        for j in range(img_size1):
            if res1[i + 1, j + 1] == 1:
                tmp = False
                for k in range(3):
                    for l in range(3):
                        if res1[i + k, j + l] == -1:
                            tmp = True
                            break
                    if tmp:
                        break
                if tmp:
                    res2[i, j] = 0

    return res2 * 255
```