# 数字图像处理 实验六

**一．Python小波变化库——PyWavelets**

关于小波变换，在python中有专门的库进行调用来处理，本次实验通过PyWalvelets来学习小波变换的实际处理过程。

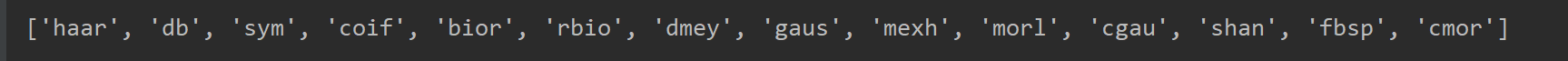
**PyWavelets API说明**

1.小波函数

查看小波族：pywt.families

1. **import** pywt
2. **print**(pywt.families())

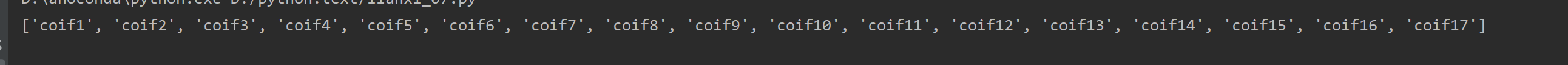
返回值如下：



查看每个小波族中提供的系数：pywt.wavelist

1. pywt.wavelist(family=None, kind=’all’)
2. family: 小波族的名称
3. kind:可以查看小波族下全部、离散或者连续的小波
4. **import** pywt
5. **print**(pywt.wavelist('coif'))

上面两行代码返回小波族中包含的小波名称



2.二维小波变换（一维和n维类似）

2.1 单层变换 pywt.dwt2

1. pywt.dwt2(data, wavelet, mode=’symmetric’, axes=(-2, -1))
2. data: 输入的数据
3. wavelet:小波基
4. mode:  默认是对称的
5. **return**: (cA, (cH, cV, cD))要注意返回的值，分别为低频分量，水平高频、垂直高频、对角线高频。高频的值包含在一个tuple中

2.2 单层逆变换 pywt.idwt2

1. pywt.idwt2(coeffs, wavelet, mode, axes)
2. coeffs: 经小波变换后得到的各层的系数
3. wavelet:小波基

2.3 多尺度变换 pywt.wavedec2

1. pywt.wavedec2(data, wavelet, mode=’symmetric’, level=None, axes=(-2, -1))
2. data: 输入的数据
3. wavelet:小波基
4. level: 尺度（要变换多少层）
5. **return**： 返回的值要注意，每一层的高频都是包含在一个tuple中，例如三层的话返回为 [cA3, (cH3, cV3, cD3), (cH2, cV2, cD2)， (cH1, cV1, cD1)]

2.4 阈值函数 pywt.threshold

1. pywt.threshold (data, value, mode=, substitute=)
2. data: 输入的数据
3. value：阈值
4. mode：阈值函数的类型 ，API提供四种类型（soft   hard  greater  less），具体实现效果可以参考API的实例来理解
5. substitute：要替换的值（经阈值函数处理后的值）

**相关资料**

PyWavelets 官方文档： <https://pywavelets.readthedocs.io/en/latest/ref/index.html>

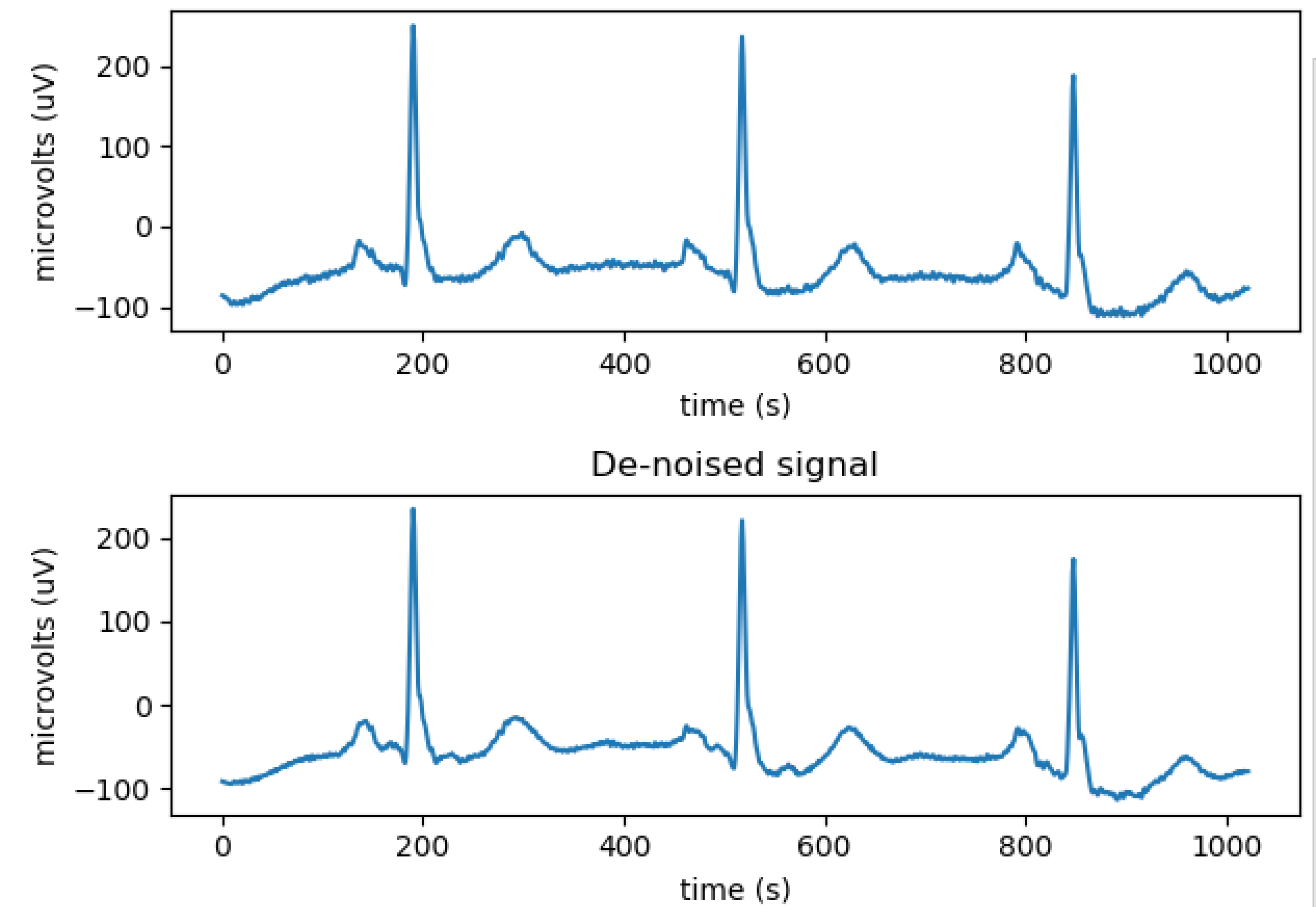
**二．小波变换实际应用**

**1.小波变换用于去噪**

我们使用pywavelets中内置的数据进行去噪分析，源程序如下：

1. **import** matplotlib.pyplot as plt
2. **import** pywt
3. # 获取数据:
4. ecg = pywt.data.ecg()  # 生成心电信号
5. index = []
6. data = []
7. **for** i **in** range(len(ecg)-1):
8. X = float(i)
9. Y = float(ecg[i])
10. index.append(X)
11. data.append(Y)
12. # 创建小波对象并且定义参数
13. w = pywt.Wavelet('db8')  # 选用Daubechies8小波
14. maxlev = pywt.dwt\_max\_level(len(data), w.dec\_len)
15. **print**("maximum level is " + str(maxlev))
16. threshold = 0.04  # 定义阈值
17. #小波分解
18. coeffs = pywt.wavedec(data, 'db8', level=maxlev)
19. plt.figure()
20. **for** i **in** range(1, len(coeffs)):
21. coeffs[i] = pywt.threshold(coeffs[i], threshold\*max(coeffs[i]))  # 将噪声滤波
22. datarec = pywt.waverec(coeffs, 'db8')  # 将信号进行小波重构
23. mintime = 0
24. maxtime = mintime + len(data) + 1
25. plt.figure()
26. plt.subplot(2, 1, 1)
27. plt.plot(index[mintime:maxtime], data[mintime:maxtime])
28. plt.xlabel('time (s)')
29. plt.ylabel('microvolts (uV)')
30. plt.title("raw signal")
31. plt.subplot(2, 1, 2)
32. plt.plot(index[mintime:maxtime], datarec[mintime:maxtime-1])
33. plt.xlabel('time (s)')
34. plt.ylabel('microvolts (uV)')
35. plt.title("De-noised signal")
36. plt.tight\_layout()
37. plt.show()

程序输出如下：

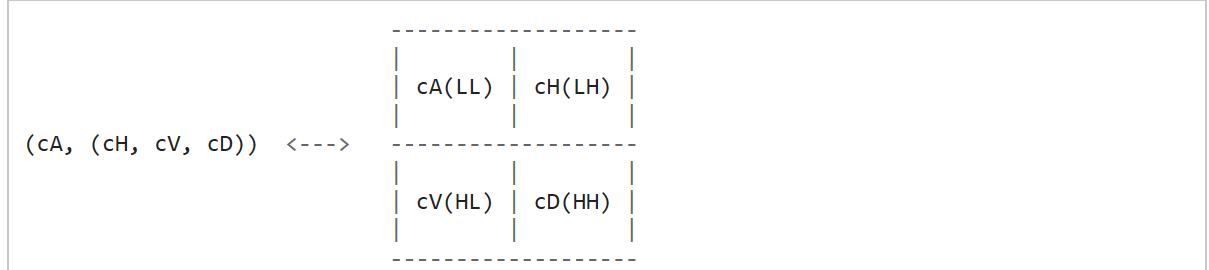


可以看到去噪后信号质量明显变好

**2.小波变换用于图像分解**

我们利用刚才的pywavelets中的pywt.dwt2()函数来进行小波变换 。

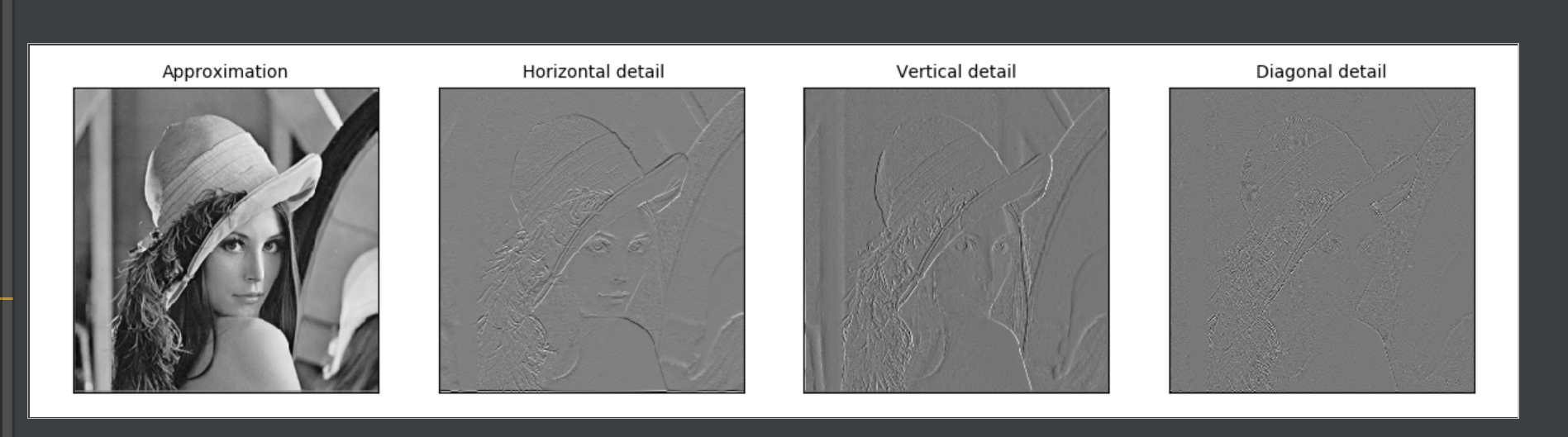
请注意：调用该函数后返回值为：LL，(LH，HL，HH)



这是每个不同的频率分量的位置，左上角为低频分量。

1. **import** cv2
2. **def** test\_pywt():
3. **import** numpy as np
4. **import** matplotlib.pyplot as plt
5. **import** pywt
6. **import** pywt.data
7. # 加载图片
8. original = cv2.imread('lena.jpg',0)
9. # 对图像进行小波转换，并且对其进行可视化
10. titles = ['Approximation', ' Horizontal detail',
11. 'Vertical detail', 'Diagonal detail']
12. coeffs2 = pywt.dwt2(original, 'bior1.3')
13. LL, (LH, HL, HH) = coeffs2
14. #plt.imshow(original)
15. #plt.colorbar(shrink=0.8)
16. fig = plt.figure(figsize=(12, 3))
17. **for** i, a **in** enumerate([LL, LH, HL, HH]):
18. ax = fig.add\_subplot(1, 4, i + 1)
19. ax.imshow(a, interpolation="nearest", cmap=plt.cm.gray)
20. ax.set\_title(titles[i], fontsize=10)
21. ax.set\_xticks([])
22. ax.set\_yticks([])
23. fig.tight\_layout()
24. plt.show()
25. test\_pywt()

程序输出结果如下：

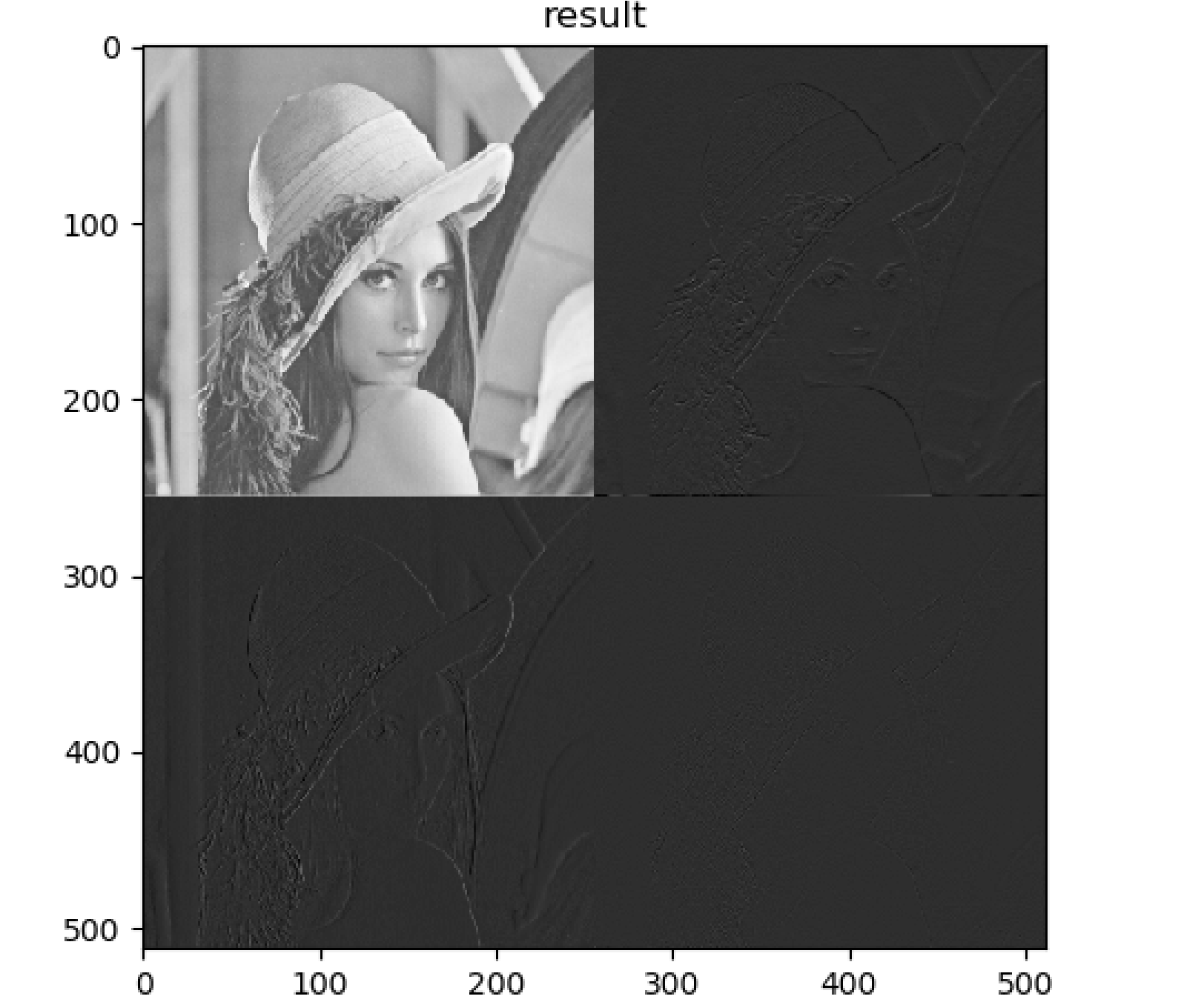


这里程序输出了Lena图像的近似图，水平细节，垂直细节，对角线细节四个部分。

我们还可以将上述可视化结果换种输出方式：

1. **import** numpy as np
2. **import** pywt
3. **import** cv2
4. **import** matplotlib.pyplot as plt
5. **def** test():
6. img = cv2.imread("lena.jpg")
7. # 将多通道图像变为单通道图像
8. img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY).astype(np.float32)
10. plt.figure('二维小波一级变换')
11. coeffs = pywt.dwt2(img, 'haar')
12. cA, (cH, cV, cD) = coeffs
14. # 将各个子图进行拼接，最后得到一张图
15. AH = np.concatenate([cA, cH], axis=1)
16. VD = np.concatenate([cV, cD], axis=1)
17. img = np.concatenate([AH, VD], axis=0)
19. # 显示为灰度图
20. plt.imshow(img,'gray')
21. plt.title('result')
22. plt.show()
23. test()

输出结果如下：

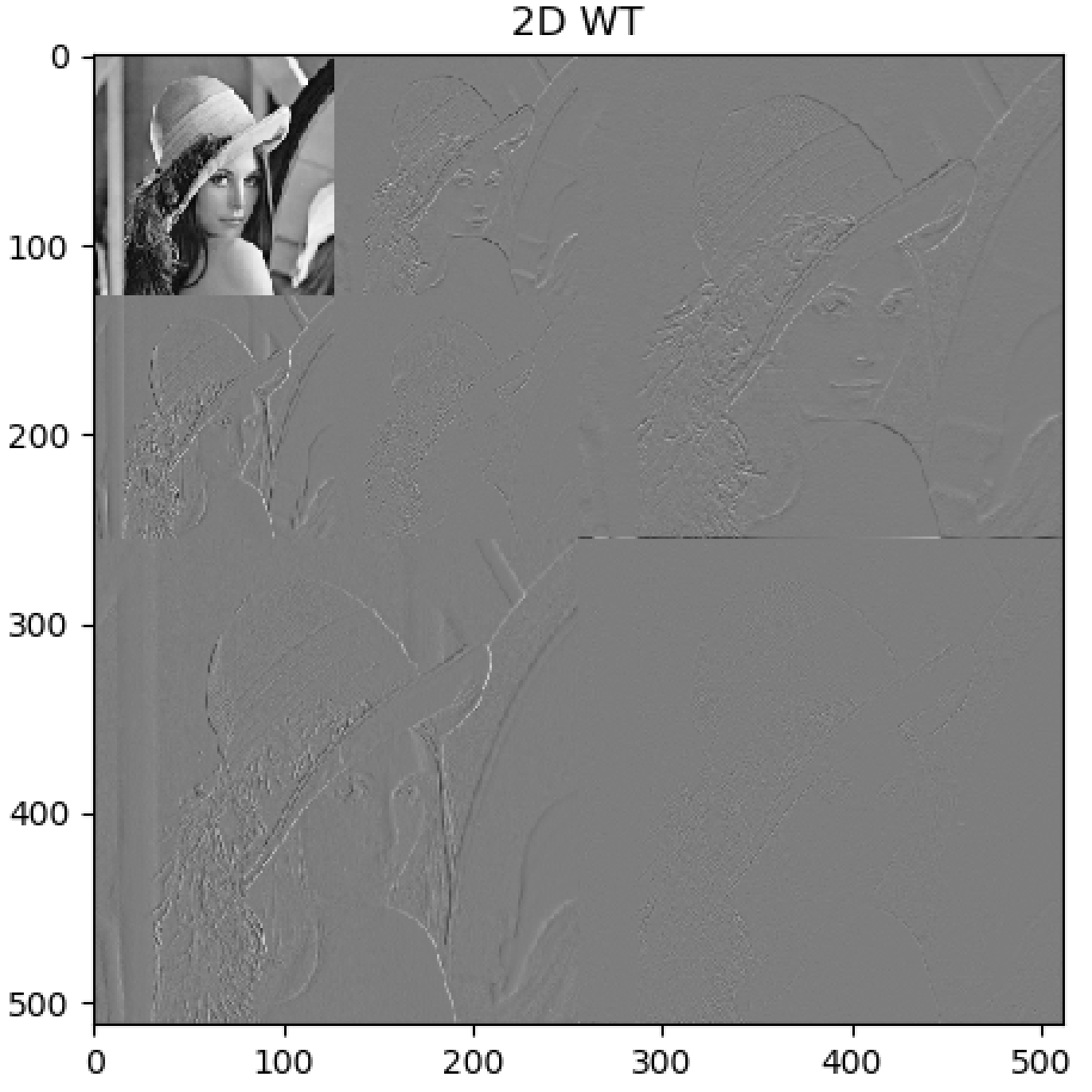


这也就是对二维图像进行单级变换得到后的图像，我们只是将上述并列的四幅图像拼接起来。

同样的，我们可以调用wavedec2()函数进行二维图像的多级分解。具体实施代码如下：

1. **import** numpy as np
2. **import** pywt
3. **import** cv2
4. **import** matplotlib.pyplot as plt
6. img = cv2.imread("lena.jpg")
7. #img = cv2.resize(img, (448, 448))
8. # img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB).astype(np.float32)
9. img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY).astype(np.float32)
11. plt.figure('二维图像多级分解')
12. coeffs = pywt.wavedec2(img, 'haar', level=2)
13. cA2, (cH2, cV2, cD2), (cH1, cV1, cD1) = coeffs
15. # 将每个子图的像素范围都归一化到与CA2一致  CA2 [0,255\* 2\*\*level]
16. AH2 = np.concatenate([cA2, cH2+510], axis=1)
17. VD2 = np.concatenate([cV2+510, cD2+510], axis=1)
18. cA1 = np.concatenate([AH2, VD2], axis=0)
20. AH = np.concatenate([cA1, (cH1+255)\*2], axis=1)
21. VD = np.concatenate([(cV1+255)\*2, (cD1+255)\*2], axis=1)
22. img = np.concatenate([AH, VD], axis=0)
24. plt.imshow(img,'gray')
25. plt.title('2D WT')
26. plt.show()

输出结果如下：



**编程实践：**

请给出对标准的lena图像进行一级分解，二级分解后的图像的与原始图像的压缩比是多少（即原图像的大小与处理后的图像的大小之比）