数字媒体实验一

任务一：OpenMv 图像处理

原图：

（1）能获取图像任意一点的像素值,getPixel(x,y)

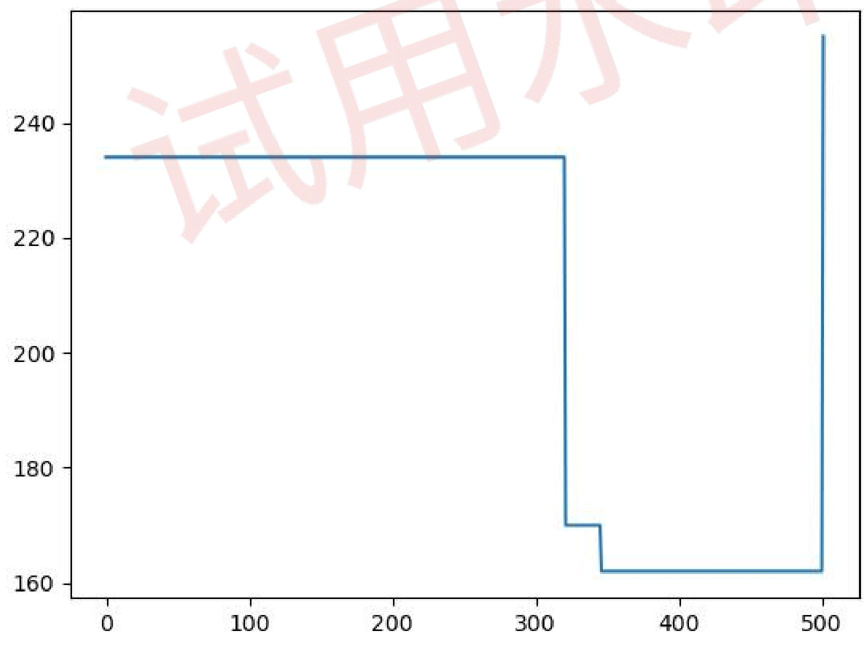
```
def getPixel(x,y):
    im=Image.open('1.bmp')#文件的路径
    im2=im.convert("RGB")
    print(im2.mode)
    print(im2.getpixel((x,y)))#（0，0）表示像素点的坐标
```

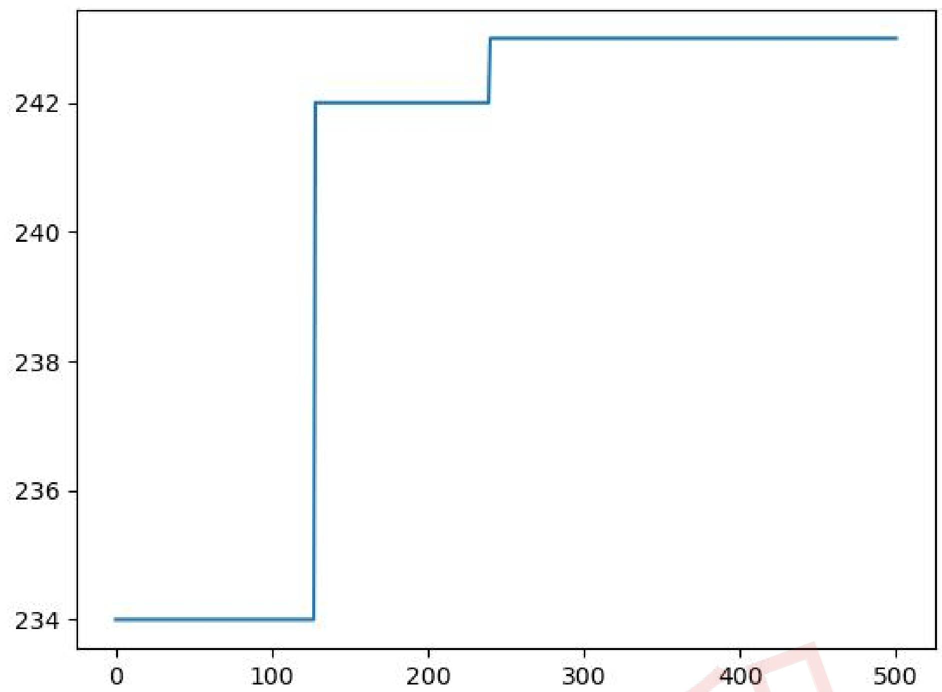
```
getPixel( x: 10, y: 10)          #要求一
```

```
(64, 160, 192)
```

（2）能将图像的任意一行和一列像素值在显示窗口画出来 drawRow(row),drawCol(col)

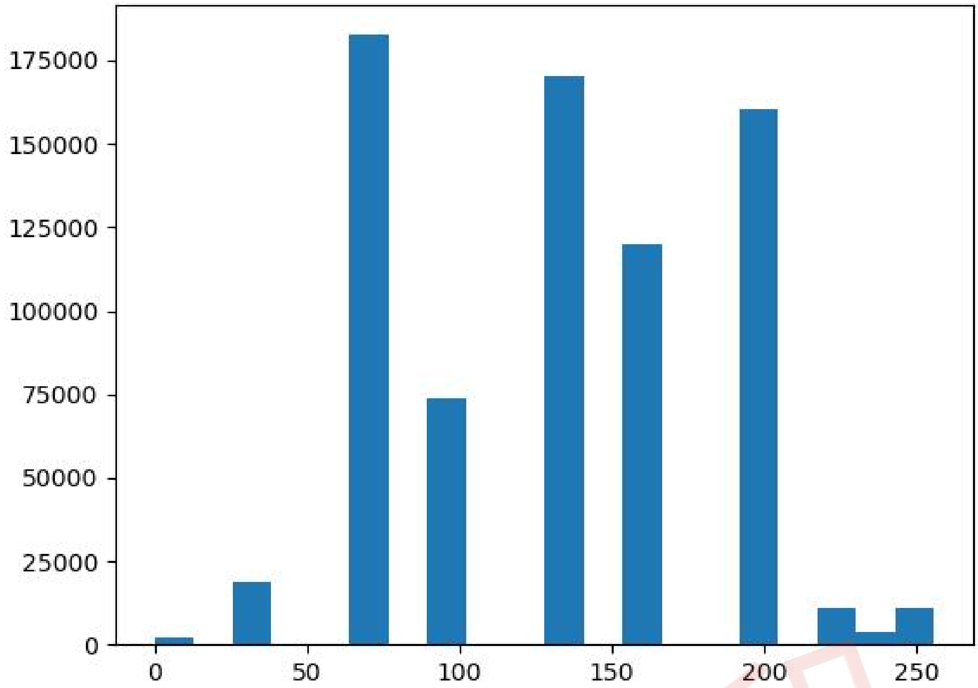
```
# 要求2： 将任意一行和一列的像素值在窗口显示
1 usage
def drawRow(row):
    pixels = list(img.getdata())
    width, height = img.size
    print(width,height)
    row_pixels = pixels[row*width:(row+1)*width]
    plt.plot(row_pixels)
    plt.show()

1 usage
def drawCol(col):
    pixels = list(img.getdata())
    width, height = img.size
    col_pixels = [pixels[i*width+col] for i in range(height)]
    plt.plot(col_pixels)
    plt.show()
```

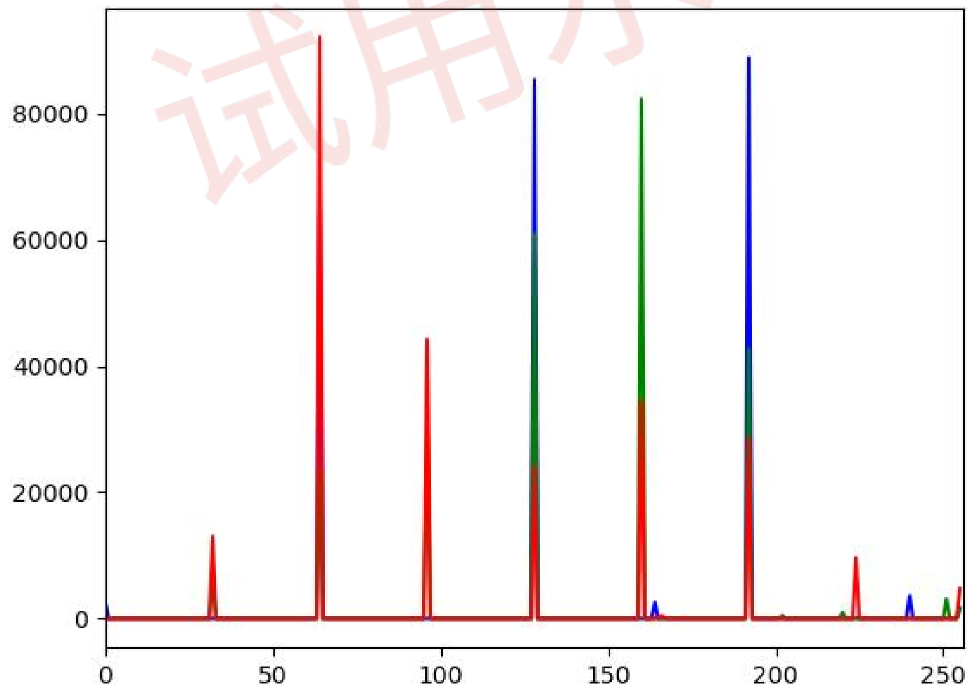




（3）能统计图像的像素直方图 getHist(image)，并计算图像的信息熵,getEntropy(image)

```
# 要求3 : 统计图像的像素直方图
1 usage
def getHist(image_path: str):
    # 一维直方图（单通道直方图）
    img = cv.imread(image_path, cv.IMREAD_COLOR)
    cv.imshow( winname: 'input', img)
    color = ('blue', 'green', 'red')
    # 使用plt内置函数直接绘制
    plt.hist(img.ravel(),  bins: 20,  range: [0, 256])
    plt.show()
    # 一维像素直方图，也即是单通道直方图
    for i, color in enumerate(color):
        hist = cv.calcHist( images: [img],  channels: [i],  mask: None,  histSize: [256],  ranges: [0, 256])
        #print(hist)
        plt.plot( *args: hist, color=color)
        plt.xlim([0, 256])
    plt.show()
    cv.waitKey(0)
    cv.destroyAllWindows()
```


175000
150000
125000
100000
75000
50000
25000
0
0
50
100
150
200
250

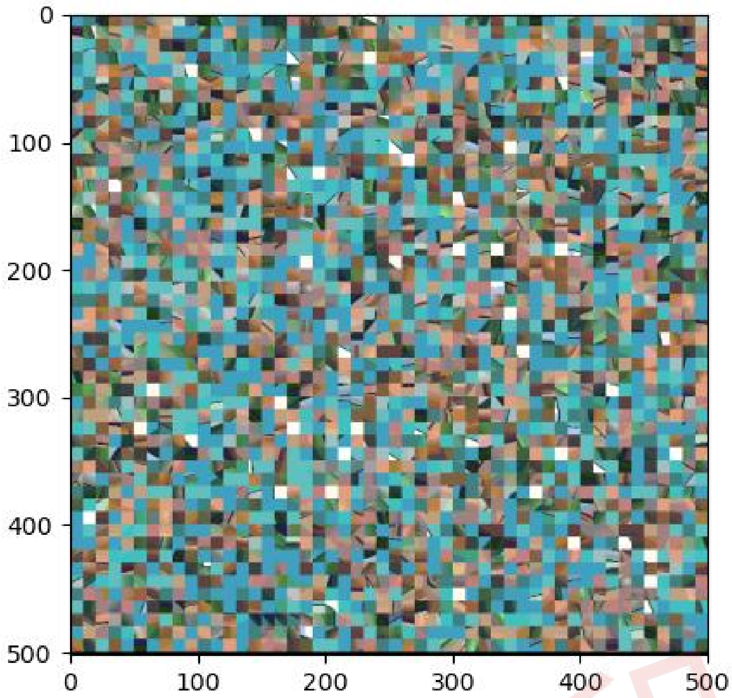

80000
60000
40000
20000
0
0
50
100
150
200
250

```python
# 要求3 ： 计算图像的信息熵
def getEntropy():
    # img = cv2.imread('20201210_3.bmp',0)
    # img = np.zeros([16,16]).astype(np.uint8)

    a = [i for i in range(256)]
    img = np.array(a).astype(np.uint8).reshape(16, 16)

    hist_cv = cv2.calcHist([img], [0], None, [256], [0, 256])  # [0,256]的范围是
    P = hist_cv / (len(img) * len(img[0]))  # 概率
    E = np.sum([p * np.log2(1 / p) for p in P])
    print("一维熵: ",E)  # 熵
```

```
一维熵:  8.0
```

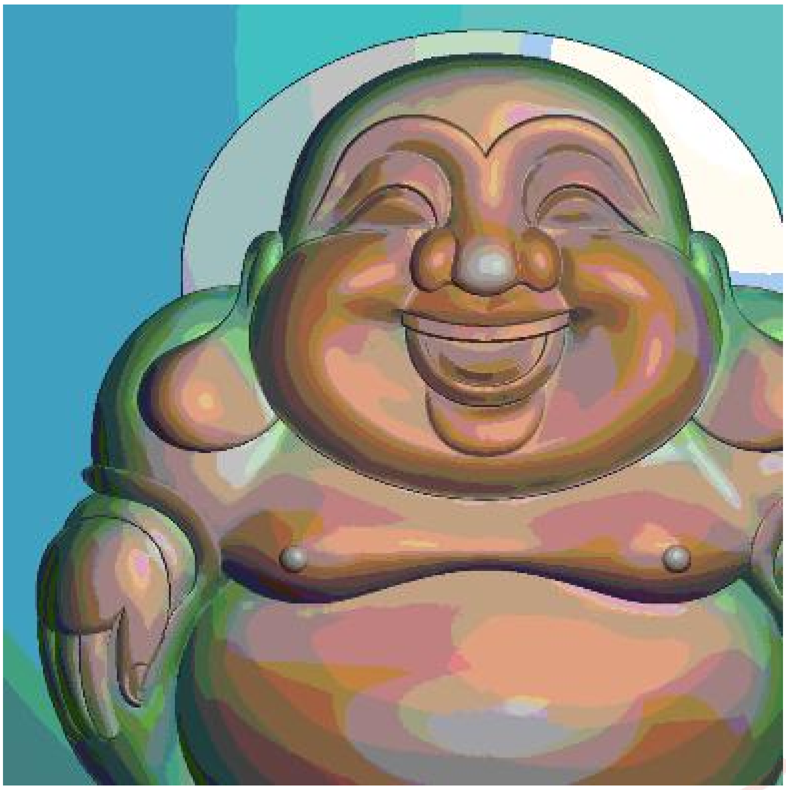
（4）能将图像分成任意块大小，PermutationFun(inputImage, blockwidth, blockheight,seed), 例如 4*4,8*8,16*16,32*32,64*64，并置乱块的位置并显示（类似马赛克效果）；能指定区域内的图像分块并置乱块的顺序再显示（本条可以调用软件或库的读图接口）

```python
def PermutationFun(inputImage, blockwidth, blockheight, sed):
    seed(sed)
    width, height = inputImage.size
    xblock = width // blockwidth
    yblock = height // blockheight
    regions = []

    for i in range(0, yblock * blockheight, blockheight):
        for j in range(0, xblock * blockwidth, blockwidth):
            region = inputImage.crop((j, i, j + blockwidth, i + blockheight))
            regions.append(region)
    shuffle(regions)
    outputImage = Image.new('RGB', (width, height))
    idx = 0
    for i in range(0, yblock * blockheight, blockheight):
        for j in range(0, xblock * blockwidth, blockwidth):
            outputImage.paste(regions[idx], (j, i))
            idx += 1

    plt.imshow(outputImage)
    plt.show()
```



（5）尝试截图图像的任意一个区域并存为另一幅图像

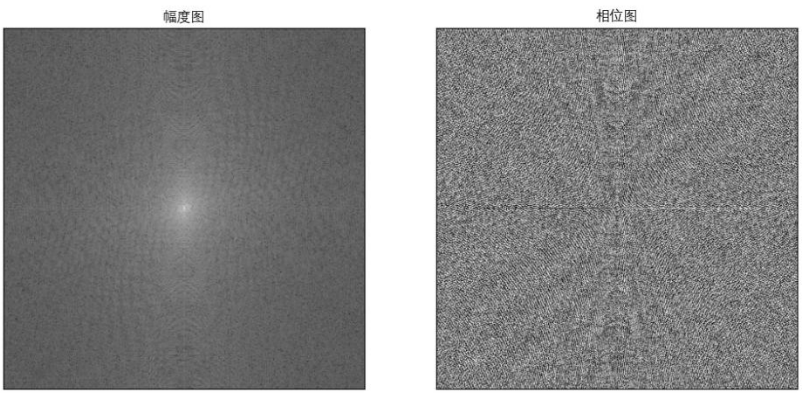
```python
def crop_bmp(input_filename, output_filename, left, top, right, bottom):
    data = read_bmp(input_filename)
    file_header = data[:14]
    info_header = data[14:54]
    pixels = data[54:]
    width = int.from_bytes(info_header[4:8], byteorder='little')
    height = int.from_bytes(info_header[8:12], byteorder='little')
    row_size = (width * 3 + 3) & ~3
    pixel_array_offset = int.from_bytes(file_header[10:14], byteorder='little')
    cropped_pixels = bytearray()
    for y in range(top, bottom):
        start = pixel_array_offset + y * row_size + left * 3
        end = start + (right - left) * 3
        cropped_pixels.extend(pixels[start:end])
    new_width = right - left
    new_height = bottom - top
    new_row_size = (new_width * 3 + 3) & ~3
    new_pixel_array_size = new_row_size * new_height
    new_file_size = 14 + 40 + new_pixel_array_size
```



（6）理解 FFT 的优势：用一维的 DFT 变换对图像块进行变换，分别显示其幅度图和相位图（注意如何可视化结果，例如，将系数区间归一化到[0,255]，或者系数取对数等）；用 FFT 变换来对图像块进行变换，看看其速度；

代码详见 img_function2.py，结果如下：

```
一维DFT变换时间： 0.017153501510620117
FFT变换时间： 0.010176420211791992
```

（7）理解 DCT 变换的能量聚集特性，8*8 变换后，保留左上角 k 个系数后（做 Zigzag 扫描，将 8*8 转化为 64 维向量），再做逆 DCT 变换，恢复原始图像，比较原始图像与恢复图像的 PSNR 值和 SSIM 值(需要查阅 PSNR 和 SSIM 的公式并实现)

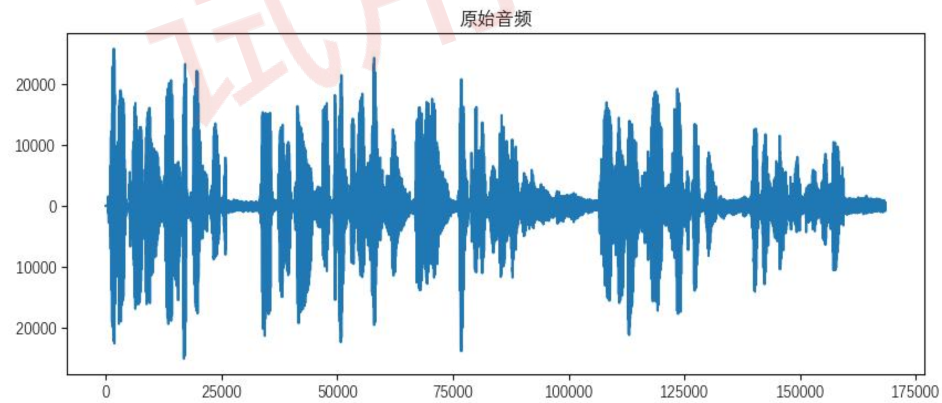
代码详见 img_function_3，测试结果如下：

```
PSNR:  20.848105719360333
SSIM:  0.6260750701162212
```

任务二：wav 音频文件处理

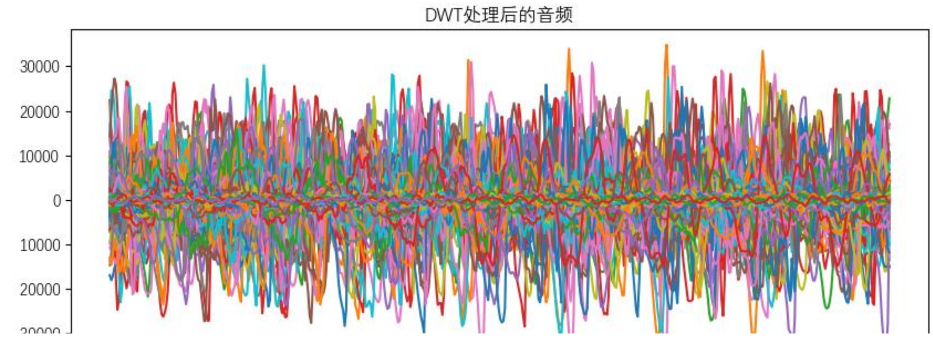
（1）能阅读 wav 音频文件，并将原始的 PCM 音频数据显示出来，并画出其大小示意图（画出波形图）

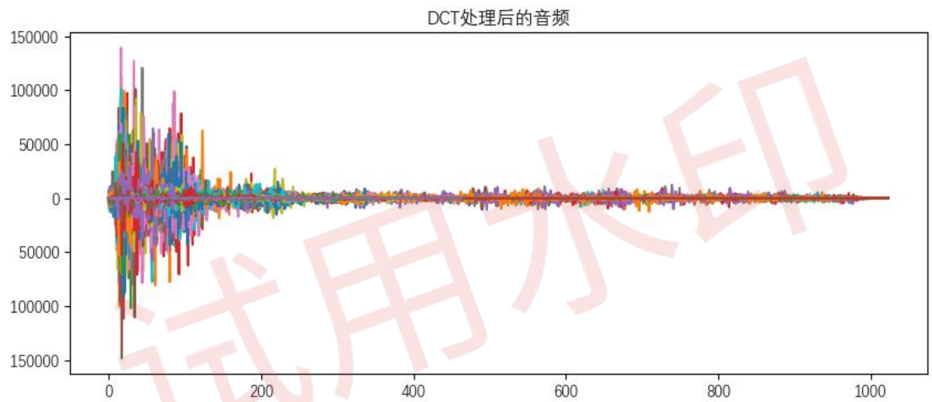
```python
# 画出原始音频，以及处理后音频的图形
plt.figure(figsize=(10, 4))
plt.plot(data)
plt.title('原始音频')
plt.show()
```

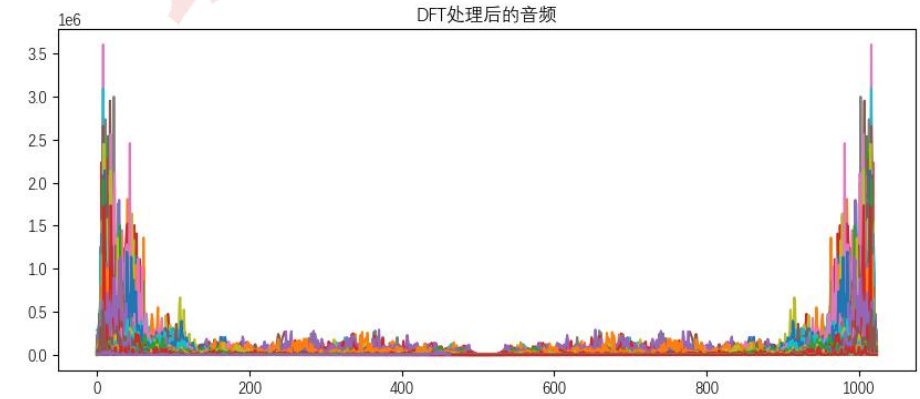



（2）读入音频文件，以 1024 长度对音频分窗处理，然后对其进行一维的 DFT，DCT，DWT 处理，然后画出原始音频，以及处理后音频的图形

代码详见 audio_function_1，结果如下：

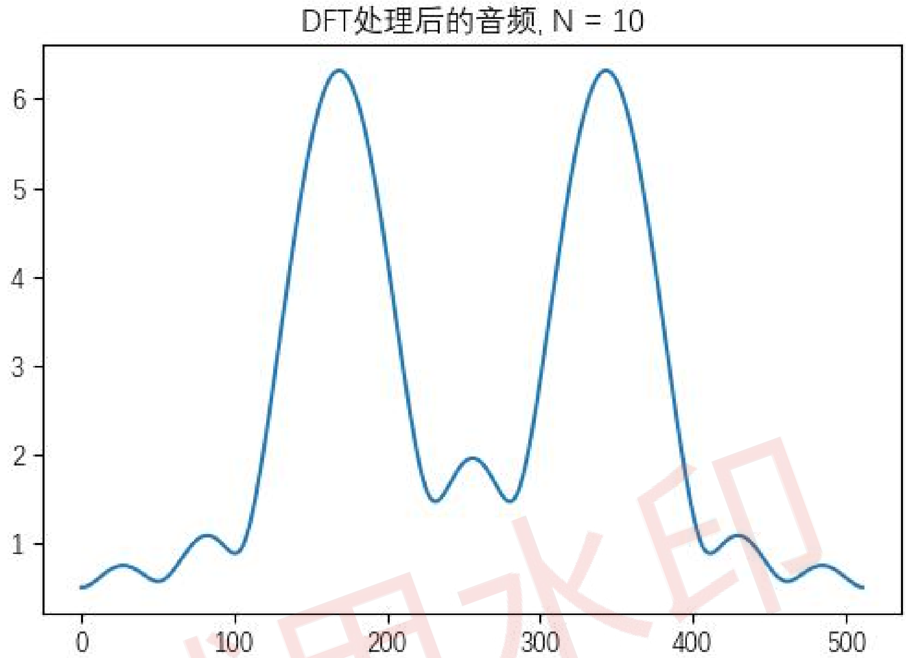


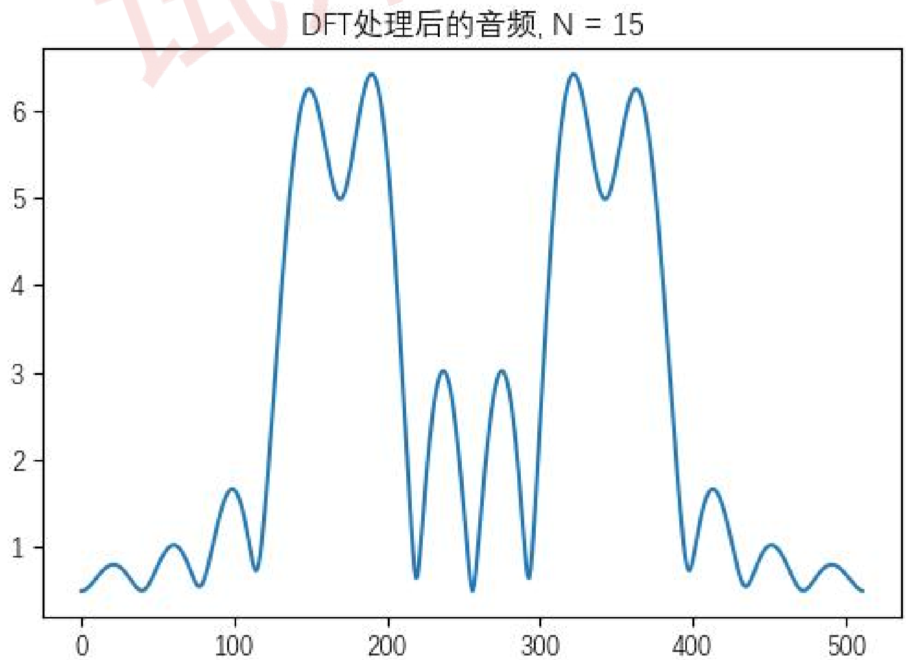


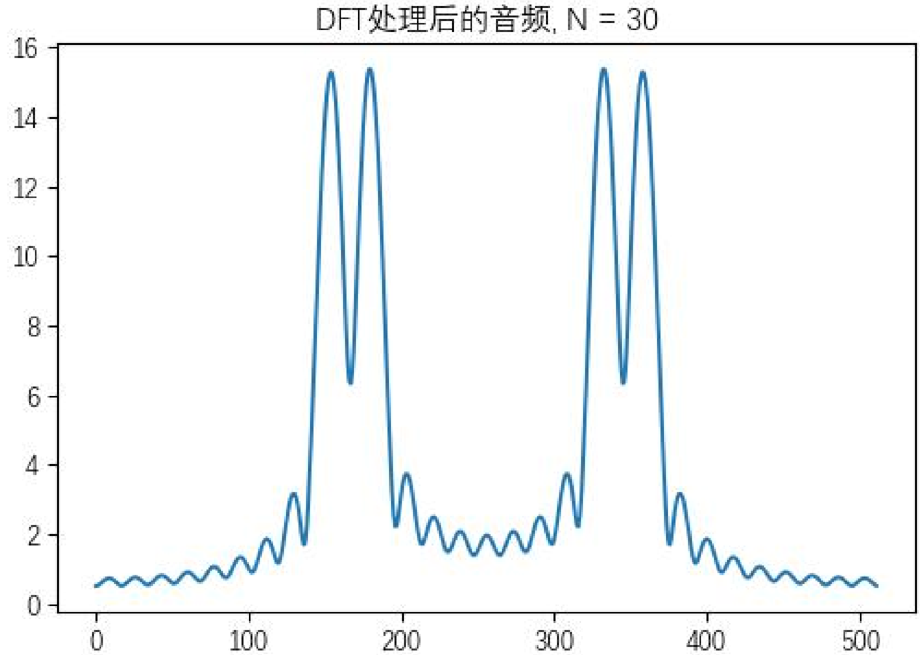


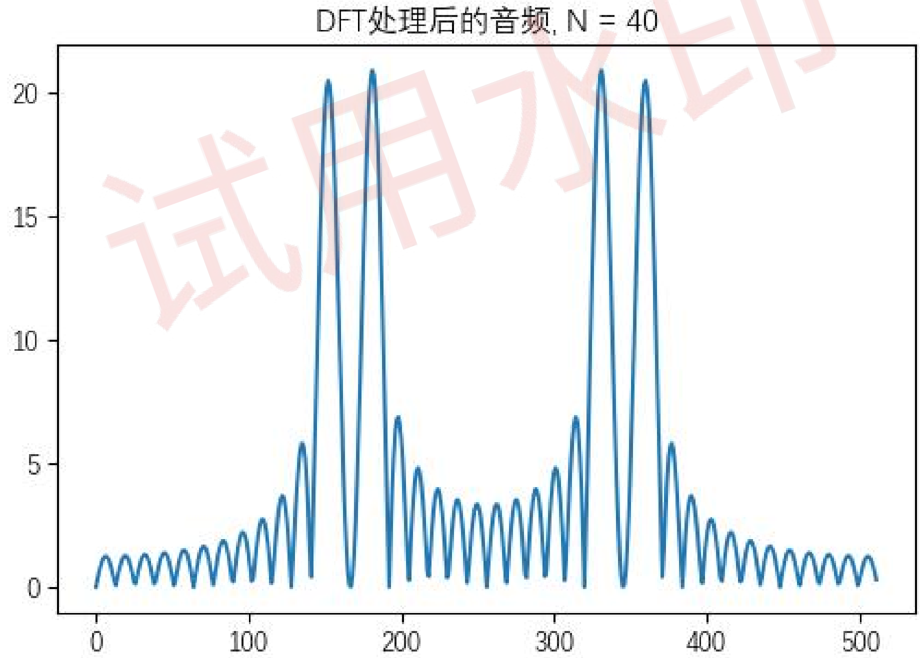
任务三：模拟信号分析

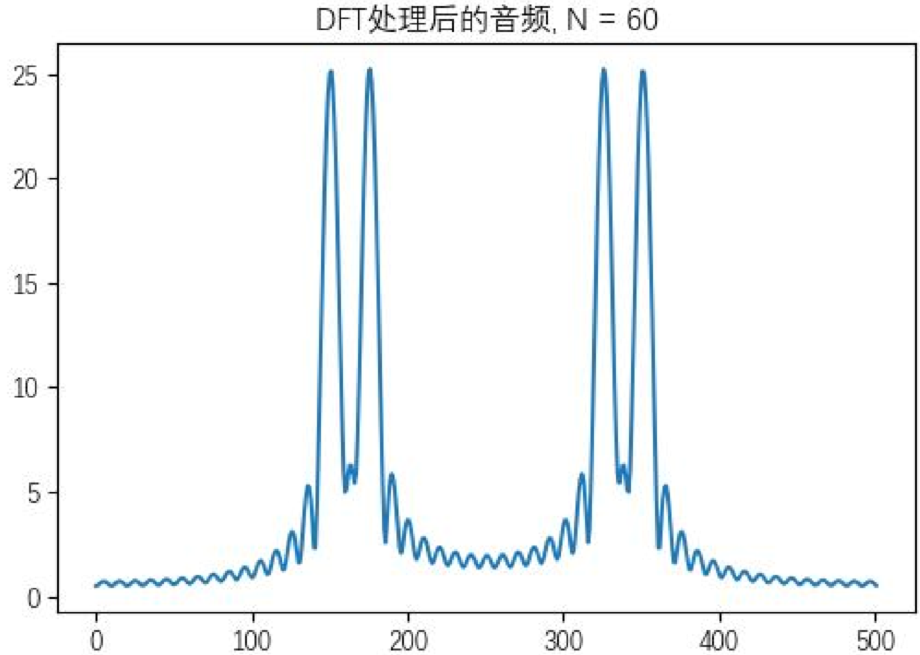
模拟两个信号的频率分别是 F1 = 30 Hz，F2 = 35 Hz，假设按照采样区间为 T = 0.02 秒进行采样.考虑不同数据长度 N = 10, 15, 30, 40,60,70，100，并填 0 扩充至 512 个点，然后分别画出其图像，并解释其现象！（参考课件内容）

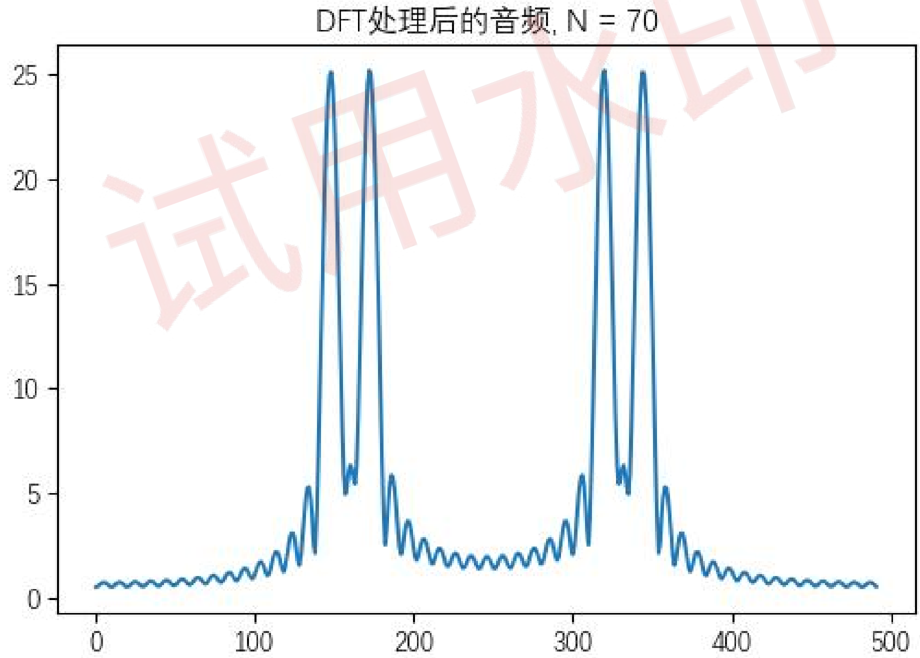


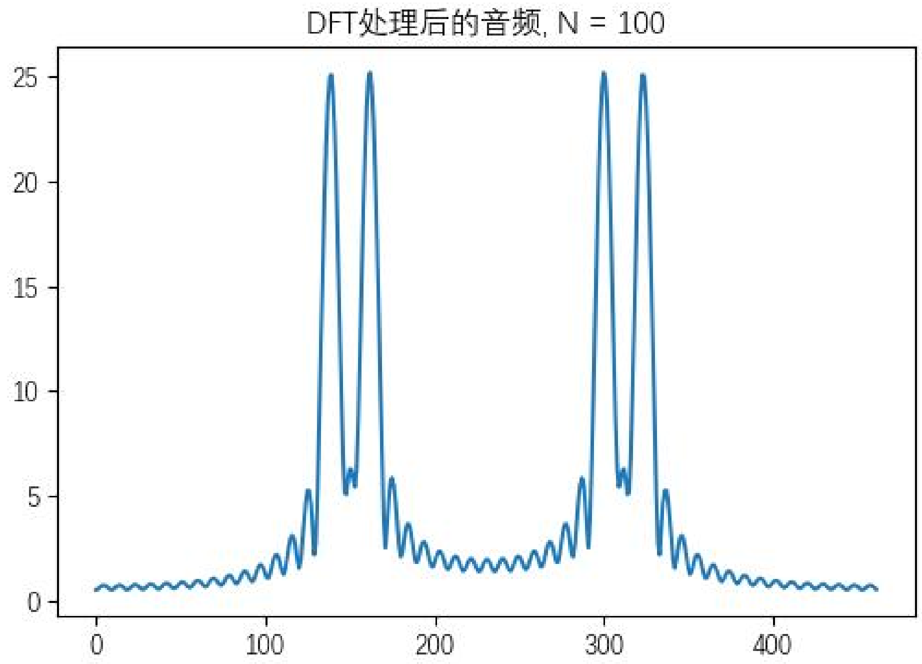



DFT处理后的音频, N = 30
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500


DFT处理后的音频, N = 40
0
5
10
15
20
0
100
200
300
400
500


DFT处理后的音频, N = 60
25
20
15
10
5
0
0
100
200
300
400
500


DFT处理后的音频, N = 70
25
20
15
10
5
0
0
100
200
300
400
500



分析：已知

DFT的频率分辨率是指：

$$F_{res} \sim \frac{1}{NT}[Hz]$$

由 F2-F1=5Hz　>　1/NT　解得 N > 10，其中 T 为固定值 0.02，所以只有在 N 大于 10 之后的图像才能观察到四段波峰的现象，并且随着 N 越大越明显。因为分辨率与采样区间、点数有关，所以可以通过：1 增加采样率⇒扩展频率区间　2　增加观察时间⇒提高频率分辨率的方法来提升分辨率，本次因为 T 是固定值，故采用的是 1 方法。