**Matlab综合实验三 图像处理实验报告**

学号：2017011090

班级：无78

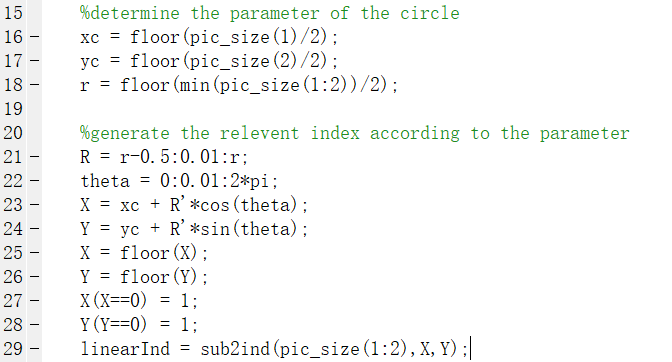
姓名：游子權

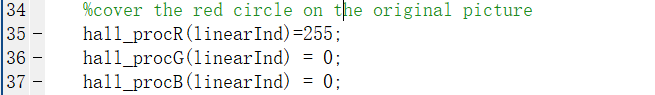
说明：在本作业第一章、第二章中，我皆以「用最少量的循环语句完成任务」为核心目标来设计程序。

1. **基础知识**

**(a)关键代码及方法如下：**

* q1\_2a.m







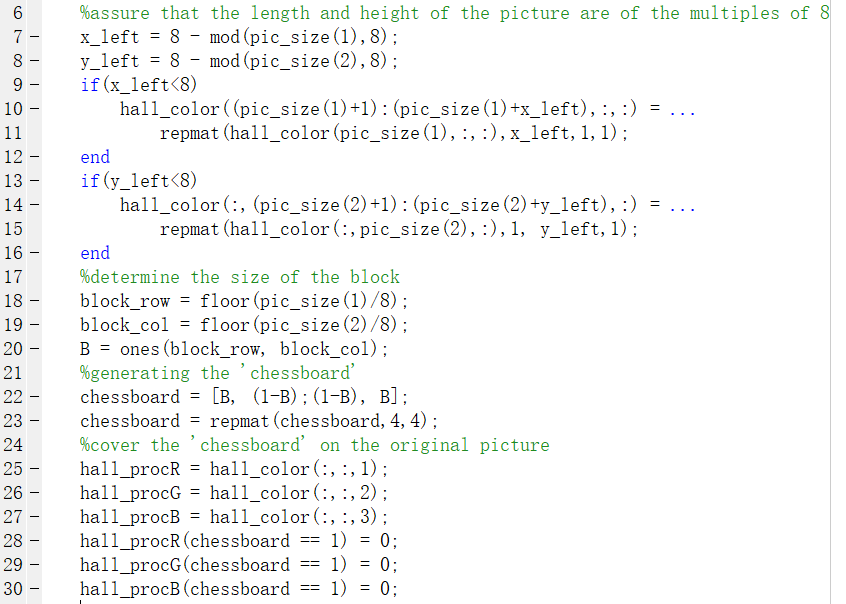
* 第16~18行：根据图片大小确定圆心位置与圆的大小。
* 第21~29行：利用圆的参数式，产生要染成红色的像素于图片矩阵中的行、列座标。并用sub2ind函式转成线性索引值（index）。
* 第35~37行：把上述的位置「染成红色」。



* 第44行：利用Matlab的Image I/O函数imwrite进行写入。
* **结果（hall\_withCircle.jpg）：成功，图片上出现红圈圈，惟其边缘锯齿状的现象较严重，是因为该图片本身就低清，离散化程度过高所致**。

**(b)关键代码及方法如下：**

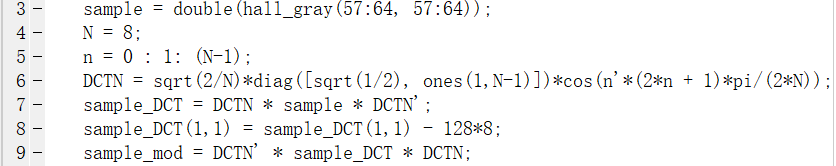
* q1\_2b.m



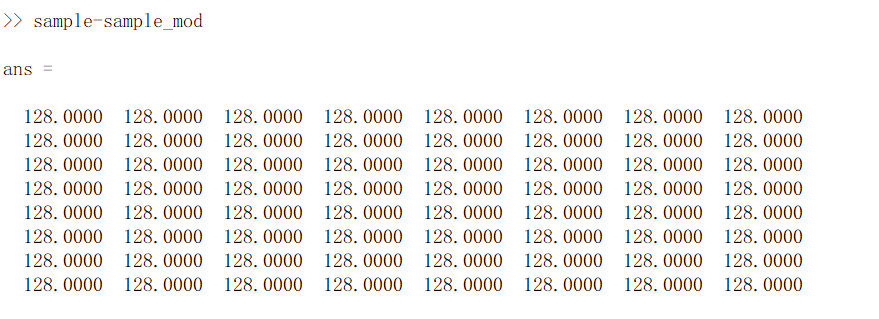
* 第7~16行：因为西洋棋棋盘大小为8×8，若能保证图片的长宽皆为8的倍数可方便划分「格子」。本人的做法是当图片不为8的倍数时，以其最后一行、最后一列的值填满至最接近的8的倍数的行、列大小。**本程序在之后第二章图像压缩编码中仍可沿用。因为其要求将图片分成若干个8×8的小块，故亦需保证长、宽为8的倍数。**
* 第18~20行：确定图像上每一「块」棋盘方格对应几行、几列。
* 第22~23行：重覆棋盘的最小单位以制作棋盘。
* 第25~30行：把做好的棋盘覆盖在原图上
* **结果（hall\_withChessboard.jpg）：成功**。

1. **图像压缩编码**
   1. 可以。二维离散余弦变换后的(1,1)元代表的是整张图片的直流分量。若我们是对N×N的图片块做二维DCT，且欲减去直流分量M，只需在其变换后系数矩阵的(1,1)元减去N×M即可。原因如下：

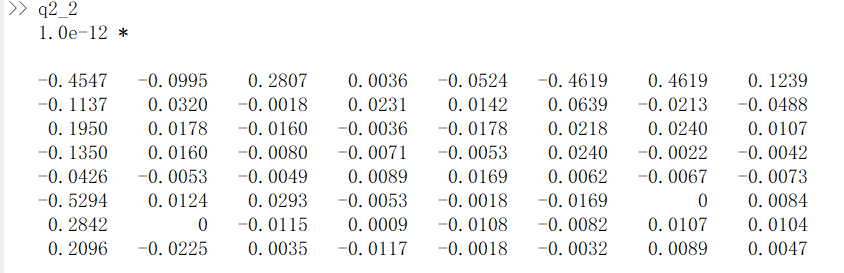
* **关键代码：q2\_1.m**



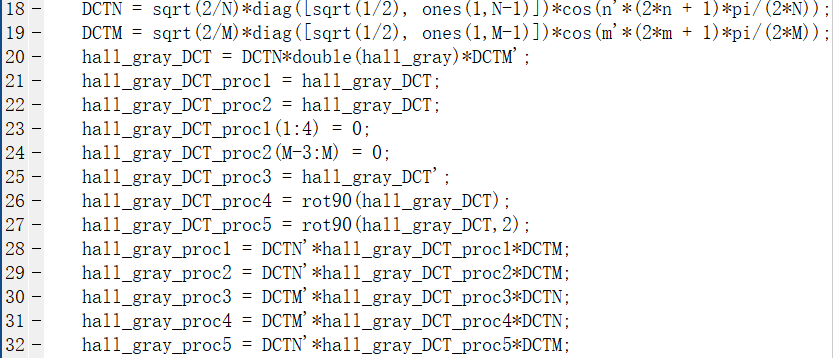
* 第6行：按照DCT的表达式生成对N维座标作DCT变换的矩阵。
* 第8行：如题要求所述，对8\*8的图片块，每个像素灰度值减去128，即相当于二维DCT系数的直流系数（第1行第1列）减去128×8。
* 第9行：对处理过的DCT系数做IDCT。以便比较逆变换后每个像素的灰度值是否皆比原本少128。
* **结果：成功。确实各像素灰度值皆比原本少128。**



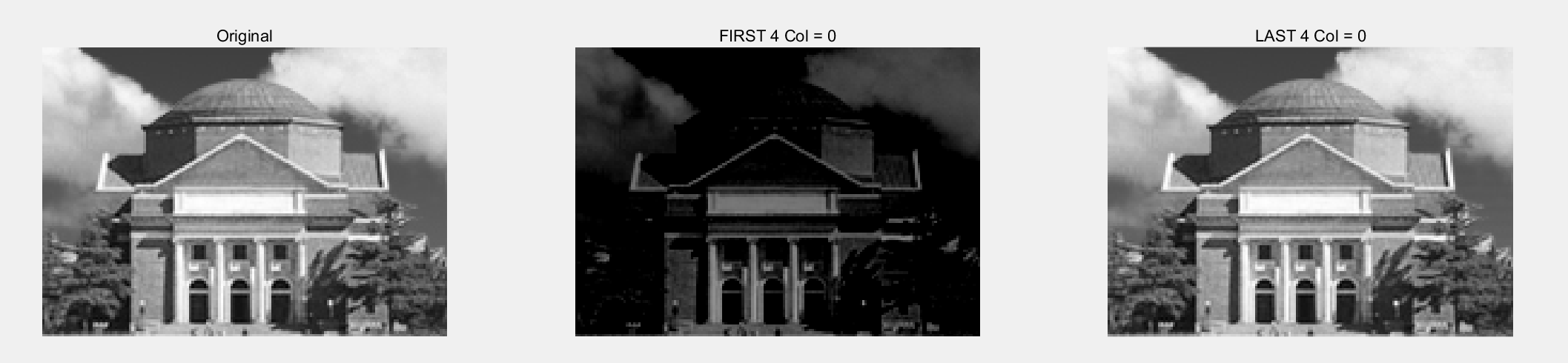
* 1. 本题的目标于上一题已经实现，本处仅比较与系统自带函数运算结果是否相同。
* **关键代码：q2\_2.m（代码附于附件中，此处从略）**
* 第7~9行：分别按自己矩阵计算，再按matlab的库函数计算，再将两者计算之差输出至命令行中。
* **结果：两者之差小于，可视为数值计算产生的舍入等误差。**



* 1. DCT系数(1)偏左上方为直流分量、(2)偏左下方为竖直方向快速振荡的分量、(3)偏右上方为水平方向快速振荡的分量、(4)偏右下方为水平、竖直方向皆快速振荡的分量。
* 将DCT后的系数前四列置0，影响的主要为如上所述的(1)(2)部分，直流分量变为0，使得处理后的图像黯淡许多；水平方向的快速振荡分量被置为0，但人眼的鉴别力有限，难以鉴别此所造成的差异。
* 将DCT后的系数的后四列系置0，影响的主要为如上所述的(3)(4)部分，但人眼的鉴别力有限，难以鉴别高频分量存在与否造成的差异。
* **本题关键代码与结果并于第4.题中说明。**
  1. 兹分述如下：
* 将DCT系数转置即相当于对原图像的灰度值矩阵转置。原因如下：
* DCT系数原本就是左上角密集（非0数值较多）、右下角稀疏，将DCT系数逆时针旋转90度，变成以左下角最为密集，而承3.(2)所述，偏左下方为竖直方向的高频分量，故经此一旋转后，可在图上观察到明显横条纹。有一种老电影画面的感觉。
* 同上所述，转180度后系数以右下角最为密集，而承3.(4)所述，偏右下方为水平、竖直两方向的高频分量，故经此一旋转后，可在图上观察到明显的网状条纹。有另一种老电影画面的感觉。
* **关键代码：q2\_3\_4.m**

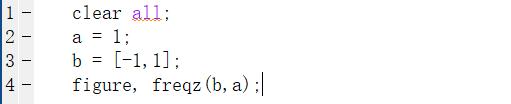


* 第18~19行：由于矩阵维度大小不一，需不同维度的DCT变换阵。
* 第21~27行：对于DCT系数阵做不同的处理。
* 第28~32行：将各个处理后的DCT系数逆变换。
* 第33~50行（从略）：将原图及处理后的各图绘在同一图窗中。
* **结果：如下图，与预期相符。**

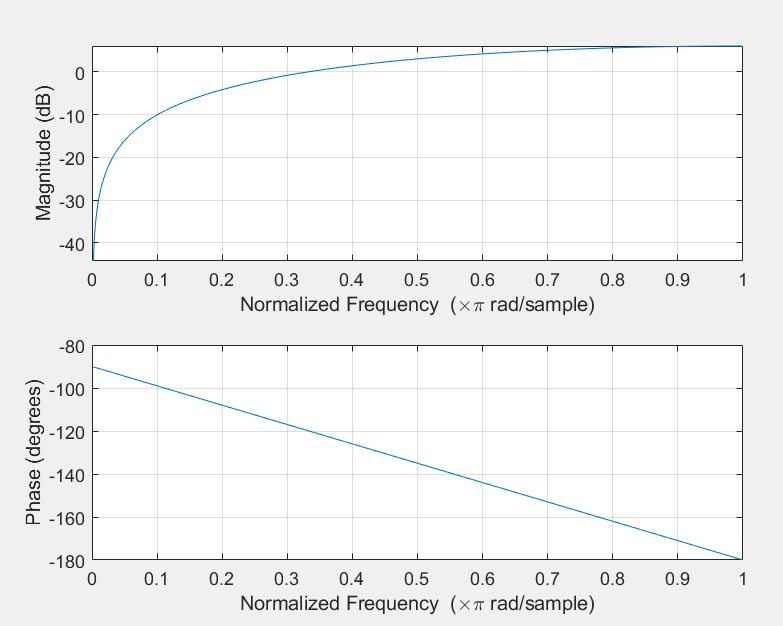




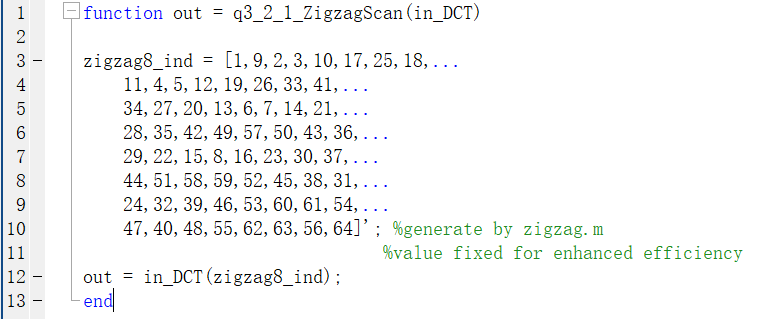
* 1. 如果认为差分编码是一个系统，则其输出与输入满足差分方程：，且。则可用Matlab中自带的函数分析其特性。
* **关键代码：q2\_5.m**



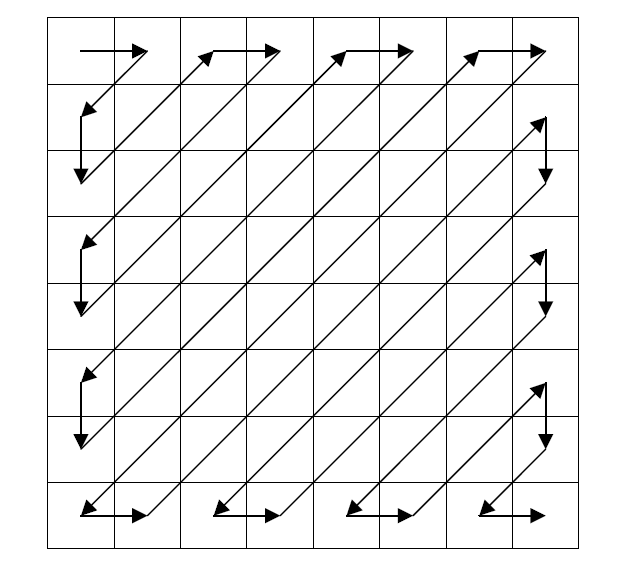
* 第4行：使用freqz函数求其频率响应。
* **结果：**上图为幅频响应，下图为相频响应。可见其**为一高通滤波器**。说明**DC系数的低频分量应该更多**。



* 1. 显然，设输入的数为，则给出该数所在的Category为
* **关键代码：getDCCategory.m（此处从略）**
  1. **思路：**Matlab擅于处理矩阵运算，故若能将Zigzag扫描路径所对应的线性索引（index）编列出来存至一数组中（下称「**顺序数组**」），让Matlab根据顺序数组中的索引在矩阵内依序取值，即可实现Zigzag扫描。对于一个8×8的矩阵，我们当然可以硬是用肉眼观察遍历的顺序编列出顺序数组，如下所示：
* **关键代码：q3\_2\_1\_ZigzagScan.m（本函数复用于本作业第三章第2题的第1种方法）**



* 第3~10行：编列顺序数组。
* 第12行：根据顺序数组在矩阵中取值。

但事实上我们并不满足于此。我们知道一旦有了顺序数组，就可以在Matlab中做高效的Zigzag扫描，但对于任意大小为n×n的方阵，有无可能编写程序自动生成顺序数组？我们可以透过观察找出Zigzag扫描中依序经过的行号、列号规律。以n = 8为例：

经过的列（column）号依序为：1,2,1, 1,2,3,4,3,2,1,

1,2,3,4,5,6,5,4,3,2,1,

1,2,3,4,5,6,7,8,7,6,5,4,3,2,1,

2,3,4,5,6,7,8,

8,7,6,5,4,3,

4,5,6,7,8,

8,7,6,5,

6,7,8,

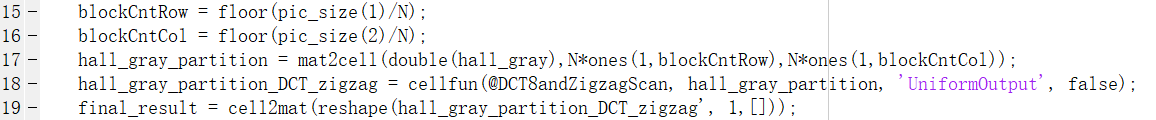
8,7,

8

如上分组后，可以见其有明显规律。先是一堆回文数列，当回文数列中的最大值恰为矩阵大小8时，之后开始从8数到k或从k数到8的数列（根据k的奇偶性不同，有不同的方向）。同理可针对行号作上述讨论。再针对n是奇数或偶数的不同状况分类讨论，即可得到任意大小方阵做Zigzag扫描的行号、列号顺序规律。有了行号、列号顺序，可调用Matlab内置的sub2ind函数计算其线性索引值。

* **关键代码：zigzag.m（代码太长、规则繁琐，在此略去）**
  1. **关键代码：**
* **q2\_8\_9.m**

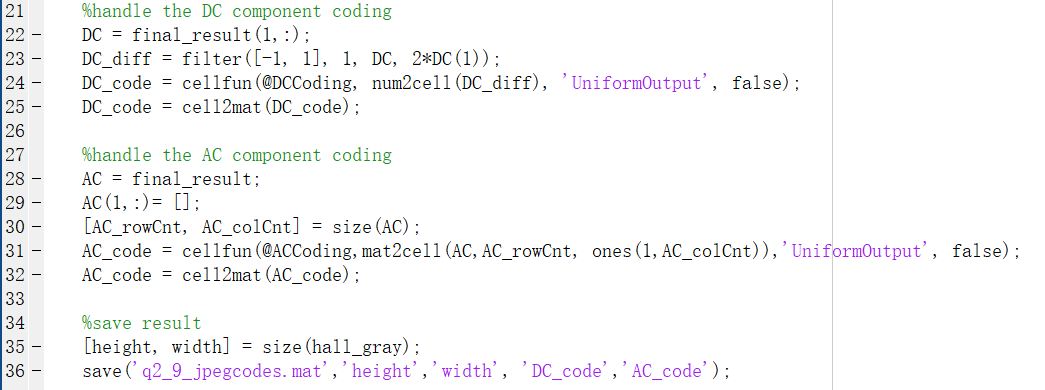




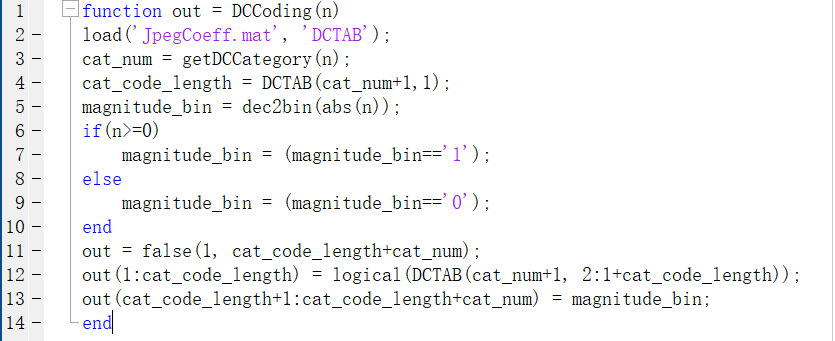
* 第17行：为了减少循环语句，我将图片灰度值矩阵以N×N(N=8)为单位，用mat2cell函数分割。
* 第18行：为了减少循环语句，我用cellfun对分割的各块进行DCT及Zigzag扫描，得到一个列向量。
* 第19行：将各块矩阵处理结果依照先逐列后逐行读取重新整形成数组形式，并用cell2mat拼接回矩阵。
* **DCT8andZigzagScan.m**



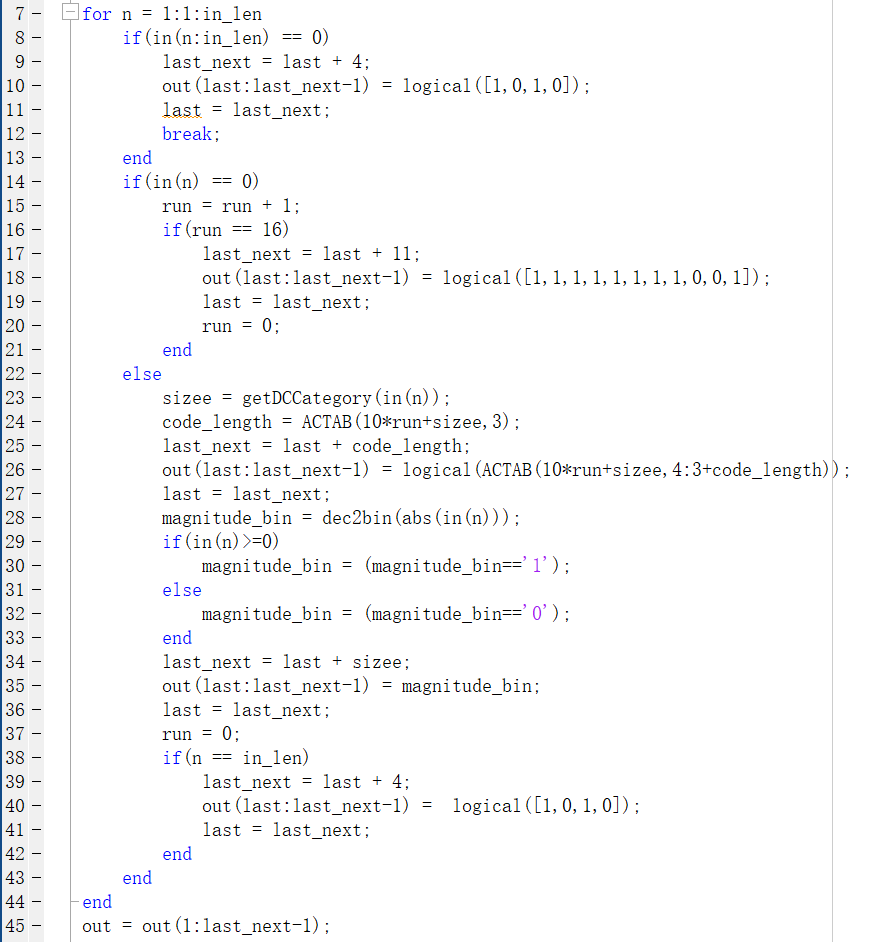
* 综合上述各题成果，第24行将输入做DCT，第26行依照量化表量化之，第27行做Zigzag扫描并输出之。
* **结果：**成功生成如题要求的矩阵final\_result。
  1. **关键代码：**
* **q2\_8\_9.m**
* 第22~23行：用滤波器进行差分编码。
* 第24~25行：为了减少循环语句，我用cellfun对直流分量各个差分编码结果进行熵编码。
* 第28~32行：为了减少循环语句，我用cellfun对交流分量进行熵编码。
* 第35~36行：依题意要求将结果存成q2\_9\_jpegcodes.mat。

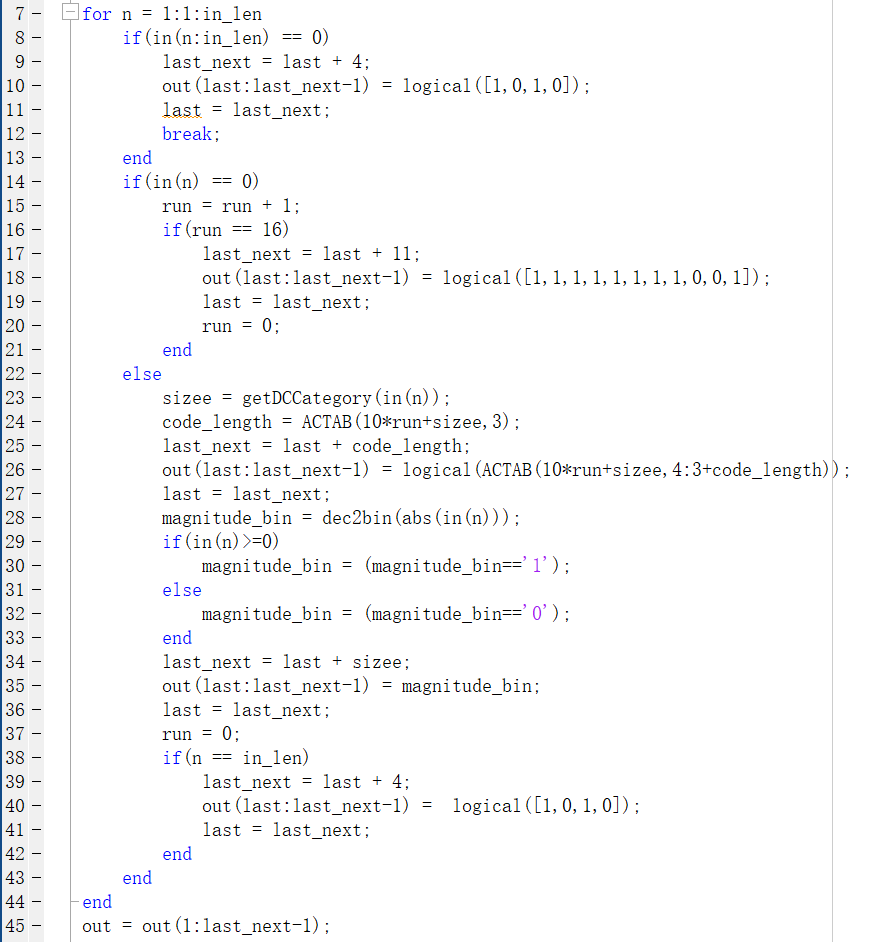


* **DCCoding.m**



* 第3~4行：计算该数对应之Category，并从DCTAB表中获取对应编码。
* 第5~10行：处理Magnitude部分，若输入的数为正数，直接取其二进制值；若为负数，则取其反码。
* 第11~13行：合并Category和Magnitude编码并输出。
* **ACCoding.m**
* 第8~13行：判断特殊状况（块结尾EOB）。
* 第14~21行：输入之数为0时，计数run递增1，如果run到达16，输出ZRL并将run归零。
* 第22~37行：其余情况，依据run、size、Magnitude查表、编码。
* 第38~42行：如果本块最后一个数非零，在编完该数后加入块结尾编码EOB。



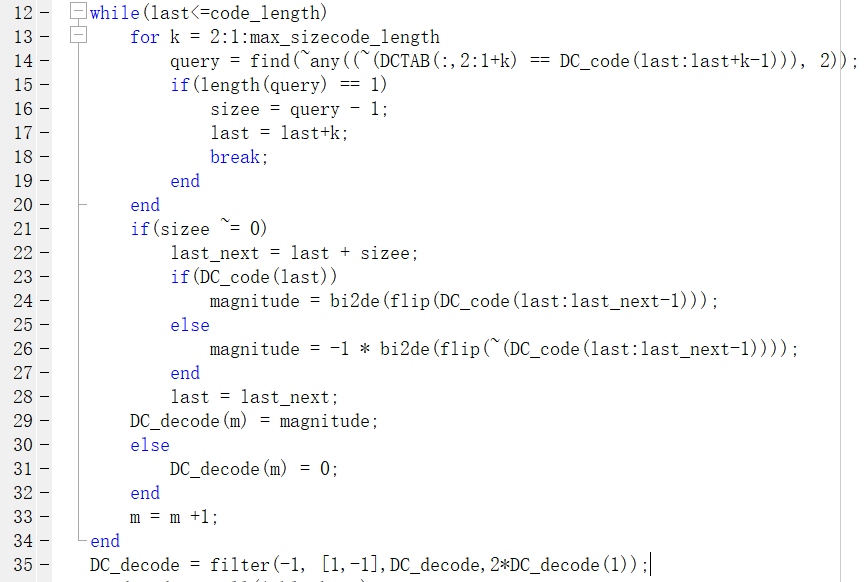


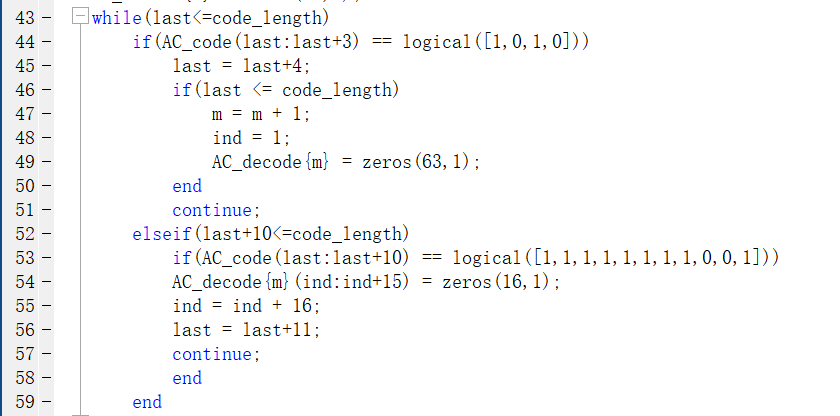
* **结果（q2\_9\_jpegcodes.mat）：成功。**
  1. 输入图像大小：bits

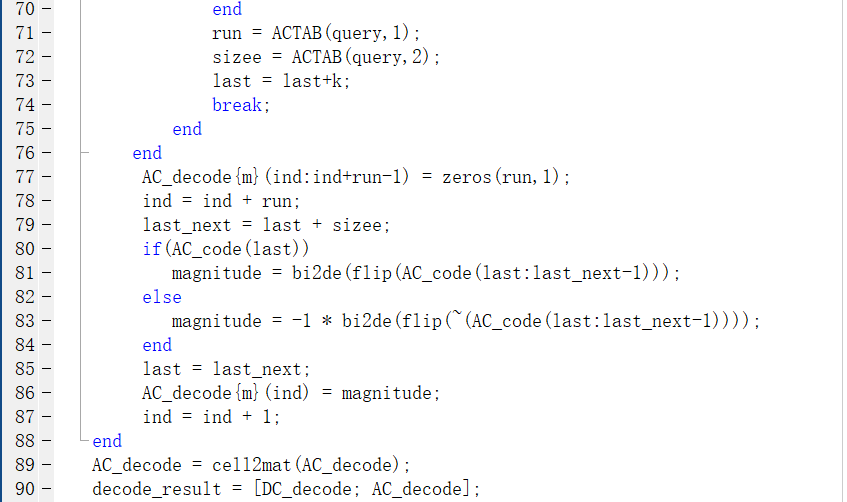
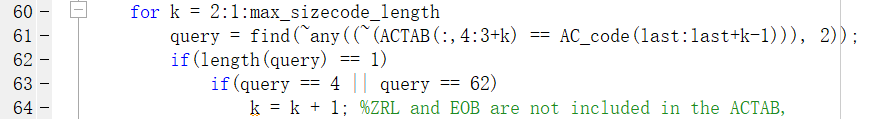
输出编码长度：bits

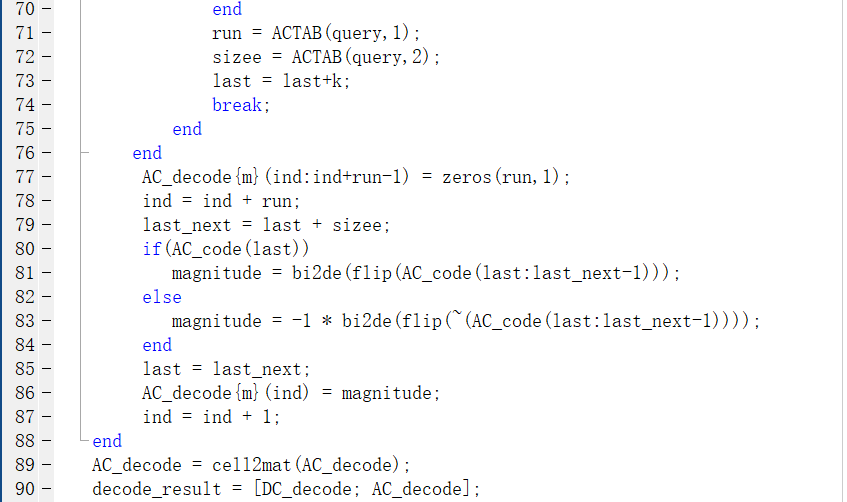
压缩比：

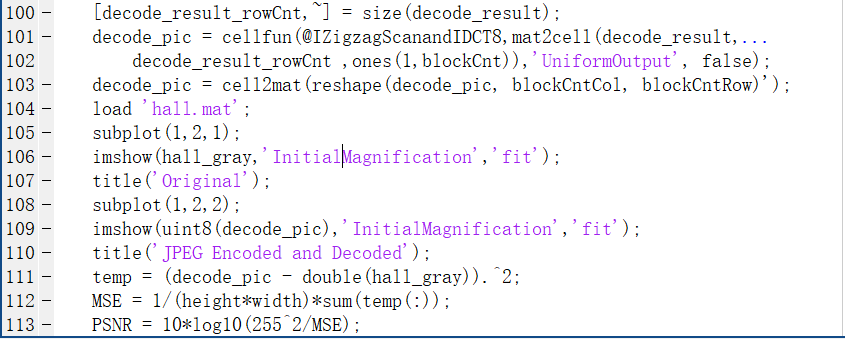
* 1. **关键代码：**
* **q2\_11.m**
* 第12~35行：DC解码。**往届有些同学在从Category码去找对应的Size的时候，竟然是在DCTAB中逐行比对，没有利用到Huffman前缀编码的特性。本人在此充分考虑Huffman前缀编码的性质，第13行至第20行仅在码长为2到最长可能码长之间做循环。**并巧用find、any函数去找到其所对应的Size大小。（先用any函数找到完全符合前缀编码的列，再用find去找到列号，当找到列号仅剩1个时，即找到该前缀编码对应的size。）第21~35行判别Magnitude之正负并据以解码。第35行利用filter从差分编码还原DC分量。
* 第42~89行：AC解码。第44~51行考虑特殊情形EOB；第52~59行考虑特殊情形ZRL。第60~76行考虑常规情形，仿上述方法，根据Huffman编码前缀性，找出Run/Size值并解码。**注意到本实验指导书虽写明ACTAB包含完整的码本，经查却不包含EOB、ZRL**，导致Huffman编码的前缀性受到破坏。例如编码1011对应Run/Size为0/4，但因1010不含于表中，导致搜索至101时即可判定Run/Size，使程序误以为Run/Size编码已读毕，从而自原本应包含于Run/Size编码中的1开始读取为Magnitude，导致之后解码完全错位，而产生错误。对于此种情形，应手动校正Magnitude的开始位，如第64行所示。第77~86行处理补零和Magnitude的解码。
* 第89~90行将解码结果拼接回如本章题8.要求之矩阵。
* 第101行对其做逆Zigzag扫描和IDCT，还原灰度值。
* 第104~113行：画原图与压缩图像做比较，计算MSE和PSNR。



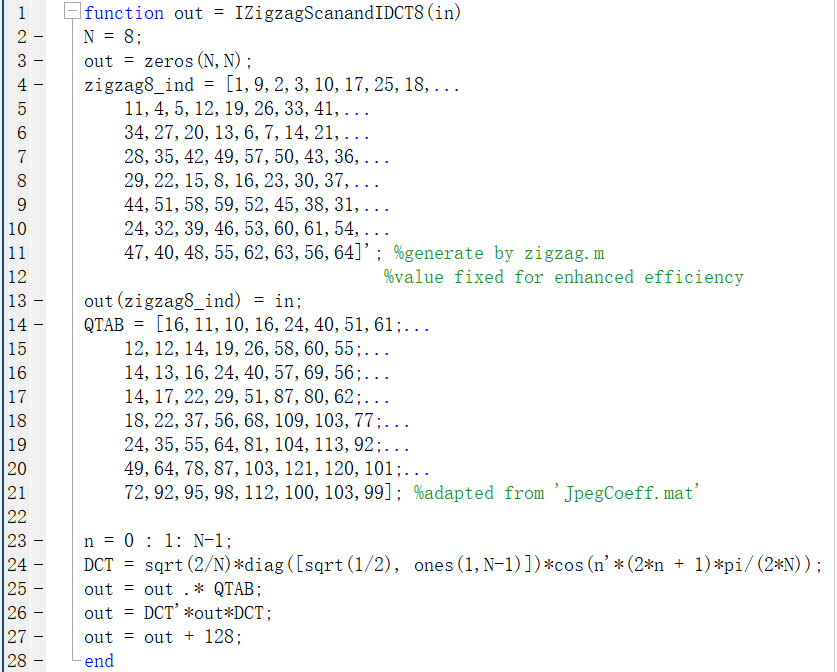






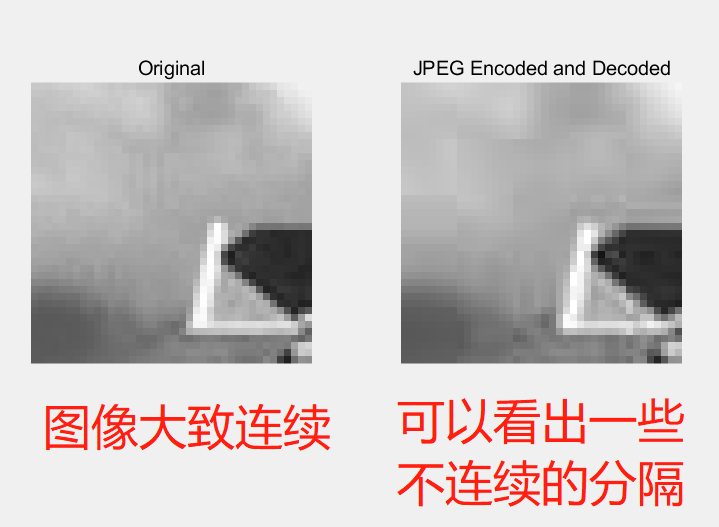


* **IZigzagScanandDCT8.m**



* 第13行：作逆Zigzag扫描
* 第25~27行：完成反量化、逆变换、加回直流分量。
* **结果：**如下图所示，压缩图像与原图乍看之下并无二致，但细看可发现图中原本分割成8\*8的小块更加「棱角分明」了。这应是我们在量化过程中舍入导致的差异。学理上称为压缩失真产生的「方块现象」。

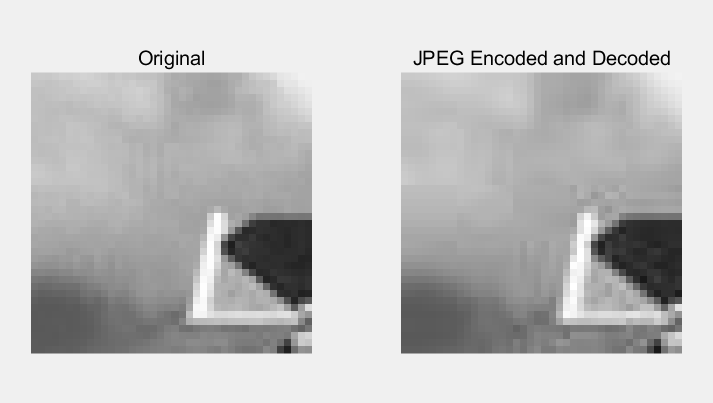




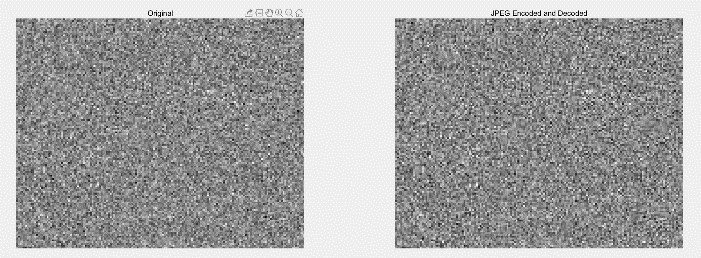
**经计算，MSE=49.6；PSNR=31.2。**

* 1. **关键代码：**
* **q2\_12\_encode.m**
* **q2\_12\_decode.m**
* 与前述代码基本上并无二致，仅调用以下改变量化步长之函数。
* **q2\_12\_DCT8andZigzagScan.m**
* **q2\_12\_IZigzagScanandIDCT8.m**
* 相较前述代码，于量化/反量化之前加入「QTAB = QTAB ./ 2;」。
* **结果：**图像与原本更加接近，至少「棱角分明」的情形稍有改善。压缩比。MSE=24.7；PSNR=34.2。较第11.题而言，PSNR略为上升，但压缩比却显着下降，仅为第11.题之68.7%。





* 1. **关键代码：**
* **q2\_13\_encode.m**
* **q2\_13\_decode.m**
* 与第8. 9. 11. 题所述代码基本上并无二致，仅将输入图像改为snow.mat。（话说这个图像是哪里美丽啊？）
* **结果：**雪／一片一片一片一片／在屏幕静静缤纷／眼看ＤＤＬ就要来了／而我也将／也将不再生存（<https://music.163.com/#/song?id=5257150>）

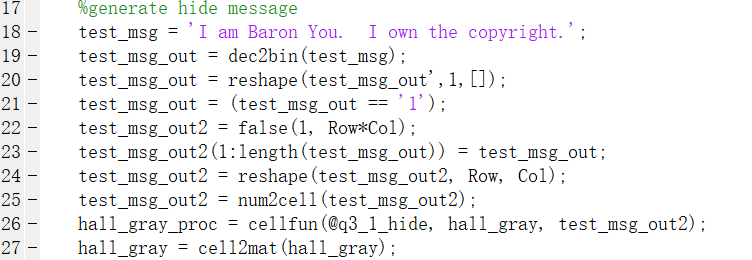
压缩比。MSE=331.6；PSNR=22.9。本例中，PSNR并没有前面的图高，但压缩比却也再度下降。观察可以发现本图的DC编码比之前的图短；AC编码却比之前的图来得长许多，可以知道本图高频分量多，迥异于一般图像，而量化矩阵是针对一般图像设计，高频分量的量化步长较大，导致耗费较长AC编码，却仍未能很好的保存图片的原貌。

1. **信息隐藏**

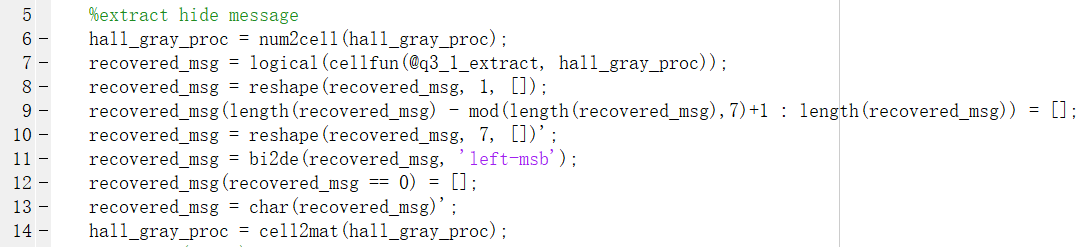
说明：考虑到信息隐藏具有保护版权的初衷，本章中本人隐藏的讯息皆为 “I am Baron You. I own the copyright.”

1. **关键代码：**

* q3\_1\_encode.m



* 利用dec2bin、reshape等函数将欲隐藏的讯息对应到图片的每一个位置上，再利用q3\_1\_hide函数逐一替换灰度值的最低位。
* q3\_1\_decode.m

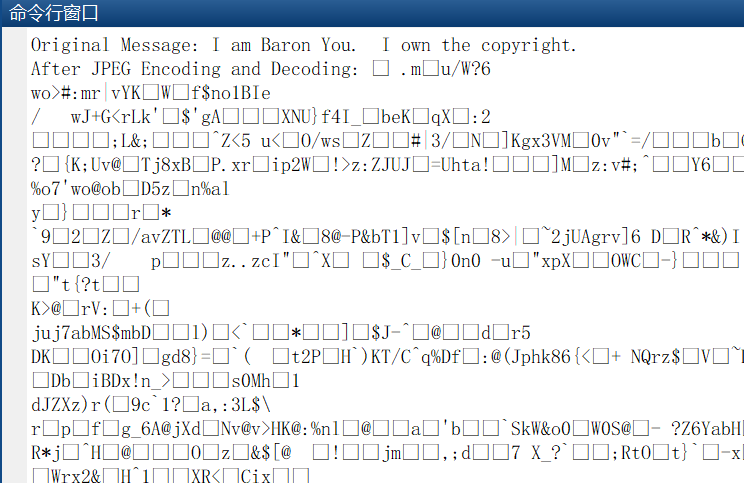


* 利用q3\_1\_extract函数提取出各灰度值的最低位，再用reshape整形回原本的数组形状、用bi2de函数、char转型，恢复原本信息。
* q3\_1\_hide.m（从略）：将欲隐藏的bit替换输入之灰度值的最后一位。
* q3\_1\_extract.m（从略）：将输入之灰度值的最后一位提取出来。
* **结果**：

1. 图像上：肉眼无法鉴别差异。



1. 讯息恢复：用未压缩图像传输，讯息可以完整恢复。图像一经压缩，即便解码（解压缩），讯息亦已遗失，不能恢复。



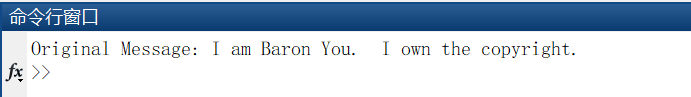
1. **【方法一】关键代码：**

* q3\_2\_1\_encode.m
* q3\_2\_1\_decode.m
* 与本章第1.题类似，只是将隐藏的时机放到DCT与量化之后，Zigzag扫描之前。从而原本的DCT8andZigzagScan.m和IZigzagScanandIDCT8.m要分开来写。
* q3\_2\_1\_hide.m
* q3\_2\_1\_extract.m
* 与本章第1.题类似，但要注意hide的时候，量化后的系数可能会出现负数，此时直接用de2bi或dec2bin函数会出问题，须先判断。
* q3\_2\_1\_DCT8.m
* q3\_2\_1\_IDCT8.m
* q3\_2\_1\_ZigzagScan.m
* q3\_2\_1\_IZigzagScan.m
* 将原本的DCT8andZigzagScan.m和IZigzagScanandIDCT8.m中的Zigzag扫描部分与变换及量化部分分离。
* **结果**：

1. 图像上：压缩比。MSE=121.6；PSNR=27.3。因为刻意藏了讯息，PSNR比第二章第9.题低。且肉眼可见第一列有不自然的横条纹，估计是在高频分量中藏了「1」所致。

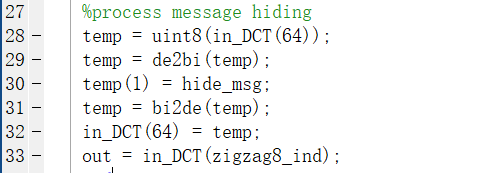


1. 讯息恢复：讯息可以完整恢复。



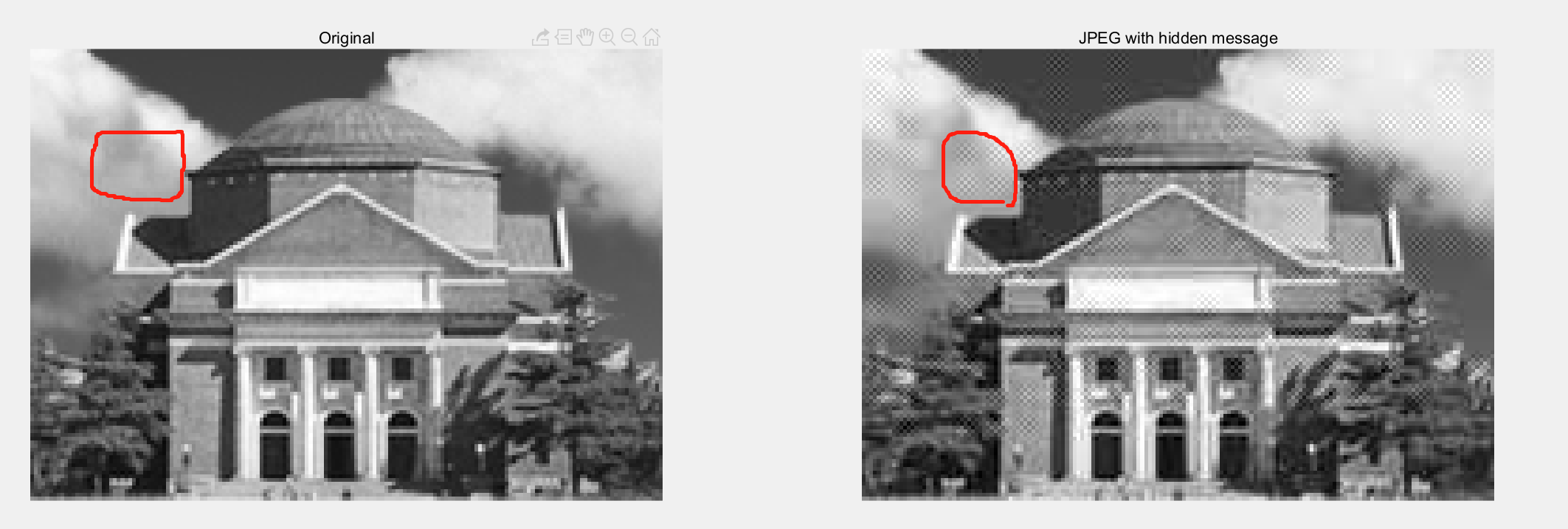
**【方法二】关键代码：**

* q3\_2\_2\_encode.m
* q3\_2\_2\_decode.m
* 与【方法一】类似，只是本次只将讯息隐藏在8\*8小块做完DCT的最右下方的一个元素的最后一位。故本人改写DCT8andZigzagScan.m使其同时肩负置入隐藏信息位的功能。
* q3\_2\_2\_extract.m
* 与【方法一】类似。
* q3\_2\_2\_DCT8andZigzagScan.m

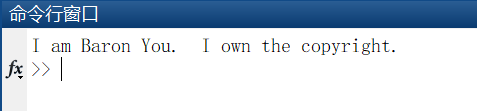


* 于输出前替换欲隐藏之信息位。
* **结果**：

1. 图像上：压缩比。压缩比较【方法一】下降，是因为当分块DCT、量化后，最后一个数的最后一位所藏的是「1」时，即便前面都是「0」，不能直接编入EOB，而必须编入多个ZRL。使编码增长。MSE=113.5；PSNR=27.6。因为刻意藏了讯息，PSNR比第二章第9.题低。且肉眼可见部分区域有不自然的网格状，此乃因在DCT最后一个分量藏入「1」所致。



1. 讯息恢复：讯息可以完整恢复。



**【方法三】关键代码：**

* q3\_2\_3\_encode.m
* q3\_2\_3\_decode.m
* 与【方法二】类似，只是本次藏入的位置在Zigzag扫描后，每列最后一个非零系数之后，或最后一位。
* q3\_2\_3\_findInd.m

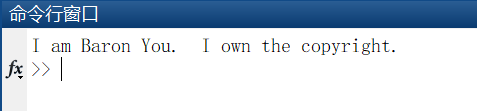


* 利用flip和find找到一列向量中最后一个非零元素的index。
* q3\_2\_3\_DCT8andZigzagScan.m
* 与【方法一】类似。
* **结果**：

1. 图像上：压缩比。MSE=84.4；PSNR=28.9。本方法是三个方法中压缩比最高、PSNR亦最高的方案。且生成的图像除了方块现象略为明显外，与一般压缩图像几无差别。



1. 讯息恢复：讯息可以完整恢复。

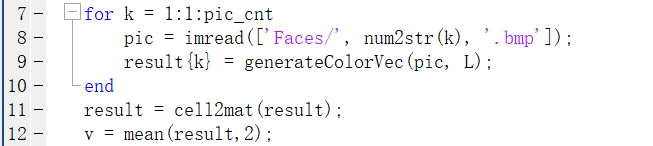


1. **人脸检测**
   1. (a)大可不必。因为我们的向量**v**中，各个元素的计算方式都是「落在该颜色范围内的像素点占总像素点的『比例』」，其为一比值，自不应受图片大小影响。

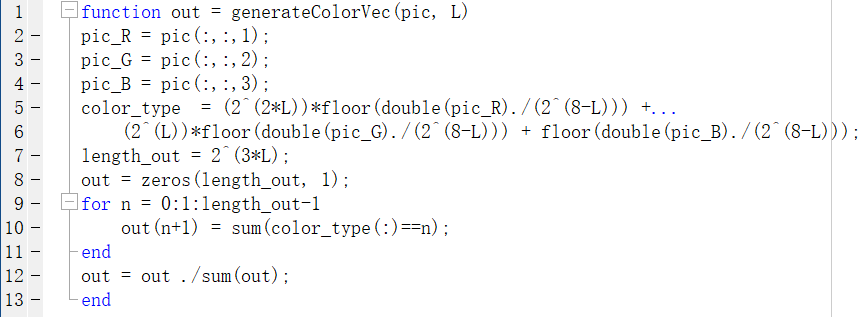
(b)L=4的向量将L=3时的颜色更细分8倍；L=5的向量将L=4时的颜色更细分8倍。因此其长度（维数）也不同。L=3时为512个元素，L=4时为L=3时的8倍，L=5时为L=4时的8倍。

关键代码如下：

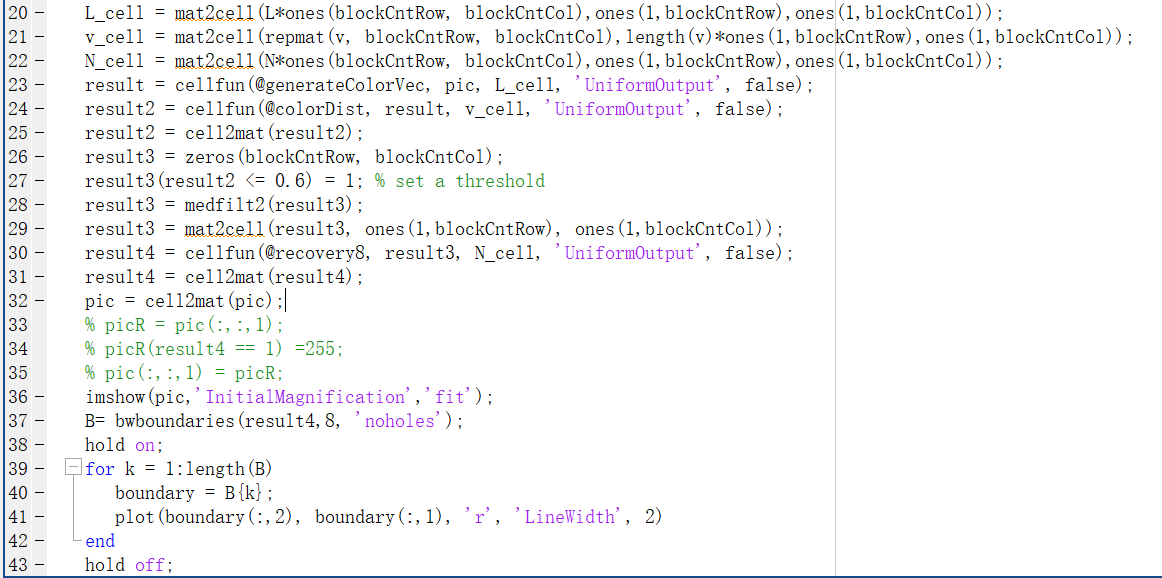
* q4\_1.m



* 对附录的33张人脸逐一调用generateColorVec生成颜色直方图向量。并将其取平均。
* generateColorVec.m



* 对图片根据L值制作颜色直方图列向量。第5~6行根据颜色RGB值计算在当前L值的规范下属于何「种」颜色。第9~11行，统计各「种」颜色出现的次数，计入输出向量的位置中。第12行对该向量「正规化」，即除以总像素点数。
* **结果：**储存于q4\_1\_v.mat中。
  1. L值愈大，对颜色鉴别的愈细致，但人脸在不同光照、不同环境下颜色本来就有略微差异，如果鉴别得太细致，反而无法成功辩认出人脸所在位置。经测试，在我提供的照片下，在L=3时能成功辨识人脸和部分颜色相近的人体部分，在L=4时有时可以辩识人脸，但会有遗漏的部分。L=5时更是因为循环太多而卡死。本人采用的方法是先将输入图片划分为N\*N的小块，每块调用第1.题中generateColorVec函数计算其冲色直方图列向量，并与第1.题中训练出之向量v相比，若「距离」大于一定阈值则将该块结果设为0，反之则设为1。最后用bwboundaries函数寻找0-1矩阵中各连通分支的边界并绘制于原图上。关键代码如下：
* q4\_2.m
* 第23~24行逐块生成直方图向量并比对距离；第27行检测该距离是否小于阈值；第28行作中值滤波减小杂讯；第30~31行将判定结果拼接回与原本同样大小的短阵。第37行用bwboundaries找各连通分支的边界；第39~42行将各边界画出。



* colorDist.m



* 依题意，利用Bhattacharyya度量方法给出两直方图向量间的「距离」。
* recovery8.m
* 将N\*N分块的判定结果回复为N\*N大小的矩阵。
* **结果：猜猜我在哪？**

|  |  |  |
| --- | --- | --- |
|  | **L=3** | **L=4** |
| **图1** | 25张显着人脸识别出来24张。但同时有些手也被当成了脸。 | 一张脸都没辨别出来。 |
| **图2** | 6张脸辨别出来6张。但同时有些手甚至地上的一部分也被当成了脸。 | 6张脸辨别出来4张。但同时地上的一部分也被当成了脸。 |
| **图3** | 32张脸辩别出来31张。但同时手、脖子等部位有时也被当成了脸。 | 32张脸辩别出来26张。但同时有些手也被当成了脸。 |

* 1. 本题所有部分以L=3，图1为例。所有代码（q4\_2.m）类似本章第2.题。故不再赘述。仅概述结果如下：

(a)辨识成功，如下图(a)。

(b)辨识成功，如下图(b)。

(c)辨识失败，如下图(c)。因本算法是以颜色分布为核心，未更动图片颜色的项目皆能辨识成功。一旦更动到图片颜色，辨识失败的机率大增。

* 1. 应选取各种不同光照下的人脸，若识别的主要对象为黄种人，也应多选取黄种人的人脸照片。



**▲图4.3.(a)**



**▲图4.3.(b)**



**▲图4.3.(c)**