信号与系统——MATLAB 综合实验之图像处理

实验报告

学号： 2014011216

班级： 无 47

姓名： 刘 前

日期： 2016-8-14

第一章 基础知识

在 MATLAB 中，像素值用 uint8 类型表示，参与浮点数运算前需要转成 double 型。本章练习题中“测试图像”指的是 hall.mat 中的彩色图像。

1. MATLAB 提供了图像处理工具箱，在命令窗口输入 help images 可查看该工具箱内的所有函数。请阅读并大致了解这些函数的基本功能。

分析：

help images 能够查看图像处理工具箱（Image Processing Toolbox）内的所有函数。函数主要包括以下大类：

函数类别	基本功能
Image display and exploration	图像显示和浏览
Image file I/O	图像文件的输入/输出
Image arithmetic	图像数学算法
Geometric transformations	几何变换
Image registration	图像配准
Pixel values and statistics	像素值与统计
Image analysis	图像分析
Image enhancement	图像增强
Linear filtering	线性滤波器
Linear 2-D filter design	线性二维滤波器设计
Image deblurring	图像去模糊
Image segmentation	图像分割
Image transforms	图像变换
Neighborhood and block processing	邻域和块处理
Morphological operations	形状操作
Structuring element (STREL) creation and manipulation	结构元素创建于操作
Texture analysis	纹理分析
Region-based processing	基于区域的处理
Colormap manipulation	色彩表操作
Color space conversions	色彩空间转换
ICC color profiles	ICC 色彩配置
Array operations	数组操作
Image types and type conversions	图像类型和类型转换
Toolbox preferences	工具箱偏好设置
Toolbox utility functions	工具箱实用功能
Modular interactive tools	模块化交互式工具
Navigational tools for image scroll panel	图像滚动面板导航工具
Utility functions for interactive tools	交互式工具实用功能
Interactive mouse utility functions	互动鼠标实用功能

以上即为图像处理工具箱中函数的分类，每一类中包含了若干图像处理的具体函数。

2. 利用 MATLAB 提供的 Image file I/O 函数分别完成以下处理：

(a) 以测试图像的中心为圆心，图像的长和宽中较小值的一半为半径画一个红颜色的圆；

(b) 将测试图像涂成国际象棋状的“黑白格”的样子，其中“黑”即黑色，“白”即意味着保留原图。用一种看图软件浏览上述两个图，看是否达到了目标。

(a)

分析：

首先从 hall.mat 中获取彩色图像的矩阵数据。根据要求获取图像的长和宽，从而得到圆的半径。注意作出红颜色的圆时，圆内的像素点的(R, G, B)=(255, 0, 0)。

代码：

chapter1_2a.m

```
clear all,close all,clc;

load hall.mat;        %从 hall.mat 中获取数据(hall_color)
hallc=hall_color;     %测试图像为 hall.mat 中的彩色图像 hall_color
                      %另存为 hallc,避免更改原始数据 hall_color
imshow(hallc);        %展示原图
imwrite(hallc,'./chapter1_2/0hall_origin.bmp');
                      %另存为 hall_origin.bmp，看图软件浏览

[len,width,n]=size(hallc);              %获取图像的长宽
radius=min(len,width)/2;                %长和宽中较小值为半径
center_x=len/2;                       %中间点的横坐标
center_y=width/2;                       %中间点的纵坐标
for i=1:len
    for j=1:width
        dis_sq=(i-center_x)^2+(j-center_y)^2;  %计算(i,j)到中心距离的平方
        if(dis_sq<=radius^2)                   %判断点(i, j)属于圆内
            hallc(i,j,1)=255;  %将圆内的每一像素点置为红色
            hallc(i,j,2)=0;    %(R,G,B)=(255,0,0)
            hallc(i,j,3)=0;
        end
    end
end

figure;
imshow(hallc);                          %展示修改后图像
```

```
imwrite(hallc,'./chapter1_2/1hall_circle.bmp');
                                        %另存为 hall_circle.bmp,看图软件浏览
```

MATLAB 运行结果：

图 1.2.a.1

图 1.2.a.2

如上图所示，图 1.2.a.1 是 hall.mat 中 hall_color 的图像，也就是测试图像；图 1.2.a.2 是经过处理后的图像。根据要求，选取长和宽中较短的一半作为半径，画出红颜色的圆。用看图软件浏览两图，达到了实验目标。

(b)

分析：

国际象棋状的“黑白格”如图图 1.2.b.1 所示：

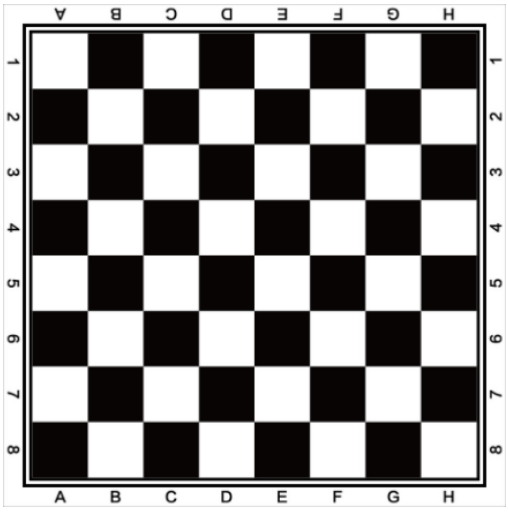


图 1.2.b.1

由于测试图像是矩形图像，因而最终实现得到的“黑白格”每一小格也是矩形。如上图所示，黑色部分的每一像素(R, G, B)=(0, 0, 0)，白色部分保留原图，可以不做处理。

代码：

chapter1_2b.m

```
clear all,close all,clc;
```

```
load hall.mat;        %从 hall.mat 中获取数据(hall_color)
hallc=hall_color;     %测试图像为 hall.mat 中的彩色图像 hall_color
                      %另存为 hallc,避免更改原始数据 hall_color
[len,width,n]=size(hallc);    %获取图像的长宽
slen=len/8;                   %国际象棋棋盘为 8*8
                              %slen 表示每一小格的长度
swidth=width/8;               %swidth 表示每一小格的宽度
for i=1:8
    for j=1:8
        if(mod((i+j),2)==1)     %8*8 的棋盘，发现棋盘中某一格(i,j)
                                %满足 i+j 为奇数则是黑色
                               %i+j 为偶数则为白色，保持原图，故不作处理
                                %黑色(R,G,B)=(0,0,0)
            hallc((i-1)*slen+1:i*slen,(j-1)*swidth+1:j*swidth,1)=0;
            hallc((i-1)*slen+1:i*slen,(j-1)*swidth+1:j*swidth,2)=0;
            hallc((i-1)*slen+1:i*slen,(j-1)*swidth+1:j*swidth,3)=0;
        end
    end
end
imshow(hallc);                %展示修改后图片
imwrite(hallc,'./chapter1_2/3hall_chess.jpg');
                              %另存为 hall_chess.jpg,看图软件浏览
```

MATLAB 运行结果：

图 1.2.b.2

图 1.2.b.3

如上图所示，图 1.2.b.2 是 hall.mat 中 hall_color 的图像，也就是测试图像；图 1.2.b.3 是经过处理后的图像。根据要求，将测试图像分成 8*8 的块，按照图 1.2.b.1 国际象棋的棋盘进行处理。用看图软件浏览两图，达到了实验目标。

第二章 图像压缩编码

本章练习题所用数据均可由“JpegCoeff.mat”导入，其内容如下表所示。本章练习题中“测试图像”指的是 hall.mat 中的灰度图像。

变量名	含义	说明
QTAB	DCT 系数的量化步长矩阵	
DCTAB	DC 系数预测误差的 Category 码本	每行对应一个 Category，第一列对应 Huffman 编码的长度 L，随后 L 列对应该码字，再后全零为填充
ACTAB	AC 系数的(Run/Size)的码本	每行对应一个(Run/Size)，第一列表示 Run，第二列表示 Size，第三列表示(Run/Size)Huffman 编码的长度 L，随后 L 列对应对应码字，再后全零为填充

表 2.1　JpegCoeff.mat 所含数据

1. 图像的预处理是将每个像素灰度值减去 128，这个步骤是否可以在变换域进行？请在测试图像中截取一块验证你的结论。

分析：

在测试图像中选取 40*40 的一小块，进行图像预处理：

方法一：将图像灰度值矩阵的每个元素(即每个像素点的灰度值)减去 128。

方法二：对 40*40 的图像进行离散余弦变换，再把 128*ones(40,40)的矩阵进行离散余弦变换，两者相减后进行离散余弦逆变换。

代码：

chapter2_1.m

```
clear all,close all,clc;

load hall.mat;          %从 hall.mat 中获取数据(hall_gray)
hallg=hall_gray;        %测试图像为 hall.mat 中的灰度图像 hall_gray
                        %另存为 hallg,避免更改原始数据 hall_gray
hallp=double(hallg(21:60,31:70)); %从灰度图像选取 40*40 块，注意类型变化
hallp_sub=hallp-128*ones(40,40);%将每个像素灰度值减去 128
subplot(1,2,1);                 %子图 1
imshow(hallp_sub);              %观察第一种方法处理后的图像
imwrite(hallp_sub,'./chapter2_1/1 减去 128.bmp');
                                %另存为“1 减去 128.bmp”,看图软件浏览
```

```
%尝试在变换域中进行
DCTa=dct2(hallp);                  %将 hallp 进行离散余弦变换
DCTb=dct2(128*ones(40,40));        %离散余弦变换
hallp_trans=idct2(DCTa-DCTb);      %进行离散余弦逆变换，得到处理结果
subplot(1,2,2);                   %子图 2
imshow(hallp_trans);               %观察变换域处理后的图像
imwrite(hallp_trans,'./chapter2_1/2 变换域.bmp');
                                   %另存为“2 变换域.bmp”,看图软件浏览

diffmax=max(max(abs(hallp_sub-hallp_trans)))
                                   %两种处理方法所得图像矩阵之差的最大值
Cor=corr2(hallp_sub,hallp_trans) %两种处理方法所得图像矩阵的相关系数
```

MATLAB 运行结果：

图 2.1.1

图 2.1.2

如以上两图所示，图 2.1.1 是直接减去 128 进行预处理所得到的图像，图 2.1.2 是在变换域内进行预处理所得的图像。主观来看，两种方法所得到的结果几乎没有差别，为了得到更加客观的比较结果，通过以下两种方法进行对比：

方法一：

对两种方法所得图像矩阵作差，并取绝对值，获得该矩阵的最大元素 diffmax。最大元素 diffmax 越小，说明两矩阵的差别越小。

方法二：

运用 MATLAB 自带的 corr2 函数比较两矩阵，结果 Cor 可以用于评价两矩阵的相似程度。函数结果 Cor 越接近 1，表明两矩阵相似程度越高；越接近 0，两矩阵越不相似。

以下是两种方法各自的结果：

```
diffmax =                    Cor =

   8.5265e-14                   1.0000
```

由此可得，diffmax 的数量级为10^{-14}，而且 corr2 的结果为 1.0000，两个结果都说明“直接减去 128”和“变换域”两种方法的效果相同。

2. 请编程实现二维 DCT，并和 MATLAB 自带的库函数 dct2 比较是否一致。

分析：

二维 DCT(离散余弦)变换的计算公式：

$C = DPD^T$

其中，D 为 DCT 算子，具体形式如下：

$$\sqrt{\frac{2}{N}}\begin{bmatrix} \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \cdots & \sqrt{\frac{1}{2}} \\ \cos\frac{\pi}{2N} & \cos\frac{3\pi}{2N} & \cdots & \cos\frac{(2N-1)\pi}{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \cos\frac{(N-1)\pi}{2N} & \cos\frac{(N-1)3\pi}{2N} & \cdots & \cos\frac{(N-1)(2N-1)\pi}{2N} \end{bmatrix}$$

P 是待处理的图像矩阵，$P(x,y)$ 表示 (x,y) 位置的亮度值。

根据以上公式，即可编程实现二维离散余弦变换的函数。

代码：

mydct2.m

```
function B= mydct2(A)
% 二维离散余弦变换函数
% 输入：二维矩阵 A
% 返回值：A 的二维离散余弦变换 B
% B = DCTtwo(A) 返回矩阵 A 的二维离散余弦变换，矩阵 B 和矩阵 A 大小相同

    [M,N]=size(A);          %获取矩阵 A 的大小
    B=zeros(M,N);           %初始化返回矩阵 B
    D=zeros(M,N);           %初始化 DCT 算子矩阵
    for i=1:M
        for j=1:N
            %DCT 算子的元素计算公式
         D(i,j)=sqrt(2/M)*cos((i-1)*(2*j-1)*pi/(2*M));
        end
    end
    D(1,:)=D(1,:)/sqrt(2);      %系数例外情况
    B=D*A*D';                   %直接根据矩阵运算
end
```

对 mydct2 函数进行测试：

chapter2_2.m

```
clear all,close all,clc;

load hall.mat;              %从 hall.mat 中获取数据(hall_gray)
hallg=hall_gray;            %测试图像为 hall.mat 中的灰度图像 hall_gray
                            %另存为 hallg,避免更改原始数据 hall_gray
hallp=double(hallg(21:60,31:70));    %从灰度图像选取 40*40，注意类型变化
DCT1=dct2(hallp);                    %自写二维 DCT 函数 mydct2
DCT2=mydct2(hallp);                  %MATLAB 自带 dct2

diffmax=max(max(abs(DCT1-DCT2)))
                            %自写函数 mydct2 与 MATLAB 自带 dct2 对比
Corr=corr2(dct2(hallp),mydct2(hallp))
                            %两个矩阵所得结果的相关系数
```

MATLAB 运行结果：

将自写的 mydct2 与 MATLAB 自带 dct2 对比。首先进行功能对比，为了得到更加客观的比较结果，通过以下两种简易方法进行对比：

方法一：

对两种方法所得矩阵结果作差，并取绝对值，获得该矩阵的最大元素 diffmax。最大元素 diffmax 越小，说明两矩阵的差别越小。

方法二：

运用 MATLAB 自带的 corr2 函数比较两矩阵，结果 Cor 可以用于评价两矩阵的相似程度。函数结果 Cor 越接近 1，表明两矩阵相似程度越高，说明两函数的功能越相近，

以下是两种函数各自的运行结果：

```
diffmax =

   7.3985e-12

Corr =

     1
```

由此可得，diffmax 的数量级为10^{-12}，而且 corr2 的结果为 1，两个结果都说明自写的 mydct2 函数与自带的 dct2 函数功能相一致。

再比较两函数的实现方式：

自写的 mydct2 函数完全根据二维 DCT 的公式加以实现；

MATLAB 自带的 dct2 函数则调用了 dct 函数，即一维离散余弦变换函数。因而两函数的实现方式存在较大差异。

3. 如果将 DCT 系数矩阵中右侧四列的系数全部置零，逆变换后的图像会发生什么变化？选取一块图验证你的结论。如果左侧的四列置零呢？

分析：

根据要求，先选取一块图，得到 DCT 系数矩阵，再将 DCT 系数矩阵中右侧四列的系数全部置零，逆变换后得到结果。同样地，将 DCT 系数矩阵中左侧四列的系数全部置零，逆变换得到相应结果。

代码：

chapter2_3a.m　　chapter2_3b.m　　chapter2_3and4.m

```
clear all,close all,clc;
%DCT 系数矩阵右侧 4 列全部置零(选取 35*35 一小块)
load hall.mat;              %从 hall.mat 中获取数据(hall_gray)
hallg=hall_gray;            %测试图像为 hall.mat 中的灰度图像 hall_gray
                            %另存为 hallg,避免更改原始数据 hall_gray
hallp=hallg(26:60,66:100);      %选取原图的一部分
subplot(1,2,1);                 %子图 1
imshow(hallp);                  %观察 DCT 系数矩阵不变时处理后的图像
imwrite(hallp,'./chapter2_3and4_35×35/1 原图_35×35.bmp');

DCT=dct2(hallp);            %将 hallp 进行离散余弦变换
DCT(:,37:40)=0;             %将 DCT 系数矩阵右侧 4 列全部置零
iDCT=idct2(DCT);            %作离散余弦逆变换
subplot(1,2,2);             %子图 2
imshow(uint8(iDCT));        %观察 DCT 系数矩阵改变后处理后的图像
    %注意像素值用 uint8 类型表示，参与浮点运算要转换为 unit8 类型
imwrite(uint8(iDCT),'./chapter2_3and4_35×35/2 右侧 4 列置零_35×35.bmp');

clear all,close all,clc;
%DCT 系数矩阵左侧 4 列全部置零(选取 35*35 一小块)
load hall.mat;              %从 hall.mat 中获取数据(hall_gray)
hallg=hall_gray;            %测试图像为 hall.mat 中的灰度图像 hall_gray
                            %另存为 hallg,避免更改原始数据 hall_gray
hallp=hallg(26:60,66:100);      %选取原图的一部分
subplot(1,2,1);                 %子图 1
imshow(hallp);              %观察 DCT 系数矩阵不变时处理后的图像

DCT=dct2(hallp);            %将 hallg 进行离散余弦变换
DCT(:,1:4)=0;               %将 DCT 系数矩阵左侧 4 列全部置零
iDCT=idct2(DCT);            %作离散余弦逆变换
```

```
subplot(1,2,2);                %子图 2
imshow(uint8(iDCT));           %观察 DCT 系数矩阵改变后处理后的图像
    %注意像素值用 uint8 类型表示，参与浮点运算要转换为 unit8 类型
imwrite(uint8(iDCT),'./chapter2_3and4_35×35/3 左侧 4 列置零_35×35.bmp');
```

MATLAB 运行结果：

图 2.3.1

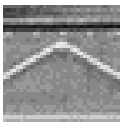


图 2.3.2

图 2.3.3

如以上三个图像所示，选取测试图像中的一小块，图 2.3.1 是原图，图 2.3.2 是将 DCT 系数矩阵中右侧四列的系数全部置零后逆变换所得到的图像，图 2.3.3 是将 DCT 系数矩阵中左侧四列的系数全部置零后逆变换所得的图像。

通过直观地比较，可以发现，将 DCT 系数矩阵中右侧四列的系数全部置零所得图像与原图差别不大，只有一点点失真；但是，将 DCT 系数矩阵中右侧四列的系数全部置零所得图像明显变暗，整体色调变为灰黑色，但仍能基本看清原图的轮廓。分析可知：DCT 系数矩阵靠左侧的列表示图像的 DC 直流分量，靠右侧的列表示图像的高频分量。直流分量置零对图像整体的影响很大，发生较严重的失真；高频分量置零则只会稍微失真。

4. 若对 DCT 系数分别做转置、旋转 90 度和旋转 180 度操作(rot90)，逆变换后恢复的图像有何变化？选取一块图验证你的结论。

分析：

按照要求对 DCT 系数矩阵进行处理，逆变换后后恢复出图像。

代码：

chapter2_4a.m

```
clear all,close all,clc;
%DCT 系数矩阵进行转置操作(选取 35*35 一小块)
load hall.mat;              %从 hall.mat 中获取数据(hall_gray)
hallg=hall_gray;            %测试图像为 hall.mat 中的灰度图像 hall_gray
                            %另存为 hallg,避免更改原始数据 hall_gray
hallp=hallg(26:60,66:100);          %选取原图的一部分
subplot(1,2,1);                     %子图 1
imshow(hallp);              %观察 DCT 系数矩阵不变时处理后的图像

DCT=dct2(hallp);            %将 hallp 进行离散余弦变换
DCT=DCT';                   %将 DCT 系数矩阵进行转置操作
```

```
iDCT=idct2(DCT);          %作离散余弦逆变换
subplot(1,2,2);           %子图 2
imshow(uint8(iDCT));      %观察 DCT 系数矩阵改变后处理后的图像
        %注意像素值用 uint8 类型表示，参与浮点运算要转换为 unit8 类型
imwrite(uint8(iDCT),'./chapter2_3and4_35×35/4 转置_35×35.bmp');
```

---

chapter2_4b.m

```
clear all,close all,clc;
%DCT 系数矩阵进行旋转 90 度操作(选取 35*35 一小块)
load hall.mat;          %从 hall.mat 中获取数据(hall_gray)
hallg=hall_gray;        %测试图像为 hall.mat 中的灰度图像 hall_gray
                        %另存为 hallg,避免更改原始数据 hall_gray
hallp=hallg(26:60,66:100);          %选取原图的一部分
subplot(1,2,1);                     %子图 1
imshow(hallp);          %观察 DCT 系数矩阵不变时处理后的图像
DCT=dct2(hallp);        %将 hallp 进行离散余弦变换
DCT=rot90(DCT);         %将 DCT 系数矩阵旋转 90 度
iDCT=idct2(DCT);        %作离散余弦逆变换
subplot(1,2,2);         %子图 2
imshow(uint8(iDCT));      %观察 DCT 系数矩阵改变后处理后的图像
        %注意像素值用 uint8 类型表示，参与浮点运算要转换为 unit8 类型
imwrite(uint8(iDCT),'./chapter2_3and4_35×35/5 逆时针旋转 90 度_35×35.bmp');
```

---

chapter2_4c.m

```
clear all,close all,clc;
%DCT 系数矩阵进行旋转 180 度操作(选取 35*35 一小块)
load hall.mat;          %从 hall.mat 中获取数据(hall_gray)
hallg=hall_gray;        %测试图像为 hall.mat 中的灰度图像 hall_gray
                        %另存为 hallg,避免更改原始数据 hall_gray
hallp=hallg(26:60,66:100);      %选取原图的一部分
subplot(1,2,1);                     %子图 1
imshow(hallp);                  %观察 DCT 系数矩阵不变时处理后的图像
DCT=dct2(hallp);        %将 hallp 进行离散余弦变换
DCT=rot90(DCT);         %将 DCT 系数矩阵旋转 90 度
DCT=rot90(DCT);         %将 DCT 系数矩阵再旋转 90 度（共旋转 180 度）
iDCT=idct2(DCT);        %作离散余弦逆变换
subplot(1,2,2);         %子图 2
imshow(uint8(iDCT));      %观察 DCT 系数矩阵改变后处理后的图像
    %注意像素值用 uint8 类型表示，参与浮点运算要转换为 unit8 类型
imwrite(uint8(iDCT),'./chapter2_3and4_35×35/6 旋转 180 度_35×35.bmp');
```

MATLAB 运行结果：

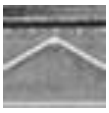
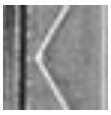
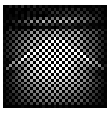
 　  　  　 

图 2.4.1　　图 2.4.2　　图 2.4.3　　图 2.4.4

如以上四个图像所示，选取测试图像中的一小块，图 2.4.1 是原图，图 2.4.2 是将 DCT 系数矩阵转置后逆变换所得到的图像，图 2.4.3 是将 DCT 系数矩阵逆时针旋转 90 度后逆变换所得的图像，图 2.4.4 是将 DCT 系数矩阵旋转 180 度后逆变换得到的图像。

通过直观地比较，可以发现，DCT 系数矩阵转置时，图像逆时针旋转 90 度；DCT 系数矩阵逆时针旋转 90 度时，图像逆时针旋转 90 度，且有很明显的失真(黑色条纹状)；DCT 系数矩阵旋转 180 度时，图像有很明显的失真(点状)。

为了更直观地看出以上变换对整个图像的影响，将选取的图像改为完整的测试图像，以下是变换后的图像效果。

图 2.4.5　　图 2.4.6　　图 2.4.7

图 2.4.8　　图 2.4.9　　图 2.5.10

图 2.4.5	原测试图像
图 2.4.6	DCT 系数矩阵右侧四列置零
图 2.4.7	DCT 系数矩阵左侧四列置零
图 2.4.8	DCT 系数矩阵转置
图 2.4.9	DCT 系数矩阵逆时针旋转 90 度
图 2.5.10	DCT 系数矩阵旋转 180 度

图像效果与选取一小块的效果相同。

5. 如果认为差分编码是一个系统，请绘出这个系统的频率响应，说明它是一个_____（低通、高通、带通、带阻）滤波器。DC 系数先进行差分编码再进行熵编码，说明 DC 系数的_____ 频率分量更多。

分析：

差分编码的运算公式如下：

$$\hat{c}_D(n)=\begin{cases}\tilde{c}_D(n) & n=1;\\ \tilde{c}_D(n-1)-\tilde{c}_D(n) & \text{elsewhere}\end{cases}$$

把差分编码当作一个系统，则表征系统的差分方程为：

$$y(n)=x(n)-x(n-1)$$

由 freqz 函数即可得到该系统的频率响应。

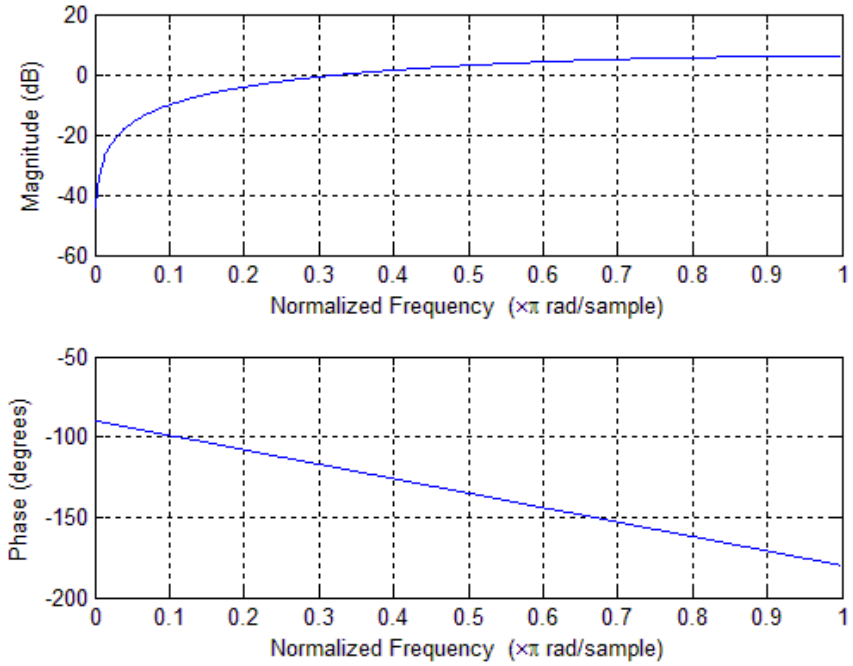
代码：

chapter2_5.m

```
clear all,close all,clc;

a=1;          %定义输出信号系数
b=[-1,1];     %定义输入信号系数
freqz(b,a)    %系统频率响应
```

MATLAB 运行结果：



根据所得频率响应的幅频响应特性曲线，可以判断该系统是一个高通系统。DC 系数先进行差分编码再进行熵编码，说明 DC 系数低频的频率分量更多，因而需要先经过一个高通系统进行滤波。

6. DC 预测误差的取值和 Category 值有何关系？如何利用预测误差计算出其 Category？

分析：

DC 预测误差的取值和 Category 值的对应关系：

预测误差	Category
0	0
-1,1	1
-3, -2, 2, 3	2
-7, ···, -4, 4, ···, 7	3
-15, ···, -8, 8, ···, 15	4
-31, ···, -16, 16, ···, 31	5
-63, ···, -32, 32, ···, 63	6
-127 ,···, -64, 64, ···, 127	7
-255, ···, -128, 128, ···, 255	8
-511, ···, -256, 256, ···, 511	9
-1023, ···, -512, 512, ···, 1023	10
-2047, ···, -1024, 1024, ···, 2047	11

设预测误差为 E_p，则根据 E_p 计算 Category 的计算公式为：

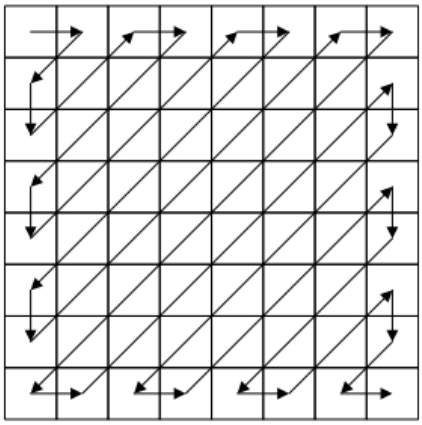
$$Category = ceil(\log_2 |E_p| + 1)$$

其中 ceil 函数的功能是：返回大于或者等于指定表达式的最小整数。

7. 你知道哪些实现 Zig-Zag 扫描的方法？请利用 MATLAB 的强大功能设计一种最佳方法。

分析：

Zig-Zag 扫描将二维的 DCT 系数矩阵变为一维矢量。分成 8*8 的块后，各块 DCT 系数矩阵大小也是 8*8。



Zig-Zag 的扫描方式如上图所示。据此图可以得到两种实现 Zig-Zag 扫描的方法。

方法一：

思路：

将 Zig-Zag 扫描的顺序预先存储在一个向量中，根据向量元素的顺序访问原矩阵的元素，从而实现 Zig-Zag 扫描。值得注意的是，矩阵元素角标的排列如下图所示：

1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63
8	16	24	32	40	48	56	64

代码：

zigzag1.m

```
--------------------------------------------------------------------------------------------------------------
function B=zigzag1(A)
% 函数 zigzag1，对 8*8 矩阵进行 Zig-Zag 扫描
% 输入变量： 8*8 矩阵 A
% 返回值：　　对 A 进行 zigzag 扫描得到的 64*1 向量

  [a,b]=size(A);　　%获取输入矩阵大小
  if(a==8&&b==8)　%满足矩阵为 8*8 的要求%设定扫描的顺序

    zigzag_index= [ 1,9,2,3,10,17,25,18,11,4,5,12,19,26,33,41,34,27,20,13,6,7,14,...

    21,28,35,42,49,57,50,43,36,29,22,15,8,16,23,30,37,44,51,58,59,...

    52,45,38,31,24,32,39,46,53,60,61,54,47,40,48,55,62,63,56,64];
  else error('函数的输入不符合要求!要求输入为 8*8 矩阵');
                    %输入不符合要求
  end
  B=A(zigzag_index); %按照设定的顺序得到矩阵 A 的 zig-zag 扫描结果
end
--------------------------------------------------------------------------------------------------------------
```

方法二：

思路：

从第一角度出发，将 1 到 64 按照 Zig-Zag 扫描的顺序放入 8*8 的矩阵，使得扫描后得到的向量元素按照 1 至 64 的顺序排列。由角标的对应关系，即可得到任意 8*8 矩阵 Zig-Zag 扫描后的向量。

代码：

zigzag2.m

```
function B=zigzag2(A)
% 函数 zigzag2，对 8*8 矩阵进行 Zig-Zag 扫描
% 输入变量：  8*8 矩阵 A
% 返回值：  对 A 进行 zigzag 扫描得到的 64*1 向量

    [a,b]=size(A);          %获取输入矩阵大小
    if(a==8&&b==8)      %满足矩阵为 8*8 的要求
        %设定扫描顺序与矩阵的对应关系
        zigzag_index=[      1, 2, 6, 7, 15,16,28,29;
                            3, 5, 8, 14,17,27,30,43;
                            4, 9, 13,18,26,31,42,44;
                           10,12,19,25,32,41,45,54;
                           11,20,24,33,40,46,53,55;
                           21,23,34,39,47,52,56,61;
                           22,35,38,48,51,57,60,62;
                           36,37,49,50,58,59,63,64;];
        else error('函数的输入不符合要求!\n 要求输入为 8*8 矩阵');
                        %输入不符合要求
    end
    for i=1:8
        for j=1:8
            B(zigzag_index(i,j))=A(i,j);
              %按照 zigzag_index 与矩阵 A 的 zig-zag 的对应输出扫描结果
        end
     end
end
```

MATLAB 运行结果：

对于上面两种方法实现的 Zig-Zag 扫描函数(zigzag1 和 zigzag2)进行测试，详见 **zigzag_test.m** 文件。测试代码运行结果显示，两函数运行结果相同，都能实现实验要求的 Zig-Zag 扫描功能。

8. 对测试图像分块、DCT 和量化，将量化后的系数写成矩阵的形式，其中每一列为一个块的 DCT 系数 Zig-Zag 扫描后形成的列矢量。第一行为各个块的 DC 系数。

分析：

根据之前习题的准备，按照要求对测试图像依次进行：分块、离散余弦变换、量化、Zig-Zag 扫描的处理。

代码：

chapter2_8.m

```
clear all,close all,clc;

load hall.mat;                %获取 hall.mat 数据
load JpegCoeff.mat;           %获取 JpegCoeff.mat 数据

%处理前准备
[leny,lenx]=size(hall_gray);  %获取 hall_gray 大小
hallg=hall_gray;          %测试图像为 hall.mat 中的灰度图像 hall_gray
                              %另存为 hallg,避免更改原始数据 hall_gray
XNum=ceil(lenx/8);        %水平方向每 8 个像素,不是 8 的倍数需补全
YNum=ceil(leny/8);        %竖直方向每 8 个像素,不是 8 的倍数需补全
Total_Num=XNum*YNum;          %8*8 为 1 块，Total_Num 计算总个数
index_y=8*ones(1,YNum);       %cell 单元数组每个矩阵元素的竖直高度均为 8
index_x=8*ones(1,XNum);  %cell 单元数组每个矩阵元素的水平长度均为 8

%分块
Cell=mat2cell(hallg,index_y,index_x);     %将每个 8*8 的块当作 cell 单元元素
QC=zeros(8*8,Total_Num);      %对每块量化并 Zig-Zag 扫描得到系数矩阵

%减少灰度、DCT 量化、Zig-Zag 扫描
for i=1:YNum                          %依次对每个 8*8 小块进行处理
    for j=1:XNum
    Cell{i,j}=double(Cell{i,j})-128;   %对每小块图像进行预处理
    Cell_DCT=dct2(Cell{i,j});     %DCT：离散余弦变换，得到二维 DCT 系数矩阵
    Cell_QT=round(Cell_DCT./QTAB);    %量化处理
    QC(:,(i-1)*XNum+j)=zigzag2(Cell_QT);
     %对每小块的 DCT 系数矩阵扫描得到一个列矢量，所有列矢量构成矩阵
    end
end
DC=QC(1,:);               %第一行元素为各个块的 DC 系数
```

MATLAB 运行结果：

名称	值	最小值	最大值
ACTAB	*160x19 double*	0	16
Cell	*15x21 cell*		
Cell_DCT	*8x8 double*	-509....	70.37...
Cell_QT	*8x8 double*	-32	5
DC	*1x315 double*	-43	57
DCTAB	*12x10 double*	0	9
hall_color	*120x168x3 uint8*	0	255
hall_gray	*120x168 uint8*	4	255
hallg	*120x168 uint8*	4	255
QC	*64x315 double*	-43	57
QTAB	*8x8 double*	10	121

如上图所示，MATLAB 运行结果中得到的 QC 矩阵和 DC 系数向量。QC 矩阵是量化系数写成的矩阵，其中每一列为一个块的 DCT 系数 Zig-Zag 扫描后形成的列矢量。

9. 请实现本章介绍的 JPEG 编码（不包括写 JFIF 文件），输出为 DC 系数的码流，AC 系数的码流、图像高度和图像宽度，将这四个变量写入 jpegcodes.mat 文件。

分析：

按要求完后 2.8 中的分块、DCT、量化和 Zig-Zag 扫描之后，再分别对 DC 系数和 AC 系数进行编码。

代码：

chapter2_9.m

```
clear all,close all,clc;

load hall.mat;          %获取 hall.mat 数据
load JpegCoeff.mat;  %获取 JpegCoeff.mat 数据

%处理前准备
[leny,lenx]=size(hall_gray);      %获取 hall_gray 大小
hallg=hall_gray;                  %测试图像为 hall.mat 中的灰度图像 hall_gray
                                  %另存为 hallg,避免更改原始数据 hall_gray
XNum=ceil(lenx/8);   %水平方向每 8 个像素,不是 8 的倍数需补全
YNum=ceil(leny/8);   %竖直方向每 8 个像素,不是 8 的倍数需补全
```

```
Total_Num=XNum*YNum;     %8*8 为 1 块，Total_Num 计算总个数
index_y=8*ones(1,YNum);  %cell 单元数组每个矩阵元素的竖直高度均为 8
index_x=8*ones(1,XNum);  %cell 单元数组每个矩阵元素的水平长度均为 8

%分块
Cell=mat2cell(hallg,index_y,index_x); %将每个 8*8 的块当作 1 个 cell 单元元素
QC=zeros(8*8,Total_Num);  %对每块量化并进行 Zig-Zag 扫描之后的系数矩阵

%减少灰度、DCT 量化、Zig-Zag 扫描
for i=1:YNum                  %依次对每个 8*8 小块进行处理
    for j=1:XNum
    Cell{i,j}=double(Cell{i,j})-128;     %对每小块图像进行预处理
    Cell_DCT=dct2(Cell{i,j}); %DCT：离散余弦变换，得到二维 DCT 系数矩阵
    Cell_QT=round(Cell_DCT./QTAB); %量化处理
    QC(:,(i-1)*XNum+j)=zigzag1(Cell_QT);
    end
end
%---------------DC 系数编码--------------%
Err=QC(1,:);                          %亮度直流分量预测误差矢量
Category=zeros(1,Total_Num);          %Category 参量
Category(1)=ceil(log2(abs(Err(1))+1)); %对第一项预测误差进行单独处理
Magnitude=deci2bin(Err(1));           %Magnitude 是预测误差的二进制表示
                                      %负数则用 1 反码表示
                          %deci2bin 是专门为编码写的函数，详见 deci2bin.m
DCCode=[DCTAB(Category(1)+1,2:1+DCTAB(Category(1)+1,1)) Magnitude];
                                      %第一项预测误差的编码
for i=2:Total_Num                     %从第 2 项 DC 系数开始依次进行处理
    Err(i)=QC(1,i-1)-QC(1,i);         %预测误差的计算
    Category(i)=ceil(log2(abs(Err(i))+1));  %Category 参量的计算
    Magnitude=deci2bin(Err(i));       %Magnitude 是预测误差的二进制表示
    DCCode=[DCCodeDCTAB(Category(i)+1,2:1+DCTAB(Category(i)+1,1))Magnitude];
                                      %将每项 DC 系数的编码并入 DCCode 矢量中
                                      %包含 Category 和 Magnitude 两部分
end
%-----------AC 系数编码------------%
ACCode=[];                    %ACCode 初始化为空矩阵
ZRL=[1,1,1,1,1,1,1,1,0,0,1];  %连续 16 个 0，则插入 ZRL，(F/0)编码为 11111111001
EOB=[1,0,1,0];                %编完最后一个非零 AC 系数，插入块结束符 EOB
for i=1:Total_Num             %对扫描后所得的 QC 矩阵逐列进行处理
    Run=0;                    %行程 Run 初始化
    AC=QC(2:end,i);           %每列的第一个元素为 DC 系数，剩余为 AC 系数
    last_nonz=find(AC~=0,1,'last');
```

```
                        %先找到最后一个非零系数，之后元素均为 0 元素
    if ~isempty(last_nonz)    %last_nonz 可能为空矩阵（当 AC 全为 0 时）
        for j=1:last_nonz     %从 AC 系数向量的第一个到最后一个非零系数
            if AC(j)==0       %每逢零元素
                if Run<15 %判断是否连续 16 个零
                  Run=Run+1; %未到连续 16 个零时，每逢零元素，行程加 1
                else
                    ACCode=[ACCode ZRL];%连续 16 个零，则插入 ZRL
                    Run=0;                %并且将 Run 重新置零
                end
            else              %当 AC 系数是非零元素时
                Size=ceil(log2(abs(AC(j))+1));    %Size 的计算/与 Catagory 相同
                Amplitude=deci2bin(AC(j)) ;%Amplitude 计算与 Magnitude 相同
                L=ACTAB(10*Run+Size,3);   %获取 AC 编码的长度
                ACCode=[ACCode ACTAB(10*Run+Size,4:3+L) Amplitude];
                        %得到该非零元素的编码
                        %包括(Run/Size)联合体的 Huffman 编码和 Amplitude
                Run=0;    %将 Run 重新置零
            end
        end
    end
    ACCode=[ACCode EOB];  %每一列处理完都要加 EOB，先验
    end
%图像尺寸
Height=leny;          %图像高度
Width=lenx;           %图像宽度
%输出为 DC 系数的码流、AC 系数的码流、图像的高度和宽度
%将四个变量写入 jepgcodes.mat 文件
save('jpegcodes.mat','DCCode','ACCode','Height','Width');
```

MATLAB 运行结果：

名称	值	最小值	最大值
ACCode	*1x23072 double*	0	1
DCCode	*1x2054 double*	0	1
Height	120	120	120
Width	168	168	168

如上所示，MATLAB 运行结果中 jpegcodes 包含 DC 系数的码流、AC 系数的码流、图像的高度和宽度。

10. 计算压缩比（输入文件长度/输出码流长度），注意装转换为相同进制。

分析：

根据上一题中得到的 DC 系数的码流、AC 系数的码流长度，即可计算得到压缩比。

代码：

chapter2_10.m

```
clear all,close all,clc;

load jpegcodes.mat;
DCL=length(DCCode);        %DC 系数编码的码流长度
ACL=length(ACCode);        %AC 系数编码的码流长度
InputL=Height*Width*8;     %输入文件长度，转换为二进制，每个像素需要 8 位
OutputL=DCL+ACL;       %输出码流长度，包含 DC 系数码流和 AC 系数码流
COMR=InputL/OutputL;       %压缩比=输入文件长度/输出码流长度
```

MATLAB 运行结果：

压缩比计算公式：

$$\text{压缩比}=\frac{\text{输入文件长度}}{\text{输出码流长度}}$$

名称	值	最小值	最大值
ACCode	*1x23072 double*	0	1
COMR	6.4188	6.4188	6.4188
DCCode	*1x2054 double*	0	1
Height	120	120	120
Width	168	168	168

根据上面计算公式，如上图所示得到压缩比 COMR=6.4188。

11. 请实现本章介绍的 JPEG 解码，输入是你生成的 jpegcodes.mat 文件。分别用客观(PSNR)和主观方式评价编解码效果如何。

分析：

JPEG 解码逆着编码的思路逐步实现码流的复原。具体的实现方案在代码注释部分进行了详细的分析和解释。注意解码时，需要用到逆 Zig-Zag 函数将向量转换为 8*8 的矩阵。

逆 Zig-Zag 函数代码：

izigzag.m

```
function A=izigzag(B)
% 函数 izigzag，将 zigzag 扫描得到的 64 元素向量恢复为 8*8 矩阵
% 输入变量：  1*64 的矢量
```

```
% 返回值：      Zig-Zag 扫描之前的 8*8 矩阵

    len=length(B);    %获取输入矩阵大小
     A=zeros(8);
    if(len==64)       %满足矩阵为 8*8 的要求
        %设定扫描顺序与矩阵的对应关系
        zigzag_index=[   1, 2, 6, 7, 15,16,28,29;
                         3, 5, 8, 14,17,27,30,43;
                         4, 9, 13,18,26,31,42,44;
                         10,12,19,25,32,41,45,54;
                         11,20,24,33,40,46,53,55;
                         21,23,34,39,47,52,56,61;
                         22,35,38,48,51,57,60,62;
                         36,37,49,50,58,59,63,64;];
        else error('函数的输入不符合要求!\n 要求输入为 64 元素的向量');
    end
    for i=1:8
        for j=1:8
            A(i,j)=B(zigzag_index(i,j));
        end
     end
end
```

逆 Zig-Zag 函数测试代码：

izigzag_test.m

```
    clear all,close all,clc;

    %测试矩阵
    t=[ 1, 2, 3, 4, 5, 6, 7, 8 ;
        9, 10,11,12,13,14,15,16;
        17,18,19,20,21,22,23,24;
        25,26,27,28,29,30,31,32;
        33,34,35,36,37,38,39,40;
        41,42,43,44,45,46,47,48;
        49,50,51,52,53,54,55,56;
        57,58,59,60,61,62,63,64;];
    A=zigzag1(t);     %函数 zigzag1 结果
    C=izigzag(A);
    diffmax=max(max(abs(t-C)))
```

MATLAB 运行结果：

先调用 zigzag1 函数将 8*8 矩阵 Zig-Zag 扫描为向量，再调用上面的逆 Zig-Zag 扫描函数将向量还原为 8*8 矩阵，将还原后的矩阵与原来的矩阵作差，发现两矩阵完全相同，即 diffmax=max(max(abs(t-C)))=0，说明逆 Zig-Zag 函数能够正常实现功能。

JPEG 解码：

chapter2_11a.m

```
clear all,close all,clc;
load jpegcodes.mat;  %获取编码得到的码流和图像宽高信息
load JpegCoeff.mat;  %获取 JpegCoeff.mat 数据
%DCCode 解码得到 DC 系数
ErrDec=[];                %解码所得的预测误差向量 ErrDec 初始化
DLTAB=DCTAB(:,1);         %DC 系数预测误差的码本中编码的长度
NumDC=length(DLTAB);      %DC 系数预测误差的码本的 Category 总数
DCL=length(DCCode);       %DC 系数预测误差编码后 DCCode 的码流总长度
code_start=1;             %解码时，起始是 Huffman 编码部分
                          %故使用 code_start 表示 Huffman 码的起始位
while(code_start<=DCL)    %判断解码未完成时
    for CT=1:NumDC        %在码本寻找与码流匹配的 Category，与 Category 对比
        code_end=code_start+DLTAB(CT)-1;
                %Huffman 码的终止位，与尝试匹配时所选的 Category 有关
        if (DCTAB(CT,2:DLTAB(CT)+1))==DCCode(code_start:code_end)
                %完全匹配，找到码本中对应的行
            Category=CT-1;      %获取 Category
            break;              %不再与码本之后的内容对比
        end
    end
    bit_start=code_end+1;       %对于每个被编码的 DC 系数预测误差
    if Category==0
        ErrDec=[ErrDec 0];      %当 Category 为 0 时，单独处理
        code_start=bit_start+1;
                %下一位是下一个 DC 预测误差的 Huffman 码的起始位
    else
        bit_end=bit_start+Category-1;
                %其他情况下，由 Category 计算出 Magnitude 的终止位
        bits=DCCode(bit_start:bit_end);
                %起始位和终止位之间的部分是预测误差对应的 Magnitude
        ErrDec=[ErrDec bin2deci(bits)];
    %根据 bits 由函数 bin2cdeci 得到十进制形式的预测误差，并入预测误差向量
        code_start=bit_end+1;
```

```
    %该预测误差解码完毕，更新下一预测误差的 Huffman 码起始位置
     end
end
DCDec=ErrDec;                          %准备由预测误差向量得到 DC 系数
for i=2:length(ErrDec)
    DCDec(i)=DCDec(i-1)-ErrDec(i);    %DC 系数向量，即为解码后的 DC 系数部分
end
%AC 解码
ZRL=[1,1,1,1,1,1,1,1,0,0,1];     %ZRL 的编码，表示 AC 系数中 16 个连零
EOB=[1,0,1,0];                   %结束符 EOB 的编码，每块结尾都有结束符
ACDec=[];                        %解码所得的 AC 系数向量初始化
ALTAB=ACTAB(:,3);                %AC 系数码本中不同 Run/Size 的码长 L
NumAC=length(ALTAB);             %AC 系数码本中不同 Run/Size 的总个数
ACL=length(ACCode);              %AC 编码码流总长度
zero_dbg=zeros(1,5);
    %特别说明：由于解码时 code_end 可能访问到超过 ACCode 长度的部分
    %为了防止 MATLAB 报错，延长 ACCode 长度,事实上对结果没有任何影响
ACCode=[ACCode zero_dbg];        %延长后得到的 ACCode
code_start=1;                    %解码时，起始是 Huffman 编码部分
while(code_start<=ACL)           %判断解码未完成时,对 ACCode 码流进行解码
    if(ACCode(code_start:code_start+3)==EOB)    %解码遇到 EOB 结束符
        currL=length(ACDec);    %获取当前解码得到的 ACDec 的长度
        if(mod(currL,63)==0&&ACDec(end)~=0)
          %如果当前 ACDec 长度已经为 63 的倍数并且最后一位是非零数
                                 %表明不需要再补零
          code_start=code_start+4;   %直接进入下一段 Huffman 码的解码
        else                     %其他情况需要将 ACDec 用零补全 63 位
          zero=zeros(1,63-mod(currL,63));
          %用于补全的零向量，63-mod(currL,63)计算需要零的个数
          ACDec=[ACDec zero];        %用零向量补全 63 位
          code_start=code_start+4;   %处理 EOB 完毕，进入下一段解码
        end
    elseif (ACCode(code_start:code_start+10)==ZRL)
          %解码遇到 ZRL，表明 AC 系数矩阵中有连续 16 个零
        zero=zeros(1,16);            %16 个零的向量
        ACDec=[ACDec zero];          %解码向量 ACDec 中增加 16 个零
        code_start=code_start+11;  %处理 ZRL 完毕，进入下一段 Huffma 解码
    else     %除去 EOB 和 ZRL 之外，其他情况需要根据码流依次解码
        for k=1:NumAC                %在码本中寻找与码流匹配的 Run/Size
          code_end=code_start+ALTAB(k)-1;       %Huffman 码的终止位
          if (ACTAB(k,4:ALTAB(k)+3))==ACCode(code_start:code_end)
                                 %完全匹配，找到码本中对应的行
```

```
                Run=ACTAB(k,1);   %获取行程 Run，即非零系数之前零的个数
                Size=ACTAB(k,2);  %获取 Size，AC 系数二进制表示时的比特数
                break;
            end
        end
        zero=zeros(1,Run);          %生成长度等于 Run 的零矩阵
        ACDec=[ACDec zero];         %解码向量 ACDec 中增加该矩阵
        bit_start=code_end+1;       %下一位对应 AC 系数二进制形式的起始位
        bit_end=bit_start+Size-1;   %由 Size 计算 AC 系数二进制形式的终止位
        bits=ACCode(bit_start:bit_end);
                        %从 ACCode 码流中得到 AC 系数对应的二进制形式
        ACDec=[ACDec bin2deci(bits)];
            %根据 bits 由函数 bin2cdeci 得到十进制形式的 AC 系数
        code_start=bit_end+1;       %该 AC 系数解码完毕
    end
end
ACcount=length(ACDec);          %获取解码得到的系数向量总长度
ACMat=reshape(ACDec,63,ACcount/63);  %将向量转换为 63 行的矩阵
CoMat=[DCDec;ACMat];  %DC 系数和 AC 系数合并，恢复得到 DCT 系数矩阵
%后续处理
YNum=Height/8;          %8*8 为一块
XNum=Width/8;
Cell=cell(YNum,XNum);   %分块处理，得到元胞数组（cell 单元数组）
for i=1:YNum            %从左向右-从上至下逐块处理
    for j=1:XNum
        Cell{i,j}=izigzag(CoMat(:,(i-1)*XNum+j));
            %逆 Zig-Zag 函数，根据扫描后 1*64 的向量还原 8*8 矩阵
        Cell_DCT=Cell{i,j}.*QTAB;     %反量化处理
        Cell{i,j}=idct2(Cell_DCT);    %idct2 二维离散余弦逆变换
        Cell{i,j}=round(Cell{i,j}+128); %加 128 进行复原
    end
end
Recovery=cell2mat(Cell);      %拼接操作，将 Cell 元胞数组还原为矩阵形式
hallg_rec=uint8(Recovery);    %转换为 uint8 类型，hall_rec 表示还原得到图像矩阵
save('jpegdecodes.mat','hallg_rec');    %另存为 jpegdecodes.mat
```

客观（PSNR）和主观方法评价编解码效果

chapter2_11b.m

```
clear all,close all,clc;
%用客观（PSNR）和主观方法评价编解码效果
load jpegdecodes.mat;
load hall.mat;
```

```
hallg=hall_gray;  %测试图像为 hall.mat 中的灰度图像 hall_gray
                  %另存为 hallg,避免更改原始数据 hall_gray
[Height,Width]=size(hallg);     %获取图像大小
PixelNum=Height*Width;          %总像素数目，用于计算 MSE
MSE=1/PixelNum*sum(sum((double(hallg_rec)-double(hallg)).^2));
                                %根据公式计算 MSE
PSNR=10*log10(255^2/MSE); %根据公式计算 PSNR
subplot(1,2,1);                 %子图 1
imshow(hallg);                  %作出原图
title('原图');                  %设定标题
imwrite(hallg,'./chapter2_11/1 原图.bmp');   %另存用图片浏览器浏览
subplot(1,2,2);                 %子图 2
imshow(hallg_rec);              %作出经过 JPEG 编码和解码之后复原得到的图
title('JPEG 编解码复原图');     %设定标题
imwrite(hallg_rec,'./chapter2_11/2JPEG 编解码复原图.bmp');
                                %另存用图片浏览器浏览
```

MATLAB 运行结果：

图 2.11.1

图 2.11.2

如上面两图所示：图 2.11.1 是原图，图 2.11.2 是 JPEG 解码后所得的图。从主观、客观方式两个方面评价编解码效果：

① 主观方式评价编解码效果：

观察 JPEG 编解码后的图与原图对比，可以看出两幅图几乎没有区别，只有微小的失真。

② 客观方式(PSNR)评价编解码效果：

名称	值	最小值	最大值
hall_color	*120x168x3 uint8*	0	255
hall_gray	*120x168 uint8*	4	255
hallg	*120x168 uint8*	4	255
hallg_rec	*120x168 uint8*	0	255
Height	120	120	120
MSE	49.4698	49.46...	49.46...
PixelNum	20160	20160	20160
PSNR	31.1874	31.18...	31.18...
Width	168	168	168

根据 MATLAB 的计算结果，即可计算得到 PSNR=31.1874。峰值信噪比相对较高，可见 JPEG 编解码的效果较好。

12. 将量化步长减小为原来的一半，重做编解码。同标准量化步长的情况比较压缩比和图像质量。

分析：

JpegCoeff.mat 提供了量化步长矩阵 QTAB，将量化步长减小为原来的一半，即：QTAB=QTAB/2。JPEG 编解码代码中其他部分不作其他变动。

代码：

代码与 2.11 的实现代码改动不大，故报告中省略。详见 **chapter2_12.m**。

MATLAB 运行结果：

图 2.12.1

图 2.12.2

如上面两图所示：

图 2.12.1 是原图，图 2.12.2 是 JPEG 解码后所得的图。从主观、客观方式两个方面评价编解码效果：

① 主观方式评价编解码效果：

观察 JPEG 编解码后的图与原图对比，可以看出两幅图几乎没有区别，目测没有失真；可以看出，比 2.11 中实现的 JPEG 编解码的失真小。

② 客观方式(PSNR)评价编解码效果：

名称	值	最小值	最大值
ACCode	*1x34169 double*	0	1
ACMat	*63x315 double*	-79	113
ACTAB	*160x19 double*	0	16
Cell	*15x21 cell*		
CoMat	*64x315 double*	-86	115
COMR	4.4081	4.4081	4.4081
DCCode	*1x2423 double*	0	1
DCTAB	*12x10 double*	0	9
hall_gray	*120x168 uint8*	4	255
Height	120	120	120
MSE	24.6837	24.68...	24.68...
PixelNum	20160	20160	20160
PSNR	34.2067	34.20...	34.20...
QC	*64x315 double*	-86	115
QTAB	*8x8 double*	5	60.50...

根据 MATLAB 的计算结果，即可计算得到 PSNR=34.2067，比之前 2.11 所得的信噪比高。峰值信噪比相对更高，可见 JPEG 编解码的效果更好。

13. 看电视时偶尔能看到美丽的雪花图像(见 snow.mat)，请对其编解码，和测试图像的压缩比和图像质量进行比较，并解释比较结果。

分析：

按照相同的处理方式处理 snow.mat 中的数据，代码只需作维系微小改动。

代码：

代码与 2.11 的实现代码改动不大，故报告中省略。详见 **chapter2_13.m**。

MATLAB 运行结果：

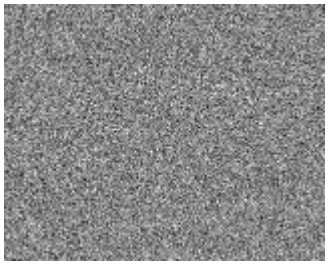


图 2.13.1

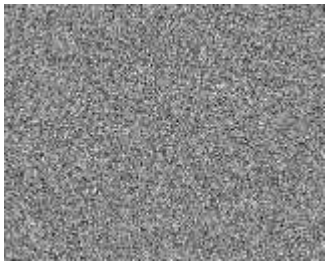


图 2.13.2

如上面两图所示：

图 2.13.1 是原图，图 2.13.2 是 JPEG 解码后所得的图。从主观、客观方式两个方面评价编解码效果：

① 主观方式评价编解码效果：

雪花图像从观察角度基本看不出什么差别。

② 客观方式(PSNR)评价编解码效果：

名称	值	最小值	最大值
ACCode	*1x43551 double*	0	1
ACMat	*63x320 double*	-11	9
ACTAB	*160x19 double*	0	16
Cell	*16x20 cell*		
CoMat	*64x320 double*	-11	9
COMR	3.6407	3.6407	3.6407
DCCode	*1x1456 double*	0	1
DCTAB	*12x10 double*	0	9
MSE	331.6164	331.6...	331.6...
PixelNum	20480	20480	20480
PSNR	22.9244	22.92...	22.92...
QC	*64x320 double*	-11	9
QTAB	*8x8 double*	10	121
snow	*128x160 uint8*	0	255

根据 MATLAB 的计算结果，即可计算得到压缩比为 3.6407，峰值信噪比 PSNR=22.9244，比之前 2.12 所得的信噪比低。 雪花图像噪声过大，且高频分量过多，从而导致压缩比有限，峰值信噪比相对较低，可见 JPEG 编解码的效果较差。

第三章 信息隐藏技术

本章练习题所用测试图像同上一章，本章练习题所指代隐藏信息可自由选择。

1. 实现本章介绍的空域隐藏方法和提取方法，验证其抗 JPEG 编码能力。

分析：

空域信息隐藏技术直接将信息数据插到像素的亮度中。先将图像信息表示成二进制码流，再依次用每位信息替换掉图像中各亮度分量的最低位。

优点：简单且数据隐藏量很大(8 个 bits 可以隐藏 1 个 bit)；

缺点：只能以未压缩的格式存储图像，否则附着信息会被当作是高频分量而舍弃掉。

先实现空域隐藏技术和提取方法，将隐藏了信息的图像进行 JPEG 编码，再尝试从 JPEG 解码后的图像中提取出隐藏的信息，从而验证空域信息隐藏技术的抗 JPEG 编码能力。

代码：

空域信息隐藏技术和提取方法的实现：

chapter3_1a.m

```
clear all,close all,clc;
load hall.mat;              %获取 hall.mat 中 hall_gray 的数据
hallg=double(hall_gray);  %uint8 到 double 的类型转换
%对信息进行空域隐藏
info='Hello World';         %需要隐藏的信息(字符串)
info_bin=dec2bin(double(info));   %将字符串转换为 char 型，存储二进制
[height,width]=size(info_bin);      %获取 info_bin 的宽高
len=height*width;           %获取二进制字符串的长度
info_bit=[];                %二进制数组初始化
for i=1:height
    for j=1:width
    info_bit=[info_bit str2num(info_bin(i,j))];
    end     %得到 info_bit 数组，存储了需要隐藏的字符串的二进制数组形式
end
for i=1:len                 %隐藏到图像中，用信息位替换图像各像素灰度最低位
    bit=deci2bin(hallg(i));     %获取图像像素点的灰度值，并转换为二进制数组
    bit(end)=info_bit(i);       %将该像素点灰度的最低位替换为隐藏的信息位
    hallg(i)=bin2deci(bit);     %再将改变后的像素灰度转换回十进制数形式
end
%imshow(uint8(hallg));          %查看空域隐藏信息之后的图像

%提取隐藏的信息
```

```
for i=1:len                    %len 为隐藏信息的字符串长度
    bit=deci2bin(hallg(i));    %对该长度范围内的每一像素点的灰度值进行处理
    exc(i)=bit(end);           %提取出每个灰度值的最低位
end
exc=reshape(exc,width,height);   %改变矩阵的长宽
info_exc=[];                     %提取出的信息 初始化
for i=1:height                   %获取隐藏的字符串中每一个字符的 ASII 值
    info_exc=[info_exc uint8(bin2deci(exc(:,i)))];
end                              %并且转换为 uint8 类型
info_exc=char(info_exc)          %将 ASCII 数组转换为字符串
```

MATLAB 运行结果：

隐藏的信息位字符串“Hello World”；

将信息进行空域隐藏之后，再进行信息提取，所得结果为：

```
info_exc =

Hello World
```

表明成功实现了空域信息隐藏技术和提取方法。

信息隐藏后进行 JPEG 编解码，检测抗 JPEG 能力：

chapter3_1b.m

```
clear all,close all,clc;

load hall.mat;          %获取 hall.mat 中 hall_gray 的数据
load JpegCoeff.mat;     %获取 JpegCoeff.mat 数据
hallg=double(hall_gray);  %uint8 到 double 的类型转换

%对信息进行空域隐藏
info='Hello World';          %需要隐藏的信息(字符串)
info_bin=dec2bin(double(info));   %将字符串转换为 char 型，存储二进制
[height,width]=size(info_bin);       %获取 info_bin 的宽高
len=height*width;     %获取二进制字符串的长度
info_bit=[];          %二进制数组初始化
for i=1:height
    for j=1:width
    info_bit=[info_bit str2num(info_bin(i,j))];
    end               %得到存储了需要隐藏的字符串的 info_bit 数组
end
for i=1:len           %隐藏到图像中，用信息位逐一替换图像各像素灰度最低位
    bit=deci2bin(hallg(i));     %获取图像像素点的灰度值，并转换为二进制数组
```

```
    bit(end)=info_bit(i);      %将该像素点灰度的最低位替换为隐藏的信息位
    hallg(i)=bin2deci(bit);    %再将改变后的像素灰度转换回十进制数形式
end
%imshow(uint8(hallg));         %查看空域隐藏信息之后的图像
%----------------------------------------编码开始----------------------------------------%
%处理前准备
[leny,lenx]=size(hall_gray);   %获取 hall_gray 大小
hallg=hall_gray;       %测试图像为 hall.mat 中的灰度图像 hall_gray
                       %另存为 hallg,避免更改原始数据 hall_gray
XNum=ceil(lenx/8);     %水平方向每 8 个像素,不是 8 的倍数需补全
YNum=ceil(leny/8);     %竖直方向每 8 个像素,不是 8 的倍数需补全
Total_Num=XNum*YNum;   %8*8 为 1 块，Total_Num 计算总个数
index_y=8*ones(1,YNum);    %cell 单元数组每个矩阵元素的竖直高度均为 8
index_x=8*ones(1,XNum);    %cell 单元数组每个矩阵元素的水平长度均为 8
%分块
Cell=mat2cell(hallg,index_y,index_x);  %将每个 8*8 的块当作 1 个 cell 单元元素
QC=zeros(8*8,Total_Num);   %对每块量化并进行 Zig-Zag 扫描之后的系数矩阵
%减少灰度、DCT 量化、Zig-Zag 扫描
for i=1:YNum               %依次对每个 8*8 小块进行处理
    for j=1:XNum
    Cell{i,j}=double(Cell{i,j})-128;     %对每小块图像进行预处理
    Cell_DCT=dct2(Cell{i,j});  %DCT：离散余弦变换，得到二维 DCT 系数矩阵
    Cell_QT=round(Cell_DCT./QTAB);  %量化处理
    QC(:,(i-1)*XNum+j)=zigzag1(Cell_QT);     %对每小块的 DCT 系数矩阵扫描
    end
end
%---------------DC 系数编码--------------%
Err=QC(1,:);                       %亮度直流分量预测误差矢量
Category=zeros(1,Total_Num);       %Category 参量
Category(1)=ceil(log2(abs(Err(1))+1));  %对第一项预测误差进行单独处理
Magnitude=deci2bin(Err(1));
   %Magnitude 是预测误差的二进制表示，负数则用 1 反码表示
   %deci2bin 是专门为编码写的函数，详见 deci2bin.m
DCCode=[DCTAB(Category(1)+1,2:1+DCTAB(Category(1)+1,1)) Magnitude];
   %第一项预测误差的编码
for i=2:Total_Num           %从第 2 项 DC 系数开始依次进行处理
    Err(i)=QC(1,i-1)-QC(1,i);  %预测误差的计算
    Category(i)=ceil(log2(abs(Err(i))+1));  %Category 参量的计算
    Magnitude=deci2bin(Err(i));  %Magnitude 是预测误差的二进制表示
    DCCode=[DCCodeDCTAB(Category(i)+1,2:1+DCTAB(Category(i)+1,1)) Magnitude];
                           %将每项 DC 系数的编码并入 DCCode 矢量中
                           %包含 Category 和 Magnitude 两部分
```

```
end
%----------AC 系数编码----------%
ACCode=[];                    %ACCode 初始化为空矩阵
ZRL=[1,1,1,1,1,1,1,1,0,0,1];  %连续 16 个 0，则插入 ZRL，(F/0)编码为 11111111001
EOB=[1,0,1,0];                %编完最后一个非零 AC 系数，插入块结束符 EOB
for i=1:Total_Num          %对扫描后所得的 QC 矩阵逐列进行处理
    Run=0;                 %行程 Run 初始化
    AC=QC(2:end,i);        %每列的第一个元素为 DC 系数，剩余元素为 AC 系数
    last_nonz=find(AC~=0,1,'last');
                           %先找到最后一个非零系数，之后元素均为 0 元素
    if ~isempty(last_nonz)    %last_nonz 可能为空矩阵
        for j=1:last_nonz     %从 AC 系数向量的第一个最后一个非零系数
            if AC(j)==0       %每逢零元素
                if Run<15 %判断是否连续 16 个零
                    Run=Run+1;  %无连续 16 个零时，每逢零元素，行程加 1
                else
                    ACCode=[ACCode ZRL];     %连续 16 个零，则插入 ZRL
                    Run=0;          %并且将 Run 重新置零
                end
            else                          %当 AC 系数是非零元素时
                Size=ceil(log2(abs(AC(j))+1));    %Size 的计算/与 Catagory 相同
                Amplitude=deci2bin(AC(j));
                L=ACTAB(10*Run+Size,3);         %获取 AC 编码的长度
                ACCode=[ACCode ACTAB(10*Run+Size,4:3+L) Amplitude];
                    %得到该非零元素的编码
                    %包括(Run/Size)联合体的 Huffman 编码和 Amplitude
                Run=0;    %将 Run 重新置零
            end
        end
    end
    ACCode=[ACCode EOB]; %每一列处理完都要加 EOB
end
%图像尺寸
Height=leny;            %图像高度
Width=lenx;             %图像宽度
%-----------------------------------------编码结束-----------------------------------%
%-----------------------------------------计算压缩比-----------------------------------%
DCL=length(DCCode);     %DC 系数编码的码流长度
ACL=length(ACCode);     %AC 系数编码的码流长度
InputL=Height*Width*8; %输入文件长度，转换为二进制，每个像素需要 8 位
OutputL=DCL+ACL;        %输出码流长度，包含 DC 系数码流和 AC 系数码流
COMR=InputL/OutputL;   %压缩比=输入文件长度/输出码流长度
```

```
%----------------------------------------计算结束----------------------------------------%
%----------------------------------------解码开始----------------------------------------%
%DCCode 解码得到 DC 系数
ErrDec=[];                  %解码所得的预测误差向量 ErrDec 初始化
DLTAB=DCTAB(:,1);           %DC 系数预测误差的码本中编码的长度
NumDC=length(DLTAB);        %DC 系数预测误差的码本的 Category 总数
DCL=length(DCCode);         %DC 系数预测误差编码后 DCCode 的码流总长度
code_start=1;               %解码时，起始是 Huffman 编码部分，故使用 code_start
表示 Huffman 码的起始位
while(code_start<=DCL)      %判断解码未完成时
    for CT=1:NumDC          %在码本中寻找匹配的 Category
        code_end=code_start+DLTAB(CT)-1;
        %Huffman 码的终止位，与尝试匹配时所选的 Category 有关
        if (DCTAB(CT,2:DLTAB(CT)+1))==DCCode(code_start:code_end)
        %完全匹配，找到码本中对应的行
            Category=CT-1;          %获取 Category
            break;                  %不再与码本之后的内容对比
        end
    end
    bit_start=code_end+1;   %对于每个被编码的 DC 系数预测误差
        if Category==0
        ErrDec=[ErrDec 0];  %当 Category 为 0 时，单独处理
        code_start=bit_start+1; %下一个 DC 预测误差的 Huffman 码的起始位
    else
        bit_end=bit_start+Category-1;
        %其他情况下，由 Category 计算出 Magnitude 的终止位
        bits=DCCode(bit_start:bit_end);
        %起始位和终止位之间的部分是预测误差对应的 Magnitude，即 bits
        ErrDec=[ErrDec bin2deci(bits)];
        %由函数 bin2cdeci 得到十进制形式的预测误差，并入预测误差向量
        code_start=bit_end+1;
        %该预测误差解码完毕，更新下一预测误差的 Huffman 码起始位置
    end
end
DCDec=ErrDec;               %准备由预测误差向量得到 DC 系数
for i=2:length(ErrDec)
    DCDec(i)=DCDec(i-1)-ErrDec(i);
        %倒推得到 DC 系数向量 DCDec,即为解码后的 DC 系数部分
end
%AC 解码
ZRL=[1,1,1,1,1,1,1,1,0,0,1];    %ZRL 的编码，表示 AC 系数中 16 个连零
EOB=[1,0,1,0];                  %结束符 EOB 的编码，AC 系数每块结尾都有结束符
```

```
ACDec=[];                    %解码所得的 AC 系数向量初始化
ALTAB=ACTAB(:,3);            %AC 系数码本中不同 Run/Size 的码长 L
NumAC=length(ALTAB);         %AC 系数码本中不同 Run/Size 的总个数
ACL=length(ACCode);          %AC 编码码流总长度
zero_dbg=zeros(1,5);
    %由于解码时 code_end 可能访问到超过 ACCode 长度的部分。
    %为了防止 MATLAB 报错，延长 ACCode 长度,事实上对结果没有任何影响
ACCode=[ACCode zero_dbg]; %延长后得到的 ACCode
code_start=1;                %解码时，起始是 Huffman 编码部分
while(code_start<=ACL)       %判断解码未完成时,对 ACCode 码流进行解码
     if(ACCode(code_start:code_start+3)==EOB)     %解码遇到 EOB 结束符
          currL=length(ACDec);    %获取当前解码得到的 ACDec 的长度
          if(mod(currL,63)==0&&ACDec(end)~=0)
    %如果当前 ACDec 长度已经为 63 的倍数并且最后一位是非零数
    %表明不需要再补零
               code_start=code_start+4;
    %直接进入下一段 Huffman 码的解码,更新 code_start
          else %其他情况需要将 ACDec 用零补全 63 位(ACDec 长度为 63 的倍数)
               zero=zeros(1,63-mod(currL,63));
    %用于补全的零向量，63-mod(currL,63)计算需要零的个数
               ACDec=[ACDec zero];         %用零向量补全 63 位
               code_start=code_start+4;    %处理 EOB 完毕
          end
     elseif (ACCode(code_start:code_start+10)==ZRL)
    %解码遇到 ZRL，表明 AC 系数矩阵中有连续 16 个零
          zero=zeros(1,16);              %16 个零的向量
          ACDec=[ACDec zero];            %解码向量 ACDec 中增加 16 个零
          code_start=code_start+11;      %处理 ZRL 完毕
     else       %除去 EOB 和 ZRL 之外，其他情况需要根据码流依次解码
          for k=1:NumAC                  %在码本中寻找与码流匹配的 Run/Size
               code_end=code_start+ALTAB(k)-1;
    %Huffman 码的终止位，与码本中对应的码长有关
               if (ACTAB(k,4:ALTAB(k)+3))==ACCode(code_start:code_end)
    %完全匹配，找到码本中对应的行
                    Run=ACTAB(k,1);   %获取行程 Run，非零系数之前零的个数
                    Size=ACTAB(k,2);  %获取 Size，即 AC 系数二进制表示的比特数
                    break;
               end
          end
          zero=zeros(1,Run);         %生成长度等于 Run 的零矩阵
          ACDec=[ACDec zero];        %解码向量 ACDec 中增加该矩阵
          bit_start=code_end+1;      %下一位对应该 AC 系数二进制形式的起始位
```

```
            bit_end=bit_start+Size-1;     % AC 系数二进制形式的终止位
            bits=ACCode(bit_start:bit_end);
        %从 ACCode 码流中得到 AC 系数对应的二进制形式
            ACDec=[ACDec bin2deci(bits)];
        %由函数 bin2cdeci 得到十进制形式的 AC 系数，并入 AC 系数向量
            code_start=bit_end+1;         %该 AC 系数解码完毕
        end
    end
    ACcount=length(ACDec);                    %获取解码得到的系数向量总长度
    ACMat=reshape(ACDec,63,ACcount/63);    %将向量转换为 63 行的矩阵
    CoMat=[DCDec;ACMat];
        %DC 系数和 AC 系数合并，恢复得到 DCT 系数矩阵(Zig-Zag 处理之后)

    %后续处理
    YNum=Height/8;                %8*8 为一块
    XNum=Width/8;
    Cell=cell(YNum,XNum);         %分块处理，得到元胞数组（cell 单元数组）
    for i=1:YNum                  %从左向右-从上至下逐块处理
        for j=1:XNum
            Cell{i,j}=izigzag(CoMat(:,(i-1)*XNum+j));
        %逆 Zig-Zag 函数，根据扫描后 1*64 的向量还原 8*8 矩阵
            Cell_DCT=Cell{i,j}.*QTAB;     %反量化处理
            Cell{i,j}=idct2(Cell_DCT);    %idct2 二维离散余弦逆变换
            Cell{i,j}=round(Cell{i,j}+128); %加 128 进行复原
        end
    end
    Recovery=cell2mat(Cell);
        %拼接操作，将 Cell 元胞数组还原为矩阵形式，宽高与原图的完全相同
    hallg_rec=uint8(Recovery);          %转换为 uint8 类型
    %----------------------------------------解码结束---------------------------------------%
    %提取隐藏的信息
    for i=1:len                         %len 为隐藏信息的字符串长度
        bit=deci2bin(Recovery(i));      %对该长度范围内的每一像素灰度值进行处理
        exc(i)=bit(end);                %提取出每个灰度值的最低位
    end
    exc=reshape(exc,width,height);      %改变矩阵的长宽
    info_exc=[];                        %提取出的信息 初始化
    for i=1:height                      %获取隐藏的字符串中每一个字符的 ASII 值
        info_exc=[info_exc uint8(bin2deci(exc(:,i)))];
    end                                 %并且转换为 uint8 类型
    info_exc=char(info_exc)             %将 ASCII 数组转换为字符串
```

MATLAB 运行结果：

隐藏的信息位字符串“Hello World”；

将信息进行空域隐藏，然后进行 JPEG 编解码操作，之后再进行信息提取，所得结果为：

```
info_exc =

  Nm u W
```

表明空域信息隐藏技术和提取方法的抗 JPEG 编码能力很差。因而，空域信息隐藏技术的缺点是只能以未压缩的格式存储图像；否则，图像一经有损压缩，附着信息就会被当作高频分量而被舍弃掉。

2. 依次实现本章介绍的变换域信息隐藏方法和提取方法，分析嵌密方法的隐蔽以及嵌密后 JPEG 图像的质量变化和压缩比变化。

（一）、用信息位逐一替换掉每个量化后的 DCT 系数的最低位，再进行熵编码。

分析：

变换域信息隐藏方法是将信息隐藏在 DCT 域中；与空域信息隐藏技术类似，变换域信息隐藏是用信息位逐一替换掉每个量化后的 DCT 系数的最低位，之后再进行熵编码。

代码：

代码较大部分与空域信息隐藏方法的代码相同，核心部分是变换域信息隐藏技术和提取方法的实现。代码如下：

chapter3_2a.m

```
%------------------对信息进行 DCT 域隐藏--------------------%
info='Hello World';                    %需要隐藏的信息(字符串)
info_bin=dec2bin(double(info));        %将字符串转换为 char 型，存储二进制
[height,width]=size(info_bin);   %获取 info_bin 的宽高
len=height*width;        %获取二进制字符串的长度
info_bit=[];             %二进制数组初始化
for i=1:height
    for j=1:width
    info_bit=[info_bit str2num(info_bin(i,j))];
    end                  %info_bit 存储了需要隐藏的字符串的二进制数组形式
end
for i=1:len      %隐藏到图像中，用信息位逐一替换图像各像素点的灰度最低位
    bit=deci2bin(QC(i));    %获取图像像素点的灰度值，并转换为二进制数组
    bit(end)=info_bit(i);   %将该像素点灰度的最低位替换为隐藏的信息位
    QC(i)=bin2deci(bit);    %再将改变后的像素灰度转换回十进制数形式
end
```

```
%------------------在信息 DCT 域隐藏完毕--------------------%
%--------------------提取隐藏的信息-------------------------%
for i=1:len                          %len 为隐藏信息的字符串长度
    bit=deci2bin(CoMat(i));          %对该长度范围内的每一像素灰度值进行处理
    exc(i)=bit(end);                 %提取出每个灰度值的最低位
end
exc=reshape(exc,width,height); %改变矩阵的长宽
info_exc=[];                         %提取出的信息 初始化
for i=1:height                       %获取隐藏的字符串中每一个字符的 ASII 值
    info_exc=[info_exc uint8(bin2deci(exc(:,i)))];
end                                  %并且转换为 uint8 类型
info_exc=char(info_exc)      %将 ASCII 数组转换为字符串
%--------------------隐藏的信息提取完毕-----------------------%
```

MATLAB 运行结果：

图 3.2.1

图 3.2.2

如以上两图所示，图 3.2.1 是原图，图 3.2.2 是经过变换域信息隐藏之后所得的图像。

```
info_exc =

Hello World
```

隐藏的信息为字符串“Hello World”，经过变换域信息隐藏之后提取得到的信息也是“Hello World”，表明变换域信息隐藏技术的抗 JPEG 压缩能力较强。

名称	值	最小值	最大值
ACCode	*1x23245 double*	0	1
ACTAB	*160x19 double*	0	16
Cell	*15x21 cell*		
COMR	6.3762	6.3762	6.3762
DCCode	*1x2054 double*	0	1
DCTAB	*12x10 double*	0	9
hall_gray	*120x168 uint8*	4	255
MSE	57.3157	57.31...	57.31...
PixelNum	20160	20160	20160
PSNR	30.5481	30.54...	30.54...
QC	*64x315 double*	-43	57
QTAB	*8x8 double*	10	121

对变换域信息隐藏技术的评价：

从上面 MATLAB 运行的结果中能够得出，上面实现的变换域信息隐藏技术实现的压缩比为 6.3762，相对较高；而峰值信噪比 PSNR 为 30.5481，相对于不加隐藏信息的 JPEG 编解码较低。从直观角度来看，也可以看出，隐藏信息之后恢复所得的图 3.2.2 的左上角有明显的失真。

（二）、用信息位逐一替换掉若干位量化后的 DCT 系数的最低位。

分析：

与（一）中类似，只不过要求中将“每位”更改为“若干位”，基本思路和实现方法不变。

代码：

代码较大部分与（一）中信息隐藏方法的代码相同，核心部分是变换域信息隐藏技术和提取方法的实现。代码如下：

chapter3_2b.m

```
%------------------对信息进行 DCT 域隐藏--------------------%
info='Hello World';                 %需要隐藏的信息(字符串)
info_bin=dec2bin(double(info));     %将字符串转换为 char 型，存储二进制
[height,width]=size(info_bin);      %获取 info_bin 的宽高
len=height*width;                   %获取二进制字符串的长度
info_bit=[];                        %二进制数组初始化
for i=1:height
    for j=1:width
    info_bit=[info_bit str2num(info_bin(i,j))];
    end         %得到存储了需要隐藏的字符串的 info_bit 数组
end
for i=1:len         %隐藏到图像中，用信息位逐一替换图像各像素点的灰度最低位
    bit=deci2bin(QC(32,i));   %获取图像像素点的灰度值，并转换为二进制数组
    bit(end)=info_bit(i);     %将该像素灰度最低位替换为需要隐藏的信息位
    QC(32,i)=bin2deci(bit);   %再将改变后的像素灰度转换回十进制数形式
end
%------------------在信息 DCT 域隐藏完毕--------------------%
%-------------------提取隐藏的信息-------------------------%
for i=1:len                         %len 为隐藏信息的字符串长度
    bit=deci2bin(CoMat(32,i));    %对该长度范围每一像素灰度值进行处理
    exc(i)=bit(end);              %提取出每个灰度值的最低位
end
exc=reshape(exc,width,height);    %改变矩阵的长宽
info_exc=[];                        %提取出的信息 初始化
for i=1:height                      %获取隐藏的字符串中每一个字符的 ASII 值
```

```
    info_exc=[info_exc uint8(bin2deci(exc(:,i)))];
end                                    %转换为 uint8 类型
info_exc=char(info_exc)                %将 ASCII 数组转换为字符串
%-------------------隐藏的信息提取完毕-----------------------%
```

MATLAB 运行结果：

图 3.2.3

图 3.2.4

如以上两图所示，图 3.2.3 是原图，图 3.2.4 是经过变换域信息隐藏之后所得的图像。

```
info_exc =

Hello World
```

隐藏的信息为字符串“Hello World”，经过变换域信息隐藏之后提取得到的信息也是“Hello World”，表明变换域信息隐藏技术的抗 JPEG 压缩能力较强。

名称	值	最小值	最大值
ACCode	*1x24288 double*	0	1
ACTAB	*160x19 double*	0	16
Cell	*15x21 cell*		
COMR	6.1237	6.1237	6.1237
DCCode	*1x2054 double*	0	1
DCTAB	*12x10 double*	0	9
hall_gray	*120x168 uint8*	4	255
MSE	59.1558	59.15...	59.15...
PixelNum	20160	20160	20160
PSNR	30.4108	30.41...	30.41...
QC	*64x315 double*	-43	57
QTAB	*8x8 double*	10	121

对变换域信息隐藏技术的评价：

从上面 MATLAB 运行的结果中能够得出，上面实现的变换域信息隐藏技术实现的压缩比为 6.1237，比（一）中实现的变换域信息隐藏的压缩比稍低；而峰值信噪比 PSNR 为 30.4108，相对于不加隐藏信息的 JPEG 编解码较低，且低于（一）中的 PSNR。从直观角度来看，也可以看出，隐藏信息之后恢复所得的图 3.2.4 的上半部分有非常明显的波浪状的失真条纹，不能够满足实际的信息隐藏功能。

（三）先将待隐藏信息用 1，-1 的序列表示，再逐一将信息位追加在每个块 Zig-Zag 顺序的最后一个非零 DCT 系数之后；如果原本该图像块的最后一个系数就不为零，那就用信息位替换该系数。

分析：

同样是在 DCT 域进行信息的隐藏，该方法只能在每个 8*8 的块中存储 1 位信息，因而信息的存储量有所减少。根据要求实现信息的隐藏，在信息提取中，每列最后一个非零系数即为隐藏的信息位，从而做到隐藏信息的提取。

代码：

代码较大部分与（一）中信息隐藏方法的代码相同，核心部分是变换域信息隐藏技术和提取方法的实现。代码如下：

chapter3_2c.m

```
%------------------对信息进行 DCT 域隐藏--------------------%
info='Hello World';                    %需要隐藏的信息(字符串)
info_bin=dec2bin(double(info));    %将字符串转换为 char 型，存储二进制
[height,width]=size(info_bin);       %获取 info_bin 的宽高
len=height*width;          %获取二进制字符串的长度
info_bit=[];                    %二进制数组初始化
for i=1:height
    for j=1:width
    info_bit=[info_bit str2num(info_bin(i,j))];
    end     %得到存储了需要隐藏的字符串的 info_bit 数组
end
info_bit(find(info_bit==0))=-1;      %将二进制数组中的 0 元素全部更改为-1
for i=1:len    %隐藏到图像中，用信息位逐一替换图像各像素灰度最低位
    block=QC(:,i);                    %QC 的第 i 列表示第 i 块
    last_nonz=find(block~=0,1,'last'); %找到最后一个非零系数
        if(last_nonz==64)          %当最后一个系数非零时，用信息位替换
        QC(64,i)=info_bit(i);
    else     %其他情况，将信息位追加在最后一个非零 DCT 系数之后
        QC(last_nonz+1,i)=info_bit(i);
    end
end
%------------------在信息 DCT 域隐藏完毕--------------------%
%--------------------提取隐藏的信息-------------------------%
for i=1:len                    %len 为隐藏信息的字符串长度
    block=CoMat(:,i);     %CoMat 第 i 列表示第 i 块
    last_nonz=find(block~=0,1,'last');
                           %先找到最后一个非零系数，之后元素均为 0 元素
    if(last_nonz==64)     %CoMat 某列最后一个系数非零时，是信息位
        exc(i)=CoMat(64,i);
```

```
        else
            exc(i)=CoMat(last_nonz,i);    %其他情况该列最后一个非零系数是信息位
        end
end
exc(find(exc==-1))=0;                  %将 1，-1 序列还原为二进制数组形式
exc=reshape(exc,width,height);         %改变矩阵的长宽
info_exc=[];                           %提取出的信息 初始化
for i=1:height                         %获取隐藏的字符串中每一个字符的 ASII 值
    info_exc=[info_exc uint8(bin2deci(exc(:,i)))];
end                                    %并且转换为 uint8 类型
info_exc=char(info_exc)                %将 ASCII 数组转换为字符串
%--------------------隐藏的信息提取完毕-----------------------%
```

MATLAB 运行结果：

图 3.2.5

图 3.2.6

如以上两图所示，图 3.2.5 是原图，图 3.2.6 是经过变换域信息隐藏之后所得的图像。

```
info_exc =

Hello World
```

隐藏的信息为字符串“Hello World”，经过变换域信息隐藏之后提取得到的信息也是“Hello World”，表明变换域信息隐藏技术的抗 JPEG 压缩能力较强。

名称	值	最小值	最大值
ACCode	*1x23308 double*	0	1
ACTAB	*160x19 double*	0	16
Cell	*15x21 cell*		
COMR	6.3604	6.3604	6.3604
DCCode	*1x2054 double*	0	1
DCTAB	*12x10 double*	0	9
hall_gray	*120x168 uint8*	4	255
MSE	52.7845	52.78...	52.78...
PixelNum	20160	20160	20160
PSNR	30.9057	30.90...	30.90...
QC	*64x315 double*	-43	57
QTAB	*8x8 double*	10	121

对变换域信息隐藏技术的评价：

从上面 MATLAB 运行的结果中能够得出，上面实现的变换域信息隐藏技术实现的压缩比为 6.3604，比（一）中实现的变换域信息隐藏的压缩比稍低，但是比（二）的压缩比高；而峰值信噪比 PSNR 为 30.9057，相对于不加隐藏信息的 JPEG 编解码较低，但是比（一）（二）高。从直观角度来看，也可以看出，隐藏信息之后恢复所得的图 3.2.6 的上半部分有比较明显的失真，对实际的信息隐藏有一定影响。

3. (选做) 请设计实现新的隐藏算法并分析其优缺点。

分析：

设计新的隐藏算法与上题的第一二中隐藏算法类似，变动在于：用信息位替换 DC 系数的最低位。DC 系数是每块的直流分量，在 DCT 系数矩阵中位于第一行，且绝对值相对较大，预计会有较好的隐藏效果。

代码：

chapter3_3.m

```
--------------------------------------------------------------------------------------------------------------
%------------------对信息进行 DCT 域隐藏--------------------%
info='Hello World';                 %需要隐藏的信息(字符串)
info_bin=dec2bin(double(info));     %将字符串转换为 char 型，存储二进制
[height,width]=size(info_bin);      %获取 info_bin 的宽高
len=height*width;                   %获取二进制字符串的长度
info_bit=[];                        %二进制数组初始化
for i=1:height
    for j=1:width
    info_bit=[info_bit str2num(info_bin(i,j))];
    end     %得到存储了需要隐藏的字符串的二进制 info_bit 数组
end
for i=1:len    %隐藏到图像中，用信息位逐一替换图像各像素点的灰度最低位
    bit=deci2bin(QC(1,i));     %获取图像像素点的灰度值，并转换为二进制数组
    bit(end)=info_bit(i);      %将该像素灰度最低位替换为需要隐藏的信息位
    QC(1,i)=bin2deci(bit);     %再将改变后的像素灰度转换回十进制数形式
end
%------------------在信息 DCT 域隐藏完毕--------------------%

%--------------------提取隐藏的信息----------------------------%
for i=1:len                         %len 为隐藏信息的字符串长度
    bit=deci2bin(CoMat(1,i));       %对该长度范围内的每一像素灰度值进行处理
    exc(i)=bit(end);                %提取出每个灰度值的最低位
end
exc=reshape(exc,width,height);      %改变矩阵的长宽
info_exc=[];        %提取出的信息  初始化
```

```
for i=1:height     %获取隐藏的字符串中每一个字符的 ASII 值
    info_exc=[info_exc uint8(bin2deci(exc(:,i)))];
end                                %转换为 uint8 类型
info_exc=char(info_exc)            %将 ASCII 数组转换为字符串
%-------------------隐藏的信息提取完毕-----------------------%
```

MATLAB 运行结果：

图 3.3.1

图 3.3.2

如以上两图所示，图 3.3.1 是原图，图 3.2.2 是经过变换域信息隐藏之后所得的图像。

```
info_exc =

Hello World
```

隐藏的信息为字符串“Hello World”，经过变换域信息隐藏之后提取得到的信息也是“Hello World”，表明变换域信息隐藏技术的抗 JPEG 压缩能力较强。

名称	值	最小值	最大值
ACCode	*1x23077 double*	0	1
ACTAB	*160x19 double*	0	16
Cell	*15x21 cell*		
COMR	6.4181	6.4181	6.4181
DCCode	*1x2057 double*	0	1
DCTAB	*12x10 double*	0	9
hall_gray	*120x168 uint8*	4	255
MSE	50.1154	50.11...	50.11...
PixelNum	20160	20160	20160
PSNR	31.1311	31.13...	31.13...
QC	*64x315 double*	-43	57
QTAB	*8x8 double*	10	121

对变换域信息隐藏技术的评价：

从上面 MATLAB 运行的结果中能够得出，上面实现的变换域信息隐藏技术实现的压缩比为 6.4181，比之前实现的所有信息隐藏技术的压缩比都高；同样，峰值信噪比 PSNR 为 31.1311，相对于不加隐藏信息的 JPEG 编解码较低，但是在变换域信息隐藏技术中同样最高。从直观角度来看，也可以看出，隐藏信息之后恢复所得的图 3.3.1 的失真不是很明显，在没有原图参照的情况下几乎看不出失真。因此，该方法实现的变换域信息隐藏技术在已经实现的技术中性能最好。

四种变换域信息隐藏技术的比较：

实现方式	压缩比 COMR	峰值信噪比 PSNR
2（一）	6.3762	30.5481
2（二）	6.1237	30.4108
2（三）	6.3604	30.9507
3	6.4181	31.1311

通过对比可以看出，第 3 题实现的信息隐藏技术效果最佳。

第四章 人脸检测

1. 所给资料 Faces 目录下包含从网图中截取的 28 张人脸，试以其作为样本训练人脸标准 v。

(a) 样本人脸大小不一，是否需要首先将图像调整为相同大小？

(b) 假设 L 分别取 3, 4, 5，所得三个 v 之间有何关系？

分析：

(a) 样本训练人脸标准 v 所描述的是各种颜色在总颜色中出现的比率，不论样本的人脸大小如何，最终都作归一化处理，即用比率表示。因而与样本人脸图像的大小没有关系，不需要将图像调整为相同大小。

(b) 根据人脸检测样本训练的原理，L 越大，所能检测的颜色总数越多，从而对图像颜色分布的描述更加精确，所得到的人脸标准 v 也更精确，从而对后续人脸的检测也更准确。总结来说，L 越大，v 的准确度越高。

代码：

特征提取：

ColorExc.m

```
function [u] = ColorExc(L,Face)
% 函数功能：提取图像的颜色特征
% 输入：    L 用于衡量颜色的种类数，Face 是图像矩阵
% 返回值：  颜色特征，各种不同颜色的比例构成的矢量

    [width,len,dim]=size(Face);
    pixelnum=width*len;
    colornum=2^(3*L);
    adder=1/pixelnum;
    u=zeros(1,colornum);
    for i=1:width
        for j=1:len
            Color=double(Face(i,j,:));
            Color=floor(Color/2^(8-L));
            n=1+2^(2*L)*Color(1)+2^L*Color(2)+Color(3);
            u(n)=u(n)+adder;
        end
    end
end
```

训练标准特征：

chapter4_1a.m

```
clear all,close all,clc;
FaceNum=33;
L=3;                                    %L=3
colornum=2^(3*L);
v_sum=zeros(1,colornum);
 for i=1:FaceNum
     filename=sprintf('%i.bmp',i);      %各图像文件名
     Face=imread(filename);             %读取图片
     u=ColorExc(L,Face);
     v_sum=v_sum+u;
 end
 v1=v_sum/FaceNum;
 save('v1.mat','v1');
```

chapter4_1b.m

```
clear all,close all,clc;
FaceNum=33;
L=4;                                    %L=4
colornum=2^(3*L);
v_sum=zeros(1,colornum);
 for i=1:FaceNum
     filename=sprintf('%i.bmp',i);     %各图像文件名
     Face=imread(filename);             %读取图片
     u=ColorExc(L,Face);
     v_sum=v_sum+u;
 end
 v2=v_sum/FaceNum;
 save('v2.mat','v2');
```

chapter4_1c.m

```
clear all,close all,clc;
FaceNum=33;
L=5;
colornum=2^(3*L);
v_sum=zeros(1,colornum);
```

```
for i=1:FaceNum
    filename=sprintf('%i.bmp',i);     %各图像文件名
    Face=imread(filename);              %读取图片
    u=ColorExc(L,Face);
    v_sum=v_sum+u;
end
v3=v_sum/FaceNum;
save('v3.mat','v3');
```

---

2. 设计一种从任意大小的图片中检测任意多张人脸的算法并编程实现（输出图像在判定为人脸的位置加上红色的方框）。随意选取一张多人图片（比如支部活动或者足球比赛），对程序进行测试。尝试 L 分别取不同的值，评价检测结果有何区别。

分析：

根据作业要求中基于彩色直方图的人脸检测方法，按照选取特征、训练标准特征和检测的顺序，编程实现该人脸检测方法。

代码：

chapter4_2a.m　　chapter4_2b.m　　chapter4_2c.m

```
clear all,close all,clc;

load v1.mat;
v=v1;L=3;e=0.30;
pace=10;
Face_size=28;
Faces=imread('test.jpg');
[width,len,dim]=size(Faces);
YNum=ceil((width-Face_size)/pace);
XNum=ceil((len-Face_size)/pace);
ys1=[];ys2=[];xs1=[];xs2=[];mark=[];
for i=1:YNum
    for j=1:XNum
        y1=(i-1)*pace+1;
        y2=(i-1)*pace+Face_size;
        x1=(j-1)*pace+1;
        x2=(j-1)*pace+Face_size;
        Block=Faces(y1:y2,x1:x2,:);
        u=ColorExc(L,Block);
        Sqr=sqrt(u.*v);
        d=1-sum(Sqr);
```

```
            if(d<e)
                ys1=[ys1,(i-1)*pace+1];
                ys2=[ys2,(i-1)*pace+Face_size];
                xs1=[xs1,(j-1)*pace+1];
                xs2=[xs2,(j-1)*pace+Face_size];
                mark=[mark,1];
            end
        end
end
LocNum=length(ys1);
for i=1:LocNum-1
        isinter=0;
        for j=i+1:LocNum
            a1=((ys1(j)-ys1(i))<(ys2(i)-ys1(i)));
            a2=((ys1(i)-ys1(j))<(ys2(j)-ys1(j)));
            a3=((xs1(j)-xs1(i))<(xs2(i)-xs1(i)));
            a4=((xs1(i)-xs1(j))<(xs2(j)-xs1(j)));
            if(a1&&a2&&a3&&a4)
                isinter=1;
                break;
            end
        end
        if(isinter)
            ys1(j)=min(ys1(i),ys1(j));
           ys2(j)=max(ys2(i),ys2(j));
            xs1(j)=min(xs1(i),xs1(j));
          xs2(j)=max(xs2(i),xs2(j));
        else
            mark(i)=0;
        end
end
mark(LocNum)=0;
for i=1:LocNum
        if(~mark(i))
            a=ys1(i);b=ys2(i);
            c=xs1(i);d=xs2(i);
            Faces(a,c:d,1)=255;Faces(a,c:d,2)=0;Faces(a,c:d,3)=0;
            Faces(b,c:d,1)=255;Faces(b,c:d,2)=0;Faces(b,c:d,3)=0;
                Faces(a:b,c,1)=255;Faces(a:b,c,2)=0;Faces(a:b,c,3)=0;
            Faces(a:b,d,1)=255;Faces(a:b,d,2)=0;Faces(a:b,d,3)=0;
        end
end
```

```
imshow(Faces);
imwrite(Faces,'./Faces/Faces1.bmp');
```

MATLAB 运行结果：

① L=3 时，调整阈值，得到如下结果：

分析：

可以看出，编程实现的人脸检测方法能够正确识别图像中的人脸，且识别效果很好，没有发生误判。

② L=4 时，

分析：

可以看出，L=4 时，调整代码中的判断阈值 e，编程实现的人脸检测方法能够正确识别图像中的人脸，且识别效果很好，同样没有发生误判。

③ L=5 时，

分析：

可以看出，L=5 时，调整代码中的判断阈值 e，编程实现的人脸检测方法能够正确识别图像中的人脸，且识别效果很好，同样没有发生误判。

3. 对上述图像分别进行如下处理后

(a) 顺时针旋转 90°(imrotate)；

(b) 保持高度不变，宽度拉伸为原来的两倍(imresize)；

(c) 适当改变颜色(imadjust)；

再试试你的算法检测结果如何？并分析所得结果。

分析：

分别用函数 imrotate, imresize 和 imadjust 函数实现图像的变化后，再用之前实现的基于彩色直方图的人脸检测方法进行人脸检测，即可验证算法的准确性。

代码：

chapter4_3a.m

```
clear all,close all,clc;

load v1.mat;
v=v1;L=3;e=0.30;
```

```
pace=10;
Face_size=28;
Faces=imread('test.jpg');

Faces=imrotate(Faces,270);
%Faces=imrotate(Faces,60);
%Faces=imrotate(Faces,120);

[width,len,dim]=size(Faces);
YNum=ceil((width-Face_size)/pace);
XNum=ceil((len-Face_size)/pace);
ys1=[];ys2=[];xs1=[];xs2=[];mark=[];
for i=1:YNum
    for j=1:XNum
        y1=(i-1)*pace+1;
        y2=(i-1)*pace+Face_size;
        x1=(j-1)*pace+1;
        x2=(j-1)*pace+Face_size;
        Block=Faces(y1:y2,x1:x2,:);
        u=ColorExc(L,Block);
        Sqr=sqrt(u.*v);
        d=1-sum(Sqr);
        if(d<e)
            ys1=[ys1,(i-1)*pace+1];
            ys2=[ys2,(i-1)*pace+Face_size];
            xs1=[xs1,(j-1)*pace+1];
            xs2=[xs2,(j-1)*pace+Face_size];
            mark=[mark,1];
        end
    end
end
LocNum=length(ys1);
for i=1:LocNum-1
    isinter=0;
    for j=i+1:LocNum
        a1=((ys1(j)-ys1(i))<(ys2(i)-ys1(i)));
        a2=((ys1(i)-ys1(j))<(ys2(j)-ys1(j)));
        a3=((xs1(j)-xs1(i))<(xs2(i)-xs1(i)));
        a4=((xs1(i)-xs1(j))<(xs2(j)-xs1(j)));
        if(a1&&a2&&a3&&a4)
            isinter=1;
            break;
```

```
            end
        end
        if(isinter)
            ys1(j)=min(ys1(i),ys1(j));
            ys2(j)=max(ys2(i),ys2(j));
            xs1(j)=min(xs1(i),xs1(j));
            xs2(j)=max(xs2(i),xs2(j));
        else
            mark(i)=0;
        end
    end
    mark(LocNum)=0;
    for i=1:LocNum
        if(~mark(i))
            a=ys1(i);b=ys2(i);
            c=xs1(i);d=xs2(i);
            Faces(a,c:d,1)=255;Faces(a,c:d,2)=0;Faces(a,c:d,3)=0;
            Faces(b,c:d,1)=255;Faces(b,c:d,2)=0;Faces(b,c:d,3)=0;
            Faces(a:b,c,1)=255;Faces(a:b,c,2)=0;Faces(a:b,c,3)=0;
            Faces(a:b,d,1)=255;Faces(a:b,d,2)=0;Faces(a:b,d,3)=0;
        end
    end
    imshow(Faces);
    imwrite(Faces,'./Faces_rotate/Faces_rotate90.bmp');
    %imwrite(Faces,'./Faces_rotate/Faces_rotate60.bmp');
    %imwrite(Faces,'./Faces_rotate/Faces_rotate120.bmp');
```

---

chapter4_3b.m 注意：imresize 函数之后的代码与 chapter4_3a.m 基本相同，略。

```
    clear all,close all,clc;

    load v1.mat;
    v=v1;L=3;e=0.25;
    pace=10;
    Face_size=28;
    Faces=imread('test.jpg');
    [width,len,dim]=size(Faces);
    len=2*len;
    Faces=imresize(Faces,[width,len]);
```

---

chapter4_3c.m 注意：imadjust 函数之后的代码与 chapter4_3a.m 基本相同，略。

```
clear all,close all,clc;

load v1.mat;
v=v1;L=3;e=0.5;
pace=10;
Face_size=28;
Faces=imread('test.jpg');
[width,len,dim]=size(Faces);
Faces=imadjust(Faces,[0.2,0.3,0;0.6,0.7,1],[]);
```

MATLAB 运行结果：

① 瞬时针旋转 90 度：

分析：

不调整阈值会发生误判，调整阈值之后，可以看出，将图像逆时针旋转 90 度之后，编程实现的人脸检测方法仍然能够正确识别图像中的人脸，且识别效果很好，同样没有发生误判。

② 保持高度不变，宽度拉伸为原来的两倍：

分析：

不调整阈值会发生误判，调整阈值之后，可以看出，保持高度不变，宽度拉伸为原来的两倍，编程实现的人脸检测方法仍然能够正确识别图像中的人脸，且识别效果很好，同样没有发生误判。

③ 适当调整颜色：

分析：

可以看出，调整图像整体的颜色，编程实现的人脸检测方法完全无法正确识别图像中的人脸，产生了误判。这是因为该人脸检测算法的核心思想就是根据图像的颜色分布进行人脸识别，因而图像颜色一经改变就无法实现人脸识别。

4. 如果可以重新选择人脸样本训练标准，你觉得应该如何选取？

解答：

实现了基于彩色直方图的人脸检测方法，发现了该方法具有一定的不足。为了保证人脸样本训练标准的精确性，应当选取不同角度和方向的人脸图像作为样本；并且，最好根据不同肤色的人种得到不同的训练标准，以提高人脸检测的准确度。同时通过作业中的习题可以看出，该人脸检测方法对图像颜色的依赖性很强，因而需要保证人脸样本图像的整体色调为自然色调。

实验心得

本次实验实现了图像压缩编码、图像信息隐藏和人脸检测三种图像处理技术，整体实验难度较高，耗费了较长的时间。实验完成之后，也有很大的成就感。通过本次实验，自己在编程和调试代码的过程中对以上图像处理技术有了更深的了解，能够基本掌握图像处理的思路和方法，同时也对图像处理产生了浓厚的兴趣，总体来说收获很大。