多媒体技术基础及应用 图像视频实验一 实验报告

2013011427　刘智峰　计 31

【Exp1】

一、实验环境

编程语言：MATLAB

实验平台：MATLAB R2014a on windows 7

二、实验任务

1、给出 lena 图片的灰度图

2、对全图使用先行后列的一维 DCT 变换

3、对全图使用 2 维 DCT 变换

4、将图片分为 8*8 的模块，进行 2 维 DCT 变换

5、分析上述任务程序所用的时间，以及 IDCT 重构得到的图片和原图的 PSNR。

6、以 1、1/4、1/16、1/64 的比例调整 DCT 系数，重复上述任务，观察并分析实验结果。

三、实现说明

一维、二维的 DCT 变换直接使用了 MATLAB 自带的 dct()、dct2()函数；对于将图片进行 8*8 分块并进行对应的 DFT、IDFT 变换，刚开始自己实现了一个 for 循环来分割 512*512 的图片，较为繁琐，代码很冗余。后来在网上找到了按 8*8 大小分块，并进行 DCT 系数压缩的相关资料，并进行了学习，使用 MATLAB 自带的 blkproc 函数，配合适用于小矩阵的 DCT 变换函数 dctmtx(8)，大大简化了代码量。

Blkproc 函数的用法为：I = blkproc(A,[size1 , size2] , fun , arg1 ,agr2...)。其中 A 是待分块矩阵 ,size 为分块的规模 ,fun 为自己实现的函数 ,arg1 ,agr2...为 fun 的参数。在指定了 size 后 ,此函数会自动将原来的 A 矩阵按[size1 ,size2]规模进行划分，然后对每个分出来的子矩阵，将其传入 fun 函数，执行相应的功能。最后 ,会将所有子矩阵执行 fun 函数后得到的结果矩阵拼到一起 ,存入 I 中。本次实验中，我用到 I2=blkproc(A,[8 8]，“P1*x*P2”，T，T’)，其中 T=dctmtx(8)。这样就相当于将 A 矩阵中的每一个不重复的 8*8 矩阵进行一个 2 维 DCT 变换，然后将结果拼到 I2 中，这样就实现了对整个矩阵分块进行二维 DCT 变换。再将 T 和 T’置换位置，实现一个逆变换，即可得到重构的图片。

在进行 DCT 系数压缩时，通过观察可以看出，无论是 1 维 DCT 变换还是 2 维 DCT 变换，DCT 矩阵的非零值主要位于左上角，右下角几乎都是 0。所以，可以按压缩系数保留左上角的数据，把其余的数据清 0，达到压缩的目的。举个例子，如果要进行 1/4 的系数压缩，那就把 DFT 矩阵的左上角 1/4 的部分，即[1,256]*[1,256]的范围留下，其余的部分都清 0，这样就实现了只保留 1/4 的内容，即实现了我设想的 1/4 压缩。对于 1/16、1/64 的压缩系数类似。而对于 8*8 分块的矩阵，我直接使用了 MATLAB 自带的 blkproc 函数进行压缩。

上述 8*8 分块 DCT、压缩的内容，主要参考了：

http://www.doc88.com/p-2129986101481.html

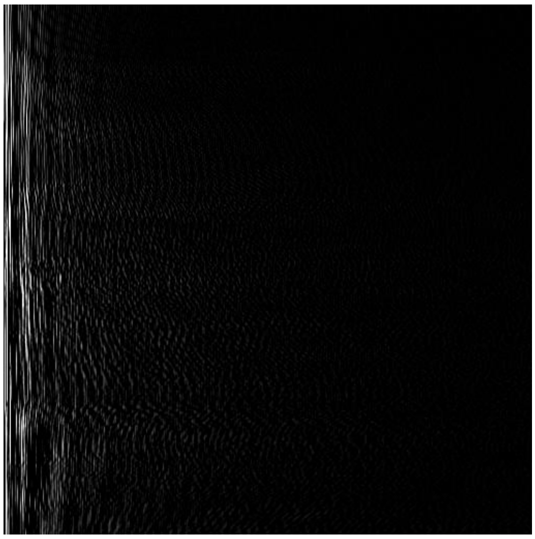
四、实验结果

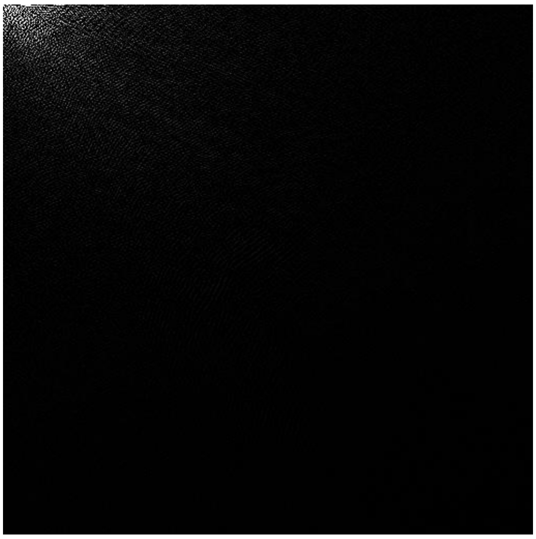
1、lena 图片的灰度图：(左边为原图，右边为灰度图)


2、对全图使用先行后列的一维 DCT 变换

如下，左边为原灰度图，右边为一维 DCT 变换，再进行逆变换后的重构图


可见两幅图几乎没有什么区别。下面附上先行后列 DCT 变换后的结果图：

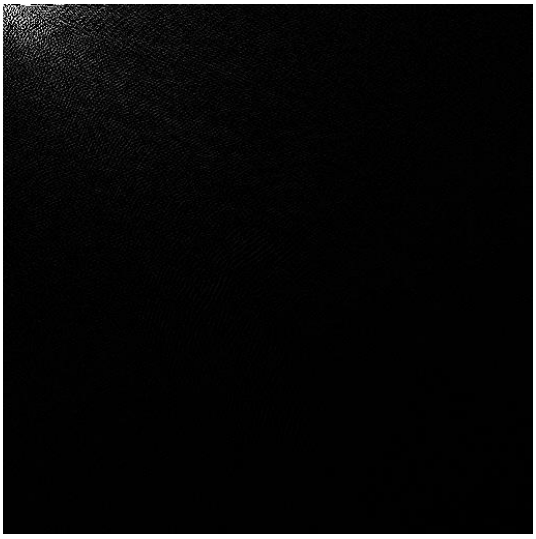




3、对全图使用 2 维 DCT 变换

如下，左边为原灰度图，右边为二维 DCT 变换，再进行逆变换后的重构图

可见两幅图几乎没有什么区别。2 维 DCT 变换后的结果如下：

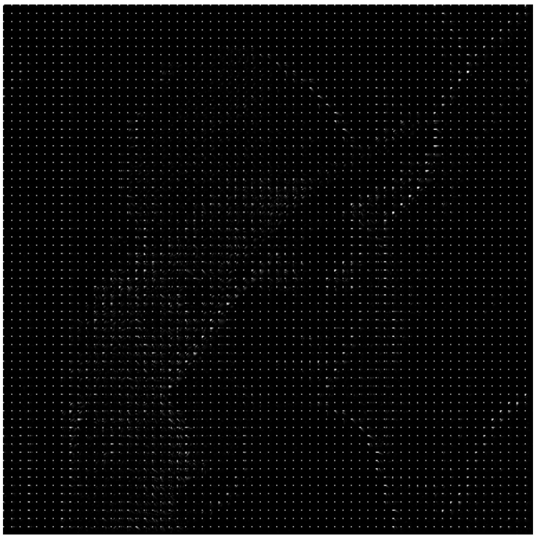


4、将图片分为 8*8 的模块，进行 2 维 DCT 变换

如下，左边为原灰度图，右边为二维 DCT 变换，再进行逆变换后的重构图

可见两幅图还是几乎没有什么区别。8*8,分块 2 维 DCT 变换后的结果如下，

可以看出，DCT 变换后得到的矩阵描绘出了 lena 的轮廓。



5、不同压缩系数对 DCT 结果的影响

如下系列图。第一张为原图，第二张为灰度图。

第二行为 1 维 DCT，第三行为 2 维 DCT，第四行为 2 维 8*8 分块 DCT；从左到右，分别是使用 1、1/4、1/16、1/64 的 DCT 系数变换后，进行逆变换得到的结果。如果将 1、1/4、1/16、1/64 称为 DCT 的压缩系数，可见从左到右，图片越来越模糊，说明压缩系数越小，图片还原得越差。同时，按列来看，缩小后图片看上去差别不是很大。但是从原图进行观察，可以很清楚地看出，当压缩系数为 1 时，3 中方法的图片与原图都差不多，肉眼难以辨别。当压缩系数减小时，可以看出 1 维 DCT 和 2 维 DCT 的重构结果没有什么区别，而 2 维 8*8 分块 DCT 的结果更加模糊，这也和同一系数下，2 维 8*8 分块 DCT 的 PSNR 值最小相符合。

放大后的 8*8 分块 1/64 压缩 DCT 结果和 2 维 1/64 压缩 DCT 对比如下：

可见，1/64 压缩 2 维 DCT 变换，再经过逆变换后结果更加模糊(左图)。

6、上述任务的运行时间、对应的 IDFT 重构图片与原始灰度图片的 PSNR 分析

统计结果如下，具体数据可查看 结果统计.xlsx

PSNR 结果：

算法 / 压缩系数	行列 1D-DCT	直接 2D-DCT	分块 2D-DCT
1	333.8549	333.8115	320.3367
1/4	57.2727	57.2727	55.922
1/16	50.9603	50.9603	49.2543
1/64	46.9023	46.9023	44.7098

已知 PSNR 值与 MSE 成反比，PSNR 越大，MSE 值越小，两张图片差别越小，越接近。从结果可以看出，没有进行压缩的情况下，3 中算法的 PSNR 值都很大，且都很接近，说明没有进行压缩的情况下，三种算法都能够通过 IDCT 重构，很好地还原出原始图片。而加上压缩系数后，PSNR 值就变小了，而且呈现压缩系数越小，PSNR 值越小的趋势，这与压缩系数越小，图片越模糊的实验结果相吻合。同时，可以看出 1 维 DCT 和 2 维 DCT 的效果差不多，而 2 维分块 DCT 的 PSNR 会略大一些，效果会稍微差一点。

运行时间结果：

首先来分析一下算法的复杂度。

对于 1 维 DCT 变换，公式为：

$$F(u)=\sqrt{\frac{2}{N}}C(u)\sum_{x=0}^{N-1}f(x)\cos\left[\frac{\pi(2x+1)u}{2N}\right]$$

$$f(x)=\sqrt{\frac{2}{N}}\sum_{u=0}^{N-1}C(u)F(u)\cos\left[\frac{\pi(2x+1)u}{2N}\right]$$

可见，每个值 u 做 DCT 的复杂度为$O(N)$，一行的复杂度为$O(N^2)$。无论是行还是列，1D-DCT 的复杂度都是$O(N^3)$。

对于 2 维 DCT 变换，公式为：

$$\text{FDCT:} \quad S_{uv} = \frac{1}{4} C_u C_v \sum_{i=0}^{7} \sum_{j=0}^{7} s_{ij} \cos\frac{(2i+1)u\pi}{16} \cos\frac{(2j+1)v\pi}{16}$$

$$\text{IDCT:} \quad s_{ij} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v S_{uv} \cos\frac{(2i+1)u\pi}{16} \cos\frac{(2j+1)v\pi}{16}$$

$$C_u C_v = \begin{cases} \frac{1}{\sqrt{2}} & for\ u,v=0 \\ 1 & otherwise \end{cases}$$

可见，计算 u、v 点的复杂度都是$O(N^2)$，所以总体复杂度为$O(N^4)$。

对于 8*8 分块 2 维 DCT 变换，每小块的复杂度为$O(8*8)$，即$O(1)$。所以总体复杂度为$O(N^2)$。

实际运行时间结果如下：

压缩系数＼算法	行列 1D-DCT	直接 2D-DCT	分块 2D-DCT
1	0.353189	0.06261	2.346848
1/4	0.345049	0.09783	4.203706
1/16	0.347657	0.0997	3.848901
1/64	0.400276	0.10576	3.849279

好像和理论的复杂度分析结果不符。按照理论分析的结果，应该是分块 2D<行列 1D<直接 2D。我认为，可能的原因有：

(1)、MATLAB 中的 dct、dct2 函数可能并没有严格按照公式来实现，而是使用了类似于快速傅里叶变换等的优化算法，导致实际复杂度与按照 1D-dct、2D-dct 公式分析的复杂度不同。

(2)、由于本次整个图片的规模也不过是 512*512，数量级不大，可能常数对时间的影响太大(因为如果数量组足够大的话，常数是不影响复杂度的)。如果图片的规模更大一些，运行时间可能会更符合时间复杂度的规模。

五、实验总结

通过对实验结果的分析，可以看出，使用 1D-DCT 或 2D-DCT 的方式进行图像压缩，可以有效地保存图片中的整体信息，丢失的只会是一些细节方面，所以逆变换重构后得到的图片和原图没有明显的差别；当对 DCT 的系数进行压缩时，图片质量会产生明显的下降；对大图片进行分块后 DCT 压缩，复杂度是最小的，可能需要的时间也是最少的，但是逆变换重构后的图片效果可能会打折扣，不如对没有分块的原图直接进行 DCT 变换的效果好。

六、实验数据

(1)、源代码：

my_psnr.m：按照公式计算两个矩阵的 PSNR 值

coefficients.m：按照传入的参数，对 DCT 系数进行压缩

DCT.m：主程序，通过调用 psnr 和 coefficients 函数，实现上述任务。

代码中都附有详细的注释。

(2)、PSNR、运行时间统计：

见 “结果统计.xlsx”。

(3)、PSNR、运行时间统计：

原图：“lena.bmp”

灰度图：“gray-lena.bmp”

一维 DCT，行变换后结果：“D1_row.bmp”

一维 DCT，先行后列变换后结果：“D1_column.bmp”

二维 DCT 变换结果：“D2_dct.bmp”

二维分块 DCT 变换结果：“D2_with_block_dct.bmp”

一维，压缩系数为 1、1/4、1/16、1/64 的 DCT 变换的逆变换结果位于“一维 DCT 图片结果”文件夹内，命名为：

“D1_m_idct.bmp”,其中 m 为小数表示的压缩系数（压缩系数为 1 时为空)

二维，压缩系数为 1、1/4、1/16、1/64 的 DCT 变换的逆变换结果位于“二维 DCT 图片结果”文件夹内，命名为：

“D2_m_idct.bmp”, 其中 m 为小数表示的压缩系数(压缩系数为 1 时为空)

二维 8*8 分块压缩系数为 1、1/4、1/16、1/64 的 DCT 变换的逆变换结果位于“二维_8_8_DCT 图片结果”文件夹内，命名为：

“D2_with_block_m_idct.bmp”,其中 m 为小数表示的压缩系数(压缩系数为 1 时为空)

【Exp2】

一、实验环境

编程语言：MATLAB

实验平台：MATLAB R2014a on windows 7

二、实验任务

1、将图片进行 8*8 分块，并使用二维 DCT 进行变换。

2、使用量化矩阵 Q 对二维 DCT 的结果进行量化。

3、对量化结果使用二维 IDCT 变换，并计算重构图片的 PSNR。

4、对所有的分块，计算平均 PSNR。

5、使用一个参数 a(0.1<a<2)乘上量化矩阵 Q，并重复上述过程。

6、画出 PSNR-a 曲线图，并解释实验结果。

7、自己尝试设计一个量化矩阵，并说明影响量化矩阵设计的因素。

8、自己选择图片，使用 Cannon 和 Nikon 两种量化矩阵，比较结果与 PSNR。

三、实现说明

本次任务主要要做的还是对矩阵进行分块量化计算 PSNR。计算 PSNR 的函数直接使用了实验一实现的函数。

进行矩阵量化时，量化相关的公式和原理参考了：

http://bbs.csdn.net/topics/310026388

查看这里的资料后得知，矩阵量化的主要公式是：

$$F^{Q}(u,v) = IntegerRound\left[\frac{F(u,v)}{Q(u,v)}\right]$$

其中，F 为经过 DCT 变换后的矩阵，Q 为量化矩阵，IntergerRound 是一个取整函数。关于取整，可以直接用 MATLAB 自带的 round 函数。而对于矩阵出发，可以通过./Q 实现。但是在量化后进行 2D-IDCT 时，我发现如果没有经过一个逆量化，即没有经过一个.*Q 的过程，最后重构得的结果很黑，几乎不能看出 lena 的轮廓，且得到的 PSNR 很小，于是我加上了逆量化的过程。

在对矩阵进行分块计算时，刚开始我还是用的 blcproc 矩阵，即自己实现一个对 8*8 矩阵进行 2D-DCT、量化、逆量化、2D-IDCT、计算 PSNR 的函数。但是最后的出来的结果老是不对，重构的图片老是白白的一片。将最后拼接出来的 IDCT 结果输出，发现几乎所有值都超过 50。刚开始我怀疑，是不是这里不能用 blcproc 函数，或者是不是我的用法出错了。所以我又实现了一种使用 mat2cell 进行 8*8 分块的方法，同样对每个子矩阵进行 2D-DCT、量化、逆量化、2D-IDCT，然后计算 PSNR，但是最终重构结果还是白的，还是不对。通过

观察，我发现这两种实现方法算出来的平均 PSNR 值是相同的，说明这两种方法应该都是可以的。调了好久，最后对得到的结果加上了一个 mat2gray，对矩阵进行一个归一化，结果就正常了。

进行作图时，只需要对每一个系数取值，计算出这种情况下量化的 PSNR 平均值，然后存到数组中，最后用 plot 绘图即可。

自己选择图片时，我选择了之前信号处理课程上老师给过的三张灰度图片，具体图片见实验结果。

四、实验结果

1、对图片进行 8*8 分块，使用量化矩阵 Q 计算每个小块的 PSNR 值，并计算 PSNR 平均值。

每个子块的 PSNR 值记录在“Q_1_data.txt”中。PSNR 平均值为 29.3266。

重构图片结果如下，左边为原图，右边为逆变换重构的图。

2、使用参数 a 进行放缩量化矩阵，并计算平均 PSNR 值，绘制 PSNR-a 曲线，分析实验结果。

本次实验，我总共取了 20 个值，a=0.1,0.2…2，对每种 a 都进行了 PSNR 平均值的计算，并绘制了 PSNR-a 的曲线图。不同 a 的取值下，3 种量化矩阵的

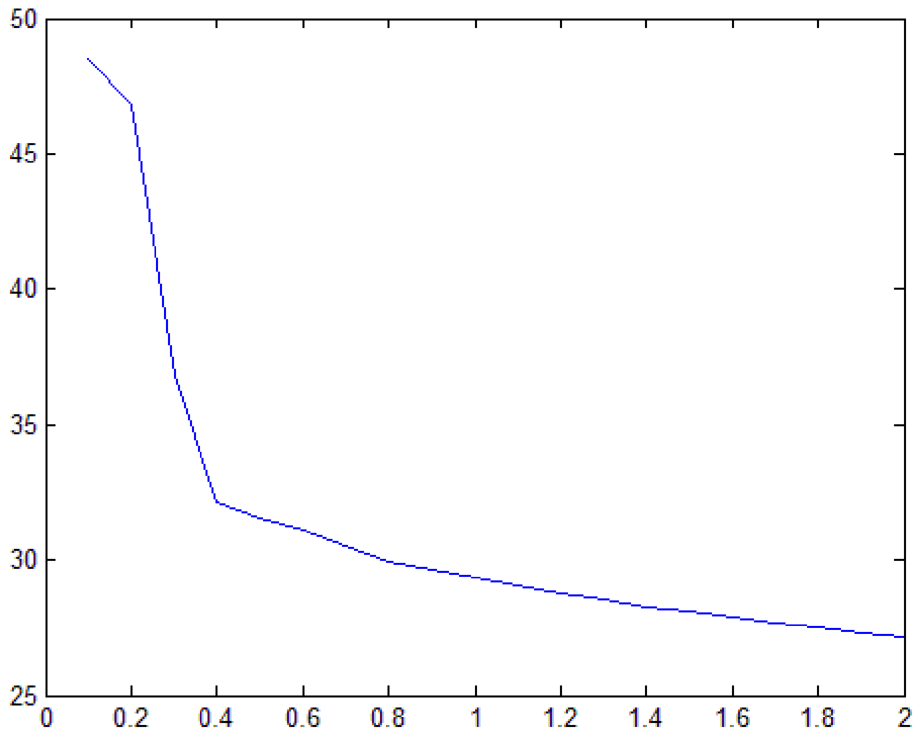
平均 PSNR 结果截图如下。具体数据见 “数据统计.xlsx”。

Name(策略名_a值)	average_PSNR	averge_20
Q_0.1	48.5262	31.32504
Q_0.2	46.7865	
Q_0.3	36.8118	
Q_0.4	32.1483	
Q_0.5	31.5342	
Q_0.6	31.0865	
Q_0.7	30.5029	
Q_0.8	29.9591	
Q_0.9	29.6249	
Q_1	29.3266	
Q_1.1	29.03	
Q_1.2	28.7515	
Q_1.3	28.5121	
Q_1.4	28.2853	
Q_1.5	28.0856	
Q_1.6	27.8857	
Q_1.7	27.6761	
Q_1.8	27.4939	
Q_1.9	27.3271	
Q_2	27.1465	
Cannon_0.1	52.4955	44.511035
Cannon_0.2	50.7522	
Cannon_0.3	49.2474	
Cannon_0.4	47.7973	
Cannon_0.5	47.4716	
Cannon_0.6	46.1882	
Cannon_0.7	47.058	
Cannon_0.8	44.8385	
Cannon_0.9	42.7397	
Cannon_1	42.6051	
Cannon_1.1	43.191	
Cannon_1.2	44.0302	
Cannon_1.3	44.8984	
Cannon_1.4	44.8769	
Cannon_1.5	43.9524	
Cannon_1.6	42.2751	
Cannon_1.7	40.7072	
Cannon_1.8	39.3782	
Cannon_1.9	38.3147	
Cannon_2	37.4031	

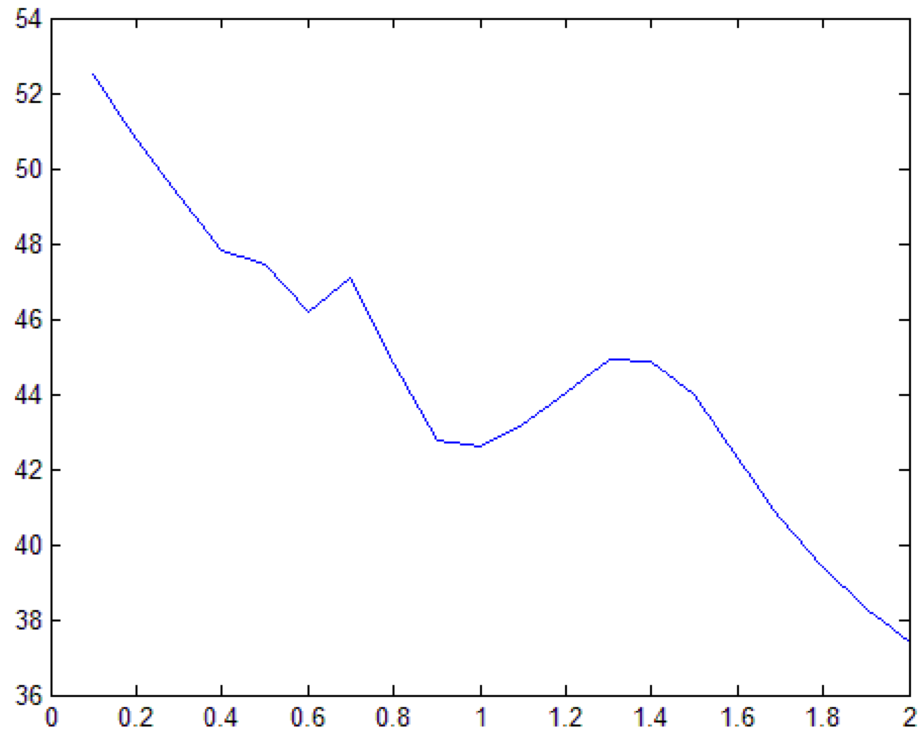
Nikon_0.1	53.3269	45.9245
Nikon_0.2	51.0912	
Nikon_0.3	49.8555	
Nikon_0.4	49.443	
Nikon_0.5	47.9426	
Nikon_0.6	47.4471	
Nikon_0.7	45.9139	
Nikon_0.8	47.7517	
Nikon_0.9	46.1249	
Nikon_1	43.6468	
Nikon_1.1	42.0538	
Nikon_1.2	42.1273	
Nikon_1.3	42.949	
Nikon_1.4	44.0441	
Nikon_1.5	45.3813	
Nikon_1.6	45.9926	
Nikon_1.7	45.4137	
Nikon_1.8	43.9794	
Nikon_1.9	42.6132	
Nikon_2	41.4017	

PSNR 随 a 的变化图如下：

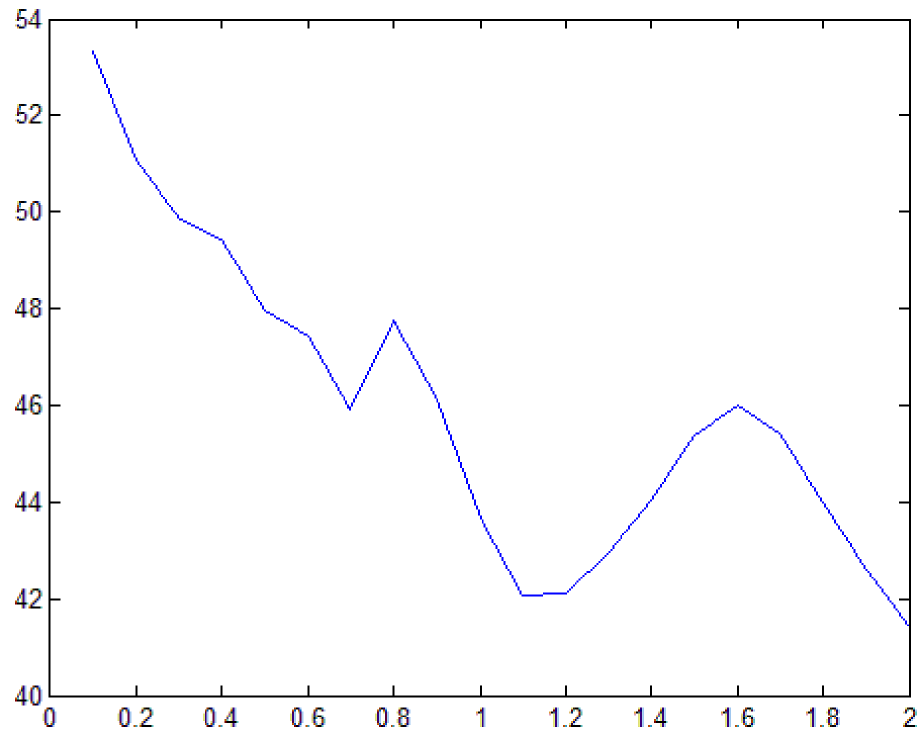
(1) 量化矩阵为 Q 时：



(2) 量化矩阵为 Cannon 时：



(3) 量化矩阵为 Nikon 时：



从 3 张 PSNR-a 的趋势图可以看出，无论量化矩阵是 Q，还是 Cannon，或者是 Nikon，随着 a 的增大，PSNR 值都呈减小的趋势。这是由于随着 a 的增大，a*Q 中的元素不断增大，导致 round(dct2(A)./Q).*Q 的值也不断增大，经过逆变换并归一化后，矩阵的值越接近于 0，丢失的信息越多，导致图片模糊度增大，PSNR 值减小。

但同时也可以看出，只有量化矩阵为 Q 时的 PSNR-a 曲线是单调的，对于 Cannon 或者 Nikon，PSNR 总体上呈减小的趋势，但是在减小过程中还是有一系列的凸值。这可能是受真正处理的图片本身的特性影响。

从三种量化矩阵的曲线图和 PSNR 均值可以看出，相同系数 a 下 Nikon 和 Cannon 的 PNSR 值会比 Q 的 PSNR 值来的大，20 种 a 的取值下 PSNR 的均值也比 Q 的值来的大，说明 Nikon 和 Cannon 量化矩阵的效果比 Q 来的好。而 Nikon 和 Cannon 相比均值差不多，会大一点，效果可能也会好一点。

20 种 a 的值的情况下，三种量化矩阵量化后重构得到的图片，附于”Q 图片结果”、“cannon 图片结果”、“Nikon 图片结果”三个文件夹中。这里贴出原图、a=0.1、1、2 时的结果。

(1) 量化矩阵为 Q 时，从左到右、从上到下依次为：原图、a=0.1、1、2 时的结果




(2) 量化矩阵为 Cannon 时，从左到右、从上到下依次为：原图、a=0.1、1、2 时的结果


(3) 量化矩阵为 Nikon 时，从左到右、从上到下依次为：原图、a=0.1、1、2 时的结果

可见，随着 a 的增大，图片越来越模糊，这和随着 a 增大，PSNR 值不断减

小的结果相符合。

3、说出一些影响量化矩阵设计的因素，并尝试自己给出一个量化矩阵。

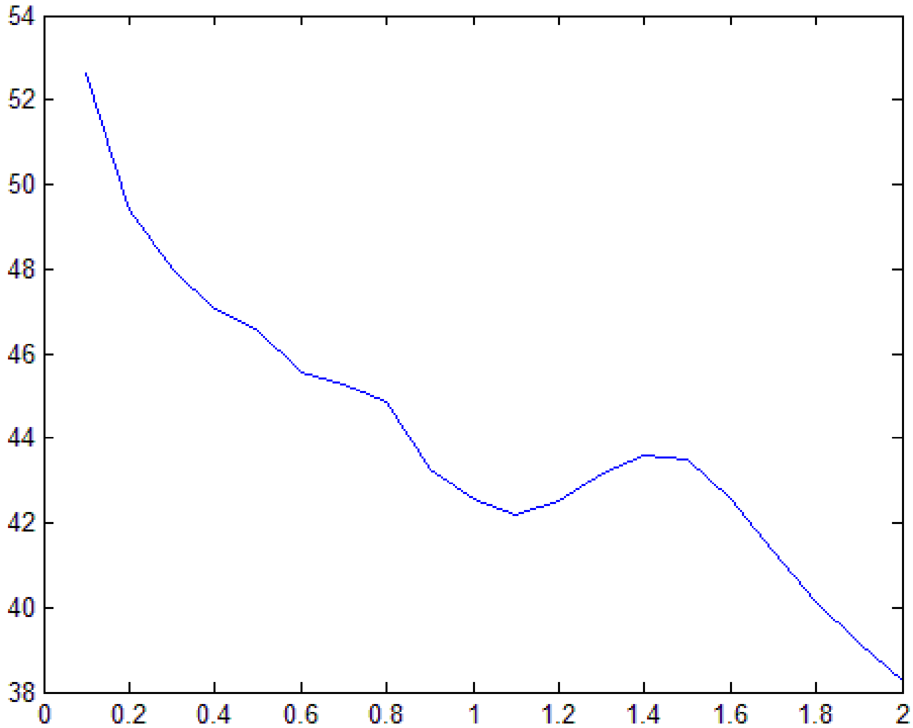
根据上述 3 种量化矩阵的经验，知 a 的值越大时，压缩的效果越好。而 a 的值越大，其实也就是使量化矩阵中的值越大。由于 Cannon 和 Nikon 量化矩阵的效果明显好于 Q，观察 Cannon 和 Nikon 矩阵可以发现，越接近左上角，元素越小；越接近右下角，元素越大。我认为，可能是左上角主要用来保存低频信号，右下角主要用来保存高频信号。由于人眼对低频信号的区分能力要强于对高频信号的区分能力，所以使用量化矩阵进行压缩时，由于图片损失不可避免，故可以尽可能地保存低频信号，而让高频信号作为损失失去。所以，在设计量化矩阵时，可以有选择性地让左上角的元素相对较小，右下角的元素相对较大。

我设计的矩阵如下：

$$\begin{bmatrix} 1 & 1 & 1 & 2 & 3 & 6 & 7 & 8 \\ 1 & 1 & 2 & 3 & 5 & 7 & 9 & 8 \\ 2 & 2 & 2 & 3 & 5 & 8 & 10 & 12 \\ 2 & 2 & 3 & 4 & 6 & 9 & 12 & 12 \\ 3 & 3 & 4 & 8 & 9 & 10 & 11 & 12 \\ 4 & 4 & 5 & 8 & 10 & 12 & 13 & 14 \\ 6 & 8 & 9 & 10 & 12 & 13 & 14 & 15 \\ 9 & 11 & 12 & 13 & 14 & 15 & 16 & 17 \end{bmatrix}$$

最后的结果为：

my_0.1	52.598	44.06975
my_0.2	49.3687	
my_0.3	47.9825	
my_0.4	47.0404	
my_0.5	46.5218	
my_0.6	45.5585	
my_0.7	45.2954	
my_0.8	44.8611	
my_0.9	43.2752	
my_1	42.5469	
my_1.1	42.1798	
my_1.2	42.4974	
my_1.3	43.1526	
my_1.4	43.5675	
my_1.5	43.5039	
my_1.6	42.5594	
my_1.7	41.3651	
my_1.8	40.1517	
my_1.9	39.1268	
my_2	38.2414	



可见，和 Cannon 和 Nikon 相比，PSNR 值略小一些，但是 PSNR-a 曲线下降的趋势更加明显，减少了一个凸峰的出现。

8、自己选择图片，使用 Cannon 和 Nikon 两种量化矩阵，比较结果与 PSNR。

本次实验，我选择了 3 张《信号处理原理》课程中老师给出的经典的图像变化的灰度图，并使用 Cannon 和 Nikon 进行量化，结果如下：

(1)shoot.bmp,从左到右从上到下为原图、Cannon、Nikon

(2)boat.bmp,从左到右从上到下为原图、Cannon、Nikon

(3)Barbara.bmp,从左到右从上到下为原图、Cannon、Nikon

2 种不同的量化矩阵下，3 种图片的 PSNR 值如下：

图片	量化矩阵	PSNR
图片1：shoot.bmp	Cannon	30.7453
	Nikon	63.1245
图片2：boat.bmp	Cannon	95.0878
	Nikon	128.1588
图片3：barbara.bmp	Cannon	161.5432
	Nikon	195.861

可见，无论对哪张图片，用 Nikon 量化的结果的 PSNR 总是会比 Cannon 的结果的 PSNR 来的大，这和用 lena 作为图片时的结论相符合。

五、实验总结

通过本次任务，我了解到，对于同一张图片来说，量化矩阵中的值越大，量化后重构得到的图片计算 PSNR 值就越小，信息丢失得越多。在设计量化矩阵时，由于压缩损失不可避免，我们可以将量化矩阵的左上角设计得小一些，右下角设计得大一下，让高频分量损失，保留低频分量，这样得出的图片结果才会更清晰。

六、实验数据

my_psnr.m：计算两个同纬度矩阵的 PSNR 值

my_2dct.m：使用 blkproc 的方法进行分块，并计算 PSNR 值

my_2dct_use_matcell.m：使用 mat2cell 的方法进行分块，并计算 PSNR

main.m：主脚本文件，实现上述任务

lena.bmp 为原图，gray-lena.bmp 为原灰度图；

Q_1_data.txt 为用 Q 进行量化，所有 8*8 矩阵的 PSNR 值。

Q.bmp 为为用 Q 进行量化再重构后得到的图片。

“数据统计.xlsx”中保存了 Q、Cannon、Nikon 以及自己实现的量化矩阵，对于 a=[0.1，2]的情况下的 PSNR 值，以及自己选择的图片，使用 Cannon 和 Nikon 量化的 PSNR 值。

a=[0.1，2]，3 中量化矩阵得到的图片位于“Q 图片结果”、“Cannon 图片结果”、“Nikon 图片结果”文件夹内。Q、Cannon、Nikon、自己实现的量化矩阵得到的 PSNR-a 曲线，位于“PSNR_a 曲线”文件夹内。

Shoot.bmp，boat.bmp，barbara.bmp 为自己选择的图片，使用 Nikon 和 Cannon 量化并重构得到的图片位于“自己选择图片量化重构结果”文件夹内。