

## 问题三十一至四十

### 问题三十一：仿射变换 (Affine Transformations) —— 倾斜

1. 使用仿射变换，输出 (1) 那样的 $x$ 轴倾斜30度的图像 ( $t_x = 30$ )，这种变换被称为X-sharing。
2. 使用仿射变换，输出 (2) 那样的 $y$ 轴倾斜30度的图像 ( $t_y = 30$ )，这种变换被称为Y-sharing。
3. 使用仿射变换，输出 (3) 那样的 $x$ 轴、 $y$ 轴都倾斜30度的图像( $t_x = 30, t_y = 30$ )。

原图像的大小为 $h \cdot w$ ，使用下面各式进行仿射变换：

- X-sharing

$$a = \frac{t_x}{h}$$
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Y-sharing

$$a = \frac{t_y}{w}$$
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ a & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

输入 (imori.jpg)	输出 (1) (answers_image/answer_31_1.jpg)	输出 (2) (answers_image/answer_31_2.jpg)	输出 (3) (answers_image/answer_31_3.jpg)
			

答案

- Python >> [answers\\_py/answer\\_31.py](#)
- C++ >> [answers\\_cpp/answer\\_31.cpp](#)

### 问题三十二：傅立叶变换 (Fourier Transform)

使用离散二维傅立叶变换 (Discrete Fourier Transformation)，将灰度化的 `imori.jpg` 表示为频谱图。然后用二维离散傅立叶逆变换将图像复原。

二维离散傅立叶变换是傅立叶变换在图像处理上的应用方法。通常傅立叶变换用于分离模拟信号或音频等连续一维信号的频率。但是，数字图像使用 $[0, 255]$ 范围内的离散值表示，并且图像使用 $H \times W$ 的二维矩阵表示，所以在这里使用二维离散傅立叶变换。

二维离散傅立叶变换使用下式计算，其中 $I$ 表示输入图像：

$$G(k, l) = \frac{1}{H \cdot W} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} I(x, y) e^{-2\pi \cdot j \cdot (\frac{kx}{W} + \frac{ly}{H})}$$

在这里让图像灰度化后，再进行离散二维傅立叶变换。

频谱图为了能表示复数 $G$ ，所以图上所画长度为 $G$ 的绝对值。这回的图像表示时，请将频谱图缩放至 $[0, 255]$ 范围。

二维离散傅立叶逆变换从频率分量 $G$ 按照下式复原图像：

$$I(x, y) = \frac{1}{H \cdot W} \sum_{l=0}^{H-1} \sum_{k=0}^{W-1} G(l, k) e^{2\pi \cdot j \cdot (\frac{k \cdot x}{W} + \frac{l \cdot y}{H})}$$

上が定義式ですが $\exp(j)$ は複素数の値をとってしまうので、実際にコードにするときはぜ下式のように絶対値を使います。

上式中 $\exp(j)$ 是个复数，实际编程的时候请务必使用下式中的绝对值形态：

这里应该有个公式的，但是它不知道去哪儿了。

——gZR

如果只是简单地使用 `for` 语句的话，计算量达到 $128^4$ ，十分耗时。如果善用 `NumPy` 的化，则可以减少计算量（答案中已经减少到 $128^2$ ）。

输入 (imori.jpg)	灰度化 (imori_gray.jpg)	输出 (answers_image/answer_32.jpg)	频谱图 (answers_image/answer_32_ps.py)
			

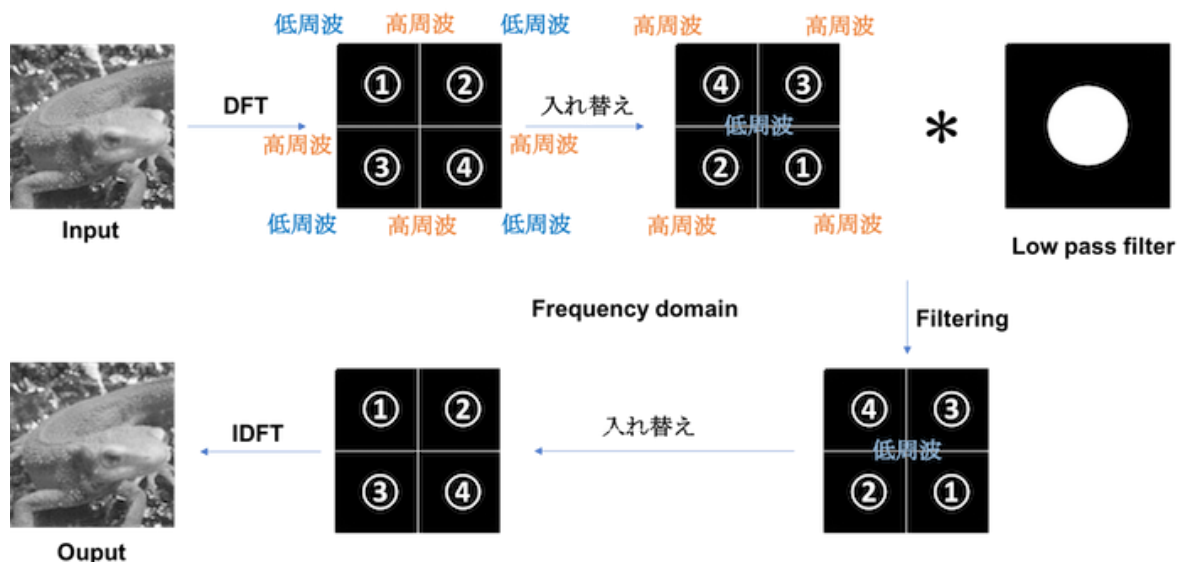
答案

- Python >> [answers\\_py/answer\\_32.py](#)
- C++ >> [answers\\_cpp/answer\\_32.cpp](#)

## 问题三十三：傅立叶变换——低通滤波

将 `imori.jpg` 灰度化之后进行傅立叶变换并进行低通滤波，之后再用傅立叶逆变换复原吧！

通过离散傅立叶变换得到的频率在左上、右上、左下、右下等地方频率较低，在中心位置频率较高。





上面图里的文字意思是：

- 高周波=高频
- 低周波=低频
- 入れ替え=替换

——gZR

在图像中，高频成分指的是颜色改变的地方（噪声或者轮廓等），低频成分指的是颜色不怎么改变的部分（比如落日的渐变）。在这里，使用去除高频成分，保留低频成分的**低通滤波器**吧！

在这里，假设从低频的中心到高频的距离为 $r$ ，我们保留 $0.5 \cdot r$ 的低频分量。

输入 (imori.jpg)	输出(answers_image/answer_33.jpg)
	



答案

- Python >> [answers\\_py/answer\\_33.py](#)
- C++ >> [answers\\_cpp/answer\\_33.cpp](#)

## 问题三十四：傅立叶变换——高通滤波

将 `imori.jpg` 灰度化之后进行傅立叶变换并进行高通滤波，之后再用傅立叶逆变换复原吧！

在这里，我们使用可以去除低频部分，只保留高频部分的**高通滤波器**。假设从低频的中心到高频的距离为 $r$ ，我们保留 $0.2 \cdot r$ 的低频分量。

输入 (imori.jpg)	输出(answers_image/answer_34.jpg)
	



答案

- Python >> [answers\\_py/answer\\_34.py](#)
- C++ >> [answers\\_cpp/answer\\_34.cpp](#)

## 问题三十五：傅立叶变换——带通滤波

将 `imori.jpg` 灰度化之后进行傅立叶变换并进行带通滤波，之后再用傅立叶逆变换复原吧！

在这里，我们使用可以保留介于低频成分和高频成分之间的分量的**带通滤波器**。在这里，我们使用可以去除低频部分，只保留高频部分的高通滤波器。假设从低频的中心到高频的距离为 $r$ ，我们保留 $0.1 \cdot r$ 至 $0.5 \cdot r$ 的分量。

输入 (imori.jpg)	输出(answers_image/answer_34.jpg)
	

答案

- Python >> [answers\\_py/answer\\_35.py](#)
- C++ >> [answers\\_cpp/answer\\_35.cpp](#)

## 问题三十六: JPEG 压缩——第一步: 离散余弦变换 (Discrete Cosine Transformation)

`imori.jpg` 灰度化之后, 先进行离散余弦变换, 再进行离散余弦逆变换吧!

离散余弦变换 (Discrete Cosine Transformation) 是一种使用下面式子计算的频率变换:

$$0 \leq u, v \leq T$$

$$F(u, v) = \frac{2}{T} \cdot C(u) \cdot C(v) \sum_{y=0}^{T-1} \sum_{x=0}^{T-1} I(x, y) \cdot \cos\left(\frac{(2 \cdot x + 1) \cdot u \cdot \pi}{2 \cdot T}\right) \cdot \cos\left(\frac{(2 \cdot y + 1) \cdot v \cdot \pi}{2 \cdot T}\right)$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & (\text{if } u = 0) \\ 1 & (\text{else}) \end{cases}$$

离散余弦逆变换 (Inverse Discrete Cosine Transformation) 是离散余弦变换的逆变换, 使用下式定义。

在这里,  $K$  是决定图像复原时分辨率高低的参数。  $K = T$  时, DCT 的系数全被保留, 因此 IDCT 时分辨率最大。  $K = 1$  或  $K = 2$  时, 图像复原时的信息量 (DCT 系数) 减少, 分辨率降低。如果适当地设定  $K$ , 可以减小文件大小。

$$1 \leq K \leq T$$

$$f(x, y) = \frac{2}{T} \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} C(u) C(v) F(u, v) \cos\left(\frac{(2 \cdot x + 1) \cdot u \cdot \pi}{2 \cdot T}\right) \cos\left(\frac{(2 \cdot y + 1) \cdot v \cdot \pi}{2 \cdot T}\right)$$



$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & (\text{if } u = 0) \\ 1 & (\text{else}) \end{cases}$$

在这里我们先将图像分割成  $8 \times 8$  的小块, 在各个小块中使用离散余弦变换编码, 使用离散余弦逆变换解码, 这就是 JPEG 的编码过程。现在我们也同样地, 把图像分割成  $8 \times 8$  的小块, 然后进行离散余弦变换和离散余弦逆变换。

这一整段我整体都在瞎\*\*译, 原文如下:

ここでは画像を  $8 \times 8$  ずつの領域に分割して、各領域で以上の DCT, IDCT を繰り返すことで、JPEG 符号に応用される。今回も同様に  $8 \times 8$  の領域に分割して、DCT, IDCT を行え。

——gZR

输入 (imori.jpg)	输出(answers_image/answer_36.jpg)
	

答案 >> [answers/answer\\_36.py](#)

## 问题三十七： PSNR

离散余弦逆变换中如果不使用8作为系数，而是使用4作为系数的话，图像的画质会变差。来求输入图像和经过离散余弦逆变换之后的图像的峰值信噪比吧！再求出离散余弦逆变换的比特率吧！

峰值信噪比（Peak Signal to Noise Ratio）缩写为PSNR，用来表示信号最大可能功率和影响它的表示精度的破坏性噪声功率的比值，可以显示图像画质损失的程度。

峰值信噪比越大，表示画质损失越小。峰值信噪比通过下式定义。MAX表示图像点颜色的最大数值。如果取值范围是[0, 255]的话，那么MAX的值就为255。MSE表示均方误差（Mean Squared Error），用来表示两个图像各个像素点之间差值平方和的平均数：

$$\text{PSNR} = 10 \cdot \log_{10} \frac{v_{max}^2}{\text{MSE}}$$

$$\text{MSE} = \frac{\sum_{y=0}^{H-1} \sum_{x=0}^{W-1} [I_1(x, y) - I_2(x, y)]^2}{H \cdot W}$$

如果我们进行 $8 \times 8$ 的离散余弦变换，离散余弦逆变换的系数为 $K \text{ times } K$ 的话，比特率按下式定义：

$$\text{bit rate} = 8 \cdot \frac{K^2}{8^2}$$

输入 (imori.jpg)	输出 (answers_image/answer_37.jpg) (PSNR = 27.62, Bitrate=2.0)
	

答案

- Python >> [answers\\_py/answer\\_37.py](#)
- C++ >> [answers\\_cpp/answer\\_37.cpp](#)

## 问题三十八：： JPEG 压缩——第二步：离散余弦变换+量化

量化离散余弦变换系数并使用 离散余弦逆变换恢复。再比较变换前后图片的大小。





量化离散余弦变换系数是用于编码 JPEG 图像的技术。

量化即在对值在预定义的区间内舍入，其中 `floor`、`ceil`、`round` 等是类似的计算。

在 JPEG 图像中，根据下面所示的量化矩阵量化离散余弦变换系数。该量化矩阵取自 JPEG 软件开发联合会组织颁布的标准量化表。在量化中，将  $8 \times 8$  的系数除以（量化矩阵） $Q$  并四舍五入。之后然后再乘以  $Q$ 。对于离散余弦逆变换，应使用所有系数。

```
1 Q = np.array(((16, 11, 10, 16, 24, 40, 51, 61),
2               (12, 12, 14, 19, 26, 58, 60, 55),
3               (14, 13, 16, 24, 40, 57, 69, 56),
4               (14, 17, 22, 29, 51, 87, 80, 62),
5               (18, 22, 37, 56, 68, 109, 103, 77),
6               (24, 35, 55, 64, 81, 104, 113, 92),
7               (49, 64, 78, 87, 103, 121, 120, 101),
8               (72, 92, 95, 98, 112, 100, 103, 99))), dtype=np.float32)
```

由于量化降低了图像的大小，因此可以看出数据量已经减少。

输入 (imori.jpg)	出力 (answers_image/answer_38.jpg) (7kb)
	

答案

- Python >> [answers\\_py/answer\\_38.py](#)
- C++ >> [answers\\_cpp/answer\\_38.cpp](#)

## 问题三十九：JPEG 压缩——第三步：YCbCr 色彩空间

在 YCbCr 色彩空间内，将  $Y$  乘以 0.7 以使对比度变暗。

YCbCr 色彩空间是用于将图像由表示亮度的  $Y$ 、表示蓝色色度  $Cb$  以及表示红色色度  $Cr$  表示的方法。



这用于 JPEG 转换。

使用下式从 RGB 转换到 YCbCr：

$$\begin{aligned} Y &= 0.299 \cdot R + 0.5870 \cdot G + 0.114 \cdot B \\ Cb &= -0.1687 \cdot R - 0.3313 \cdot G + 0.5 \cdot B + 128 \\ Cr &= 0.5 \cdot R - 0.4187 \cdot G - 0.0813 \cdot B + 128 \end{aligned}$$

使用下式从 YCbCr 转到 RGB：

$$\begin{aligned} R &= Y + (Cr - 128) \cdot 1.402 \\ G &= Y - (Cb - 128) \cdot 0.3441 - (Cr - 128) \cdot 0.7139 \\ B &= Y + (Cb - 128) \cdot 1.7718 \end{aligned}$$

入力 (imori.jpg)	出力 (answers_image/answer_39.jpg)
	

答案

- Python >> [answers\\_py/answer\\_39.py](#).
- C++ >> [answers\\_cpp/answer\\_39.cpp](#)

## 问题四十：JPEG 压缩——第四步：YCbCr+离散余弦变换+量化

将图像转为 YCbCr 色彩空间之后，进行 离散余弦变换再对 Y 用 Q1 量化矩阵量化，Cb 和 Cr 用 Q2 量化矩阵量化。最后通过离散余弦逆变换对图像复原。还需比较图像的容量。算法如下：



1. 将图像从RGB色彩空间变换到YCbCr色彩空间；
2. 对YCbCr做DCT；
3. DCT之后做量化；
4. 量化之后应用IDCT；
5. IDCT之后从YCbCr色彩空间变换到RGB色彩空间。

这是实际生活中使用的减少 JPEG 数据量的方法，Q1 和 Q2 根据 JPEG 规范由以下等式定义：

```

1  Q1 = np.array(((16, 11, 10, 16, 24, 40, 51, 61),
2                  (12, 12, 14, 19, 26, 58, 60, 55),
3                  (14, 13, 16, 24, 40, 57, 69, 56),
4                  (14, 17, 22, 29, 51, 87, 80, 62),
5                  (18, 22, 37, 56, 68, 109, 103, 77),
6                  (24, 35, 55, 64, 81, 104, 113, 92),
7                  (49, 64, 78, 87, 103, 121, 120, 101),
8                  (72, 92, 95, 98, 112, 100, 103, 99))), dtype=np.float32)
9
10 Q2 = np.array(((17, 18, 24, 47, 99, 99, 99, 99),
11                 (18, 21, 26, 66, 99, 99, 99, 99),
12                 (24, 26, 56, 99, 99, 99, 99, 99),
13                 (47, 66, 99, 99, 99, 99, 99, 99),
14                 (99, 99, 99, 99, 99, 99, 99, 99),
15                 (99, 99, 99, 99, 99, 99, 99, 99),
16                 (99, 99, 99, 99, 99, 99, 99, 99),
17                 (99, 99, 99, 99, 99, 99, 99, 99))), dtype=np.float32)

```

输入 (imori.jpg) (13kb)	输出 (answers/answer_40.jpg) (8kb)
	

#### 答案

- Python >> [answers\\_py/answer\\_40.py](#).
- C++ >> [answers\\_cpp/answer\\_40.cpp](#)