フーリエ解析　期末課題

離散コサイン変換の応用例，jpeg 変換

機械科学•航空宇宙学科　　1w192224　田久健人

2021 年 12 月 31 日

目　　次

1　目的

普段我々が身近に利用している DCT 変換の一種である jpeg について，DTC 変換の理論背景と，理論背景に基づいた簡易的な二次元配列の変換の実装の二つの観点から理解を深める．

2　DCT の理論

2.1　DFT

DCT(離散コサイン変換) を理解するにあたり，まず DFT(離散フーリエ変換) の基礎理論を整理する．
波形 $f(t)$ を周期 τ でサンプリングすると，$f(n\tau)$ と表せる．
このとき，周期関数 $f(t)$ は,

$$f_\tau(t) = \sum_{n=0}^{N-1} f(n\tau)\delta(t - n\tau) \tag{1}$$

と表せる．ここで，(1) に Fourier 変換を適用すると,

$$\begin{aligned} F_\tau(f) &= \int_{-\infty}^{\infty} f_\tau(t)^{-j\omega t} dt \\ &= \int_{-\infty}^{\infty} \sum_{n=0}^{N-1} f(n\tau)\delta(t - n\tau)^{-j\omega t} dt \\ &= \sum_{n=0}^{N-1} f(n\tau) e^{-j\omega n\tau} \end{aligned} \tag{2}$$

$\omega = \frac{2\pi k}{n\tau}$，$k = 0, 1, ..., N-1$ より,

$$F_\tau(k) = \sum_{n=0}^{N-1} W_N^{nk} \tag{3}$$

$$(W_N = e^{j\frac{1\pi}{N}})$$

2.2　DCT

f_τ を，中央線対象のデータに拡張する．$f'_\tau = f_\tau(0), f_\tau(1), ..., f_\tau(N-2), f_\tau(N-1), f_\tau(N-1), f_\tau(N-2), ..., f_\tau(1), f_\tau(0)$ のように定義する．
f'_τ の DFT に対し，$w(k) = e^{-j\frac{\pi}{2N}k}$ をかけ，式を展開する．

$$\begin{aligned} F'_\tau(k)w(k) &= \sum_{n=0}^{2N-1} f'_\tau(n) e^{-j\frac{\pi}{N}nk} e^{-j\frac{\pi}{2N}k} \\ &= \sum_{n=0}^{N-1} f'_\tau(n) e^{-j\frac{\pi}{2N}(2n+1)k} + \sum_{n=N}^{2N-1} f'_\tau(n) e^{-j\frac{\pi}{N}(2n+1)k} \end{aligned} \tag{4}$$

f'_τ の対称性より，

$$\sum_{n=N}^{2N-1} f'_\tau(n) e^{-j\frac{\pi}{2N}(2n+1)k} = \sum_{n=0}^{N-1} f'_\tau(n) e^{j\frac{\pi}{2N}(2n+1)k} \tag{5}$$

よって，(4) は，

$$\begin{aligned} F_\tau(k)w(k) &= \sum_{n=0}^{N-1} f'_\tau(n) e^{-j\frac{\pi}{2N}(2n+1)k} + \sum_{n=0}^{N-1} f'_\tau(n) e^{j\frac{\pi}{2N}(2n+1)k} \\ &= \sum_{n=0}^{N-1} f'_\tau(n) \left\{ e^{-j\frac{\pi}{2N}(2n+1)k} + e^{j\frac{\pi}{2N}(2n+1)k} \right\} \\ &= 2\sum_{n=0}^{N-1} f'_\tau(n) \cos\left(\frac{\pi}{2N}(2n+1)k\right) \end{aligned} \tag{6}$$

$N = 0, 1, ..., N-1$ で $f'_\tau = f_\tau$ より，

$$F_\tau(k)w(k) = 2\sum_{n=0}^{N-1} f_\tau(n) \cos\left(\frac{\pi}{2N}(2n+1)k\right) \tag{7}$$

このような，データを 1/2 で右にシフトして，前半分を離散フーリエ変換したものは，離散コサイン変換の中でも．DCT-II と呼ばれる．DCT には他にも複数種類存在するが，その中でも DCT-II は信号の圧縮分野で多く用いられる．

3　画像圧縮の実装

3.1　二次元の対象への応用

(7) で示した DCT を，画像に対して利用することを考える．

3.1.1　二次元の DFT

まず，DFT を二次元に拡張し，行列表示することを考える．フーリエ変換の次元を増やして，

$$F_\tau(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_\tau(x,y) e^{-j2\pi(ux+vy)} dxdy \tag{8}$$

指数関数の性質から，

$$F_\tau(u,v) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f_\tau(x,y) e^{-j2\pi ux} dx \right] e^{-j2\pi vy} dy \tag{9}$$

のように分離可能である．これを離散化して，

$$F_\tau(u,v) = \sum_{n=0}^{N-1} f(x,y) e^{-j2\pi\frac{u}{N}x} \sum_{n=0}^{N-1} e^{-j2\pi\frac{v}{N}y} \tag{10}$$

行列表示を考える．

変換対象を f，DFC の行列表示を R，変換後を F とすると，$F = Rf$ と表す．R は，

$$R = \frac{1}{N}\begin{pmatrix} W_N^0 & W_N^0 & W_N^0 & \dots & W_N^0 \\ W_N^0 & W_N^{-1} & W_N^{-2} & \dots & W_N^{-(N-1)} \\ W_N^0 & W_N^{-2} & W_N^{-4} & \dots & W_N^{-2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ W_N^0 & W_N^{-(N-1)} & W_N^{-2(N-1)} & \dots & W_N^{-(N-1)(N-1)} \end{pmatrix} \tag{11}$$

このとき，$W_N = e^{-j\frac{2\pi}{N}}$ とおいている．

rank$R = N$ より，R は正則である．

逆変換は，R が正則であることから，R の逆行列を左からかけることに相当する．

3.1.2 二次元の DCT

同様の手順で DCT を行列で表現する．

DTC の行列表示 R は,

$$R = \begin{pmatrix} \ddots & & \\ & \phi(k,i) & \\ & & \ddots \end{pmatrix} \tag{12}$$

$$\phi(k,i) = \begin{cases} \frac{1}{\sqrt{N}} & (k=0) \\ \sqrt{\frac{2}{N}}\cos\frac{(2i+1)k\pi}{2N} & (k=1,2,...,N-1) \end{cases}$$

である．DCT では複素数だった部分が実部のみで表されている．

3.2 二次元配列に対する DCT の実装

(12) までの理論式を，python で愚直に実装する．

ソース全体は `https://github.com/tkyawa/fourier-report/tree/main/src` を別途参照する．

まず，基底関数 (12) を表現する．

Listing 1 dct.phi

```
def phi(k, N):
    if k == 0:
        return np.ones(N)/np.sqrt(N)
    else:
        return np.sqrt(2.0/N)*np.cos((k*np.pi/(2*N))*(np.arange(N)*2+1))
```

次に，基底関数を用いて二次元の基底ベクトルを出力する関数を用意する．

Listing 2　dct.base

```
def base(N):
    phi_1 = np.array([phi(i,N) for i in range(N)])
    phi_2 = np.zeros((N, N, N, N))
    for i in range(N):
        for j in range(N):
            phi_i, phi_j = np.meshgrid(phi_1[i], phi_1[j])
            phi_2[i, j] = phi_i*phi_j
    return phi_2
```

基底関数を使って一次元の基底ベクトル phi_1 を定義した後，一次元の基底ベクトル 2 つを格子状に配置して，二次元の基底ベクトルとした．

次に，基底ベクトルを用いて DCT を行う関数を実装する．

Listing 3　dct.dct

```
def dct(original_data, N):
    return np.sum(base(N).reshape(N*N, N*N)*original_data.reshape(N*N), axis=1).reshape
        (N, N)
```

基底ベクトル base(N) に元のデータをかけあわせている．

次に，逆変換を行う関数を実装する．

Listing 4　dct.idct

```
def idct(tranced_data, N):
    return np.sum((tranced_data.reshape(N, N, 1)*base(N).reshape(N,N, N*N)).reshape(N*N
        , N*N), axis=0).reshape(N, N)
```

必要な関数群が揃ったので，テストを行う．今回は，対象がただの二次元配列なのに加え，愚直に実装した影響でサイズ N が 30 程度しか取れないため，手動でテスト画像に変わる行列を用意する．

Listing 5　main.py

```
N = 10
original_img = np.array([
        [0,0,0,0,0,0,0,0,0,0],
        [0,0,0,1,1,1,1,0,0,0],
        [0,0,1,1,0,0,1,1,0,0],
        [0,1,1,1,0,0,1,1,1,0],
        [0,1,0,0,1,1,0,0,1,0],
        [0,1,0,0,1,1,0,0,1,0],
        [0,1,1,1,0,0,1,1,1,0],
        [0,0,1,1,0,0,1,1,0,0],
        [0,0,0,1,1,1,1,0,0,0],
        [0,0,0,0,0,0,0,0,0,0]
])
base = dct.base(N)
transed_img = dct.dct(original_img,N)
resorted_img = dct.idct(transed_img, N)
```

サイズ N を 10 と定めた後，手動で 10×10 のテスト行列 original_img を用意し，先に用意した各関数に値を与えて，original_img に対し DCT，逆変換をする．最後に，変換前，変換後の行列に対し，PyPlot の imshow によって画像形式で表示し，見た目を比較する．

変換前の img，逆変換後の img を図 1 に示す．

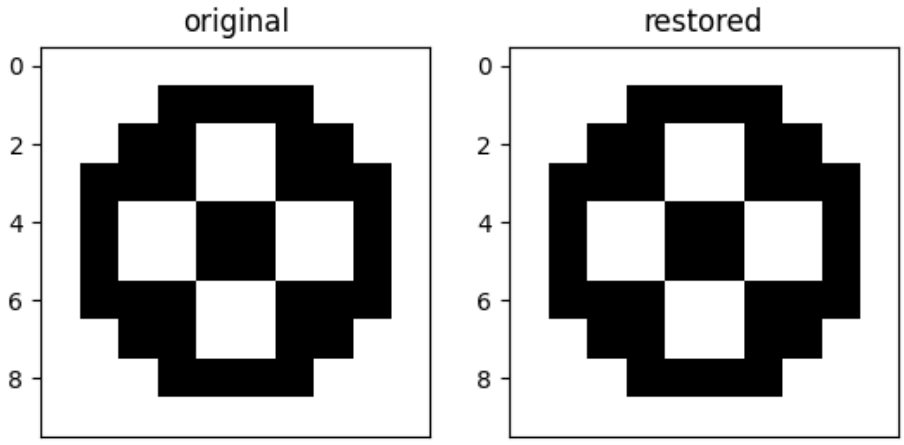

図 1　テスト実行結果

変換前に対して目視では違いがわからない程度の画像が得られているのがわかる．

また，N=10 の場合の基底ベクトル phi_2d の imshow() による出力を図 2 に示す．

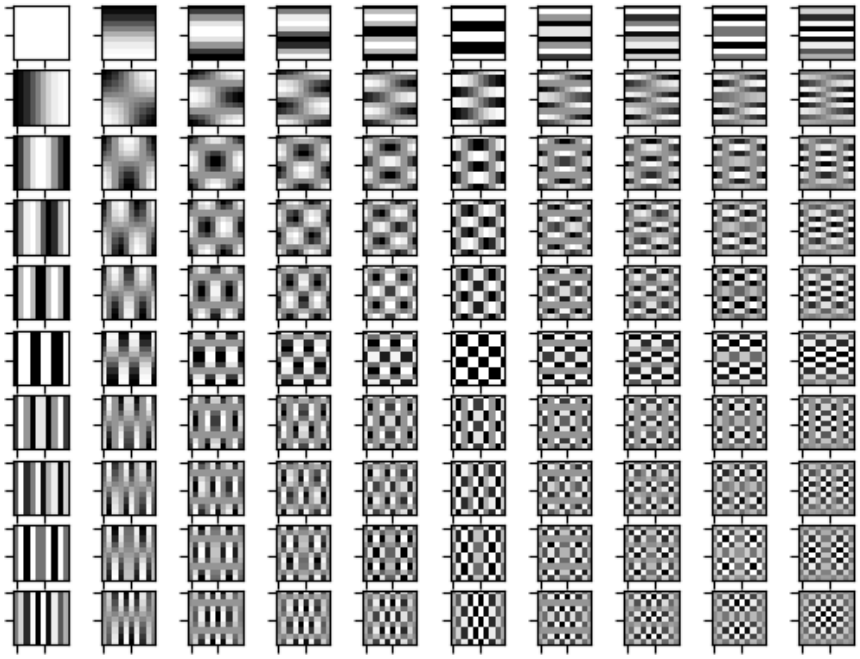

図 2　基底ベクトル (N=10)

入力画像を，これらの画像 (基底ベクトル) の重ね合わせで表現している．これにより，実際のピクセルデータと比べて少ないデータ量で画像を評伝できることが期待される．

3.3 実際の jpeg 変換

先の実装では，が，実際の画像は座標データに加えて色を表すデータも存在するため，もう少し複雑になる．図 3 に，簡単な流れを示す．

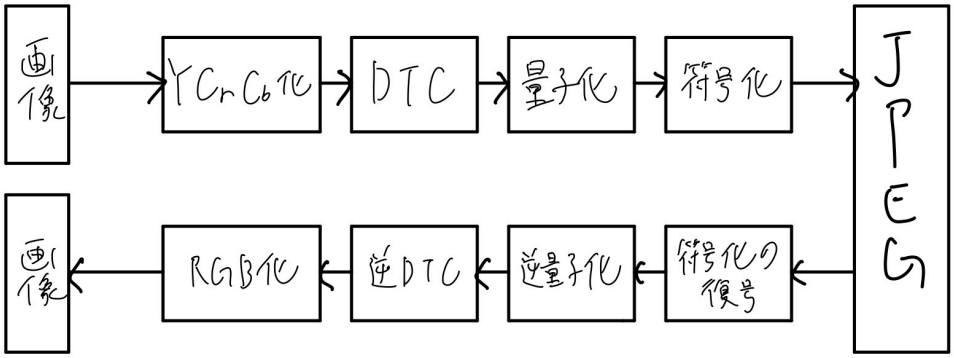

図 3　jpeg 変換のフロー

3.3.1 RGB データの YC_rC_b 形式への変換

まず，元の画像を，RGB 形式よりも処理が容易な YC_rC_b 形式に変換する．このとき，

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B - 128 \\ C_b &= -0.1687R - 0.3313G + 0.5B + 128 \\ C_r &= 0.5R - 0.4187G - 0.0813B + 128 \end{aligned} \tag{13}$$

の公式を用いることができる．

3.3.2 YC_rC_b 形式のデータのサンプリング

データ削減のために，色素情報を 1 ピクセルおきに間引きする．輝度の情報のブロックに対する赤方向，青方向の情報ブロックの数の割合を 4 : 1 : 1 とする．

3.3.3 DCT

DCT を，8 ピクセル四方単位で行う．

3.3.4 データの量子化

DCT によって出た細かい誤差を切り捨て，あらかじめ与えられる量子化テーブルをもとに周波数成分を 64 段階に分ける．jpeg 画像におけるボヤけの大半は，この量子化が原因である．

3.3.5 データの整列

変換後の 8×8 のデータを一列に並べ直す．行列の 0 番成分を基点とし，ジグザグスキャンと呼ばれる規則により一列にする．

3.3.6 データの符号化

エントロピー符号化をすることにより，データを圧縮する．

3.3.7 データの復号

符号化に用いた符号表を用いて復号，量子化に用いた量子テーブルをもとに逆量子化を行う．その後，逆DCT を行い，もとのデータを取り出す．

4 文献

1) 大石進一，”フーリエ解析”，(2007) 岩波書店

2) https://www.onosokki.co.jp/HP-WK/eMM_back/emm138.pdf
最終アクセス:2021 年 12 月 29 日

3) https://www.marguerite.jp/Nihongo/Labo/Image/JPEG.html
最終アクセス:2021 年 12 月 30 日

4) http://racco.mikeneko.jp/Kougi/2016a/IPPR/2016a_ippr09_slide_ho.pdf
最終アクセス:2021 年 12 月 30 日

5) https://tony-mooori.blogspot.com/2016/02/dctpythonpython.html
最終アクセス:2021 年 12 月 30 日

6) 酒井幸市，”画像処理とパターン認識入門”，(2006)，共立出版