

Q. 31 - 40

Q.31. アフィン変換(スキュー)

(1)アフィン変換を用いて、出力(1)のようなX-sharing(dx = 30)画像を作成せよ。

(2)アフィン変換を用いて、出力2のようなY-sharing(dy = 30)画像を作成せよ。

(3)アフィン変換を用いて、出力3のような幾何変換した(dx = 30, dy = 30)画像を作成せよ。

このような画像はスキュー画像と呼ばれ、画像を斜め方向に伸ばした画像である。

出力(1)の場合、x方向にdxだけ引き伸ばした画像はX-sharingと呼ばれる。

出力(2)の場合、y方向にdyだけ引き伸ばした画像はY-sharingと呼ばれる。


それぞれ次式のアフィン変換で実現できる。ただし、元画像のサイズがh x wとする。

(1) X-sharing
a = dx / h

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

(2) Y-sharing
a = dy / w

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ a & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

入力	出力 (1)	出力 (2)	出力 (3)
(imori.jpg)	(answers_image/answer_31_1.jpg)	(answers_image/answer_31_2.jpg)	(answers_image/answer_31_3.jpg)
			

答え

- Python >> [answers_py/answer_31.py](#)
- C++ >> [answers_cpp/answer_31.cpp](#)

Q.32. フーリエ変換

二次元離散フーリエ変換(DFT)を実装し、*imori.jpg*をグレースケール化したものの周波数のパワースペクトルを表示せよ。また、逆二次元離散フーリエ変換(IDFT)で画像を復元せよ。

二次元離散フーリエ変換(DFT: Discrete Fourier Transformation)とはフーリエ変換の画像に対する処理方法である。

通常のフーリエ変換はアナログ信号や音声などの連続値かつ一次元を対象に周波数成分を求める計算処理である。

一方、デジタル画像は[0,255]の離散値をとり、かつ画像はHxWの二次元表示であるので、二次元離散フーリエ変換が行われる。

二次元離散フーリエ変換(DFT)は次式で計算される。

$K = [0, W-1]$, $l = [0, H-1]$, 入力画像を I として

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

ここでは画像をグレースケール化してから二次元離散フーリエ変換を行え。

パワースペクトルとはGは複素数で表されるので、Gの絶対値を求めることである。今回のみ画像表示の時はパワースペクトルは[0,255]にスケールリングせよ。

逆二次元離散フーリエ変換(IDFT: Inverse DFT)とは周波数成分Gから元の画像を復元する手法であり、次式で定義される。

x = [0, W-1], y = [0, H-1] として

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

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。



シンプルに全部for文で回すと128^4の計算になるので、時間がかかってしまいます。numpyをうまく活用すれば計算コストを減らすことができます。（解答は128^2まで減らしました。）

入力 (imori.jpg) 出力 (answers_image/answer_32.jpg) パワースペクトル (answers_image/answer_32_ps.py)



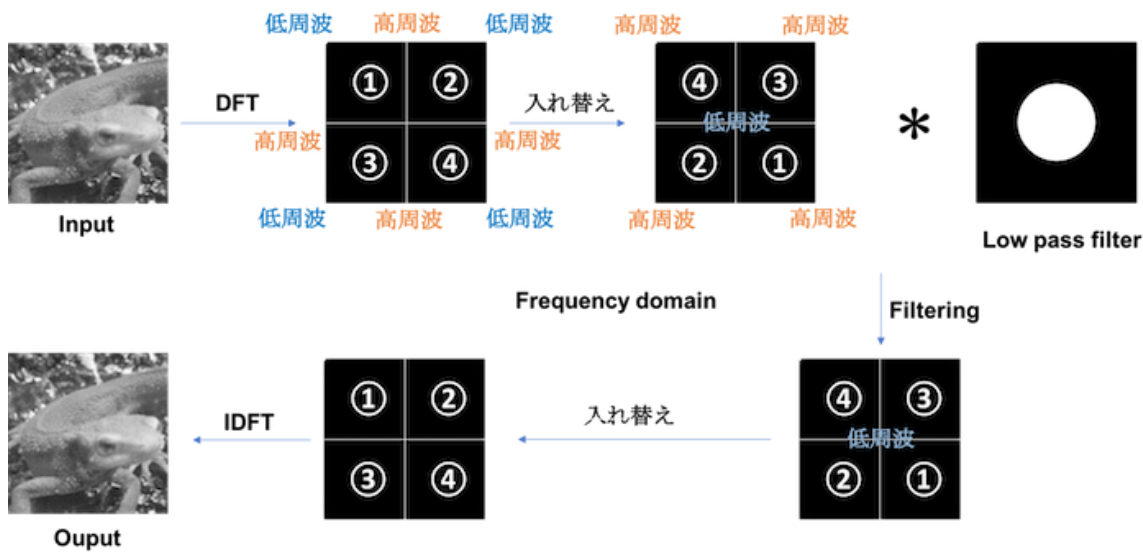
答え

- Python >> [answers_py/answer_32.py](#)
- C++ >> [answers_cpp/answer_32.cpp](#)

Q.33. フーリエ変換 ローパスフィルタ

imori.jpgをグレースケール化したものをDFTし、ローパスフィルタを通してIDFTで画像を復元せよ。

DFTによって得られた周波数成分は左上、右上、左下、右下に近いほど低周波数の成分を含んでいることになり、中心に近いほど高周波成分を示す。



画像における高周波成分とは色が変わっている部分（ノイズや輪郭など）を示し、低周波成分とは色があまり変わっていない部分（夕日のグラデーションなど）を表す。ここでは、高周波成分をカットし、低周波成分のみを通すローパスフィルタを実装せよ。

ここでは低周波数の中心から高周波までの距離をrとすると0.5rまでの成分を通すとする。

入力 (imori.jpg) 出力 (answers_image/answer_33.jpg)



答え

- Python >> [answers_py/answer_33.py](#)
- C++ >> [answers_cpp/answer_33.cpp](#)

Q.34. フーリエ変換 ハイパスフィルタ

*imori.jpg*をグレースケール化したものをDFTし、ハイパスフィルタを通してIDFTで画像を復元せよ。

ここでは、低周波成分をカットし、高周波成分のみを通す**ハイパスフィルタ**を実装せよ。

ここでは低周波数の中心から高周波までの距離を r とすると $0.1r$ からの成分を通すとする。

入力 (imori.jpg) 出力 (answers_image/answer_34.jpg)



答え

- Python >> [answers_py/answer_34.py](#)
- C++ >> [answers_cpp/answer_34.cpp](#)

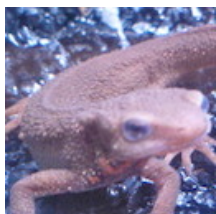
Q.35. フーリエ変換 バンドパスフィルタ

*imori.jpg*をグレースケール化したものをDFTし、ハイパスフィルタを通してIDFTで画像を復元せよ。

ここでは、低周波成分と高周波成分の中間の周波数成分のみを通す**ハイパスフィルタ**を実装せよ。

ここでは低周波数の中心から高周波までの距離を r とすると $0.1r$ から $0.5r$ までの成分を通すとする。

入力 (imori.jpg) 出力 (answers_image/answer_35.jpg)



答え

- Python >> [answers_py/answer_35.py](#)
- C++ >> [answers_cpp/answer_35.cpp](#)

Q.36. JPEG圧縮 (Step.1)離散コサイン変換

*imori.jpg*をグレースケール化し離散コサイン変換を行い、逆離散コサイン変換を行え。

離散コサイン変換(DCT: Discrete Cosine Transformation)とは、次式で定義される周波数変換の一つである。

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

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

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

逆離散コサイン変換(IDCT: Inverse Discrete Cosine Transformation)とは離散コサイン変換の逆（復号）であり、次式で定義される。ここでいう K は復元時にどれだけ解像度を良くするかを決定するパラメータである。 $K = T$ の時は、DCT係数を全部使うのでIDCT後の解像度は最大になるが、 K が1や2などの時は復元に使う情報量（DCT係数）が減るので解像度が下がる。これを適度に設定することで、画像の容量を減らすことができる。

$$1 \leq K \leq T$$

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

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

ここでは画像を8x8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8x8の領域に分割して、DCT, IDCTを行え。

入力 (imori.jpg) 出力 (answers_image/answer_36.jpg)



答え

- Python >> [answers_py/answer_36.py](#)
- C++ >> [answers_cpp/answer_36.cpp](#)

Q.37. PSNR

IDCTで用いるDCT係数を8でなく、4にすると画像の劣化が生じる。入力画像とIDCT画像のPSNRを求めよ。また、IDCTによるビットレートを求めよ。

PSNR(Peak Signal to Noise Ratio)とは信号対雑音比と呼ばれ、画像がどれだけ劣化したかを示す。

PSNRが大きいくほど、画像が劣化していないことを示し、次式で定義される。MAXは取りうる値の最大値で[0,255]の表示なら MAX=255 とする。また、MSEはMean Squared Error(平均二乗誤差)と呼ばれ、二つの画像の差分の二乗の平均値を示す。

v_{max} : *max value*

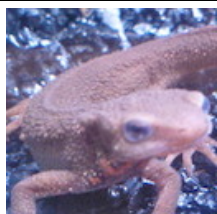
$$PSNR = 10 \log_{10} \frac{v_{max}^2}{MSE}$$

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

ビットレートとは8x8でDCTを行い、IDCTでKxKの係数までを用いた時に次式で定義される。

```
bitrate = 8 * K^2 / 8^2
```

入力 (imori.jpg) 出力 (answers_image/answer_37.jpg) (PSNR = 27.62, Bitrate=2.0)



答え

- Python >> [answers_py/answer_37.py](#)
- C++ >> [answers_cpp/answer_37.cpp](#)

Q.38. JPEG圧縮 (Step.2)DCT+量子化

DCT係数を量子化し、IDCTで復元せよ。また、その時の画像の容量を比べよ。

DCT係数を量子化することはjpeg画像にする符号化で用いられる手法である。

量子化とは、値を予め決定された区分毎に値をだまかに丸め込む作業であり、floorやceil, roundなどが似た計算である。

JPEG画像ではDCT係数を下記で表される量子化テーブルに則って量子化する。この量子化テーブルはjpeg団体の仕様書から取った。量子化では8x8の係数をQで割り、四捨五入する。その後Qを掛けることで行われる。IDCTでは係数は全て用いるものとする。

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

量子化を行うと画像の容量が減っていることから、データ量が削減されたことが伺える。

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

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



答え

- Python >> [answers_py/answer_38.py](#)
- C++ >> [answers_cpp/answer_38.cpp](#)

Q.39. JPEG圧縮 (Step.3) YCbCr表色系

YCbCr表色系において、Yを0.7倍してコントラストを暗くせよ。

YCbCr表色系とは、画像を明るさを表すY、輝度と青レベルの差Cb、輝度と赤レベルの差Crに分解する表現方法である。

これはJPEG変換で用いられる。

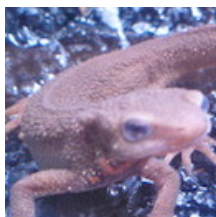
RGBからYCbCrへの変換は次式。

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

YCbCrからRGBへの変換は次式。

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

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



答え

- Python >> [answers_py/answer_39.py](#)
- C++ >> [answers_cpp/answer_39.cpp](#)

Q.40. JPEG圧縮 (Step.4) YCbCr+DCT+量子化

YCbCr表色系にし、DCT後、Yを量子化テーブルQ1、CbとCrをQ2で量子化し、IDCTで画像を復元せよ。また、画像の容量を比較せよ。

アルゴリズムは、

1. RGB を YCbCrに変換
2. YCbCrをDCT
3. DCTしたものを量子化
4. 量子化したものをIDCT

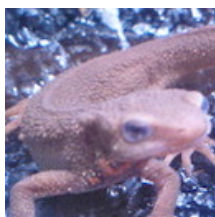
5. IDCTしたYCbCrをRGBに変換

これはJPEGで実際に使われるデータ量削減の手法であり、Q1,Q2はJPEGの仕様書に則って次式で定義される。

```
Q1 = np.array(((16, 11, 10, 16, 24, 40, 51, 61),
                (12, 12, 14, 19, 26, 58, 60, 55),
                (14, 13, 16, 24, 40, 57, 69, 56),
                (14, 17, 22, 29, 51, 87, 80, 62),
                (18, 22, 37, 56, 68, 109, 103, 77),
                (24, 35, 55, 64, 81, 104, 113, 92),
                (49, 64, 78, 87, 103, 121, 120, 101),
                (72, 92, 95, 98, 112, 100, 103, 99))), dtype=np.float32)

Q2 = np.array(((17, 18, 24, 47, 99, 99, 99, 99),
                (18, 21, 26, 66, 99, 99, 99, 99),
                (24, 26, 56, 99, 99, 99, 99, 99),
                (47, 66, 99, 99, 99, 99, 99, 99),
                (99, 99, 99, 99, 99, 99, 99, 99),
                (99, 99, 99, 99, 99, 99, 99, 99),
                (99, 99, 99, 99, 99, 99, 99, 99),
                (99, 99, 99, 99, 99, 99, 99, 99))), dtype=np.float32)
```

入力 (imori.jpg) (13kb) 出力 (answers_image/answer_40.jpg) (9kb)



答え

- Python >> [answers_py/answer_40.py](#)
- C++ >> [answers_cpp/answer_40.cpp](#)