

# 画像処理・画像処理工学 レポート課題 1

画像処理工学科 学籍番号: 21239 組番号: 234 5E 氏名: 柳原 魁人

2026 年 1 月 23 日

# 1 課題 1

## 1.1 問題 1-1: メディアンカット量子化法

### 1.1.1 理論

ピクセル値をソートして目標色数のグループに均等分割し、各グループの中央値を代表色として設定する手法です。

### 1.1.2 計算・導出過程

図 A-1 に示す  $4 \times 5$  画素の画像に対してメディアンカット量子化法を適用し、4 色に量子化します。

102	179	92	14	106
74	202	87	116	99
151	130	149	52	1
235	157	37	129	191

図 1 量子化前の  $4 \times 5$  ピクセル画像 (数値表示)

全ピクセル値は以下の通りです。

102, 179, 92, 14, 106, 74, 202, 87, 116, 99, 151, 130, 149, 52, 1, 235, 157, 37, 129, 191

これをソートすると以下のようになります。

1, 14, 37, 52, 74, 87, 92, 99, 102, 106, 116, 129, 130, 149, 151, 157, 179, 191, 202, 235

これを 4 つのグループに均等分割（各 5 ピクセル）すると、以下の表のようになります。

グループ	ピクセル値	代表色 (中央値)
1	1, 14, <b>37</b> , 52, 74	37
2	87, 92, <b>99</b> , 102, 106	99
3	116, 129, <b>130</b> , 149, 151	130
4	157, 179, <b>191</b> , 202, 235	191

上記に基づく量子化前後の比較を図で示します。

102	179	92	14	106	(a) 量子化前	99	191	99	37	99
74	202	87	116	99		37	191	99	130	99
151	130	149	52	1		130	130	130	37	37
235	157	37	129	191		191	191	37	130	191
(グループ別色分け)					(代表色)					

図 2 量子化前 (左) と量子化後 (右) の比較

### 1.1.3 結果

#### 問題1-1 限定色表示

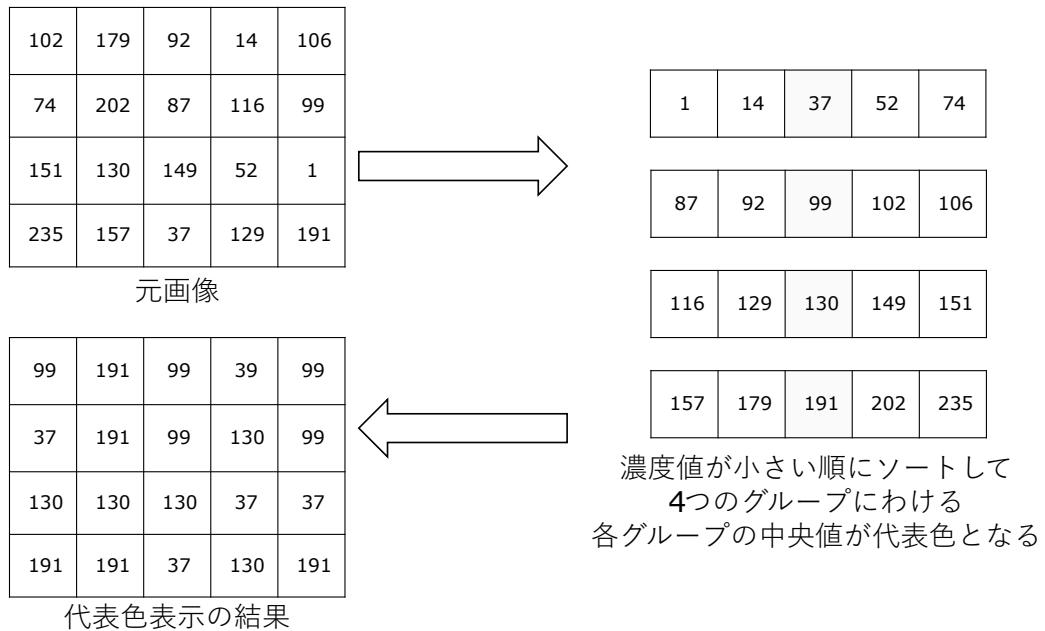


図3 メディアンカット量子化の結果

## 1.2 問題 1-2: ラベリング処理

### 1.2.1 理論

2 値画像で連結成分に同一ラベルを割り当てます。2 回走査法では、第 1 回で仮ラベル割当と等価関係記録を行い、第 2 回で統合します。

### 1.2.2 計算・導出過程

図 A-2 の  $10 \times 10$  画素画像に対してラベリングを実行。

第 1 回走査では、左上から右下へ走査し、各白ピクセルについて左上・上・右上・左のピクセルを確認します。

- 全て黒 → 新規ラベル割当
- いずれかが白 → そのラベル継承
- 複数が異なるラベル → 最小値割当、等価関係記録

等価ラベル関係： $A - B - C, E - F, D$

第 2 回走査：等価関係に基づいて代表ラベルに統一。

仮ラベル	等価関係	代表ラベル
A, B, C	$A - B - C$	A
D	単独	D
E, F	$E - F$	E

### 1.2.3 結果

#### 問題1-2 ラベリング

1回目用										2回目用									
				A	A														
			A	A	A	A	A		B										
		A	A	A	A	A	A	A											
			A	A	A	A	A	A	A										
C	A	A	A	A	A					D									
		E		F		D	D	D											
		E	E	E			D												

同じ連結成分であると記録された組：  
A-B-C, E-F, D

図4 ラベリング処理の結果

### 1.3 問題 1-3: ハフマン符号化

#### 1.3.1 理論

高確率シンボルに短い符号を割り当てる可変長符号化を行います。ハフマン木を構築することで符号を生成します。

#### 1.3.2 計算・導出過程

8 個のシンボルの出現確率を以下に示します。

シンボル	確率	符号
0	0.30	10
1	0.02	00001
2	0.06	0011
3	0.04	0001
4	0.01	00000
5	0.05	0010
6	0.20	01
7	0.32	11

結合ステップを低い確率から順に実行すると以下の通りです。

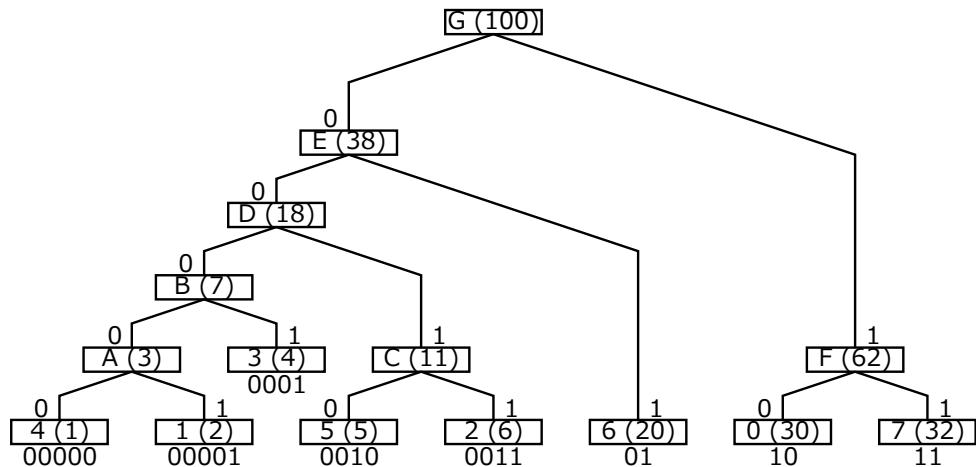
1.  $4(0.01)+1(0.02)=0.03 \rightarrow A$
2.  $A(0.03)+3(0.04)=0.07 \rightarrow B$
3.  $5(0.05)+2(0.06)=0.11 \rightarrow C$
4.  $B(0.07)+C(0.11)=0.18 \rightarrow D$
5.  $D(0.18)+6(0.20)=0.38 \rightarrow E$
6.  $0(0.30)+7(0.32)=0.62 \rightarrow F$
7.  $E(0.38)+F(0.62)=1.00 \rightarrow \text{Root}$

平均符号長計算：

$$\begin{aligned}L &= 0.30 \times 2 + 0.02 \times 5 + 0.06 \times 4 + 0.04 \times 4 \\&\quad + 0.01 \times 5 + 0.05 \times 4 + 0.20 \times 2 + 0.32 \times 2 \\&= 2.39 \text{ bits/シンボル}\end{aligned}$$

等長符号 (3 bits/シンボル) との比較：削減量 =  $3 - 2.39 = 0.61$  bits (約 20% 削減)

### 1.3.3 結果



## 2 課題 2

### 3 問題 1: モルフォロジー処理によるノイズ除去

#### 3.1 概要

開処理と閉処理を用いてノイズを除去。

#### 3.2 結果

開処理 (Opening = 収縮→膨張) により、白色の孤立ノイズを除去した。収縮処理で小領域が消失し、その後の膨張で主要な図形領域を復元する。本画像では白色ノイズが支配的であるため開処理を選択した。閉処理 (Closing = 膨張→収縮) は黒いノイズの埋め込みに有効だが、本課題では不要と判断した。

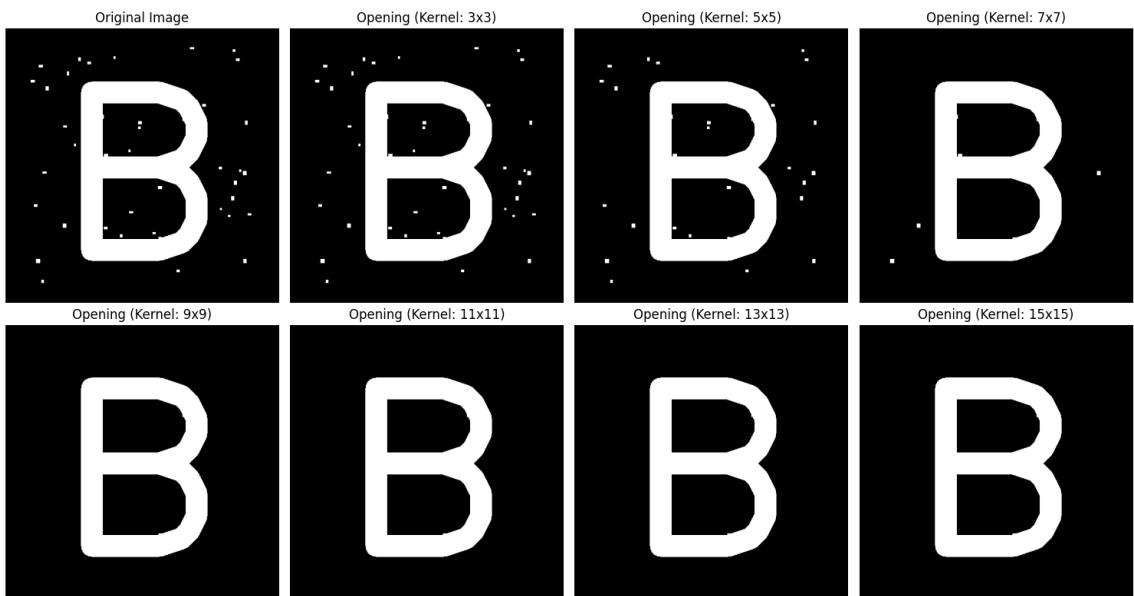


図 6 Kernel size 15 での開閉処理結果（ノイズ除去後の 2 値画像）

カーネルサイズ 9 以降は見た目の変化が小さく、9 で十分。15 にする必要はない。

## 4 問題 2: JPEG 品質と圧縮率の関係

### 4.1 概要

JPEG 品質と圧縮率、SSIM 値の関係を調査。

### 4.2 結果

下表は本レポートで得られた数値（品質 Q、サイズ比、SSIM）を整理したもの。

表 1 品質 Q によるサイズ比と SSIM

Q	サイズ比 (%)	SSIM
0	0.9	0.327
10	2.2	0.520
20	3.6	0.609
30	4.8	0.656
40	5.8	0.686
50	6.7	0.709
60	7.7	0.729
70	9.3	0.753
80	11.8	0.783
90	17.7	0.825
100	45.5	0.871

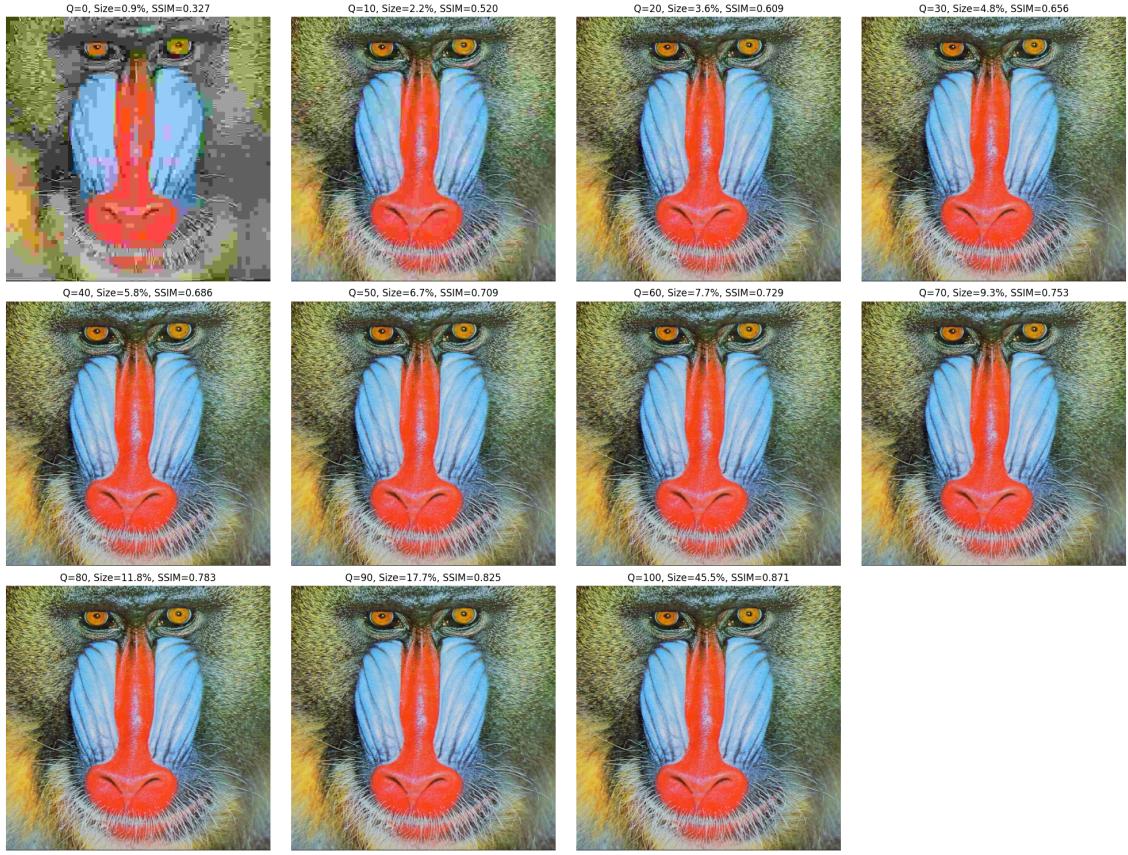


図 7 JPEG 品質 Q ごとの視覚比較とサイズ比・SSIM（提示画像）

**Q=80** で SSIM=0.783、**Q=90** で SSIM=0.825 と向上するが、サイズ比は 11.8% → 17.7% へ増加。視覚・数値の両面から、実用的には **Q=80~90** が妥当。**Q=100** は SSIM 向上が小さい一方でサイズ比 45.5% と非効率。ただし、視覚的には **Q=10** の画像でも許容できると判断される場合がある（図 7 を参照）。SSIM は 0.520 と低めのため数値的には劣るが、用途によっては容量削減を優先して **Q=10** を採用することは妥当である。

## 5 問題 3: 2 次元 FFT と振幅スペクトル

### 5.1 概要

グレースケール画像を 2 次元 FFT で周波数領域に変換し、振幅スペクトルを分析。

### 5.2 結果

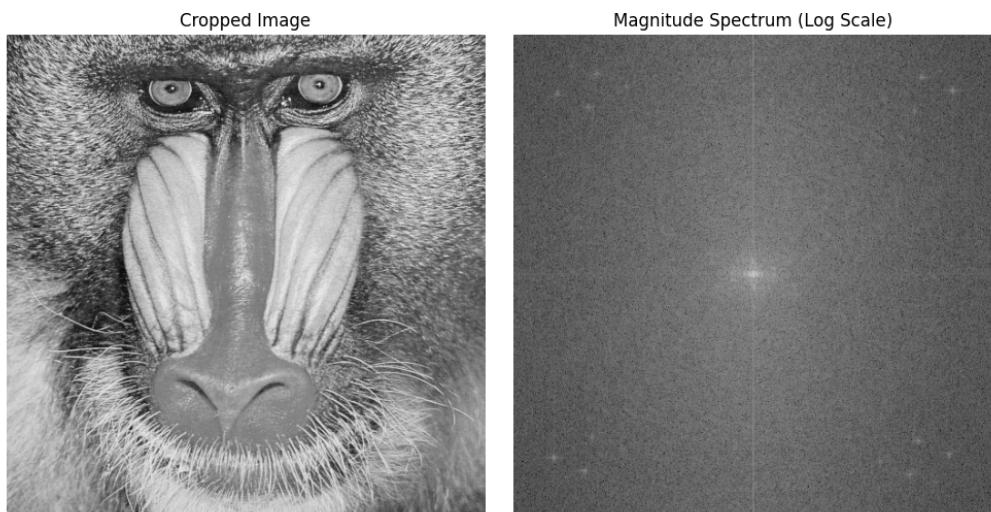


図 8 グレースケール画像の 2 次元振幅スペクトル（中央化）

振幅スペクトルの中心に強い明るさを示す領域がある。これは DC 成分（0 周波数）と低周波成分が画像の主要成分であることを示す。周辺部は暗く、高周波成分（細かなエッジやノイズ）が少ないことを表す。垂直・水平方向に弱い構造が見られるのは、画像内に規則的な境界や勾配が存在することを示唆している。

## 6 問題 4: 周波数フィルタの応用

### 6.1 概要

理想的ローパスフィルタ (ILPF) とガウシアンハイパスフィルタ (GHPF) を Cutoff=30 で適用し、効果を比較。

### 6.2 結果

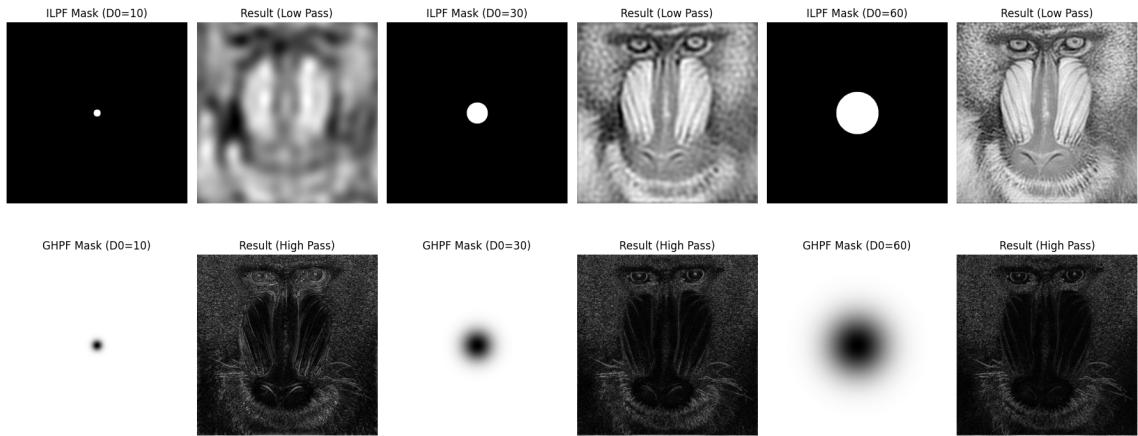


図 9 ローパスフィルタとハイパスフィルタの適用結果比較 (Cutoff=30)

**ローパスフィルタ (ILPF)**：低周波成分のみを通し、高周波（ノイズ・細部）を除去。結果は全体的にぼかされ、目や顔の大型構造は保持される一方、毛並みなどの細かいテクスチャは消失。ノイズ除去やスマージングに有効。

**ハイパスフィルタ (GHPF)**：DC 成分と低周波を除去し、高周波のみを通す。結果はエッジが明るく浮き出、目・口・毛並みなどの細部が強調される。大きな領域は暗くなり、画像全体はコントラストが低下。エッジ検出と細部抽出に有効。

**カットオフ周波数 D0 の影響:** D0 を 10/30/60 と変化させた結果、D0 が小さいほど通過する周波数帯域が狭まる。ローパスでは D0=10 で極端なぼかし効果、D0=60 で細部を一定程度保持する。ハイパスでは D0=10 でエッジ成分のみが抽出され、D0=60 では中間周波数も強調されより豊かなテクスチャが得られた。D0 の選択は目的（ノイズ除去 vs 細部保持）に応じて調整すべきである。

## 付録: プログラムリスト

### 問題1: モルフォロジー処理

```
1 # -*- coding: utf-8 -*-
2 """問題2_1.ipynb"""
3
4 # ドライブのマウントとファイル操作用モジュール
5 from google.colab import drive
6 from google.colab import files
7 drive.mount('/content/drive')
8
9 # モジュールのインポート
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13 import math
14
15 # 共通のディレクトリパス
16 common_path = '/content/drive/MyDrive/img2025/image/'
17 filename = 'a2-3_binary_image.png'
18
19 # 画像読み込み
20 img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
21
22 # カーネルサイズのリスト
23 kernel_sizes = list(range(3, 17, 2))
24 total_images = 1 + len(kernel_sizes)
25
26 # レイアウト計算
27 cols = 4
28 rows = math.ceil(total_images / cols)
29
30 # 実行結果の表示設定
31 plt.figure(figsize=(15, 4 * rows))
32
33 # オリジナル画像
34 plt.subplot(rows, cols, 1)
35 plt.title('Original Image')
36 plt.imshow(img, cmap='gray')
37 plt.axis('off')
38
39 # カーネルサイズを変えて処理
40 for i, k in enumerate(kernel_sizes):
41     # カーネル作成
42     kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (k, k))
43     # オープニング処理
```

```
44     result = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
45
46     # 表示
47     plt.subplot(rows, cols, i + 2)
48     plt.title(f'Opening (Kernel: {k}x{k})')
49     plt.imshow(result, cmap='gray')
50     plt.axis('off')
51
52 plt.tight_layout()
53
54 # 画像を保存してダウンロード
55 save_filename = 'problem2_1_result.png'
56 plt.savefig(save_filename)
57 plt.show()
58 files.download(save_filename)
```

Listing 1 問題 1 モルフォロジー処理によるノイズ除去

## 問題 2: JPEG 品質と圧縮率

```
1 # -*- coding: utf-8 -*-
2 """問題2_2.ipynb"""
3
4 # ドライブのマウントとファイル操作用モジュール
5 from google.colab import drive
6 from google.colab import files
7 drive.mount('/content/drive')
8
9 # モジュールのインポート
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13 import os
14 from skimage.metrics import structural_similarity as ssim
15
16 # 共通のディレクトリパス
17 common_path = '/content/drive/MyDrive/img2025/image/'
18 filename = 'a2-4_color_image.png'
19
20 # 画像を読み込む
21 original_img = cv2.imread(common_path + filename)
22 original_size = os.path.getsize(common_path + filename)
23 original_img_rgb = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)
24
25 # 画像表示の準備
26 plt.figure(figsize=(20, 15))
27
28 # 品質0から100まで10刻みでループ
29 for i, quality in enumerate(range(0, 101, 10)):
30     # JPEG圧縮保存 (一時ファイル)
31     output_path = common_path + f'compressed_{quality}.jpg'
32     cv2.imwrite(output_path, original_img, [int(cv2.IMWRITE_JPEG_QUALITY), quality])
33
34     # 圧縮後のファイルサイズ取得と圧縮率計算
35     comp_size = os.path.getsize(output_path)
36     comp_ratio = (comp_size / original_size) * 100
37
38     # 画像読み込み
39     compressed_img = cv2.imread(output_path)
40     compressed_img_rgb = cv2.cvtColor(compressed_img, cv2.COLOR_BGR2RGB)
41
42     # SSIM計算
43     score = ssim(original_img_rgb, compressed_img_rgb, win_size=3, channel_axis=2,
44                   data_range=255)
45
46     # 表示
```

```
46 plt.subplot(3, 4, i + 1)
47 plt.imshow(compressed_img_rgb)
48 plt.title(f"Q={quality}, Size={comp_ratio:.1f}%, SSIM={score:.3f}")
49 plt.axis('off')
50
51 plt.tight_layout()
52
53 # 画像を保存してダウンロード
54 save_filename = 'problem2_2_result.png'
55 plt.savefig(save_filename)
56 plt.show()
57 files.download(save_filename)
```

Listing 2 問題 2 JPEG 品質と圧縮率の関係調査

### 問題 3: 2 次元 FFT と振幅スペクトル

```
1 # -*- coding: utf-8 -*-
2 """問題2_3.ipynb"""
3
4 # ドライブのマウントとファイル操作用モジュール
5 from google.colab import drive
6 from google.colab import files
7 drive.mount('/content/drive')
8
9 # モジュールのインポート
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13
14 # 共通のディレクトリパス
15 common_path = '/content/drive/MyDrive/img2025/image/'
16 filename = 'a2-5_gray_image.png'
17
18 # 画像を読み込む
19 gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
20
21 # トリミング (512x512)
22 h, w = gray_img.shape
23 crop_size = 512
24 start_y = h // 2 - crop_size // 2
25 start_x = w // 2 - crop_size // 2
26 cropped_img = gray_img[start_y:start_y+crop_size, start_x:start_x+crop_size]
27
28 # 2次元FFTとシフト
29 fourier = np.fft.fft2(cropped_img)
30 fshift = np.fft.fftshift(fourier)
31
32 # 振幅スペクトル (対数スケール)
33 amp_spectrum = 20 * np.log10(np.abs(fshift) + 1)
34
35 # 実行結果の表示
36 plt.figure(figsize=(10, 5))
37
38 # トリミング画像
39 plt.subplot(1, 2, 1)
40 plt.title('Cropped Image')
41 plt.imshow(cropped_img, cmap='gray')
42 plt.axis('off')
43
44 # 振幅スペクトル
45 plt.subplot(1, 2, 2)
46 plt.title('Magnitude Spectrum (Log Scale)')
```

```
47 plt.imshow(amp_spectrum, cmap='gray')
48 plt.axis('off')
49
50 plt.tight_layout()
51
52 # 画像を保存してダウンロード
53 save_filename = 'problem2_3_result.png'
54 plt.savefig(save_filename)
55 plt.show()
56 files.download(save_filename)
```

Listing 3 問題 3 2 次元 FFT と振幅スペクトル

#### 問題 4: 周波数フィルタの応用

```
1 # -*- coding: utf-8 -*-
2 """問題2_4.ipynb"""
3
4 # ドライブのマウントとファイル操作用モジュール
5 from google.colab import drive
6 from google.colab import files
7 drive.mount('/content/drive')
8
9 # モジュールのインポート
10 import cv2
11 import numpy as np
12 import matplotlib.pyplot as plt
13
14 # 共通のディレクトリパス
15 common_path = '/content/drive/MyDrive/img2025/image/'
16 filename = 'a2-5_gray_image.png'
17
18 # フィルタ生成関数
19 def create_ideal_lowpass(shape, cutoff):
20     rows, cols = shape
21     crow, ccol = rows // 2, cols // 2
22     y, x = np.ogrid[:rows, :cols]
23     dist_sq = (x - ccol)**2 + (y - crow)**2
24     mask = np.zeros(shape)
25     mask[dist_sq <= cutoff**2] = 1
26     return mask
27
28 def create_gaussian_highpass(shape, cutoff):
29     rows, cols = shape
30     crow, ccol = rows // 2, cols // 2
31     y, x = np.ogrid[:rows, :cols]
32     dist_sq = (x - ccol)**2 + (y - crow)**2
33     mask = 1 - np.exp(-dist_sq / (2 * (cutoff**2)))
34     return mask
35
36 # 画像読み込みとトリミング
37 gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
38 h, w = gray_img.shape
39 size = 512
40 img = gray_img[h//2-size//2 : h//2+size//2, w//2-size//2 : w//2+size//2]
41
42 # 2次元FFTとシフト
43 fshift = np.fft.fftshift(np.fft.fft2(img))
44
45 # 試行する遮断周波数リスト
46 cutoffs = [10, 30, 60]
```

```

47
48 # 表示設定
49 cols = len(cutoffs) * 2
50 plt.figure(figsize=(18, 8))
51
52 # --- 上段：理想ローパスフィルタ ---
53 for i, d0 in enumerate(cutoffs):
54     # マスク作成と適用
55     mask_lp = create_ideal_lowpass(img.shape, d0)
56     img_lp = np.fft.ifft2(np.fft.ifftshift(fshift * mask_lp)).real
57
58     # マスク表示
59     plt.subplot(2, cols, i * 2 + 1)
60     plt.title(f'ILPF Mask (D0={d0})')
61     plt.imshow(mask_lp, cmap='gray')
62     plt.axis('off')
63
64     # 結果表示
65     plt.subplot(2, cols, i * 2 + 2)
66     plt.title(f'Result (Low Pass)')
67     plt.imshow(img_lp, cmap='gray')
68     plt.axis('off')
69
70 # --- 下段：ガウシアンハイパスフィルタ ---
71 for i, d0 in enumerate(cutoffs):
72     # マスク作成と適用
73     mask_hp = create_gaussian_highpass(img.shape, d0)
74     img_hp = np.fft.ifft2(np.fft.ifftshift(fshift * mask_hp)).real
75
76     # マスク表示
77     plt.subplot(2, cols, cols + i * 2 + 1)
78     plt.title(f'GHPP Mask (D0={d0})')
79     plt.imshow(mask_hp, cmap='gray')
80     plt.axis('off')
81
82     # 結果表示
83     plt.subplot(2, cols, cols + i * 2 + 2)
84     plt.title(f'Result (High Pass)')
85     plt.imshow(np.abs(img_hp), cmap='gray')
86     plt.axis('off')
87
88 plt.tight_layout()
89
90 # 画像を保存してダウンロード
91 save_filename = 'problem2_4_result.png'
92 plt.savefig(save_filename)
93 plt.show()
94 files.download(save_filename)

```

---

Listing 4 問題 4 周波数フィルタの応用