

# 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

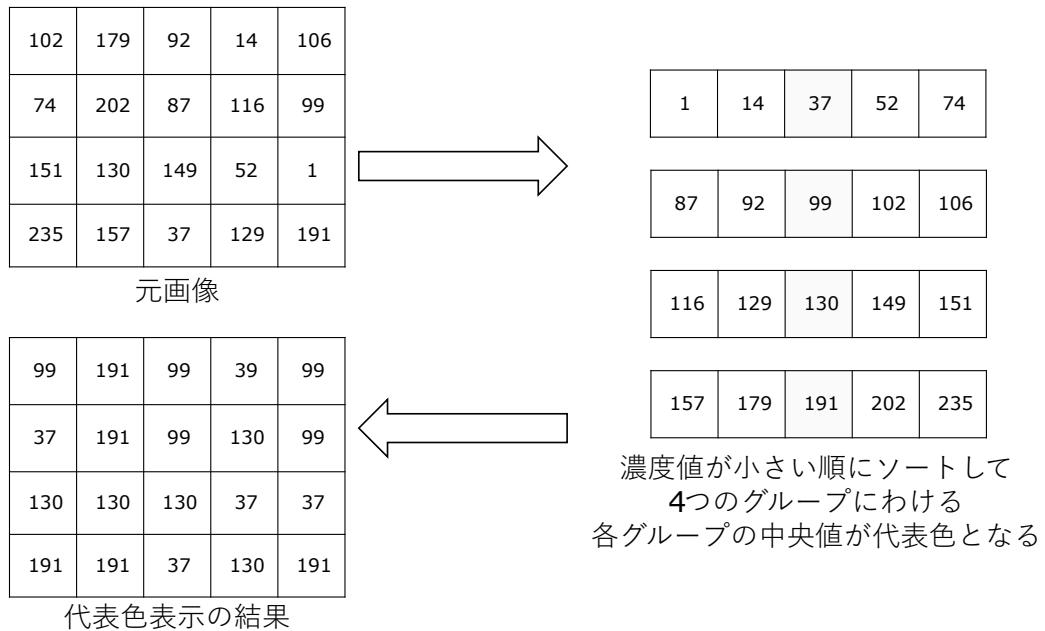


図3 メディアンカット量子化の結果

## 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。

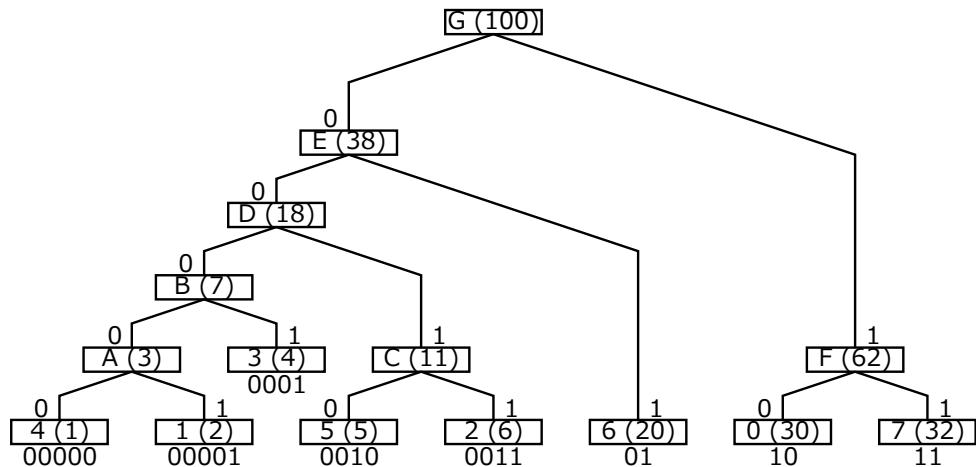
1.  $4(0.01)+1(0.02)=0.03 \rightarrow A$
2.  $A(0.03)+3(0.04)=0.07 \rightarrow B$
3.  $5(0.05)+2(0.06)=0.11 \rightarrow C$
4.  $B(0.07)+C(0.11)=0.18 \rightarrow D$
5.  $D(0.18)+6(0.20)=0.38 \rightarrow E$
6.  $0(0.30)+7(0.32)=0.62 \rightarrow F$
7.  $E(0.38)+F(0.62)=1.00 \rightarrow \text{Root}$

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

開処理により小さなノイズ除去、閉処理により黒いノイズを埋め込み。

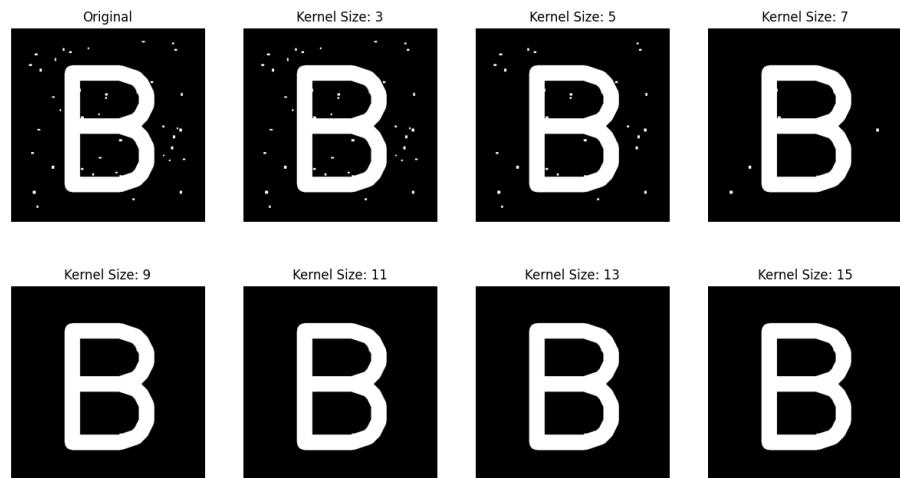


図 6 Kernel size 15 での開閉処理結果（ノイズ除去後の 2 値画像）

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 にする必然性はない。

## 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。

## 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

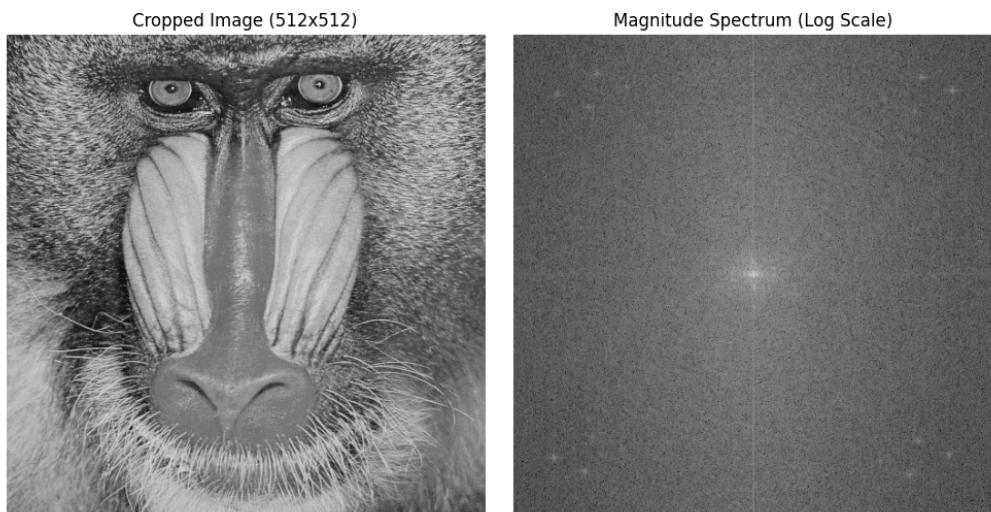


図 8 グレースケール画像の 2 次元振幅スペクトル（中央化）

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

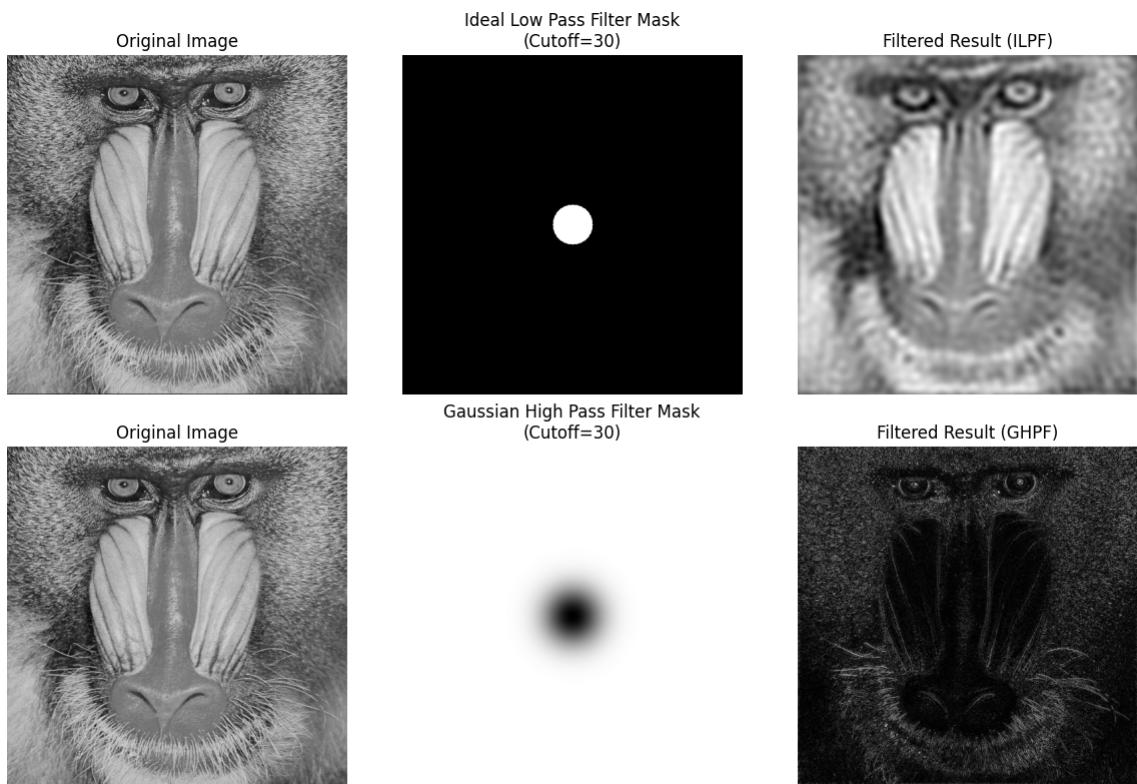


図 9 ローパスフィルタとハイパスフィルタの適用結果比較 (Cutoff=30)

**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。ノイズ除去やスムージングに有効。

**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。

## 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 Automatically generated by Colab.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1KwcAHJ1LwDpZRJEFL1gEOUwZf-fEWtg
8 """
9
10 from google.colab import drive
11 drive.mount('/content/drive')
12
13 import cv2
14 import numpy as np
15 import matplotlib.pyplot as plt
16 import math
17
18 # 画像パス（自身の環境に合わせて確認）
19 image_path = '/content/drive/MyDrive/img2025/image/a2-3_binary_image.png'
20
21 # 画像読み込み
22 img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
23
24 # 3から13まで、2ずつ増やしてリストを作る (3, 5, 7, 9, 11, 13)
25 # 終了値を大きくすれば、もっと試せる（例: range(3, 21, 2)）
26 kernel_sizes = list(range(3, 17, 2))
27
28 # 画像の総数（オリジナル + 処理結果の数）
29 total_images = 1 + len(kernel_sizes)
30
31 # レイアウト設定
32 cols = 4 # 1行に並べる数
33 rows = math.ceil(total_images / cols) # 必要な行数を自動計算
34
35 plt.figure(figsize=(15, 4 * rows))
36
37 # 1. オリジナル画像を表示
38 plt.subplot(rows, cols, 1)
39 plt.title('Original')
40 plt.imshow(img, cmap='gray')
41 plt.axis('off')
42
43 # 2. ループ処理でサイズを変えながら表示
```

```
44 for i in range(len(kernel_sizes)):
45     k = kernel_sizes[i]
46
47     # カーネル作成と処理
48     kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (k, k))
49     result = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
50
51     # 結果を表示（場所は i + 2 番目）
52     plt.subplot(rows, cols, i + 2)
53     plt.title(f'Kernel Size: {k}')
54     plt.imshow(result, cmap='gray')
55     plt.axis('off')
56
57 plt.tight_layout()
58 plt.show()
```

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#2ipynb
3
4 Automatically generated by Colab.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1La6z4iK8IV6PEYigjfX3vHFJ6G3TQe3m
8 """
9
10 # ドライブのマウント
11 from google.colab import drive
12 drive.mount('/content/drive')
13
14 # モジュールのインポート
15 import cv2
16 import numpy as np
17 import matplotlib.pyplot as plt
18 import os
19 from skimage.metrics import structural_similarity as ssim # SSIM計算用
20
21 # 共通のディレクトリパス
22 common_path = '/content/drive/MyDrive/img2025/image/'
23 filename = 'a2-4_color_image.png' # 課題指定のファイル名
24
25 # 画像を読み込む
26 original_img = cv2.imread(common_path + filename)
27
28 # 画像が正しく読み込んでいるか確認（読み込めない場合はNoneになるためエラー回避）
29 if original_img is None:
30     print(f"Error: {filename} が見つかりません。パスを確認してください。")
31 else:
32     # 元画像のファイルサイズを取得（バイト単位）
33     original_size = os.path.getsize(common_path + filename)
34     print(f"元画像読み込み完了: {filename}, サイズ: {original_size/1024:.2f} KB")
35
36 # 結果を格納するリスト
37 qualities = []
38 file_sizes = []
39 compression_ratios = []
40 ssim_scores = []
41
42 # 画像表示の準備（3行4列のグリッドで表示）
43 plt.figure(figsize=(20, 15))
44
45 # 元画像をRGB変換（SSIM計算と表示用）
46 original_img_rgb = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)
```

```

47
48 # 0から100まで10刻みでループ
49 for i, quality in enumerate(range(0, 101, 10)):
50     # 1. JPEG圧縮保存
51     output_filename = f'compressed_{quality}.jpg'
52     output_path = common_path + output_filename
53
54     # 第2引数でJPEG品質を指定
55     cv2.imwrite(output_path, original_img, [int(cv2.IMWRITE_JPEG_QUALITY), quality])
56
57     # 2. 圧縮後のファイルサイズ取得
58     comp_size = os.path.getsize(output_path)
59     comp_ratio = (comp_size / original_size) * 100 # 元画像に対するサイズ比率(%)
60
61     # 3. 圧縮画像の読み込みとRGB変換
62     compressed_img = cv2.imread(output_path)
63     compressed_img_rgb = cv2.cvtColor(compressed_img, cv2.COLOR_BGR2RGB)
64
65     # 4. SSIM（画質評価値）の計算
66     # channel_axis=2 はカラー画像(マルチチャンネル)であることを指定
67     score = ssim(original_img_rgb, compressed_img_rgb, win_size=3, channel_axis=2,
68                   data_range=255)
69
70     # リストにデータを保存
71     qualities.append(quality)
72     file_sizes.append(comp_size)
73     compression_ratios.append(comp_ratio)
74     ssim_scores.append(score)
75
76     # 5. サブプロットへの表示
77     plt.subplot(3, 4, i + 1)
78     plt.imshow(compressed_img_rgb)
79     plt.title(f"Q={quality}\nSize: {comp_ratio:.1f}%, SSIM: {score:.3f}")
80     plt.axis('off')
81
82 # レイアウト調整と表示
83 plt.tight_layout()
84 plt.show()

```

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2-3-1.ipynb
3
4 Automatically generated by Colab.
5
6 Original file is located at
7     https://colab.research.google.com/drive/10wZ09eY3hMiF-v-SEBzxZ-4SCiz3TaxX
8 """
9
10 # ドライブのマウント
11 from google.colab import drive
12 drive.mount('/content/drive')
13
14 # モジュールのインポート
15 import cv2
16 import numpy as np
17 import matplotlib.pyplot as plt
18
19 # 共通のディレクトリパス
20 common_path = '/content/drive/MyDrive/img2025/image/'
21
22 # 画像を読み込む
23 filename = 'a2-5_gray_image.png'
24 gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
25
26 # 画像を正方形（512x512）にトリミングする処理
27 # 画像の中心から指定サイズを切り出す
28 h, w = gray_img.shape
29 crop_size = 512
30
31 # 画像サイズがトリミングサイズより小さい場合の例外処理（念のため）
32 if h < crop_size or w < crop_size:
33     print(f"画像サイズが小さいため、トリミングサイズを調整します: {min(h, w)}")
34     crop_size = min(h, w)
35
36 center_y, center_x = h // 2, w // 2
37 start_x = center_x - crop_size // 2
38 start_y = center_y - crop_size // 2
39 cropped_img = gray_img[start_y:start_y+crop_size, start_x:start_x+crop_size]
40
41 # 2次元高速フーリエ変換（2D-FFT）
42 fourier = np.fft.fft2(cropped_img)
43
44 # ゼロ周波数成分（直流成分）を画像の中心にシフトする
45 fshift = np.fft.fftshift(fourier)
46
```

```

47 # 振幅スペクトルを計算する
48 amp_spectrum = np.abs(fshift)
49
50 # 対数スケールに変換（見やすくするために +1 して log をとる）
51 log_amp_spectrum = 20 * np.log10(amp_spectrum + 1)
52
53 # 実行結果の表示
54 plt.figure(figsize=(10, 5))
55
56 # トリミング後の元画像
57 plt.subplot(1, 2, 1)
58 plt.title('Cropped Image (512x512)')
59 plt.imshow(cropped_img, cmap='gray', vmin=0, vmax=255)
60 plt.axis('off')
61
62 # 振幅スペクトル（対数スケール・中心化済み）
63 plt.subplot(1, 2, 2)
64 plt.title('Magnitude Spectrum (Log Scale)')
65 plt.imshow(log_amp_spectrum, cmap='gray')
66 plt.axis('off')
67
68 plt.tight_layout()
69 plt.show()

```

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 Automatically generated by Colab.
5
6 Original file is located at
7     https://colab.research.google.com/drive/14oortqIl9hjtngj_clywKmLNDsRRMBjh
8 """
9
10 # =====
11 # 課題4: 2種類以上の周波数フィルタの適用と考察
12 # =====
13 # 1. 準備と画像読み込み
14 import cv2
15 import numpy as np
16 import matplotlib.pyplot as plt
17 from google.colab import drive
18
19 # ドライブのマウント
20 drive.mount('/content/drive')
21
22 # パスとファイル名の設定
23 common_path = '/content/drive/MyDrive/img2025/image/'
24 filename = 'a2-5_gray_image.png'
25
26 # 画像読み込み
27 gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
28
29 # 画像の正方形トリミング（課題3の処理を継承）
30 h, w = gray_img.shape
31 crop_size = 512
32 target_size = min(h, w, crop_size)
33
34 center_y, center_x = h // 2, w // 2
35 start_y = center_y - target_size // 2
36 start_x = center_x - target_size // 2
37 img = gray_img[start_y:start_y+target_size, start_x:start_x+target_size]
38
39 print(f"画像読み込み完了: {img.shape}")
40
41 # 2. FFT実行とフィルタ関数の定義・適用
42
43 # --- FFTの実行 ---
44 f = np.fft.fft2(img)
45 fshift = np.fft.fftshift(f) # 直流成分を中心に移動
46
```

```

47 # --- フィルタ生成関数の定義 ---
48 def create_ideal_lowpass(shape, cutoff):
49     """理想ローパスフィルタ"""
50     rows, cols = shape
51     crow, ccol = rows // 2, cols // 2
52     y, x = np.ogrid[:rows, :cols]
53     dist_sq = (x - ccol)**2 + (y - crow)**2
54     mask = np.zeros(shape)
55     mask[dist_sq <= cutoff**2] = 1
56     return mask
57
58 def create_gaussian_highpass(shape, cutoff):
59     """ガウシアンハイパスフィルタ"""
60     rows, cols = shape
61     crow, ccol = rows // 2, cols // 2
62     y, x = np.ogrid[:rows, :cols]
63     dist_sq = (x - ccol)**2 + (y - crow)**2
64     # ガウシアンローパスの逆(1から引く)
65     mask = 1 - np.exp(-dist_sq / (2 * (cutoff**2)))
66     return mask
67
68 # --- パラメータ設定と適用 ---
69 D0_low = 30 # ローパス用遮断周波数
70 D0_high = 30 # ハイパス用遮断周波数
71
72 # マスク作成とフィルタリング
73 mask_ilpf = create_ideal_lowpass(img.shape, D0_low)
74 mask_ghpf = create_gaussian_highpass(img.shape, D0_high)
75
76 fshift_ilpf = fshift * mask_ilpf
77 fshift_ghpf = fshift * mask_ghpf
78
79 # 逆FFT処理
80 img_ilpf = np.fft.ifft2(np.fft.ifftshift(fshift_ilpf)).real
81 img_ghpf = np.fft.ifft2(np.fft.ifftshift(fshift_ghpf)).real
82
83 print("フィルタ処理完了")
84
85 # 3. 結果の可視化
86 plt.figure(figsize=(12, 8))
87
88 # --- 上段: 理想ローパスフィルタ ---
89 plt.subplot(2, 3, 1)
90 plt.title('Original Image')
91 plt.imshow(img, cmap='gray', vmin=0, vmax=255)
92 plt.axis('off')
93
94 plt.subplot(2, 3, 2)
95 plt.title(f'Ideal Low Pass Filter Mask\n(Cutoff={D0_low})')

```

```

96 plt.imshow(mask_ilpf, cmap='gray', vmin=0, vmax=1)
97 plt.axis('off')
98
99 plt.subplot(2, 3, 3)
100 plt.title('Filtered Result (ILPF)')
101 plt.imshow(img_ilpf, cmap='gray')
102 plt.axis('off')
103
104 # --- 下段: ガウシアンハイパスフィルタ ---
105 plt.subplot(2, 3, 4)
106 plt.title('Original Image')
107 plt.imshow(img, cmap='gray', vmin=0, vmax=255)
108 plt.axis('off')
109
110 plt.subplot(2, 3, 5)
111 plt.title(f'Gaussian High Pass Filter Mask\n(Cutoff={D0_high})')
112 plt.imshow(mask_ghpf, cmap='gray', vmin=0, vmax=1)
113 plt.axis('off')
114
115 plt.subplot(2, 3, 6)
116 plt.title('Filtered Result (GHPF)')
117 plt.imshow(np.abs(img_ghpf), cmap='gray') # 振幅を表示
118 plt.axis('off')
119
120 plt.tight_layout()
121 plt.show()

```

Listing 4 問題 4 周波数フィルタの応用