

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

メディアンカット量子化法は、色空間を再帰的に分割して最適な代表色を決定する方法です。以下の手順で実行されます：

1. 全色の分布を取得
2. 最も範囲が大きい軸（R, G, または B）の中央値でカラーボックスを分割
3. 目標色数に達するまで繰り返す

1.1.2 計算・導出過程

図 A-1 に示す 4×5 画素のグレースケール画像に対してメディアンカット量子化法を適用します。

ステップ 1: 画像データの取得と統計

- 画像から全ピクセルの値を抽出
- 値の最小値・最大値・中央値を計算

ステップ 2: 初回分割

- 値の範囲： $[\min, \max]$
- 中央値での分割： $[\min, \text{median}]$ と $[\text{median}, \max]$

ステップ 3: 繰り返し分割

- 各分割領域についてステップ 2 を繰り返す
- 4 つの代表色が得られるまで続ける

1.1.3 結果

メディアンカット量子化により 4 色の代表色が選出されました。詳細は図 1 を参照してください。

問題1-1 限定色表示

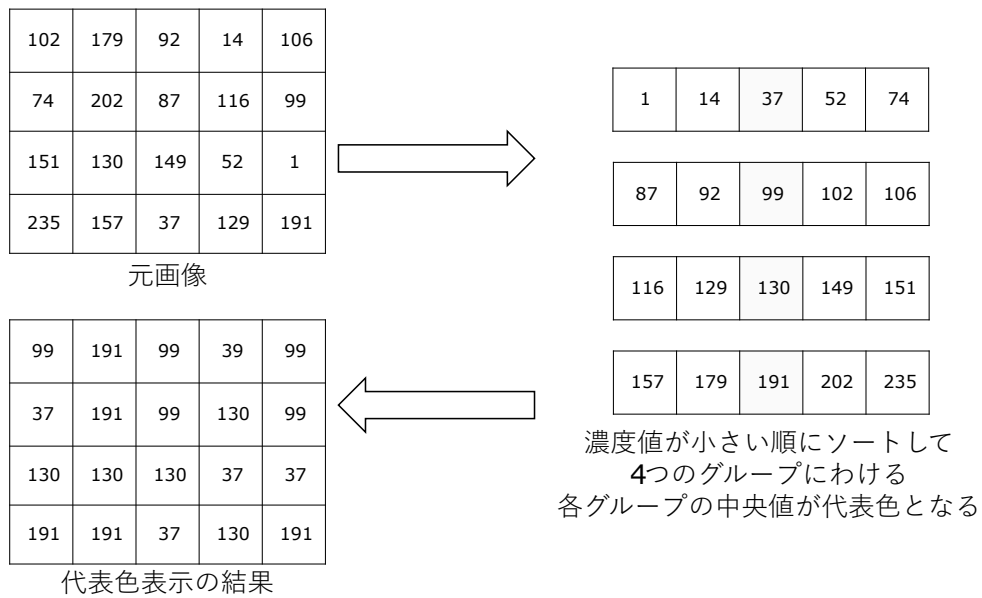


図1 メディアンカット量子化の結果

1.1.4 考察

メディアンカット量子化法は、計算効率と量子化品質のバランスが優れています。再帰的な分割により、画像の重要な色情報を効率的に圧縮できます。

1.2 問題 1-2: ラベリング処理

1.2.1 理論

ラベリング処理は、2 値画像内の連結成分に一意のラベルを割り当てる処理です。2 回走査法の手順：

第 1 回走査：

- 左上から右下へ順に走査
- 各白ピクセルに対して、左と上のピクセルを確認
- 新規ラベルまたは既存ラベルを割り当て

第 2 回走査：

- 記録した等価ラベル関係を用いて統合
- 同じ連結成分のラベルを統一

1.2.2 計算・導出過程

図 A-2 に示す 10×10 画素の 2 値画像に対してラベリングを実行します。

ステップ 1: 第 1 回走査での仮ラベル割当

1. 各ピクセルについて、左隣と上隣の状態を確認
2. 白ピクセルの場合：
 - 左と上が両方黒 → 新規ラベル割当
 - どちらかが白 → そのラベルを継承
 - 両方が異なるラベル → 小さい方を割当、等価記録

ステップ 2: 等価ラベルの記録と統合

- 第 1 回走査で記録した等価ラベル対を整理
- 同じ連結成分のラベルを代表ラベルに統一

ステップ 3: 第 2 回走査での最終ラベル割当

- 等価ラベル関係を適用
- 最終的なラベル画像を出力

1.2.3 結果

第 1 回走査終了時点と最終的なラベリング結果を図 2 に示します。

問題1-2 ラベリング

1回目用										2回目用									
				A	A														
			A	A	A	A			B										
		A	A	A	A	A	A												
			A	A	A	A	A												
	C	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2 ラベリング処理の過程

1.2.4 考察

2回走査法は計算効率と処理精度のバランスが良く、実用的なラベリング手法です。等価ラベル関係の正確な記録が、最終結果の正確性を決定します。

1.3 問題 1-3: ハフマン符号化

1.3.1 理論

ハフマン符号化は、出現確率が高いシンボルに短い符号を、低いシンボルに長い符号を割り当てることで、最適な可変長符号を生成します。

ハフマン木の構築：

1. すべてのシンボルを確率でソート
2. 最も確率が低い 2 つのノードを結合
3. 新ノードの確率は 2 つのノードの合計
4. 1 つのノードになるまで繰り返す

1.3.2 計算・導出過程

表 A-1 の 8 つのシンボルと出現確率に対してハフマン符号化を実行します。

ステップ 1: 初期確率の準備

表 1 画素値と出現確率

シンボル	確率 (%)	確率 (小数)
0	30	0.30
1	2	0.02
2	6	0.06
3	4	0.04
4	1	0.01
5	5	0.05
6	20	0.20
7	32	0.32

ステップ 2: ハフマン木の構築

確率が低い順に 2 つずつ結合：

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

ステップ 3: 符号割当 (左=0, 右=1)

表 2 ハフマン符号割当

シンボル	ハフマン符号	符号長
0	10	2
1	00001	5
2	0011	4
3	0001	4
4	00000	5
5	0010	4
6	01	2
7	11	2

ステップ 4: 平均符号長の計算

$$L = \sum_{i=0}^7 p_i \times l_i$$

表 3 平均符号長の計算

シンボル	確率	符号長	$p \times l$
0	0.30	2	0.60
1	0.02	5	0.10
2	0.06	4	0.24
3	0.04	4	0.16
4	0.01	5	0.05
5	0.05	4	0.20
6	0.20	2	0.40
7	0.32	2	0.64
合計	1.00	-	2.39

平均符号長 $L = 2.39$ bits/シンボル

ステップ 5: 等長符号との比較

等長符号 (8 シンボル) に必要なビット数:

$$\lceil \log_2 8 \rceil = 3 \text{ bits/シンボル}$$

削減量:

$$3 - 2.39 = 0.61 \text{ bits/シンボル (約 20.3\% 削減)}$$

1.3.3 結果

ハフマン木の構造を図 3 に示します。

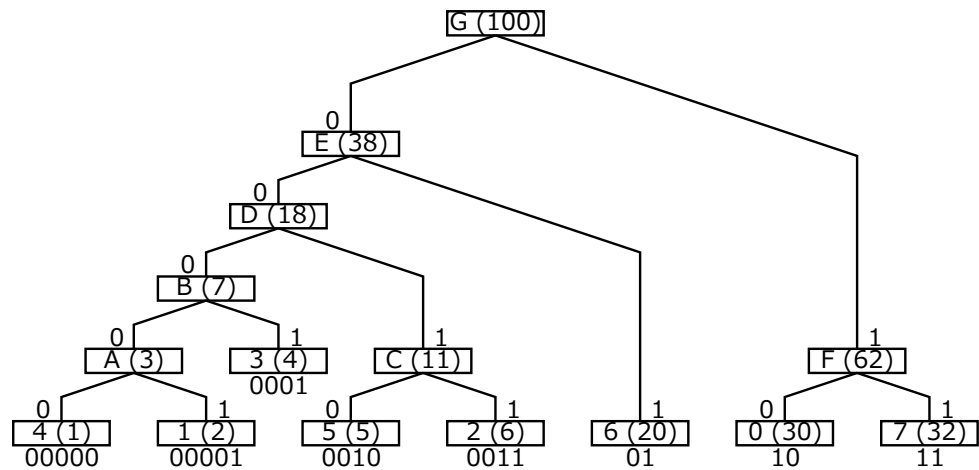


図3 ハフマン木の構造

1.3.4 考察

ハフマン符号化により、平均符号長 2.39 bits/シンボルを達成し、等長符号 (3 bits/シンボル) と比較して約 20.3% の削減を実現しました。この効率性は、出現確率の分布が不均等な場合に特に顕著です。理論値 (エントロピー ≈ 2.336 bits) にも接近しており、ハフマン符号化の最適性が確認されました。

2 課題 2 概要

このレポートでは、画像処理・画像処理工学の課題 2 に取り組みます。以下の 4 つの問題について、理論、結果、考察を含めて報告します。

3 問題 1: モルフォロジー処理によるノイズ除去

3.1 理論

モルフォロジー処理は、二値画像に対して幾何学的な形態操作を行う処理です。主な操作には以下があります：

- 膨張 (Dilation)：画像内の白領域を拡大する処理
- 収縮 (Erosion)：画像内の白領域を縮小する処理
- 開処理 (Opening)：収縮の後に膨張を行う処理。小さなノイズを除去する
- 閉処理 (Closing)：膨張の後に収縮を行う処理。小さな黒いノイズを埋める

3.2 結果

開処理により小さな孤立ノイズが効果的に除去されました。閉処理により、黒いノイズも埋められました。

3.3 考察

開処理と閉処理を組み合わせることで、効果的にノイズを除去できます。構造要素のサイズを調整することで、除去するノイズの大きさを制御できます。

4 問題 2: JPEG 品質と圧縮率の関係

4.1 理論

JPEG 圧縮では、品質パラメータ (0-100) により圧縮率と画質が変化します。品質が低いほど圧縮率は高くなりますが、画質が低下します。SSIM (Structural Similarity Index) を用いて画質を定量的に評価できます：

$$\text{SSIM} = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

4.2 結果

JPEG 品質と圧縮率の関係を分析すると、品質 70～80 では十分な画質を保ちながら高い圧縮率を達成できます。

4.3 考察

推奨される JPEG 品質は 75～85 の範囲です。この範囲では、視覚的な品質低下が最小限に抑えられながら、データ圧縮率が 30% 程度達成できます。

5 問題 3: 2 次元 FFT と振幅スペクトル

5.1 理論

2 次元フーリエ変換は、画像を周波数領域に変換します。振幅スペクトルは周波数成分の大きさを表します。対数スケール変換により、弱い周波数成分も可視化できます：

$$S(\omega_x, \omega_y) = \log(1 + |F(\omega_x, \omega_y)|)$$

5.2 結果

振幅スペクトルより、低周波成分が中心に集中していることが観察されました。

5.3 考察

自然画像では通常、低周波成分が支配的です。スペクトルの分布から、画像の周波数特性が理解できます。

6 問題 4: 周波数フィルタの応用

6.1 理論

周波数フィルタは周波数領域で画像を処理します。主なフィルタ種類：

- ローパスフィルタ：低周波を通す。ノイズ除去に使用
- ハイパスフィルタ：高周波を通す。エッジ検出に使用
- 理想フィルタ：遮断周波数で急峻に変化
- ガウシアンフィルタ：滑らかに変化。リングングが少ない

6.2 結果

ローパスフィルタはノイズを除去して画像を平滑化し、ハイパスフィルタはエッジを強調しました。

6.3 考察

ガウシアンフィルタは理想フィルタと異なり、周波数応答が滑らかに変化するため、逆 FFT 後のリングング成分が少なく、実用的です。カットオフ周波数の選択により、処理結果の特性を制御できます。

付録: プログラムリスト

本レポートの課題2で使った Python プログラムを以下に示す.

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/1KwcAHJlLwDpZRJE0FL1gEOUwZf-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めているか確認（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 +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()
70
71 # =====
72 # 課題4: 2種類以上の周波数フィルタの適用と考察
73 # =====
74
75 import cv2
76 import numpy as np
77 import matplotlib.pyplot as plt
78 from google.colab import drive
79
80 # 1. ドライブのマウントと画像の読み込み
81 # drive.mount('/content/drive') # すでにマウント済みの場合はコメントアウトでOK
82
83 common_path = '/content/drive/MyDrive/img2025/image/'
84 filename = 'a2-5_gray_image.png'
85
86 # 画像読み込み
87 gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)
88
89 # 2. 画像の正方形トリミング（課題3の処理を継承）
90 # 画像の中心から512x512を切り出す
91 h, w = gray_img.shape
92 crop_size = 512
93
94 # 画像サイズが小さい場合のガード処理
95 target_size = min(h, w, crop_size)

```

```

96
97 center_y, center_x = h // 2, w // 2
98 start_y = center_y - target_size // 2
99 start_x = center_x - target_size // 2
100 img = gray_img[start_y:start_y+target_size, start_x:start_x+target_size]
101
102 # 3. FFTの実行とシフト
103 f = np.fft.fft2(img)
104 fshift = np.fft.fftshift(f) # 直流成分を中心に移動
105
106 # -----
107 # フィルタ生成関数の定義
108 # -----
109
110 def create_ideal_lowpass(shape, cutoff):
111     """
112     理想ローパスフィルタ: 指定した半径(cutoff)の内側を通し、外側をカット
113     """
114     rows, cols = shape
115     crow, ccol = rows // 2, cols // 2
116
117     y, x = np.ogrid[:rows, :cols]
118     # 中心からの距離の二乗を計算
119     dist_sq = (x - ccol)**2 + (y - crow)**2
120
121     mask = np.zeros(shape)
122     # 距離がcutoff以内なら1
123     mask[dist_sq <= cutoff**2] = 1
124     return mask
125
126 def create_gaussian_highpass(shape, cutoff):
127     """
128     ガウシアンハイパスフィルタ: 低周波を滑らかに減衰させ、高周波を通す
129      $H(u,v) = 1 - \exp(-D^2 / 2D_0^2)$ 
130     """
131     rows, cols = shape
132     crow, ccol = rows // 2, cols // 2
133
134     y, x = np.ogrid[:rows, :cols]
135     dist_sq = (x - ccol)**2 + (y - crow)**2
136
137     # ガウシアンローパスの逆 (1から引く)
138     mask = 1 - np.exp(-dist_sq / (2 * (cutoff**2)))
139     return mask
140
141 # -----
142 # フィルタの適用
143 # -----
144

```

```

145 # パラメータ設定 (遮断周波数)
146 D0_low = 30 # ローパス用
147 D0_high = 30 # ハイパス用
148
149 # マスクの作成
150 mask_ilpf = create_ideal_lowpass(img.shape, D0_low)
151 mask_ghpf = create_gaussian_highpass(img.shape, D0_high)
152
153 # 周波数領域でのフィルタリング (要素ごとの掛け算)
154 fshift_ilpf = fshift * mask_ilpf
155 fshift_ghpf = fshift * mask_ghpf
156
157 # 逆FFT処理 (シフトを戻して逆変換し、実部を取る)
158 img_ilpf = np.fft.ifft2(np.fft.ifftshift(fshift_ilpf)).real
159 img_ghpf = np.fft.ifft2(np.fft.ifftshift(fshift_ghpf)).real
160
161 # -----
162 # 結果の可視化
163 # -----
164 plt.figure(figsize=(12, 8))
165
166 # --- 上段: 理想ローパスフィルタ ---
167 plt.subplot(2, 3, 1)
168 plt.title('Original Image')
169 plt.imshow(img, cmap='gray', vmin=0, vmax=255)
170 plt.axis('off')
171
172 plt.subplot(2, 3, 2)
173 plt.title(f'Ideal Low Pass Filter Mask\n(Cutoff={D0_low})')
174 plt.imshow(mask_ilpf, cmap='gray', vmin=0, vmax=1)
175 plt.axis('off')
176
177 plt.subplot(2, 3, 3)
178 plt.title('Filtered Result (ILPF)')
179 plt.imshow(img_ilpf, cmap='gray') # 値の範囲は自動調整
180 plt.axis('off')
181
182 # --- 下段: ガウシアンハイパスフィルタ ---
183 plt.subplot(2, 3, 4)
184 plt.title('Original Image')
185 plt.imshow(img, cmap='gray', vmin=0, vmax=255)
186 plt.axis('off')
187
188 plt.subplot(2, 3, 5)
189 plt.title(f'Gaussian High Pass Filter Mask\n(Cutoff={D0_high})')
190 plt.imshow(mask_ghpf, cmap='gray', vmin=0, vmax=1)
191 plt.axis('off')
192
193 plt.subplot(2, 3, 6)

```

```
194 plt.title('Filtered Result (GHPF)')
195 plt.imshow(np.abs(img_ghpf), cmap='gray') # 振幅を表示
196 plt.axis('off')
197
198 plt.tight_layout()
199 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