# グラフ信号処理
# 情報通信工学演習課題

2024 年 7 月 15 日

## 1　グラフの作成方法

今回のグラフの対象は画像データであり、通常格子状で表現される。そのため、画像の各ピクセルの上下左右のピクセルと連結させたグラフを構築した。

また、ピクセル毎に輝度値が割り振られているため、上下左右でのピクセル間の輝度値の差分 $+1$ をグラフの辺の重みとした。1 は輝度値の差が 0 であった場合に連結されてないとみなされないためのバイアス項である。

実装では *adjacency_matrix* 関数での *to_abs* 関数が辺の重み付けを行っている。

## 2　低域通過フィルタの応答

低域通過フィルタの応答を以下に示す。グラフ周波数領域で、$\lambda$ が一定値を超えた際に出力が 0 になるようなステップ関数として実装した。画像では Original の波形である。しきい値が 3 であり、$\lambda < 3$ であるあいだは 1 を返し、$\lambda \geq 3$ では 0 を返す。

$k = 5, 10, \cdots, 25, 50$ はチェビシェフ多項式近似による低域通過フィルタであり、$k$ は近似式の次数を表す。

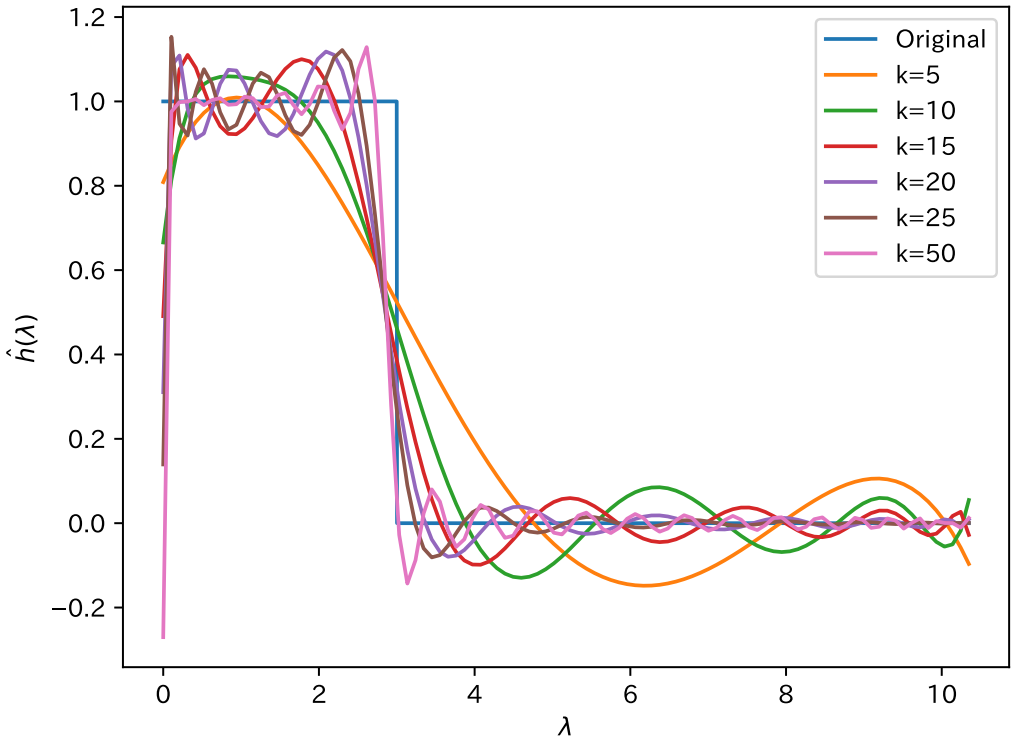


図 1　グラフ周波数領域での低域通過フィルタ

## 3　入力画像

入力画像は MNIST データセットを使用した。

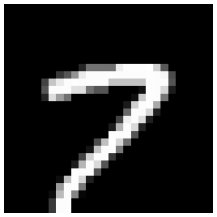

図 2　入力画像

## 4　出力画像

入力画像に低域通過フィルタを適用した結果を以下に示す。

以下はチェビシェフ多項式近似による低域通過フィルタを適用した際の画像である。次数が増えるにつれ図 1 でも見られるような振動が画像に含まれているのが見て取れる。

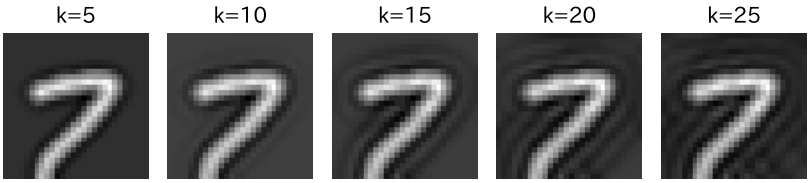


図 3　次数を変化させた出力画像

以下は低域通過フィルタのしきい値を変化させた際の画像である。各画像の上の数値は 2 低域通過フィルタの応答 で示したしきい値である。

$\lambda$ の値が大きくなるにつればやけていたエッジが明確になっていく、つまり高周波な信号が含まれていることが確認できる。

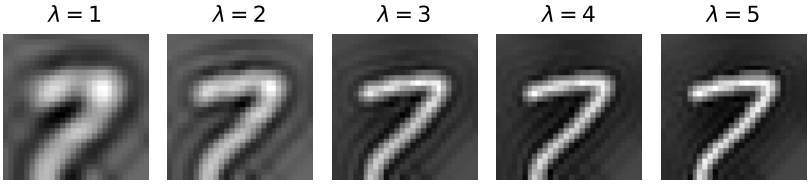


図 4　出力画像

## 5　参考文献

田中雄一. グラフ信号処理の基礎と応用. コロナ社, 2023.

“PyGSP: Graph Signal Processing in Python”.PyGSP.2017.https://pygsp.readthedocs.io/en/stable/index.html,(参照 2024-07-07)

## 6　ソースコード

実験に使用したプログラムを以下に示す。ソースコード 1 の load_mnist.py では mnist データセットを取得している。ソースコード 2 の main.py に含まれる *GraphLowPassFilter* がチェビシェフ多項式近似による低域通過フィルタの処理を行っている。

ソースコード 1　load_mnist.py

```
import inspect
import numpy as np
import os
from sklearn import datasets
# from sklearn.model_selection import train_test_split


def to_abs_path(fpath):
    dirpath = get_caller_path()
    return os.path.normpath(os.path.join(dirpath, fpath))


def get_caller_path() -> str:
    """

    Returns
    -------
    str
        _description_
    """
    caller_frame = inspect.stack()[-1]
    caller_module = inspect.getmodule(caller_frame[0])

    caller_module_path = os.path.abspath(caller_module.__file__)
    return os.path.dirname(caller_module_path)


def mnist(datapath = './data', dtype = None):
    """import mnist dataset from PyTorch

    Parameters
```

```
    ----------
    datapath : str, optional
        Description of download folder path, by default './data'

    Returns
    -------
    traindata : ndarray
    testdata : ndarray
    trainlabel : ndarray
    testlabel : ndarray

    Examples
    --------
    >>> traindata, testdata, trainlabel, testlabel = mnist_from_torch()
    >>> assert traindata.shape == (60000, 784)
    >>> assert testdata.shape == (10000, 784)
    >>> assert trainlabel.shape == (60000,)
    >>> assert testlabel.shape == (10000,)
    """
    from torchvision.datasets import MNIST

    trainset = MNIST(to_abs_path(datapath), train=True, download=True)
    testset = MNIST(to_abs_path(datapath), train=False, download=True)
    traindata:np.ndarray = trainset.data.detach().numpy().copy().reshape(-1, 784)
    testdata: np.ndarray = testset.data.detach().numpy().copy().reshape(-1, 784)

    trainlabel:np.ndarray = trainset.targets.detach().numpy().copy()
    testlabel :np.ndarray = testset.targets.detach().numpy().copy()

    if dtype is not None:
        return (
            traindata.astype(dtype),
            testdata.astype(dtype),
            trainlabel.astype(dtype),
            testlabel.astype(dtype))

    return traindata, testdata, trainlabel, testlabel



def digits():
    digits = datasets.load_digits()
    X = digits.data
    y = digits.target
    return X, y
```

ソースコード 2　main.py

```
import matplotlib.pyplot as plt
import japanize_matplotlib
import numpy as np
import warnings

from numpy.typing import NDArray
from typing import Annotated, Callable

import load_mnist


Vector = Annotated[NDArray[np.float64], "1D"]
Matrix = Annotated[NDArray[np.float64], "2D"]


def to_abs(x, y, /, *, bias = 0):
    with warnings.catch_warnings():
        warnings.filterwarnings("error")
        try:
            return bias + (x-y if x-y >= 0 else y-x)
        except Warning as e:
            print(f"{x=}{x.dtype} {y=}{y.dtype}")
            raise Exception(e)


def normalize(x:np.ndarray) -> np.ndarray:
    return x / 255


def adjacemcy_matrix(x, width = None) -> np.ndarray:
    length = len(x)
    if width is None:
        width = int(np.sqrt(length))
    height = length // width

    a = np.zeros([length, length])

    for i in range(length):
        row, column = divmod(i, width)

        if row != 0:         a[i, i-width] = a[i-width, i] = to_abs(x[i], x[i-
            width], bias=1)
        if row != height-1:  a[i, i+width] = a[i+width, i] = to_abs(x[i], x[i+
            width], bias=1)
        if column != 0:      a[i, i-1]     = a[i-1, i]     = to_abs(x[i], x[i-1],
```

```
            bias=1)
        if column !=width-1: a[i, i+1]     = a[i+1, i]     = to_abs(x[i], x[i+1],
            bias=1)
    return a


def degree_matrix(a:np.ndarray) -> np.ndarray:
    if len(a.shape) != 2:
        assert "not 2 dimentional matrix"

    diag_list = [np.sum(i) for i in a]
    return np.diag(diag_list)


class GraphLowPassFilter:
    def __init__(self,
                 x: Vector,
                 filter: Callable[[float], float],
                 L: Matrix,
                 kmax: int = 100,
                 threshold = 3
                 ) -> None:
        self._x = x
        self.filter = filter
        self._L = L
        self.lmax = np.linalg.eigvalsh(L)[-1]
        self._kmax = kmax
        self._thred = threshold
        self._cl_list = self._compute_cl()
        self._tl_list = self._compute_tl()
        print(f"lambda max = {self.lmax}")

    def apply_filter(self, k):
        y = 0
        for i in range(k):
            y += self._cl_list[i] * self._tl_list[i]
        return y

    def filter_response(self, k):
        lamda = np.linspace(0, self.lmax, 100)

        tl_list = [1, 2*lamda/self.lmax-1]
        for i in range(2, self._kmax):
            tl_list.append(
                2*(2*lamda/self.lmax - 1)*tl_list[i-1] - tl_list[i-2]
                )
```

```
88          # y = self._integrate_cl(0)/2
89          y = self._cl_list[0]
90          for i in range(1, k):
91              y += self._cl_list[i] * tl_list[i]
92              # y += self._integrate_cl(i) * tl_list[i]
93          return lamda, y
94
95      def _compute_cl(self) -> dict[float]:
96          cl_list = [self._integrate_cl(0)/2]
97
98          for i in range(1, self._kmax):
99              cl_list.append(self._integrate_cl(i))
100         return cl_list
101
102     def _integrate_cl(self, l, k = None):
103         if k is None:
104             k = self._kmax
105         k_s = k+1
106         cl = 0
107         for p in range(1, k_s):
108             theta_p = (np.pi)/k_s * (p-0.5)
109             cl += np.cos(l*theta_p) * self.filter(
110                 self.lmax/2 * (np.cos(theta_p) + 1), self._thred
111                 )
112         return 2/k_s * cl
113
114     def _compute_tl(self):
115         n_dim = len(self._x)
116         tl_list = [self._x, (2*self._L/self.lmax - np.eye(n_dim)) @ self._x]
117         for i in range(2, self._kmax):
118             tl_list.append(
119                 2*(2*self._L/self.lmax - np.eye(n_dim))@tl_list[i-1] - tl_list[i
                        -2]
120                 )
121         return tl_list
122
123
124 def func_h(lamda, thred=3):
125     return 1 if lamda < thred else 0
126
127
128 def draw_polynomial(graph:GraphLowPassFilter):
129     x = np.linspace(0, graph.lmax, 10000)
130     plt.plot(x, list(func_h(i, 3) for i in x), label="Original")
131     for i in range(5, 26, 5):
132         plt.plot(*graph.filter_response(i), label=f"k={i}")
```

```
    plt.plot(*graph.filter_response(50), label=f"k={50}")
    plt.legend()
    plt.xlabel("$\lambda$")
    plt.ylabel(r"$\hat{h}(\lambda)$")
    plt.savefig("filter_response.pdf")
    plt.close()


def main():
    traindata, *_ = load_mnist.mnist(dtype=np.int16)
    data = normalize(traindata[123])
    A = adjacemcy_matrix(data)
    D = degree_matrix(A)
    L = D - A

    plt.imshow(data.reshape(int(np.sqrt(data.shape[0])), -1), cmap="gray")
    plt.axis("off")
    plt.savefig("base_image.pdf")
    plt.close()

    graph_filter = GraphLowPassFilter(data, func_h, L, kmax=50, threshold=3)
    draw_polynomial(graph_filter)
    fig, ax = plt.subplots(1, 5)
    for i, j in enumerate(range(5, 26, 5)):
        res = graph_filter.apply_filter(j)
        ax[i].imshow(res.reshape(int(np.sqrt(data.shape[0])), -1), cmap="gray")
        ax[i].axis("off")
        ax[i].set_title(f"k={j}")
    plt.tight_layout()
    plt.savefig("output.pdf")
    plt.close()

    fig, ax = plt.subplots(1, 5)
    for i in range(1, 6):
        graph_filter = GraphLowPassFilter(data, func_h, L, kmax=20, threshold=i)
        res = graph_filter.apply_filter(20)
        ax[i-1].imshow(res.reshape(int(np.sqrt(data.shape[0])), -1), cmap="gray")
        ax[i-1].axis("off")
        ax[i-1].set_title(f"$\lambda={i}$")
    plt.tight_layout()
    plt.savefig("thershold.pdf")


if __name__=="__main__":
    main()
```