

## 第5回 日本の気象と次元削減・特徴量抽出

### 実施内容

1. 顔画像で次元削減・特徴量抽出
2. 日本の降水量データで次元削減・特徴量抽出 どのような特徴量が抽出されるか、手法毎の特性を把握する。

### ▼ 使用する次元削減・特徴量抽出手法

#### 特異値分解 (Singular Value Decomposition, SVD)

元データ:  $\mathbf{X} = [\mathbf{x}_1 \mathbf{x}_2 \dots \mathbf{x}_N] \in \mathbb{R}^{M \times N}$

$$\begin{aligned}\mathbf{X} &= \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T \\ \mathbf{U} &\in \mathbb{R}^{M \times M} \\ \boldsymbol{\Sigma} &= \text{diag} [\sigma_1 \sigma_2 \dots \sigma_{\min(M,N)}] \in \mathbb{R}^{M \times N} \\ \mathbf{V} &\in \mathbb{R}^{N \times N}\end{aligned}$$

#### SVD の特徴

- SVD はデータの軸 ( $x, y, z, \dots$ ) を取り直す手法
- $\mathbf{U}$  は  $\mathbf{X}$  の行方向、 $\mathbf{V}$  は列方向を圧縮したモードを表す。
- 軸は、その軸に対するデータの分散が最も大きくなるものから順に選ばれる
- つまり、第1軸が最も分散が大きくなる
  - = 最も分散が大きい中の真ん中を貫く軸
  - = 重要度の高いモード (重要度の大きさは  $\sigma_i$  で表される)
- 下位のモードは重要度が低いため、捨てても元データの中身を損ねない
  - = 次元削減

#### 非負値行列因子展開 (Non-negative Matrix Factorization, NMF)

$r$  をランク数として、

$$\begin{aligned}\mathbf{X} &\simeq \mathbf{W} \mathbf{H} \\ \mathbf{W} &\in \mathbb{R}^{M \times r} \\ \mathbf{H} &\in \mathbb{R}^{r \times N}\end{aligned}$$

ここで  $\mathbf{W}$  は  $r$  枚の基底画像、 $\mathbf{H}$  は基底画像を足し合わせて元画像を復元するための係数を表す行列である。

あるいは

- $\mathbf{x}_j$  を  $\mathbf{X}$  の中の1サンプル (列)
- $\mathbf{w}_i$  を  $\mathbf{W}$  の各列
- $h_{ij}$  を基底画像・サンプルのIDに対応する  $\mathbf{H}$  の中の1要素

として、

$$\mathbf{x}_j \simeq \sum_{i=1}^r h_{ij} \mathbf{w}_i$$

#### NMF の特徴

- NMF は元データ・分解後のモードが非負となる制約をかけた分解方法
- 非負の制約の上で、 $r$  個の基底画像で行列を分解
- 非負制約のため、元画像復元時に "引き算" ができないため、各基底画像は疎なイメージになりやすい。

```
# まず cartopy をインストール
!apt-get install libproj-dev proj-data proj-bin libgeos-dev
!pip install cartopy
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libgeos-dev is already the newest version (3.12.1-1~jammy0).
libgeos-dev set to manually installed.
libproj-dev is already the newest version (9.3.1-1~jammy0).
```

```

libproj-dev set to manually installed.
proj-data is already the newest version (9.3.1-1~jammy0).
proj-data set to manually installed.
The following NEW packages will be installed:
  proj-bin
0 upgraded, 1 newly installed, 0 to remove and 38 not upgraded.
Need to get 205 kB of archives.
After this operation, 521 kB of additional disk space will be used.
Get:1 https://ppa.launchpadcontent.net/ubuntu/+ppa/ubuntu jammy/main amd64 proj-bin amd64 9.3.1-1~jammy0 [205 kB]
Fetched 205 kB in 1s (158 kB/s)
Selecting previously unselected package proj-bin.
(Reading database ... 126675 files and directories currently installed.)
Preparing to unpack .../proj-bin_9.3.1-1~jammy0_amd64.deb ...
Unpacking proj-bin (9.3.1-1~jammy0) ...
Setting up proj-bin (9.3.1-1~jammy0) ...
Processing triggers for man-db (2.10.2-1) ...
Collecting cartopy
  Downloading cartopy-0.25.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (6.1 kB)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.12/dist-packages (from cartopy) (2.0.2)
Requirement already satisfied: matplotlib>=3.6 in /usr/local/lib/python3.12/dist-packages (from cartopy) (3.10.0)
Requirement already satisfied: shapely>=2.0 in /usr/local/lib/python3.12/dist-packages (from cartopy) (2.1.2)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.12/dist-packages (from cartopy) (25.0)
Requirement already satisfied: pyshp>=2.3 in /usr/local/lib/python3.12/dist-packages (from cartopy) (3.0.2.post1)
Requirement already satisfied: pyproj>=3.3.1 in /usr/local/lib/python3.12/dist-packages (from cartopy) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.6->cartopy) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.6->cartopy) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.6->cartopy) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.6->cartopy) (1.4.9)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.6->cartopy) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.6->cartopy) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.6->cartopy) (2.9.0.post0)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from pyproj>=3.3.1->cartopy) (2025.10.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib>=3.6->cartopy) (1.1.0)
Downloading cartopy-0.25.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (11.8 MB)
11.8/11.8 MB 63.7 MB/s eta 0:00:00

```

Installing collected packages: cartopy  
Successfully installed cartopy-0.25.0

```

# モジュールのインポート
from datetime import datetime, timedelta
import os

from cartopy.mpl.ticker import LongitudeFormatter, LatitudeFormatter
from matplotlib import pyplot as plt
from sklearn.decomposition import NMF
from sklearn.preprocessing import minmax_scale
import cartopy.crs as ccrs
import numpy as np

```

## 1. まず顔画像に対して次元削減・徳量量抽出を行ってみる

### 参考文献

SL Brunton & JN Kutz (2022): Data-Driven Science and Engineering Machine Learning, Dynamical Systems, and Control

<https://www.cambridge.org/highereducation/books/data-driven-science-and-engineering/6F9A730B7A9A9F43F68CF21A24BEC339>

### 使用するデータセット: the Extended Yale Face Database B Cropped

提供元のページがクラッシュしているため、下記のリンクより取得する。

<https://www.kaggle.com/datasets/tbourton/extyalebcroppedpng>

このデータセットは元々、38人の顔画像が、各被験者につき64枚、照明を変化・ポーズも変えながら、様々な条件下で撮影して作成されたものである。

本演習で使用するデータセットは、その元データセットから、正面向きの顔部分のみを切り出して整形したデータセットである。

一つ一つの顔画像は、 $192 \times 168$  ピクセルのグレースケールの画像である。

```

!curl -L -o extyalebcroppedpng.zip¥
· https://www.kaggle.com/api/v1/datasets/download/tbourton/extyalebcroppedpng
!unzip extyalebcroppedpng.zip -d extyalebcroppedpng

```

```

inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+025E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+035E+15.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+035E+40.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+035E+65.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+035E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+050E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+050E+40.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+060E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+060E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+070E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+070E+45.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+070E+35.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+085E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+085E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+095E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+110E+15.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+110E+40.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+110E+65.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+120E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A+130E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-005E+10.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-005E-10.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-010E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-010E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-015E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-020E+10.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-020E-10.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-020E+40.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-025E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-035E+15.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-035E+40.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-035E+65.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-035E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-050E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-050E+40.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-060E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-060E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-070E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-070E+45.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-070E+35.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-085E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-085E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-095E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-110E+15.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-110E+40.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-110E+65.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-120E+00.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00A-130E+20.png
inflating: extyalebcroppedpng/CroppedYalePNG/yaleB39_P00 Ambient.png

```

## ▼ 顔画像データを読み込む

すべてのデータを読み込むと、データサイズが大きくなってしまいます。

ですので、一人当たりの使用する画像枚数を制限し、処理の高速化を図ります。

下記のコードブロックの n は一人当たりの使用する画像の枚数を表す変数ですので、適宜この変数を変更してください。

```

# パラメータ（適宜変更してください）
# 一人につき基本的に64パターンの顔画像があるので、一人につき n 枚だけを使用する
n = 32

```

## ▼ 実際に顔画像データを読み込み + 訓練・テストに分割

合計38人の顔画像がありますが、最後の2人のデータは特徴量抽出に使用せず、テスト用に取っておくことにします。

```

# 顔画像ファイル一覧を取得
dir_faces = 'extyalebcroppedpng/CroppedYalePNG'
f_faces = os.listdir(dir_faces)
f_faces.sort()
# f_faces = [f for f in f_faces if 'Ambient' not in f]
f_faces = [f for f in f_faces if 'Ambient' not in f]
print('Total number of facial images = ', len(f_faces))

# 一人につき基本的に64パターンの顔画像があるので、一人につき最初の n 枚だけを使用する（多少欠損あり）
# また、最初の36人分を訓練データ、最後の2人の最初の画像をテストデータとする
f_train = []
f_test = []
for i in range(1, 40):
    file_name_head = f'yaleB{i:02}'
    f_person = []
    for f in f_faces:

```

```

if file_name_head in f:
    f_person.append(f)
if i < 38:
    if len(f_person) == n:
        f_train.extend(f_person)
        break
else:
    f_test.extend(f_person)
    break

print('The number of facial images for training = ', len(f_train))
print('The number of facial images for test      = ', len(f_test))

train_faces = []
for f in f_train:
    face = plt.imread(f'{dir_faces}/{f}')
    face = minmax_scale(face.ravel(), feature_range=(0, 1)).reshape(face.shape)
    train_faces.append(face)
train_faces = np.stack(train_faces)
print('shape of training data: ', train_faces.shape)

test_faces = []
for f in f_test:
    face = plt.imread(f'{dir_faces}/{f}')
    face = minmax_scale(face.ravel(), feature_range=(0, 1)).reshape(face.shape)
    test_faces.append(face)
test_faces = np.stack(test_faces)
print('shape of test data: ', test_faces.shape)

face_shape = train_faces.shape[1:]

# 一人につき最初の1枚の顔画像をまとめて表示（14番目の人は欠損）
n_row, n_col = 6, 7
multi_faces = np.full((face_shape[0] * n_row, face_shape[1] * n_col), np.nan)
for j in range(n_row):
    for i in range(n_col):
        image_id = j * n_col + i
        file_name_head = f'yaleB{image_id+1:02}'
        for f in f_faces:
            if file_name_head in f:
                if image_id < len(f_faces):
                    face = plt.imread(f'{dir_faces}/{f}')
                    face = minmax_scale(face.ravel(), feature_range=(0, 1)).reshape(face.shape)
                    multi_faces[j*face_shape[0] : (j+1)*face_shape[0], i*face_shape[1] : (i+1)*face_shape[1]] = face
                    break

plt.figure()
plt.imshow(multi_faces, cmap='grey')
plt.axis("off")
plt.show()

```

Total number of facial images = 2414  
The number of facial images for training = 1152  
The number of facial images for test = 2  
shape of training data: (1152, 192, 168)  
shape of test data: (2, 192, 168)



## ▼ 顔画像に次元削減を適用する

直下のコードブロックの中の n\_components はチューニングパラメータです。

これは次元削減後の次元数ですので、値を適宜変更して実行してみてください。

```
# 次元削減後の次元数（モード数）
n_components =49
n_components =32 # for rimosen enfinnering

# 行列形式へ変形
X = train_faces.reshape(train_faces.shape[0], -1).T
print(f'Matrix shape: {train_faces.shape} -> {X.shape}')

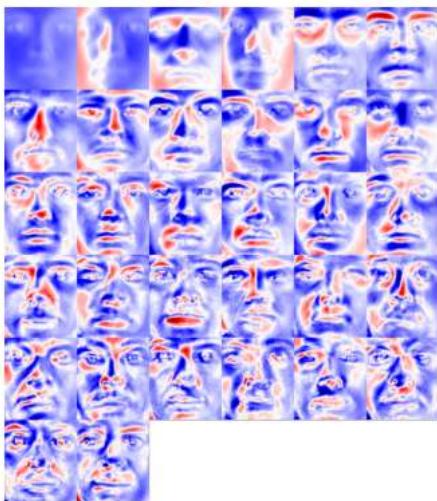
# SVD
U, S, VT = np.linalg.svd(X, full_matrices=False)

# Dimensionality reduction
Ur, Sr, Vr = U[:, :n_components], S[:n_components], VT.T[:, :n_components]

# Reshape Ur for plotting
Ur_reshaped = Ur.reshape(face_shape[0], face_shape[1], n_components)

# 基底画像をまとめて表示
n_row, n_col = round(np.sqrt(n_components)), int(np.ceil(np.sqrt(n_components)))
multi_images = np.full((face_shape[0] * n_row, face_shape[1] * n_col), np.nan)
for j in range(n_row):
    for i in range(n_col):
        image_id = j*n_col+i
        if image_id < n_components:
            multi_images[j*face_shape[0] : (j+1)*face_shape[0], i*face_shape[1] : (i+1)*face_shape[1]] = Ur_reshaped[:, :, image_id]
plt.figure()
plt.imshow(multi_images, cmap='seismic')
plt.axis("off")
# plt.colorbar(location='bottom', orientation='horizontal', aspect=60, extend='both')
plt.show()
```

Matrix shape: (1152, 192, 168) -> (32256, 1152)



## ▼ NMF を顔画像に適用してみる

- 比較的計算が重いので、注意すること
- 一人につき32枚の画像を使用したNMFでは、16分かかりました。
- 基底画像の数を49とすると、顔のパーツが基底画像として出力されます。
- 元画像の数と比べて少ない数の基底画像で全体を表そうとすると、情報圧縮度合いが大きくなるため、顔のパーツが出力されるようになります。
- 一方、例えば一人につき1枚の画像のみを使用して、それに近い数の基底画像を使用するとすると、一つの基底画像につき一人の顔を概ね反映させればよく、顔のパーツに分解されません。

```
# NMF
model = NMF(n_components=n_components, init='nndsvd', max_iter=5000)
model.fit(X)
W = model.transform(X)
H = model.components_

# Reshape W for plotting
W_reshaped = W.reshape(face_shape[0], face_shape[1], n_components)

# 基底画像をまとめて表示
```

```

n_row, n_col = round(np.sqrt(n_components)), int(np.ceil(np.sqrt(n_components)))
multi_images = np.full((face_shape[0] * n_row, face_shape[1] * n_col), np.nan)
for j in range(n_row):
    for i in range(n_col):
        image_id = j * n_col + i
        if image_id < n_components:
            W_scaled = W_reshaped[:, :, image_id]
            W_scaled /= np.max(W_scaled)
            multi_images[j*face_shape[0] : (j+1)*face_shape[0], i*face_shape[1] : (i+1)*face_shape[1]] = W_scaled
plt.figure()
plt.imshow(multi_images, cmap='Greys')
plt.axis("off")
plt.show()

```



## ▼ 抽出した特徴量から訓練していない顔を再構成

- 訓練に使用していない顔画像も、抽出したモード（基底画像）一つ一つに係数をかけ合わせて、すべて足し合わせれば、近似して表現することができる。
- 実際に抽出したモード（基底画像）で近似してみよう。
- そのためにはまず各モード（基底画像）に掛け合わせる係数を求める。

### SVD でのモードの係数の求め方

- テスト画像を「モード × 係数」で近似することを式で表すと、下記のようになる。

$$\mathbf{x}_{test} \simeq \mathbf{U}_r \mathbf{a}$$

- ここで係数ベクトル  $\mathbf{a}$  を求めたい。
- そのためには、SVD で得られるモードの各列は直交するため、転置をとって左から作用させれば簡単に求められる。

$$\mathbf{a} = \mathbf{U}_r^T \mathbf{x}_{test}$$

- あくまでこのようにして求められた  $\mathbf{a}$  はテスト画像を近似的に表すための係数であることに注意。
- あとはこの  $\mathbf{a}$  をモードに掛け合わせれば、テスト画像を抽出した特徴量を使って著した結果を確認することができる。

### NMF での基底画像の係数の求め方

- SVD と同様、テスト画像を「基底画像 × 係数」で近似することを式で表すと、下記のようになる。

$$\mathbf{x}_{test} \simeq \mathbf{W} \mathbf{h}$$

- ここで係数ベクトル  $\mathbf{h}$  を求めたい。
- ただし SVD と異なり、 $\mathbf{W}$  の各列は直交しないため、ムーア・ペンローズの疑似逆行列を使用して求める。

$$\mathbf{h} = \mathbf{W}^\dagger \mathbf{x}_{test}$$

- 後の操作は SVD と同様である。

```

n_test = test_faces.shape[0]
# n_test = 1
for i_test in range(n_test):
    # SVD のモードで近似
    a = Ur.T @ test_faces[i_test].reshape(-1, 1)
    x_test_svd = Ur @ a
    x_test_svd = x_test_svd.reshape(test_faces[i_test].shape)

    # NMF の基底画像で近似
    h = np.linalg.pinv(W) @ test_faces[i_test].reshape(-1, 1)
    x_test_nmf = W @ h
    x_test_nmf = x_test_nmf.reshape(test_faces[i_test].shape)

```

```
# 全モードを使って近似すると？
a_full = U.T @ test_faces[i_test].reshape(-1, 1)
x_test_full = U @ a_full
x_test_full = x_test_full.reshape(test_faces[i_test].shape)

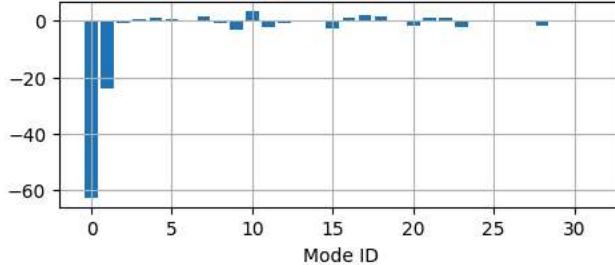
fig, axes = plt.subplots(1, 4, figsize=(10, 20))
axes[0].imshow(test_faces[i_test], cmap='gray')
axes[0].set_title('Original')
axes[0].axis('off')
axes[1].imshow(x_test_svd, cmap='gray')
axes[1].set_title('Reconstructed (SVD)')
axes[1].axis('off')
axes[2].imshow(x_test_nmf, cmap='gray')
axes[2].set_title('Reconstructed (NMF)')
axes[2].axis('off')
axes[3].imshow(x_test_full, cmap='gray')
axes[3].set_title('with full mode')
axes[3].axis('off')
plt.show()

# 各モード・基底画像の重みを棒グラフで表示
fig, axes = plt.subplots(1, 2, figsize=(12, 2))
axes[0].bar(np.arange(n_components), a.reshape(-1))
axes[0].set_title('Weight of SVD modes')
axes[0].set_xlabel("Mode ID")
axes[0].grid(True)
axes[1].bar(np.arange(n_components), h.reshape(-1))
axes[1].set_title('Weight of NMF basis images')
axes[1].set_xlabel("Basis image ID")
axes[1].grid(True)
plt.show()

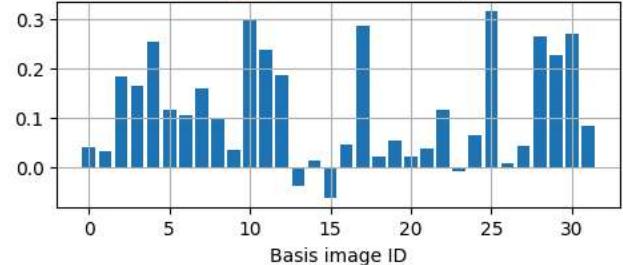
# どちらの分解方法で近似した画像が元画像によく近似できているか、Root Mean Square Error (RMSE) で評価
rmse_svd = np.sqrt(np.mean((test_faces[i_test] - x_test_svd) ** 2))
rmse_nmf = np.sqrt(np.mean((test_faces[i_test] - x_test_nmf) ** 2))
print(f'RMSE (SVD) = {rmse_svd:.3f}')
print(f'RMSE (NMF) = {rmse_nmf:.3f}'')
```



Weight of SVD modes



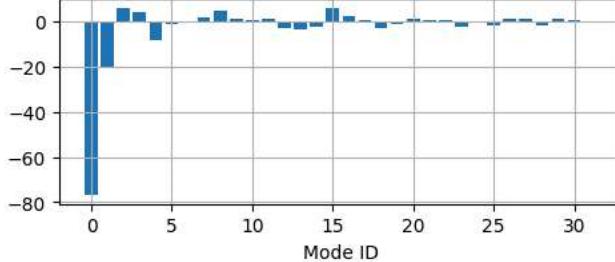
Weight of NMF basis images



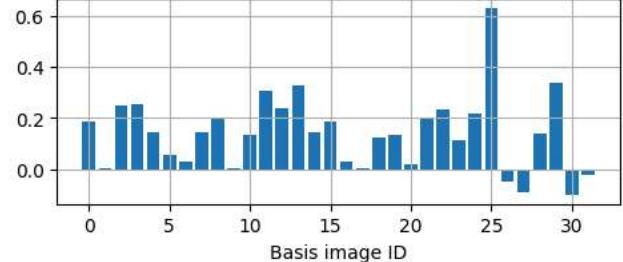
RMSE (SVD) = 0.056525  
RMSE (NMF) = 0.057607



Weight of SVD modes



Weight of NMF basis images



RMSE (SVD) = 0.087010  
RMSE (NMF) = 0.085585

## ▼ 顔画像以外の画像もモード・基底画像で近似してみる

上述の近似手法を用いることで、例えば犬の画像であってもモード・基底画像で近似的に表現することができます。もちろん、精度はそれほど出ませんが、遠目で見ると確かに顔と犬が合成されたような画像を得ることができます。

```

import requests
from PIL import Image
from io import BytesIO

# 公開URLから犬の画像をダウンロード
url = "https://images.unsplash.com/photo-1518717758536-85ae29035b6d" # 犬の写真
# url = "https://images.unsplash.com/photo-1517849845537-4d257902454a" # 他の犬の写真
response = requests.get(url)
dog = Image.open(BytesIO(response.content))

# グレースケールに変換
dog_gray = dog.convert("L")

# 顔画像に合わせて解像度を落とす
target_shape = test_faces[i_test].shape
dog_resized = dog_gray.resize((target_shape[1], target_shape[0]), Image.BILINEAR)

# 0~1に変換
dog_array = np.asarray(dog_resized, dtype=np.float32) / 255.0

```

```
# SVD のモードで近似
a = Ur.T @ dog_array.reshape(-1, 1)
dog_svd = Ur @ a
dog_svd = dog_svd.reshape(dog_array.shape)

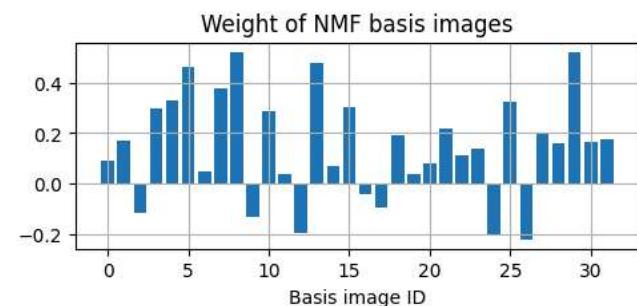
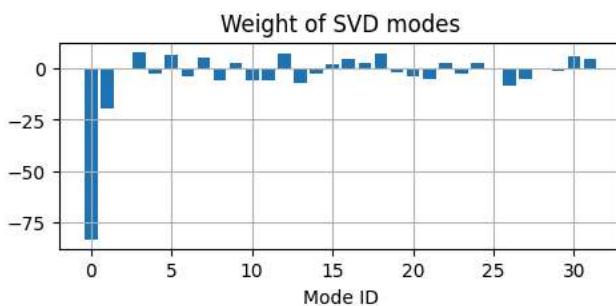
# NMF の基底画像で近似
h = np.linalg.pinv(W) @ dog_array.reshape(-1, 1)
dog_nmf = W @ h
dog_nmf = dog_nmf.reshape(dog_array.shape)

# 全モードを使って近似すると？
a_full = U.T @ dog_array.reshape(-1, 1)
dog_full = U @ a_full
dog_full = dog_full.reshape(dog_array.shape)

fig, axes = plt.subplots(1, 4, figsize=(10, 20))
axes[0].imshow(dog_array, cmap='gray')
axes[0].set_title('Original')
axes[0].axis('off')
axes[1].imshow(dog_svd, cmap='gray')
axes[1].set_title('Reconstructed (SVD)')
axes[1].axis('off')
axes[2].imshow(dog_nmf, cmap='gray')
axes[2].set_title('Reconstructed (NMF)')
axes[2].axis('off')
axes[3].imshow(dog_full, cmap='gray')
axes[3].set_title('with full mode')
axes[3].axis('off')
plt.show()

# 各モード・基底画像の重みを棒グラフで表示
fig, axes = plt.subplots(1, 2, figsize=(12, 2))
axes[0].bar(np.arange(n_components), a.reshape(-1))
axes[0].set_title('Weight of SVD modes')
axes[0].set_xlabel("Mode ID")
axes[0].grid(True)
axes[1].bar(np.arange(n_components), h.reshape(-1))
axes[1].set_title('Weight of NMF basis images')
axes[1].set_xlabel("Basis image ID")
axes[1].grid(True)
plt.show()

# どちらの分解方法で近似した画像が元画像によく近似できているか、Root Mean Square Error (RMSE) で評価
rmse_svd = np.sqrt(np.mean((dog_array - dog_svd) ** 2))
rmse_nmf = np.sqrt(np.mean((dog_array - dog_nmf) ** 2))
print(f'RMSE (SVD) = {rmse_svd:.3f}')
print(f'RMSE (NMF) = {rmse_nmf:.3f}'')
```



RMSE (SVD) = 0.199753  
RMSE (NMF) = 0.192951

## ▼ 演習課題. 日本の降水量に SVD・NMF を適用

これまでの例では、顔画像の特徴量抽出を行い、学習していない画像を近似する手法を示しました。

以上の例で重要なのは、例えば下記のような点です。

- 抽出されるモード・基底画像は、少ないモード数・ランク数でも効率よくすべての訓練画像を近似できるように重要で普遍的な特微量が抽出される点。
- 学習していない画像でも、数少ないモード・基底画像で、近似的に表現できる点

このような性質は、気象データにも有効に利用可能です。

ここでは降水量データに対して SVD・NMF を適用して特微量抽出を行う例を示します。

特に降水量においては、降水分布は疎な分布となるため、比較的 NMF の親和性が高い気象変数と考えられます。取り出された基底画像の解釈可能性が高いことを下記の例を通して確かめてください。

## ▼ 降水量に次元削減を適用する

直下のコードブロックの中の n\_components\_pr は、降水量用のチューニングパラメータです。

これは次元削減後の次元数ですので、値を適宜変更して実行してみてください。

```
# 次元削減後の次元数（モード数）
n_components_pr = 25 # 成分数
```

## ▼ 使用するデータ

解析雨量（レーダーと雨量計を組み合わせたプロダクト）を 5km スケールにアップスケールし、3日積算したもの。

ただしデータサイズ削減のため、下記のデータ削減も適用している。

- 陸域のみを取り出し、海上のデータを取り除く。
- 陸域全体のうち、2割より少ない領域でしか降水が発生していない時刻のデータは取り除く。

```
# 陸域のみ、全体の2割以上で降水が観測された際の3日積算雨量
f_precip = 'precip_land_threshold_area_0_2_72hrs.bin'
# 上のデータの時刻データ（タイムステップを確認するために使用）
f_time_steps = f'precip_land_threshold_area_0_2_72hrs_time_steps.npy'
# 海陸両方を含んだマスクデータ（日本地図の形状を復元するために使用）
f_mask = 'mask_x5_y6.npy'

# ダウンロード
!wget -O $f_time_steps 'https://www.dropbox.com/scl/fi/hf609re72m6p7gy2mctlu/precip_land_threshold_area_0_2_72hrs_time_steps.npy?rlkey=ilku5'
!wget -O $f_precip 'https://www.dropbox.com/scl/fi/ula3vliktx6c8ptmot2lg/precip_land_threshold_area_0_2_72hrs.bin?rlkey=q7yf8v6e71eyz8r7nht'
!wget -O $f_mask 'https://www.dropbox.com/scl/fi/qyksxwjm5tn7y98gs7cyu/mask_x5_y6.npy?rlkey=ygsmejtqyfc10qu5txyjrm2xg&st=qxxv2j9t&dl=1' :::::

--2025-10-24 03:30:59-- https://www.dropbox.com/scl/fi/hf609re72m6p7gy2mctlu/precip\_land\_threshold\_area\_0\_2\_72hrs\_time\_steps.npy?rlkey=ilku5
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.18, 2620:100:601d:18::a27d:512
Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ucf067e361c14a04ee085893a1b9.dl.dropboxusercontent.com/cd/0/inLine/Cz0zEzz0qKl\_IxqX\_ptHGfmh2KxDxEJazwvAdyQX56PxdlEg1knaEUo5f
--2025-10-24 03:30:59-- https://ucf067e361c14a04ee085893a1b9.dl.dropboxusercontent.com/cd/0/inline/Cz0zEzz0qKl\_IxqX\_ptHGfmh2KxDxEJazwvAdyQX56
Resolving ucf067e361c14a04ee085893a1b9.dl.dropboxusercontent.com (ucf067e361c14a04ee085893a1b9.dl.dropboxusercontent.com)... 162.125.5.15, 26
Connecting to ucf067e361c14a04ee085893a1b9.dl.dropboxusercontent.com (ucf067e361c14a04ee085893a1b9.dl.dropboxusercontent.com)|162.125.5.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 105296 (103K) [application/binary]
Saving to: 'precip_land_threshold_area_0_2_72hrs_time_steps.npy'

precip_land_threshold_area_0_2_72hrs_time_steps.npy 100%[=====] 102.83K --.-KB/s   in 0.04s

2025-10-24 03:30:59 (2.30 MB/s) - 'precip_land_threshold_area_0_2_72hrs_time_steps.npy' saved [105296/105296]

--2025-10-24 03:31:00-- https://www.dropbox.com/scl/fi/ula3vliktx6c8ptmot2lg/precip\_land\_threshold\_area\_0\_2\_72hrs.bin?rlkey=q7yf8v6e71eyz8r7nht
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.18, 2620:100:601d:18::a27d:512
Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://uce0201b87856c0e389b85e8e856.dl.dropboxusercontent.com/cd/0/inLine/Cz14hTludP1xBRhUz-6BfoXoM1FEzo4baCA0rTwUIW-96wQwSh30S7k1
--2025-10-24 03:31:00-- https://uce0201b87856c0e389b85e8e856.dl.dropboxusercontent.com/cd/0/inline/Cz14hTludP1xBRhUz-6BfoXoM1FEzo4baCA0rTwUIW
Resolving uce0201b87856c0e389b85e8e856.dl.dropboxusercontent.com (uce0201b87856c0e389b85e8e856.dl.dropboxusercontent.com)... 162.125.5.15, 26
Connecting to uce0201b87856c0e389b85e8e856.dl.dropboxusercontent.com (uce0201b87856c0e389b85e8e856.dl.dropboxusercontent.com)|162.125.5.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/Cz1RukkaReqydbIm2zokX2bZFSWvAvJAhcZfQ04mqcGqPyXkqcsqw2HYel2hEns i4MZgqN04wtfRPHuBu5U0FeVmIUDIDIEm40kA3lcR8NYGrkd1N6F
--2025-10-24 03:31:01-- https://uce0201b87856c0e389b85e8e856.dl.dropboxusercontent.com/cd/0/inline2/Cz1RukkaReqydbIm2zokX2bZFSWvAvJAhcZfQ04
Reusing existing connection to uce0201b87856c0e389b85e8e856.dl.dropboxusercontent.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 221974592 (212M) [application/binary]
Saving to: 'precip_land_threshold_area_0_2_72hrs.bin'

precip_land_threshold_area_0_2_72hrs.bin 100%[=====] 211.69M 72.7MB/s   in 2.9s

2025-10-24 03:31:04 (72.7 MB/s) - 'precip_land_threshold_area_0_2_72hrs.bin' saved [221974592/221974592]

--2025-10-24 03:31:04-- https://www.dropbox.com/scl/fi/qyksxwjm5tn7y98gs7cyu/mask\_x5\_y6.npy?rlkey=ygsmejtqyfc10qu5txyjrm2xg&st=qxxv2j9t&dl=1
Resolving www.dropbox.com (www.dropbox.com)... 162.125.5.18, 2620:100:601d:18::a27d:512
```

```

Connecting to www.dropbox.com (www.dropbox.com)|162.125.5.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://ucd2fa388748e70528f95383f21b.dl.dropboxusercontent.com/cd/0/inLine/Cz3nAcHA3PC1j0zBOK74_6NbCr6gTpZpBHypBFV1234ty1Np2i2CwJwI
--2025-10-24 03:31:04-- https://ucd2fa388748e70528f95383f21b.dl.dropboxusercontent.com/cd/0/inLine/Cz3nAcHA3PC1j0zBOK74_6NbCr6gTpZpBHypBFV12
Resolving ucd2fa388748e70528f95383f21b.dl.dropboxusercontent.com (ucd2fa388748e70528f95383f21b.dl.dropboxusercontent.com)... 162.125.5.15, 26
Connecting to ucd2fa388748e70528f95383f21b.dl.dropboxusercontent.com (ucd2fa388748e70528f95383f21b.dl.dropboxusercontent.com)|162.125.5.15|:2
HTTP request sent, awaiting response... 200 OK
Length: 819328 (800K) [text/plain]
Saving to: 'mask_x5_y6.npy'

mask_x5_y6.npy    100%[=====] 800.12K --.-KB/s   in 0.1s

2025-10-24 03:31:05 (8.03 MB/s) - 'mask_x5_y6.npy' saved [819328/819328]

```

```

# データのロード
data_time_steps = np.load(f_time_steps)
data_all = np.fromfile(f_precip, dtype=np.float64).reshape(len(data_time_steps), -1).T
mask = np.load(f_mask)

# 訓練データとテストデータに分割
train_range = 2006123115, 2023123115
test_range = 2005123115, 2006123115
training_start = datetime(2005, 12, 31, 15, 0)
training_end = datetime(2010, 12, 31, 14, 59)
test_start = datetime(2010, 12, 31, 15, 0)
test_end = datetime(2015, 12, 31, 14, 59)

train_time = [
    time_step for time_step in data_time_steps
    if training_start <= datetime.strptime(time_step, '%Y%m%d%H%M') < training_end
]
bool_train_time = [time_step in train_time for time_step in data_time_steps]
train_data = data_all[:, bool_train_time]

test_time = [
    time_step for time_step in data_time_steps
    if test_start <= datetime.strptime(time_step, '%Y%m%d%H%M') < test_end
]
bool_test_time = [time_step in test_time for time_step in data_time_steps]
test_data = data_all[:, bool_test_time]

```

```

# 描画用の関数
def fill_mask_from_1d(mask, data):
    mask = np.where(mask > 0, True, False)
    restored_data = np.full(mask.shape, np.nan)
    current_index = 0
    for y in range(mask.shape[0]):
        for x in range(mask.shape[1]):
            if mask[y, x]:
                restored_data[y, x] = data[current_index]
                current_index += 1
    return restored_data

def fill_mask_from_2d(mask, data):
    mask = np.where(mask > 0, True, False)
    restored_data = np.full((data.shape[1], mask.shape[0], mask.shape[1]), np.nan)
    current_index = 0
    for t in range(data.shape[1]):
        for y in range(mask.shape[0]):
            for x in range(mask.shape[1]):
                if mask[y, x]:
                    restored_data[t, y, x] = data[current_index]
                    current_index += 1
    return restored_data

def fill_mask(mask, data):
    """
    Parameters
    -----
    mask : numpy.ndarray
        Mask array (n_y, n_x)
    data : numpy.ndarray
        (n_grid, ) or (n_t, n_grid)
        Data to be filled (n_y, n_x) or (n_t, n_y, n_x)
    """
    if len(data.shape) == 1:
        return fill_mask_from_1d(mask, data)
    elif len(data.shape) == 2:
        return fill_mask_from_2d(mask, data)

```

```

def visualize_spatial(mode):
    extent = [128, 148, 30, 46] # [lon_min, lon_max, lat_min, lat_max]
    max_val = 1.0

    bool_positive = True
    mode_filled = []
    fig_title_all = []
    for i in range(n_components_pr):
        mode_max = np.max(mode[:, i])
        mode_min = np.min(mode[:, i])
        if np.abs(mode_min) > np.abs(mode_max):
            mode_max = np.abs(mode_min)
        if mode_min < 0:
            mode_min = -mode_max
            bool_positive = False
        else:
            mode_min = 0
        mode_scaled = mode[:, i] / mode_max
        mode_filled.append(fill_mask(mask, mode_scaled))
        fig_title_all.append(f'Component {i+1} (scale: {mode_max:.2f})')

    if bool_positive:
        multiple_mapping(
            extent, fig_title_all, np.array(mode_filled),
            map_color='Reds', vmin=0, vmax=max_val
        )
    else:
        multiple_mapping(
            extent, fig_title_all, np.array(mode_filled),
            map_color='seismic', vmin=-1, vmax=max_val
        )

def multiple_mapping(extent, fig_title, data, map_color='jet',
                     vmin=None, vmax=None, latlon=None, scatter_value=None,
                     xticks_interval=5, yticks_interval=5):

    n_data = data.shape[0]

    # Set the number of figures in a row and column
    n_figs_col = np.int32(np.sqrt(n_data)+1e-6)
    if n_figs_col * n_figs_col == n_data:
        n_figs_row = n_figs_col
    else:
        n_figs_row = n_figs_col + 1
        if n_figs_row * n_figs_col < n_data:
            n_figs_col += 1

    # Set the figure size
    fig_size_x = 25
    fig_size_y = fig_size_x * (extent[3] - extent[2]) / (extent[1] - extent[0]) * n_figs_col / n_figs_row
    fig_size = (fig_size_x, fig_size_y)

    # Create a figure
    plt.style.use('ggplot')

    fig, axes = plt.subplots(n_figs_col, n_figs_row, figsize=fig_size,
                           subplot_kw={'projection': ccrs.PlateCarree()})
    axes = axes.flatten() if n_data > 1 else [axes]

    xticks = np.arange(np.ceil(extent[0] / xticks_interval) * xticks_interval, extent[1]+1e-5, xticks_interval)
    yticks = np.arange(np.ceil(extent[2] / yticks_interval) * yticks_interval, extent[3]+1e-5, yticks_interval)
    if yticks[0] - extent[2] > yticks_interval:
        yticks = np.append(yticks[0] - yticks_interval, yticks)

    lon_formatter = LongitudeFormatter(zero_direction_label=True)
    lat_formatter = LatitudeFormatter()

    for i_ax, ax in enumerate(axes):
        if i_ax >= n_data:
            ax.set_visible(False)
            continue

        ax.set_extent([extent[0]-1e-4, extent[1]+1e-4, extent[2]-1e-4, extent[3]+1e-4])
        ax.coastlines(resolution='10m')

        # Set the ticks if the figure is in the bottom row or left column
        ax.set_xticks(xticks)
        ax.set_yticks(yticks)
        if i_ax % n_figs_row == 0:
            ax.tick_params(axis='y', which='both', labelright=False, labelleft=True)

```

```

else:
    ax.tick_params(axis='y', which='both', labelright=False, labelleft=False)
    if i_ax >= n_figs_row * (n_figs_col - 1):
        ax.tick_params(axis='x', which='both', labeltop=False, labelbottom=True)
    else:
        ax.tick_params(axis='x', which='both', labeltop=False, labelbottom=False)

    ax.xaxis.set_major_formatter(lon_formatter)
    ax.yaxis.set_major_formatter(lat_formatter)

    plt.setp(ax.get_xticklabels(), rotation=30, ha='right')

if vmin is None and vmax is None:
    im = ax.imshow(data[i_ax], cmap=map_color,
                    extent=[extent[0], extent[1], extent[2], extent[3]])
else:
    im = ax.imshow(
        data[i_ax], cmap=map_color, vmin=vmin, vmax=vmax,
        extent=[extent[0], extent[1], extent[2], extent[3]])
)

if latlon is not None:
    ax.scatter(latlon[1], latlon[0], c='pink', edgecolors='black', s=10)

ax.grid(True, color='white', linestyle='-', linewidth=0.5)

ax.set_title(fig_title[i_ax])

# Adjust the layout to make space for the colorbar
fig.tight_layout(rect=[0, 0.05, 1, 0.95])

# Add the colorbar at the bottom, spanning the width of the figure
cbar_ax = fig.add_axes([0.1, 0.02, 0.8, 0.02]) # [left, bottom, width, height]
cbar = fig.colorbar(im, cax=cbar_ax, orientation='horizontal')
cbar.ax.tick_params(labelsize=20)

# fig.savefig(fig_name, bbox_inches='tight', dpi=300)
plt.show()
# plt.close()

def visualize_temporal(mode):
    all_time_steps = []
    time_step = training_start
    while time_step <= training_end:
        all_time_steps.append(time_step)
        time_step += timedelta(hours=72)

    data_time = [datetime.strptime(time, '%Y%m%d%H%M') for time in train_time]

    bool_positive = True

    mode_scaled = []
    fig_title_all = []
    for i in range(n_components_pr):
        mode_max = np.max(mode[i, :])
        mode_min = np.min(mode[i, :])
        if np.abs(mode_min) > np.abs(mode_max):
            mode_max = np.abs(mode_min)
        if mode_min < 0:
            mode_min = -mode_max
            bool_positive = False
        else:
            mode_min = 0

        mode_filled = np.full(len(all_time_steps), np.nan)
        for j, time_step in enumerate(all_time_steps):
            if time_step in data_time:
                mode_filled[j] = mode[i, data_time.index(time_step)]

        mode_scaled.append(mode_filled / mode_max)
        fig_title = f'Component {i+1}, max: {mode_max:.2f}'
        fig_title_all.append(fig_title)

    if bool_positive:
        multiple_lines(fig_title_all, np.array(mode_scaled), all_time_steps, vmin=0, vmax=1)
    else:
        multiple_lines(fig_title_all, np.array(mode_scaled), all_time_steps, vmin=-1, vmax=1)

def multiple_lines(fig_title, data, xticks, vmin=0, vmax=1):
    # Number of plot areas
    n_data = data.shape[0]

```

```
# Figure size in y-direction
fig_size_y = 1 * n_data

plt.style.use('ggplot')

fig, axes = plt.subplots(n_data, 1, figsize=(10, fig_size_y))

for i in range(n_data):
    ax = axes[i]
    ax.plot(xticks, data[i], color='red', linewidth=0.5, label=fig_title[i])
    ax.set_xlim(vmin, vmax)
    if i == n_data - 1:
        ax.set_xlabel('Time')
    else:
        ax.tick_params(axis='x', which='both', labeltop=False, labelbottom=False)
    ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

## ▼ 演習 1: SVD を降水量データに適用

降水量に対して SVD を適用した場合には、果たして解釈可能なモードを取り出すことはできるでしょうか？

```
# 特異値分解（SVD）主成分分析（PCA）を訓練データに適用
# <-- 演習課題

# 描画
visualize_spatial(Ur_pr)
visualize_temporal(VrT_pr)
```

```
NameError: name 'Ur_pr' is not defined
Traceback (most recent call last)
/tmp/ipython-input-4291386157.py in <cell line: 0>()
      3
      4 # 描画
----> 5 visualize_spatial(Ur_pr)
      6 visualize_temporal(VrT_pr)
```

NameError: name 'Ur\_pr' is not defined

次のステップ: [エラーの説明](#)

## ▼ 演習 2: 降水量データに NMF を適用してみる

NMF はパーツに分解しやすい行列分解手法であるため、局所性の強い降水量に適用すると、それなりに意味のある基底画像が取り出されるのではないかと期待できます。実際に適用してみて、各モードについて解釈を試みてください。

特にいくつかのモードは明確に特定の降水パターンを反映しています。