Machine Learning HWS24

Assignment 3

Cagan Yigit Deliktas, cdelikta, 1979012

Nursultan Mamatov, nmamatov, 1983726

In [47]:
```
import numpy as np
from numpy.linalg import svd, matrix_rank
import pandas as pd
import matplotlib.pyplot as plt
from IPython import get_ipython
from util import (
    svdcomp,
    nextplot,
    plot_matrix,
    plot_xy,
    plot_cov,
    match_categories,
)  # see util.py
from sklearn.cluster import KMeans

# setup plotting
import psutil
inTerminal = not "IPKernelApp" in get_ipython().config
inJupyterNb = any(filter(lambda x: x.endswith("jupyter-notebook"), psutil.Process().parent().cmdline()))
inJupyterLab = any(filter(lambda x: x.endswith("jupyter-lab"), psutil.Process().parent().cmdline()))
if not inJupyterLab:
    from IPython import get_ipython
    get_ipython().run_line_magic("matplotlib", "" if inTerminal else "notebook" if inJupyterNb else "widget")
```

1 Intuition on SVD

The outer product $\mathbf{u} \otimes \mathbf{v}$ is equivalent to a matrix multiplication $\mathbf{u}\mathbf{v}^{\mathrm{T}}$, provided that $\mathbf{u}$ is represented as a $m \times 1$ column vector and $\mathbf{v}$ as a $n \times 1$ column vector (which makes $\mathbf{v}^{\mathrm{T}}$ a row vector).[2][3] For instance, if $m = 4$ and $n = 3$, then[4]

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^{\mathsf{T}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_2 & u_2 v_3 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 \\ u_4 v_1 & u_4 v_2 & u_4 v_3 \end{bmatrix}.$$

picture is from Wikipedia

In [48]:
```
M1 = np.array(
    [
        [1, 1, 1, 0, 0],
        [1, 1, 1, 0, 0],
        [1, 1, 1, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
    ]
)

#m1_u = np.array([[1/np.sqrt(3)], [1/np.sqrt(3)], [1/np.sqrt(3)], [0], [0]])
#m1_vt = np.array([1/np.sqrt(3), 1/np.sqrt(3), 1/np.sqrt(3), 0, 0]).reshape(1, 5)
#m1_singular = 3
#m1_construct = 3 * (m1_u @ m1_vt)

#rank=1
m1_u = 1/np.sqrt(3)*np.array([1, 1, 1, 0, 0])
m1_vt = 1/np.sqrt(3)*np.array([1, 1, 1, 0, 0])
m1_singular = 3
m1_construct = m1_singular * (np.outer(m1_u, m1_vt))
m1_construct
```

Out[48]:
```
array([[1., 1., 1., 0., 0.],
       [1., 1., 1., 0., 0.],
       [1., 1., 1., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [49]:
```
M2 = np.array(
    [
        [0, 0, 0, 0, 0],
        [0, 2, 1, 2, 0],
        [0, 2, 1, 2, 0],
        [0, 2, 1, 2, 0],
        [0, 0, 0, 0, 0],
    ]
)
#rank=1
m2_u = 1/np.sqrt(3) * np.array([0, 1, 1, 1, 0])
m2_vt = 1/3 * np.array([0, 2, 1, 2, 0])
m2_singular = np.sqrt(27)
m2_construct = m2_singular * (np.outer(m2_u, m2_vt))
m2_construct
```

Out[49]:
```
array([[0., 0., 0., 0., 0.],
       [0., 2., 1., 2., 0.],
       [0., 2., 1., 2., 0.],
       [0., 2., 1., 2., 0.],
       [0., 0., 0., 0., 0.]])
```

In [50]:
```
M3 = np.array([[0, 0, 0, 0],
               [0, 1, 1, 1],
               [0, 1, 1, 1],
               [0, 1, 1, 1],
               [0, 1, 1, 1]])
#rank=1
m3_u = 1/2 * np.array([0, 1, 1, 1, 1])
m3_vt = 1/np.sqrt(3) * np.array([0, 1, 1, 1])
m3_singular = 2*np.sqrt(3)
m3_construct = m3_singular * (np.outer(m3_u, m3_vt))
m3_construct
```

Out[50]:
```
array([[0., 0., 0., 0.],
       [0., 1., 1., 1.],
       [0., 1., 1., 1.],
       [0., 1., 1., 1.],
       [0., 1., 1., 1.]])
```

In [51]:
```
M4 = np.array(
    [
        [1, 1, 1, 0, 0],
        [1, 1, 1, 0, 0],
        [1, 1, 1, 0, 0],
        [0, 0, 0, 1, 1],
        [0, 0, 0, 1, 1],
    ]
)
# Rank 2
m4_u1 = 1/np.sqrt(3) * np.array([1, 1, 1, 0, 0])
m4_u2 = 1/np.sqrt(2) * np.array([0, 0, 0, 1, 1])
m4_vt1 = 1/np.sqrt(3) * np.array([1, 1, 1, 0, 0])
m4_vt2 = 1/np.sqrt(2) * np.array([0, 0, 0, 1, 1])
m4_singular1 = np.sqrt(3*3)
m4_singular2 = np.sqrt(2*2)

m4_construct = m4_singular1 * np.outer(m4_u1, m4_vt1) + m4_singular2 * np.outer(m4_u2, m4_vt2)
m4_construct
```

Out[51]:
```
array([[1., 1., 1., 0., 0.],
       [1., 1., 1., 0., 0.],
       [1., 1., 1., 0., 0.],
       [0., 0., 0., 1., 1.],
       [0., 0., 0., 1., 1.]])
```

In [52]:
```
M5 = np.array([
    [1, 1, 1, 0, 0],
    [1, 1, 1, 0, 0],
    [1, 1, 1, 1, 1],
    [0, 0, 1, 1, 1],
    [0, 0, 1, 1, 1]
])
#couldn't solve just by looking. rank=3, there will be three non-zero singular values.
```

In [53]:
```
M6 = np.array(
    [
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 0, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
    ]
```

```
)

#couldn't solve just by looking. rank=2, there will be two non-zero singular values.
```

1b

In [54]:
```
# YOUR PART

U, s, Vt = svd(M1)
print(f"Rank(M1): {np.linalg.matrix_rank(M1)}")
print('**************************************************')
print('Numpy U: ', U[:, 0])
print('**************************************************')
print('Numpy Sigma: ', np.round(s[0]))
print('**************************************************')
print('Numpy Vt: ', Vt[0,:])
print('**************************************************')
print('MY RESULTS: ')
print('My U: ', m1_u)
print('**************************************************')
print('My Sigma: ', m1_singular)
print('**************************************************')
print('My Vt: ', m1_vt)
print('**************************************************')
print('M1 matrix:\n', M1)
print('My reconstructed matrix:\n', m1_construct)
```

```
Rank(M1): 1
**************************************************
Numpy U:  [-0.57735027 -0.57735027 -0.57735027  0.          0.        ]
**************************************************
Numpy Sigma:  3.0
**************************************************
Numpy Vt:  [-0.57735027 -0.57735027 -0.57735027 -0.         -0.        ]
**************************************************
MY RESULTS:
My U:  [0.57735027 0.57735027 0.57735027 0.         0.        ]
**************************************************
My Sigma:  3
**************************************************
My Vt:  [0.57735027 0.57735027 0.57735027 0.         0.        ]
**************************************************
M1 matrix:
 [[1 1 1 0 0]
 [1 1 1 0 0]
 [1 1 1 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
My reconstructed matrix:
 [[1. 1. 1. 0. 0.]
 [1. 1. 1. 0. 0.]
 [1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [55]:
```
U, s, Vt = svd(M2)
print(f"Rank(M2): {np.linalg.matrix_rank(M2)}")
print('**************************************************')
print('Numpy U: ', U[:, 0])
print('**************************************************')
print('Numpy Sigma: ', np.round(s[0], 5))
print('**************************************************')
print('Numpy Vt: ', Vt[0,:])
print('**************************************************')
print('MY RESULTS: ')
print('My U: ', m2_u)
print('**************************************************')
print('My Sigma: ', m2_singular)
print('**************************************************')
print('My Vt: ', m2_vt)
print('**************************************************')
print('M2 matrix:\n', M2)
print('My reconstructed matrix:\n', m2_construct)
```

```
Rank(M2): 1
**********************************************
Numpy U:  [ 0.         -0.57735027 -0.57735027 -0.57735027  0.        ]
**********************************************
Numpy Sigma:  5.19615
**********************************************
Numpy Vt:  [-0.         -0.66666667 -0.33333333 -0.66666667 -0.        ]
**********************************************
MY RESULTS:
My U:  [0.         0.57735027 0.57735027 0.57735027 0.        ]
**********************************************
My Sigma:  5.196152422706632
**********************************************
My Vt:  [0.         0.66666667 0.33333333 0.66666667 0.        ]
**********************************************
M2 matrix:
 [[0 0 0 0 0]
 [0 2 1 2 0]
 [0 2 1 2 0]
 [0 2 1 2 0]
 [0 0 0 0 0]]
My reconstructed matrix:
 [[0. 0. 0. 0. 0.]
 [0. 2. 1. 2. 0.]
 [0. 2. 1. 2. 0.]
 [0. 2. 1. 2. 0.]
 [0. 0. 0. 0. 0.]]
```

In [56]:
```
U, s, Vt = svd(M3)
print(f"Rank(M3): {np.linalg.matrix_rank(M3)}")
print('**********************************************')
print('Numpy U: ', U[:, 0])
print('**********************************************')
print('Numpy Sigma: ', np.round(s[0], 5))
print('**********************************************')
print('Numpy Vt: ', Vt[0,:])
print('**********************************************')
print('MY RESULTS: ')
print('My U: ', m3_u)
print('**********************************************')
print('My Sigma: ', m3_singular)
print('**********************************************')
print('My Vt: ', m3_vt)
print('**********************************************')
print('M3 matrix:\n', M3)
print('My reconstructed matrix:\n', m3_construct)
```

```
Rank(M3): 1
**********************************************
Numpy U:  [ 0.  -0.5 -0.5 -0.5 -0.5]
**********************************************
Numpy Sigma:  3.4641
**********************************************
Numpy Vt:  [-0.         -0.57735027 -0.57735027 -0.57735027]
**********************************************
MY RESULTS:
My U:  [0.  0.5 0.5 0.5 0.5]
**********************************************
My Sigma:  3.4641016151377544
**********************************************
My Vt:  [0.         0.57735027 0.57735027 0.57735027]
**********************************************
M3 matrix:
 [[0 0 0 0]
 [0 1 1 1]
 [0 1 1 1]
 [0 1 1 1]
 [0 1 1 1]]
My reconstructed matrix:
 [[0. 0. 0. 0.]
 [0. 1. 1. 1.]
 [0. 1. 1. 1.]
 [0. 1. 1. 1.]
 [0. 1. 1. 1.]]
```

In [57]:
```
U, s, Vt = svd(M4)
print(f"Rank(M4): {np.linalg.matrix_rank(M4)}")
print('**********************************************')
print('Numpy U1: ', U[:, 0])
print('Numpy U2: ', U[:, 1])
print('**********************************************')
print('Numpy Sigma1: ', np.round(s[0], 5))
print('Numpy Sigma2: ', np.round(s[1], 5))
print('**********************************************')
```

```
print('Numpy Vt1: ', Vt[0, :])
print('Numpy Vt2: ', Vt[1, :])
print('************************************************')
print('MY RESULTS: ')
print('My U1: ', m4_u1)
print('My U2: ', m4_u2)
print('************************************************')
print('My Sigma1: ', m4_singular1)
print('My Sigma2: ', m4_singular2)
print('************************************************')
print('My Vt1: ', m4_vt1)
print('My Vt2: ', m4_vt2)
print('************************************************')
print('M4 matrix:\n', M4)
print('My reconstructed matrix:\n', m4_construct)
```

```
Rank(M4): 2
************************************************
Numpy U1:  [-0.57735027 -0.57735027 -0.57735027  0.          0.        ]
Numpy U2:  [ 0.          0.          0.         -0.70710678 -0.70710678]
************************************************
Numpy Sigma1:  3.0
Numpy Sigma2:  2.0
************************************************
Numpy Vt1:  [-0.57735027 -0.57735027 -0.57735027 -0.         -0.        ]
Numpy Vt2:  [-0.         -0.         -0.         -0.70710678 -0.70710678]
************************************************
MY RESULTS:
My U1:  [0.57735027 0.57735027 0.57735027 0.         0.        ]
My U2:  [0.         0.         0.         0.70710678 0.70710678]
************************************************
My Sigma1:  3.0
My Sigma2:  2.0
************************************************
My Vt1:  [0.57735027 0.57735027 0.57735027 0.         0.        ]
My Vt2:  [0.         0.         0.         0.70710678 0.70710678]
************************************************
M4 matrix:
 [[1 1 1 0 0]
 [1 1 1 0 0]
 [1 1 1 0 0]
 [0 0 0 1 1]
 [0 0 0 1 1]]
My reconstructed matrix:
 [[1. 1. 1. 0. 0.]
 [1. 1. 1. 0. 0.]
 [1. 1. 1. 0. 0.]
 [0. 0. 0. 1. 1.]
 [0. 0. 0. 1. 1.]]
```

In [58]:
```
# YOUR PART

U, s, Vt = svd(M5)
print(f"Rank(M5): {np.linalg.matrix_rank(M5)}")
print('************************************************')
print('Numpy U1: ', U[:, 0])
print('Numpy U2: ', np.round(U[:, 1], 4))
print('Numpy U3: ', U[:, 2])
print('************************************************')
print('Numpy Sigma(1,2,3): ', np.round(s[:3], 3))
print('************************************************')
print('Numpy Vt1: ', Vt[0, :])
print('Numpy Vt2: ', np.round(Vt[1, :], 4))
print('Numpy Vt3: ', Vt[2, :])
print('************************************************')
print("SVD reconstructed: ")
print(np.round(svdcomp(M5, range(3)), 5))
print('Original matrix: ')
print(M5)
```

```
Rank(M5): 3
************************************************
Numpy U1:  [-0.39410272 -0.39410272 -0.61541221 -0.39410272 -0.39410272]
Numpy U2:  [-0.5 -0.5 -0.   0.5  0.5]
Numpy U3:  [ 0.3077061   0.3077061  -0.78820544  0.3077061   0.3077061 ]
************************************************
Numpy Sigma(1,2,3):  [3.562 2.    0.562]
************************************************
Numpy Vt1:  [-0.39410272 -0.39410272 -0.61541221 -0.39410272 -0.39410272]
Numpy Vt2:  [-0.5 -0.5 -0.   0.5  0.5]
Numpy Vt3:  [-0.3077061  -0.3077061   0.78820544 -0.3077061  -0.3077061 ]
************************************************
SVD reconstructed:
[[ 1.  1.  1. -0. -0.]
 [ 1.  1.  1. -0. -0.]
 [ 1.  1.  1.  1.  1.]
 [-0. -0.  1.  1.  1.]
 [-0. -0.  1.  1.  1.]]
Original matrix:
[[1 1 1 0 0]
 [1 1 1 0 0]
 [1 1 1 1 1]
 [0 0 1 1 1]
 [0 0 1 1 1]]
```

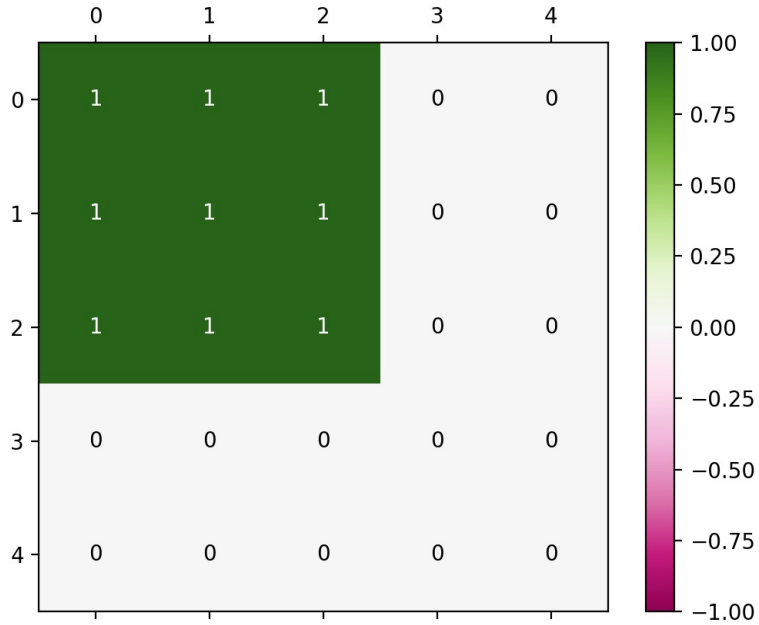
In [59]:
```
U, s, Vt = svd(M6)
print(f"Rank(M6): {np.linalg.matrix_rank(M6)}")
print('************************************************')
print('Numpy U1: ', U[:, 0])
print('Numpy U2: ', np.round(U[:, 1], 4))
print('************************************************')
print('Numpy Sigma(1,2): ', np.round(s[:2], 3))
print('************************************************')
print('Numpy Vt1: ', Vt[0, :])
print('Numpy Vt2: ', np.round(Vt[1, :], 4))
print('************************************************')
print("SVD reconstructed: ")
print(np.round(svdcomp(M6, range(2)), 5))
print('Original matrix: ')
print(M6)
```

```
Rank(M6): 2
************************************************
Numpy U1:  [-0.46193977 -0.46193977 -0.38268343 -0.46193977 -0.46193977]
Numpy U2:  [-0.1913 -0.1913  0.9239 -0.1913 -0.1913]
************************************************
Numpy Sigma(1,2):  [4.828 0.828]
************************************************
Numpy Vt1:  [-0.46193977 -0.46193977 -0.38268343 -0.46193977 -0.46193977]
Numpy Vt2:  [ 0.1913  0.1913 -0.9239  0.1913  0.1913]
************************************************
SVD reconstructed:
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1. -0.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
Original matrix:
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 0 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

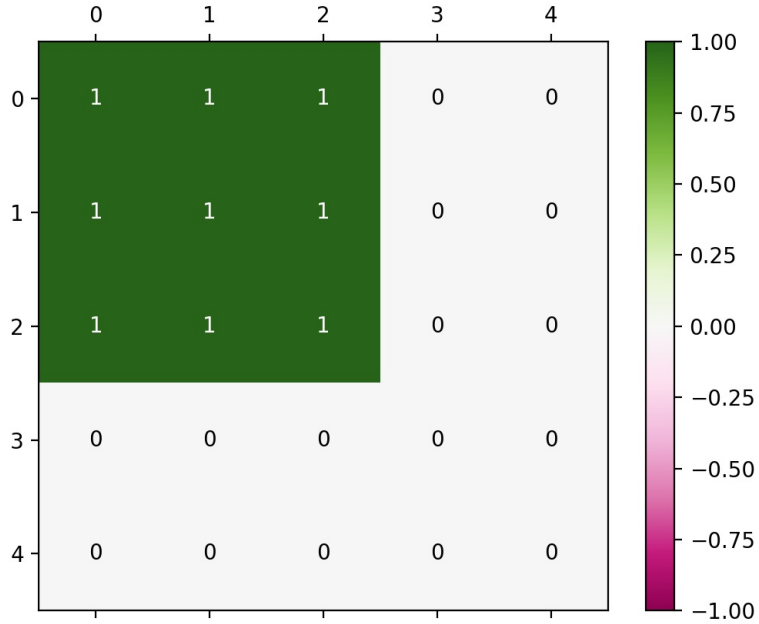
1c

In [60]:
```
# You can use the functions svdcomp and plot_matrix from util.py
# YOUR PART
matrices = {'M1': M1, 'M2': M2, 'M3': M3, 'M4': M4, 'M5': M5, 'M6': M6}

for matrix_n, matrix in matrices.items():
    print(f'{matrix_n} Reconstructed: ')
    plot_matrix(svdcomp(matrix, range(1)))
    print(f'{matrix_n} Original: ')
    plot_matrix(matrix)
    print('************************************************************')
    print('************************************************************')
```

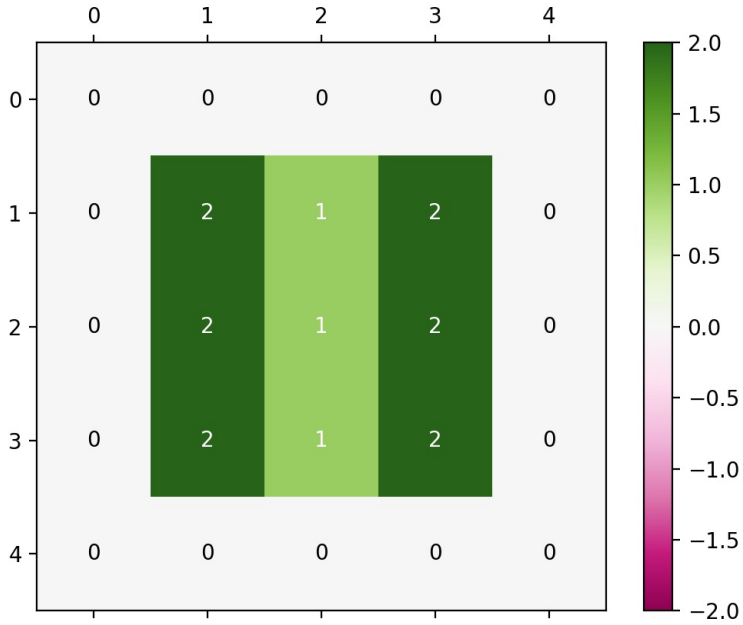
```
M1 Reconstructed:
```



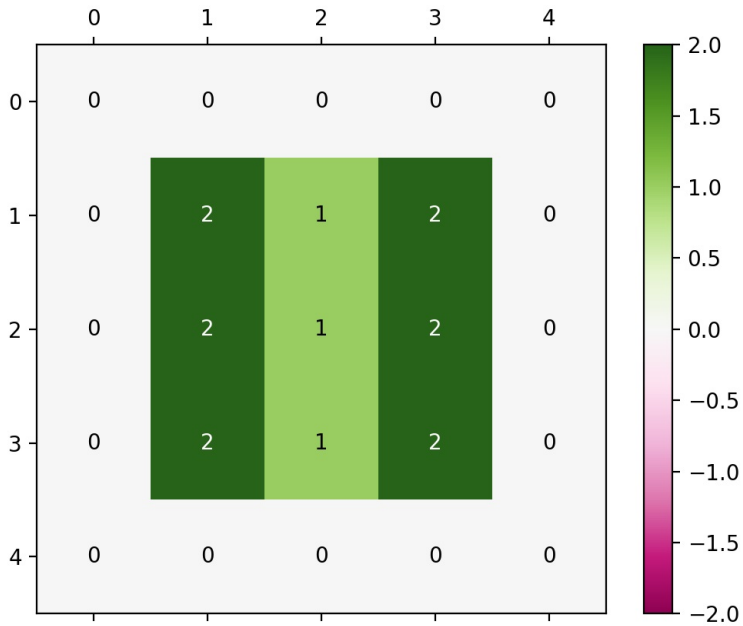
M1 Original:



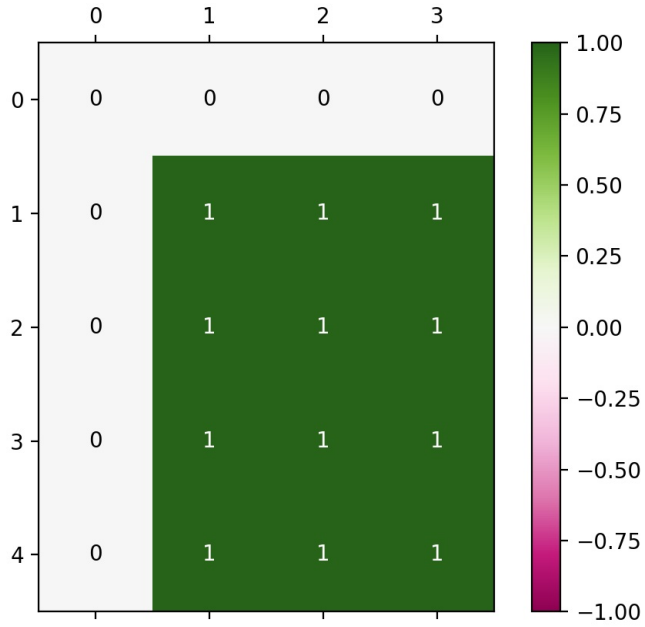
****
****
M2 Reconstructed:



M2 Original:

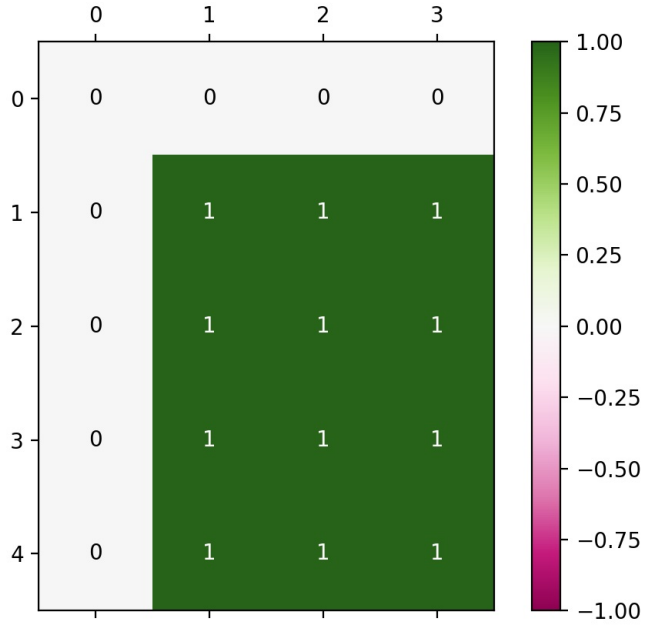


****
****
M3 Reconstructed:



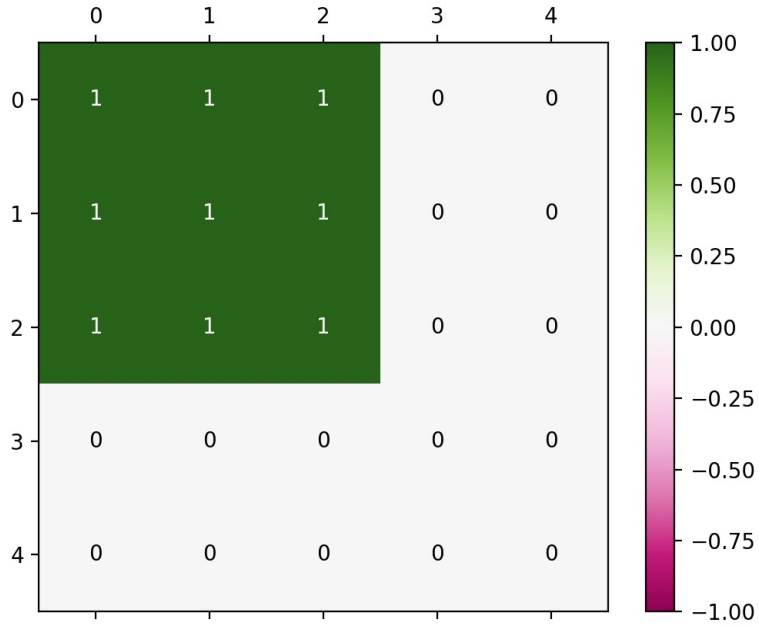
M3 Original:

```
/Users/cagandeliktas/Desktop/Mannheim Uni Master/Machine Learning/homeworks/ML_Assignments/assignment_3/util.py:
33: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplo
tlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warn
ing, see the rcParam `figure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
  plt.figure()  # this creates a new plot
```

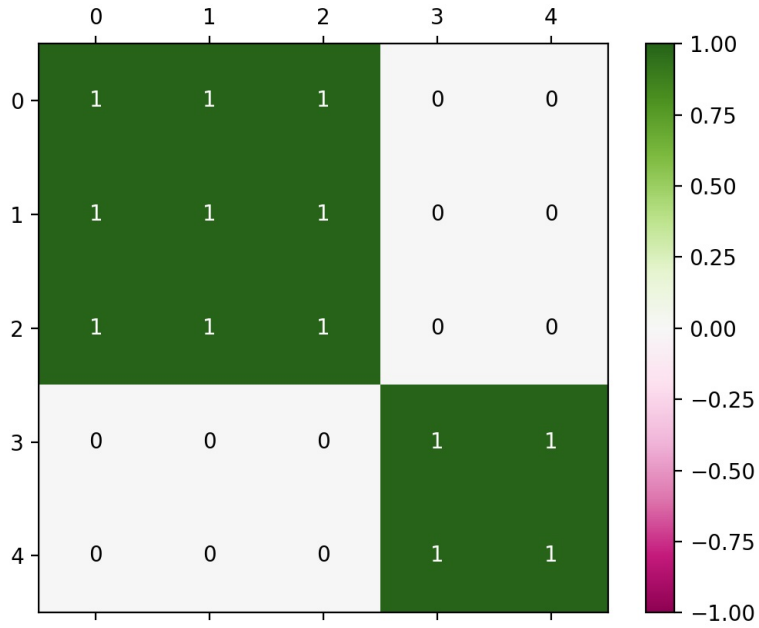



```
************************************************************
************************************************************
```

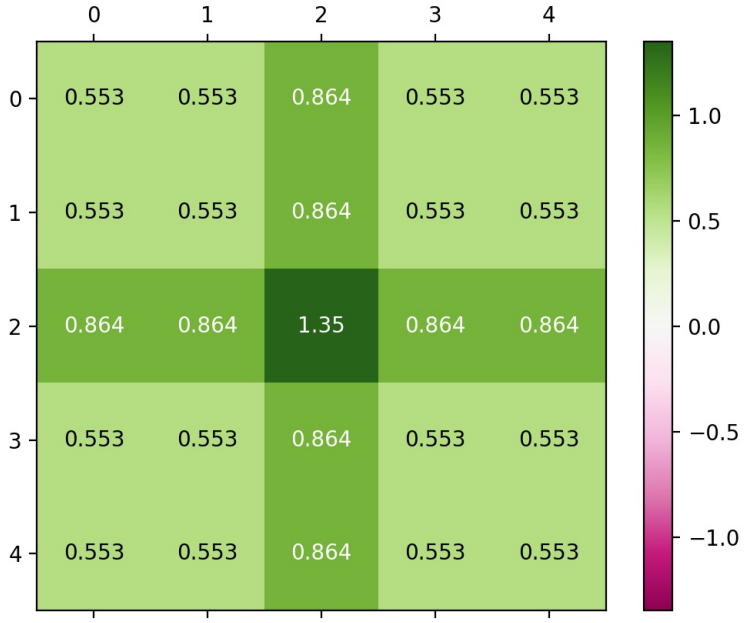
M4 Reconstructed:



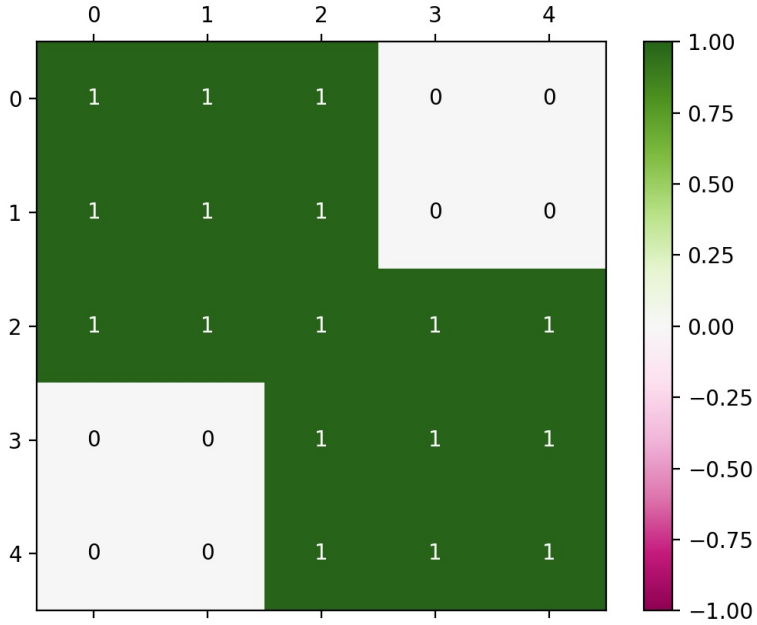
M4 Original:



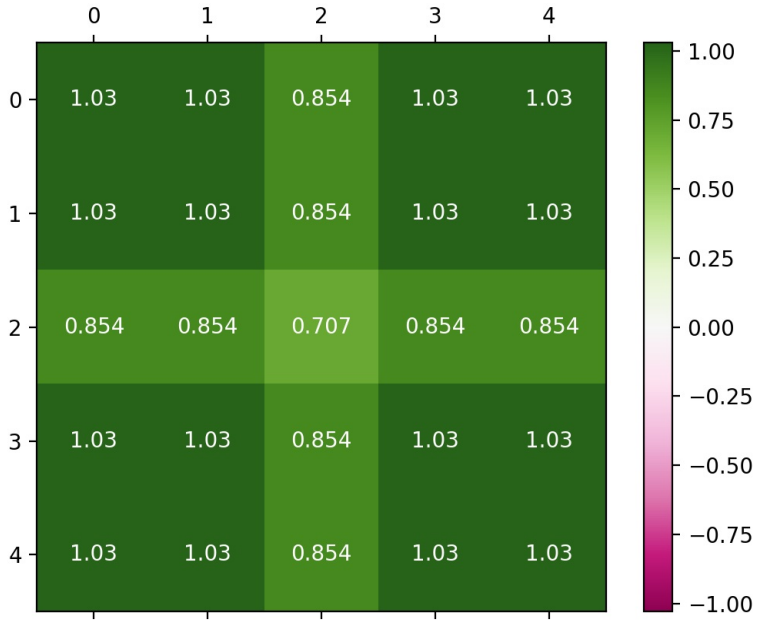
****
****
M5 Reconstructed:



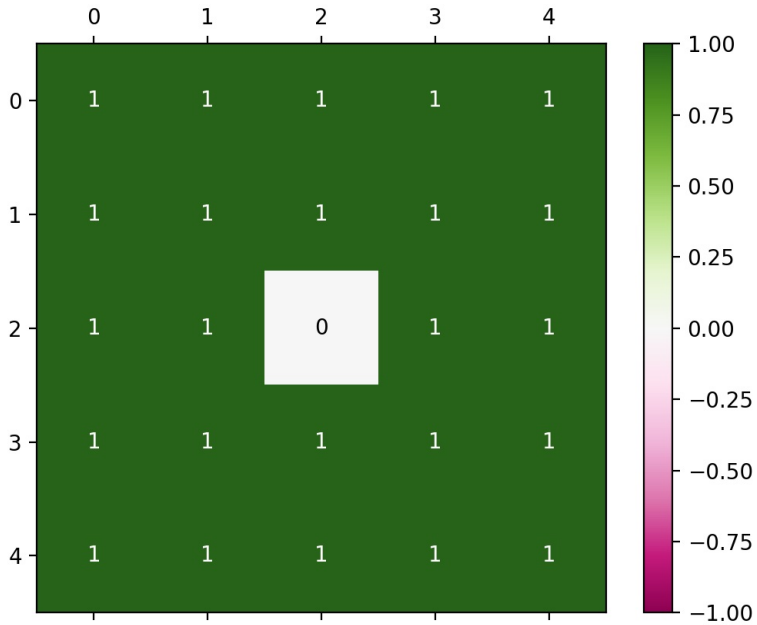
M5 Original:



****
****
M6 Reconstructed:



M6 Original:



****
****

1d

In [62]:
```
# Another method to compute the rank is matrix_rank.
# YOUR PART

U, s, Vt = svd(M6)
print(f"Rank(M6): {np.linalg.matrix_rank(M6)}")
print('**********************************************')
```

```
print('Numpy Sigma (rounded): ', np.round(s, 5))
print('Numpy Sigma: ', s)
print('************************************************')
print("SVD reconstructed: ")
print(np.round(svdcomp(M6, range(2)), 5))
print('Original matrix: ')
print(M6)
```

```
Rank(M6): 2
************************************************
Numpy Sigma (rounded):  [4.82843 0.82843 0.      0.      0.     ]
Numpy Sigma:  [4.82842712e+00 8.28427125e-01 9.95090019e-17 2.18529703e-17
 5.31822283e-50]
************************************************
SVD reconstructed:
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1. -0.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
Original matrix:
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 0 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

In [63]:
```
s > 0
```

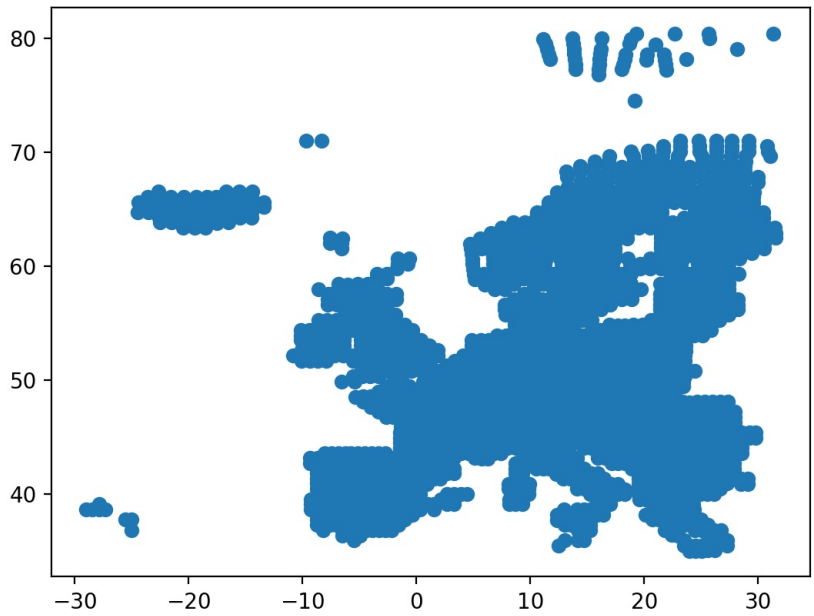
Out[63]:
```
array([ True,  True,  True,  True,  True])
```

numpy didn't return any nonzero singular values, but the last three singular values are really small: 9.95090019e-17, 2.18529703e-17, 5.31822283e-50. we should consider them as zeros. So, we rounded the singular values up to 5 decimals.

2 The SVD on Weather Data

In [64]:
```
# Load the data
climate = pd.read_csv("data/worldclim.csv")
coord = pd.read_csv("data/worldclim_coordinates.csv")
lon = coord["lon"]
lat = coord["lat"]
```

In [65]:
```
# Plot the coordinates
plot_xy(lon, lat)
```



2a

In [66]:
```
climate.describe().iloc[[3, 7],:] #data has to be normalized because the ranges are different.
```

Out[66]:

	min1	min2	min3	min4	min5	min6	min7	min8	min9	min10	...	rain3	rain4	rain5	rain6	rain7	rain8	rai
min	-23.2	-23.6	-22.5	-19.6	-11.4	-5.5	-2.1	-2.6	-6.7	-12.8	...	18.50	14.407	7.50	1.6667	0.00	0.00	8.2
max	11.9	11.3	11.8	13.3	16.4	20.9	23.1	23.8	22.2	19.0	...	188.11	141.170	158.33	181.1700	173.75	186.67	278.3

2 rows × 48 columns

In [67]:
```
# YOUR PART
# Normalize the data to z-scores. Store the result in X.
def z_score_scaling(data):
    data = np.array(data)
    mean = np.mean(data, axis=0)
    std_dev = np.std(data, axis=0)

    z_scores = (data - mean) / std_dev

    return z_scores

display(climate.head())
X = pd.DataFrame(z_score_scaling(climate), columns=climate.columns)
display(X.head())
display(X.describe().iloc[[3, 7],:])
```

	min1	min2	min3	min4	min5	min6	min7	min8	min9	min10	...	rain3	rain4	rain5	rain6	rain7	rain8	rain9	rain1
0	10.6	9.9	10.5	11.0	12.5	14.7	16.9	18.0	17.1	15.0	...	103.00	74.000	66.000	53.000	41.000	57.000	92.000	118
1	8.3	7.6	8.1	8.6	10.2	12.4	14.7	15.9	14.8	12.7	...	108.29	72.429	63.286	50.571	38.857	53.714	88.571	118
2	10.1	9.5	9.9	10.5	12.0	14.2	16.4	17.4	16.6	14.6	...	119.00	75.000	60.400	48.400	38.400	50.400	84.400	120
3	10.2	9.7	9.9	10.7	12.2	14.3	16.4	17.4	16.6	14.7	...	141.20	80.400	54.200	44.000	36.600	42.600	75.600	123
4	11.7	11.1	11.5	12.1	13.5	15.7	17.8	18.9	18.1	16.1	...	119.50	74.000	58.500	47.500	38.000	48.500	83.500	121

5 rows × 48 columns

	min1	min2	min3	min4	min5	min6	min7	min8	min9	min10	...	rain3	rain4	r
0	2.365957	2.200220	2.104881	1.861889	1.545324	1.280310	1.320737	1.604380	1.927607	2.212659	...	1.721196	1.053074	0.33
1	2.011593	1.845426	1.694031	1.384333	1.014846	0.706090	0.769095	1.102594	1.416389	1.724273	...	1.913735	0.975600	0.21
2	2.288921	2.138517	2.002169	1.762398	1.430003	1.155479	1.195364	1.461012	1.816472	2.127722	...	2.303543	1.102389	0.08
3	2.304328	2.169369	2.002169	1.802194	1.476131	1.180445	1.195364	1.461012	1.816472	2.148956	...	3.111550	1.368689	-0.19
4	2.535435	2.385330	2.276069	2.080768	1.775967	1.529971	1.546408	1.819431	2.149875	2.446234	...	2.321741	1.053074	-0.00

5 rows × 48 columns

	min1	min2	min3	min4	min5	min6	min7	min8	min9	min10	...	rain3	r
min	-2.841644	-2.967432	-3.544305	-4.226950	-3.967037	-3.762841	-3.443443	-3.317905	-3.362382	-3.690441	...	-1.354324	-1.88
max	2.566249	2.416182	2.327425	2.319546	2.444831	2.828207	2.875364	2.990266	3.061175	3.062025	...	4.818918	4.36

2 rows × 48 columns

In [68]:
```
# Plot histograms of attributes
#nextplot()
#X.hist(ax=plt.gca())
```

2b

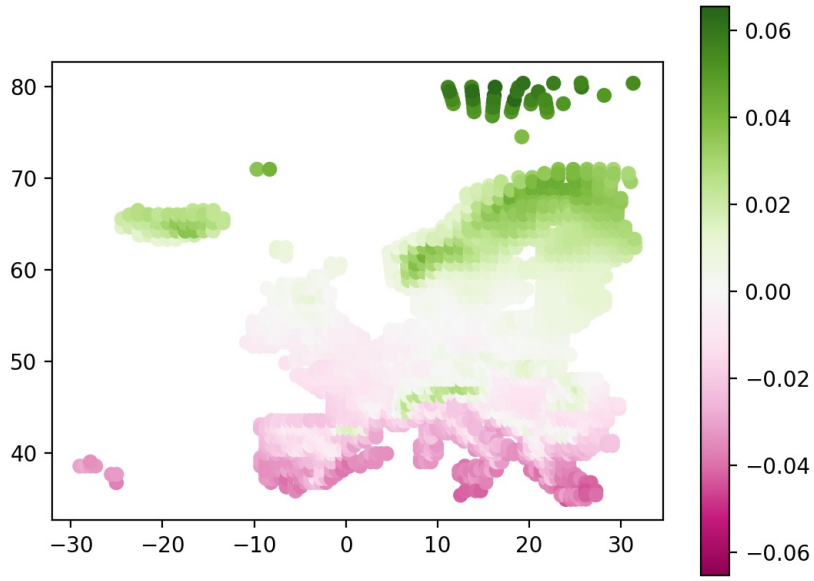
In [69]:
```
# Compute the SVD of the normalized climate data and store it in variables U,s,Vt. What
# is the rank of the data?
# YOUR PART

U, s, Vt = np.linalg.svd(X)
rank = np.sum(s > 1e-10)

print(f"Rank of the normalized climate data: {rank}")
print(f"Rank of the normalized climate data (numpy): {np.linalg.matrix_rank(X)}")

print("Singular values:")
print(s)
```

```
Rank of the normalized climate data: 48
Rank of the normalized climate data (numpy): 48
Singular values:
[2.90222389e+02 1.50668824e+02 8.84936404e+01 5.91859882e+01
 5.21202132e+01 2.74621244e+01 2.21341436e+01 1.52406513e+01
 1.41321813e+01 1.20289877e+01 1.14767046e+01 1.09209834e+01
 9.14704009e+00 8.39692373e+00 7.93211636e+00 7.06774614e+00
 6.74240524e+00 6.51838587e+00 5.76805648e+00 5.39678641e+00
 5.06878890e+00 4.21038123e+00 3.88507570e+00 3.37992885e+00
 3.12011424e+00 2.88184606e+00 2.53346089e+00 2.48165895e+00
 2.32967767e+00 2.07730350e+00 1.90548668e+00 1.86392296e+00
 1.72330112e+00 1.60158454e+00 1.28336386e+00 1.12607554e+00
 1.04958416e+00 9.84527428e-01 8.39920385e-01 6.56703416e-01
 4.94031887e-01 4.13336481e-01 3.78242622e-01 3.47242119e-01
 3.20887328e-01 3.06204289e-01 3.00736590e-01 2.55298845e-01]
```

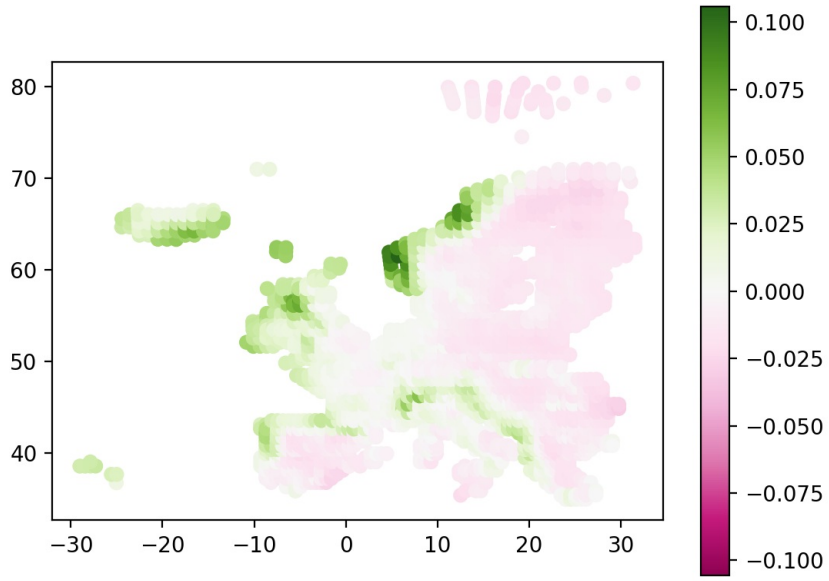
2c

In [70]:
```python
# For interpretation, it may also help to look at the other component matrices and
# perhaps use other plot functions (e.g., plot_matrix).
# YOUR PART
num_columns = 5

# Plot each of the first 5 columns of U
for i in range(num_columns):
    print(f'Left singuler vector {i+1} ')
    plot_xy(lon, lat, U[:, i])
```

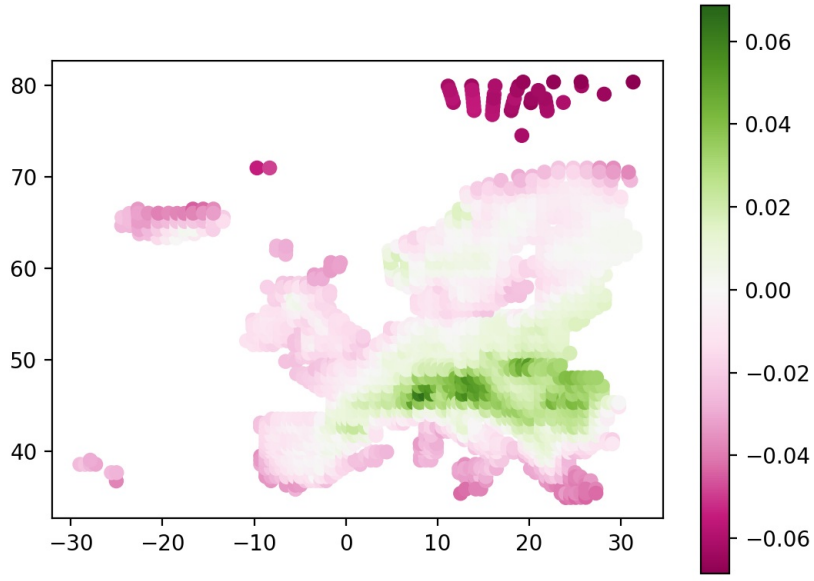
```
Left singuler vector 1
```



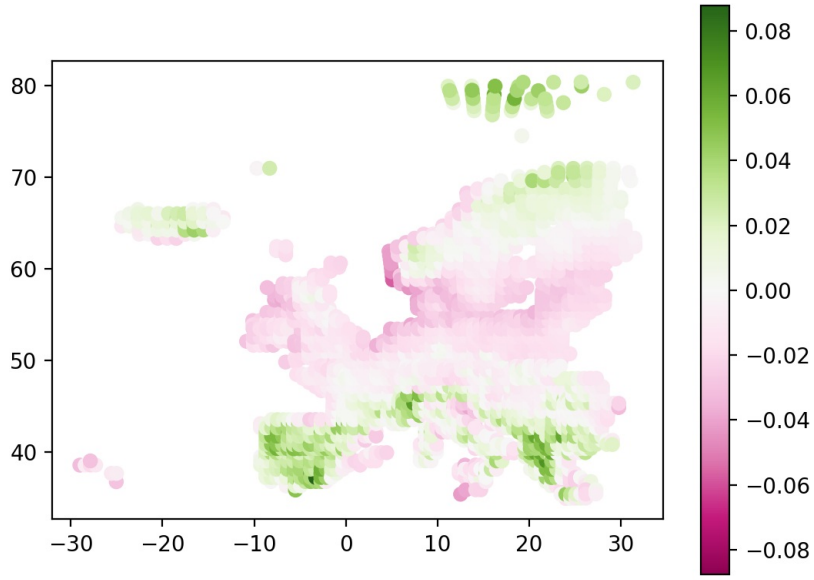

```
Left singuler vector 2
```



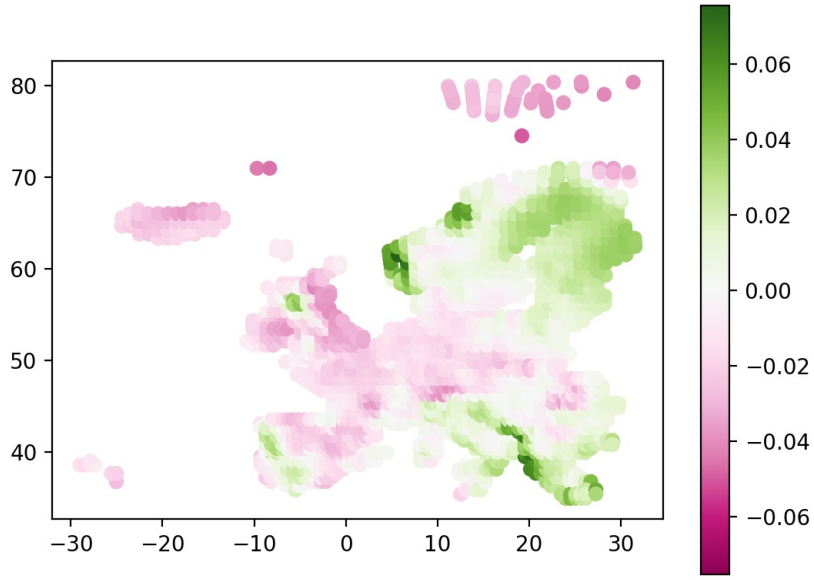
Left singuler vector 3



Left singuler vector 4



Left singuler vector 5

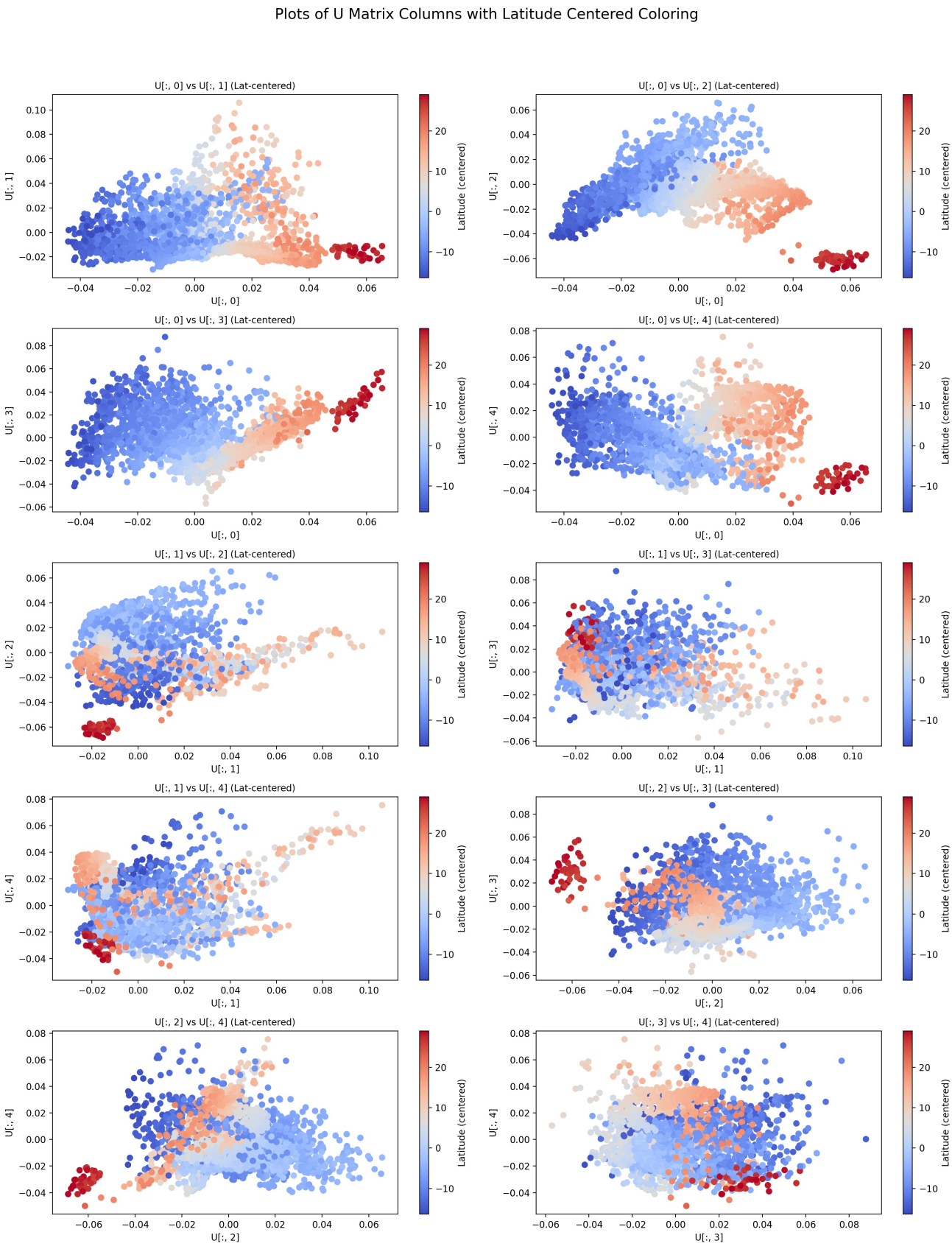


2d

"When (u[:,0]) is on the x-axis, we can observe that locations with higher latitudes (further north) appear on the right side of the plot, colored in red, while lower latitudes (closer to the south) appear on the left side, colored in blue. This indicates that (u[:,0]) corresponds to the north-south direction, with positive values of (u[:,0]) representing regions with higher latitudes, closer to the North Pole."

In [71]:
```
num_columns = 5

lat_centered = lat - np.mean(lat)
lon_centered = lon - np.mean(lon)

fig, axes = plt.subplots(5, 2, figsize=(15, 20))
fig.suptitle("Plots of U Matrix Columns with Latitude Centered Coloring", fontsize=16)

count = 0
for i in range(num_columns - 1):
    for j in range(i + 1, num_columns):
        row, col = divmod(count, 2)
        ax = axes[row, col]
        scatter_lat = ax.scatter(U[:, i], U[:, j], c=lat_centered, cmap='coolwarm')
        ax.set_xlabel(f'U[:, {i}]')
```

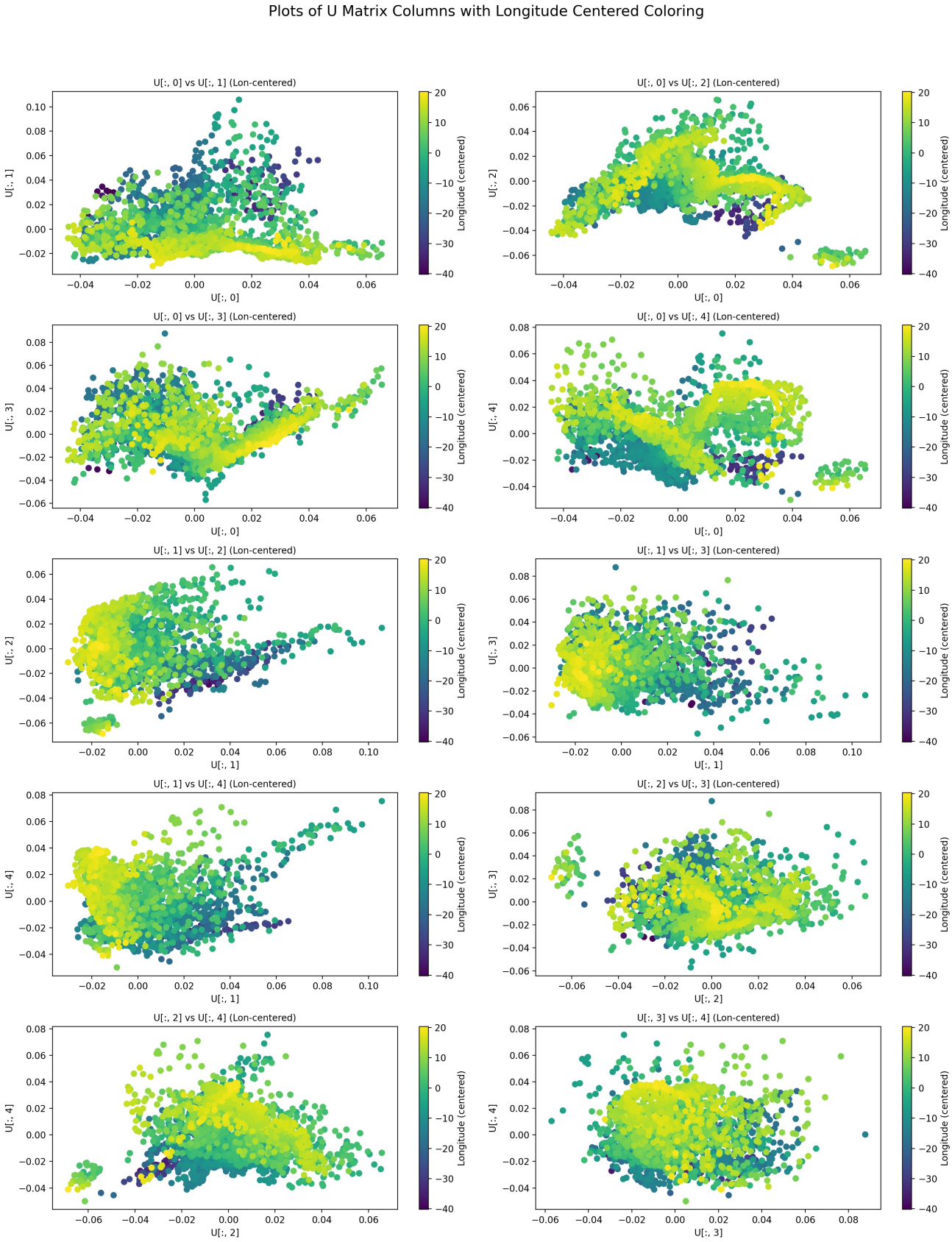
```
        ax.set_ylabel(f'U[:, {j}]')
        ax.set_title(f'U[:, {i}] vs U[:, {j}] (Lat-centered)', fontsize=10)

        fig.colorbar(scatter_lat, ax=ax, orientation="vertical", label="Latitude (centered)")
        count += 1

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Lower values of u[:, 1] correspond to higher longitudes (further east, represented by yellow on the legend). As the values of u[:, 1] increase, the locations represent lower longitudes (further west, represented by green-blue on the legend).

In [72]:
```
fig_lon, axes_lon = plt.subplots(5, 2, figsize=(15, 20))
fig_lon.suptitle("Plots of U Matrix Columns with Longitude Centered Coloring", fontsize=16)
```

```
count = 0
for i in range(num_columns - 1):
    for j in range(i + 1, num_columns):
        row, col = divmod(count, 2)
        ax = axes_lon[row, col]
        scatter_lon = ax.scatter(U[:, i], U[:, j], c=lon_centered, cmap='viridis')
        ax.set_xlabel(f'U[:, {i}]')
        ax.set_ylabel(f'U[:, {j}]')
        ax.set_title(f'U[:, {i}] vs U[:, {j}] (Lon-centered)', fontsize=10)

        fig_lon.colorbar(scatter_lon, ax=ax, orientation="vertical", label="Longitude (centered)")
        count += 1

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```



2e

In [73]:
```
# 2e(i) Guttman-Kaiser
# YOUR PART
print('index of the last singular value which is greater than 1: ', sum(s >= 1)-1)
```

```
index of the last singular value which is greater than 1:  36
```

In [74]:
```
print(f'k={sum(s >= 1)}') #37 singular values, k=37
```

```
k=37
```

In [75]:
```
k_gutmann = s[:sum(s >= 1)]
k_gutmann
```

Out[75]:
```
array([290.22238931, 150.66882365,  88.49364039,  59.18598821,
        52.12021318,  27.46212444,  22.13414365,  15.24065125,
        14.13218128,  12.02898772,  11.47670461,  10.92098344,
         9.14704009,   8.39692373,   7.93211636,   7.06774614,
         6.74240524,   6.51838587,   5.76805648,   5.39678641,
         5.0687889 ,   4.21038123,   3.8850757 ,   3.37992885,
         3.12011424,   2.88184606,   2.53346089,   2.48165895,
         2.32967767,   2.0773035 ,   1.90548668,   1.86392296,
         1.72330112,   1.60158454,   1.28336386,   1.12607554,
         1.04958416])
```

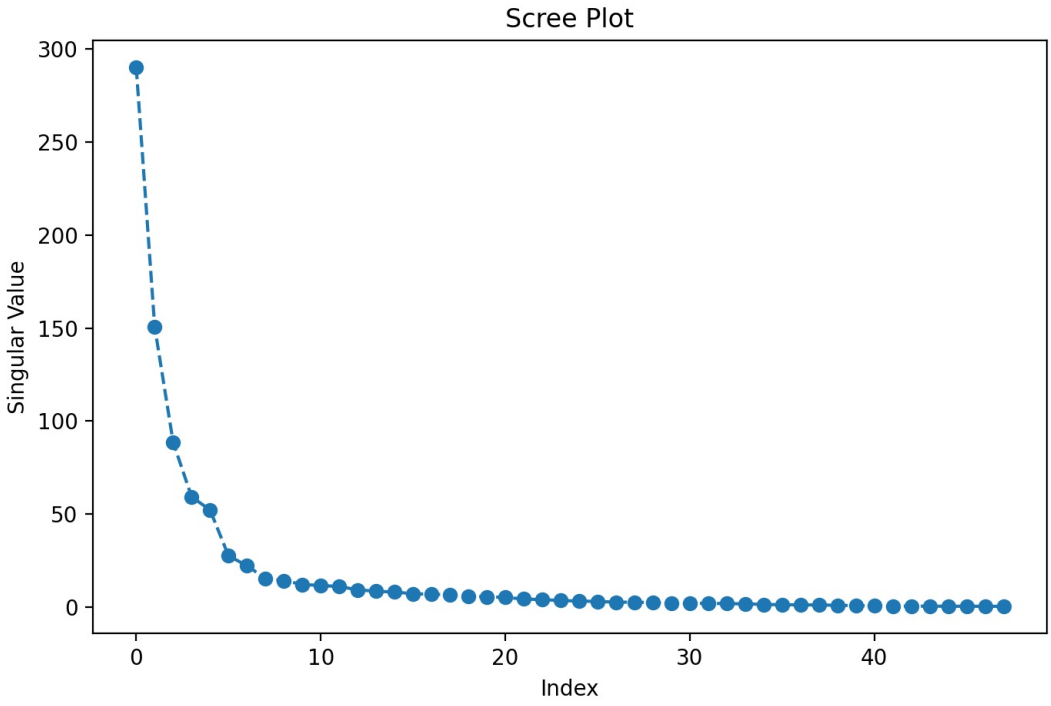
In [76]:
```
# 2e(ii) 90% squared Frobenius norm
# YOUR PART
def frobenius_norm(singular_values):
    return np.sqrt(np.sum(singular_values**2))

def rank_90_percent_frobenius(S, threshold=0.9):
    total_norm = frobenius_norm(S)
    squared_norms = np.cumsum(S**2)
    rank = np.argmax(squared_norms >= threshold * total_norm) + 1
    return rank

print(f'k={rank_90_percent_frobenius(s)}')
print('Only selects the first singular value')

ninety_percent_fro_norm = s[rank_90_percent_frobenius(s)-1]
print('first singular value: ', ninety_percent_fro_norm)
```

```
k=1
Only selects the first singular value
first singular value:  290.2223893074994
```

In the scree plot below, we can either select where there is a clear drop in magnitudes, or we can select the point where the values are even out.

k=1 or k=6

In [77]:
```
# 2e(iii) Scree test
# YOUR PART
def scree_test(S):
    plt.figure(figsize=(8, 5))
    plt.plot(S, marker='o', linestyle='--')
    plt.title("Scree Plot")
    plt.xlabel("Index")
    plt.ylabel("Singular Value")
    plt.show()

scree_test(s)
```



In [86]:
```python
# 2e(iv) entropy
# YOUR PART
def entropy_based_k(S):
    squared_singular_values = S ** 2
    total_frobenius_norm = np.sum(squared_singular_values)

    relative_contributions = squared_singular_values / total_frobenius_norm
    entropy = -np.sum(relative_contributions * np.log(relative_contributions)) / np.log(len(relative_contributi

    print(f"Entropy of singular values: {entropy:.4f}")

    cumulative_sum = np.cumsum(relative_contributions)
    rank_k = np.argmax(cumulative_sum >= entropy) + 1

    return rank_k

print(f'k={entropy_based_k(s)}')
```

```
Entropy of singular values: 0.2752
k=1
```

In [87]:
```python
# 2e(v) random flips
# Random sign matrix: np.random.choice([-1,1], X.shape)
# YOUR PART
from numpy.linalg import norm
np.random.seed(15)

def random_flip_signs(matrix):
    flip_matrix = np.random.choice([1, -1], size=matrix.shape)
    return matrix * flip_matrix

def residual_matrix(U, S, Vt, k):
```

```
    S_rest = S[k:]
    U_residual = U[:, k:]
    Vt_residual = Vt[k:, :]

    S_residual = np.zeros((U_residual.shape[1], Vt_residual.shape[0]))
    np.fill_diagonal(S_residual, S_rest)

    return U_residual @ S_residual @ Vt_residual

def random_flip_rank_selection(A, max_rank=None):
    U, S, Vt = svd(A)
    max_rank = max_rank or min(A.shape)

    best_rank = 1
    min_difference = float("inf")

    for k in range(1, max_rank + 1):
        X_minus_k = residual_matrix(U, S, Vt, k)

        X_minus_k_flipped = random_flip_signs(X_minus_k)

        spectral_norm_diff = abs(norm(X_minus_k, ord=2) - norm(X_minus_k_flipped, ord=2))
        frobenius_norm_residual = norm(X_minus_k, ord='fro')

        if frobenius_norm_residual != 0:
            normalized_difference = spectral_norm_diff / frobenius_norm_residual
        else:
            normalized_difference = float("inf")

        if normalized_difference < min_difference:
            min_difference = normalized_difference
            best_rank = k

    return best_rank
```

In [88]:
```
print(f'k={random_flip_rank_selection(X)}')
```

k=7

2f

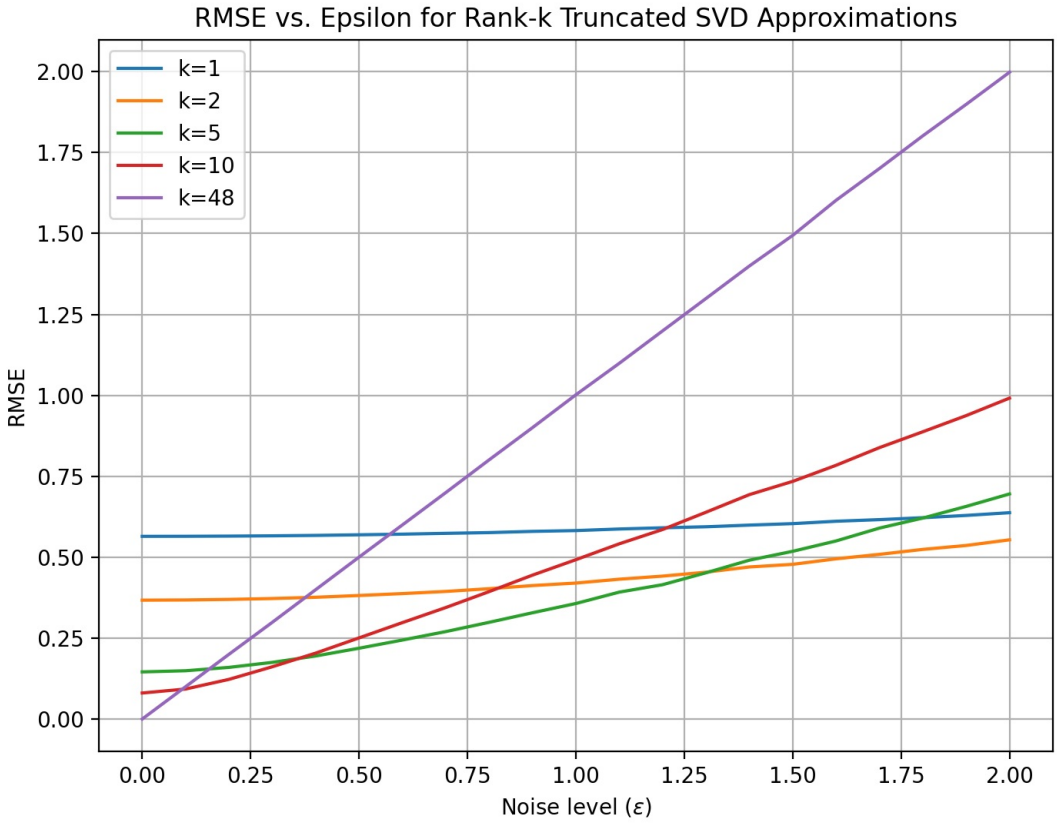
In [89]:
```
# Here is the empty plot that you need to fill (one line per choice of k: RSME between
# original X and the reconstruction from size-k SVD of noisy versions)
# YOUR PART
```

In [90]:
```
def rmse(A, A_hat):
    return np.sqrt(np.sum((A - A_hat)**2) / (A.shape[0] * A.shape[1]))

def add_noise(X, epsilon):
    return np.array(X) + np.random.randn(*X.shape) * epsilon

def rank_k_approximation(X, k):
    U, S, Vt = svd(X)
    S_k = np.diag(S[:k])
    U_k = U[:, :k]
    Vt_k = Vt[:k, :]
    return U_k @ S_k @ Vt_k
```

In [91]:
```
epsilons = np.linspace(0, 2, 21)
ranks = [1, 2, 5, 10, 48]
rmse_values = {k: [] for k in ranks}
```

In [92]:
```
for epsilon in epsilons:
    X_noise = add_noise(X, epsilon)

    for k in ranks:
        X_k = rank_k_approximation(X_noise, k)
        rmse_value = rmse(np.array(X), X_k)
        rmse_values[k].append(rmse_value)
```

In [93]:
```
plt.figure(figsize=(8, 6))
for k in ranks:
    plt.plot(epsilons, rmse_values[k], label=f'k={k}')

plt.xlabel(r"Noise level ($\epsilon$)")
plt.ylabel("RMSE")
plt.title('RMSE vs. Epsilon for Rank-k Truncated SVD Approximations')
plt.legend()
plt.grid(True)
plt.show()
```



3 SVD and k-means

In [94]:
```
# Cluster the normalized climate data into 5 clusters using k-means and store
# the vector giving the cluster labels for each location.
X_clusters = KMeans(5).fit(X).labels_
```

In [95]:
```
X_with_cluster = pd.concat([X, pd.DataFrame({'cluster':X_clusters})], axis=1)
X_with_cluster.head()
```

Out[95]:

	min1	min2	min3	min4	min5	min6	min7	min8	min9	min10	...	rain4	rain5	
0	2.365957	2.200220	2.104881	1.861889	1.545324	1.280310	1.320737	1.604380	1.927607	2.212659	...	1.053074	0.339022	-0.
1	2.011593	1.845426	1.694031	1.384333	1.014846	0.706090	0.769095	1.102594	1.416389	1.724273	...	0.975600	0.215637	-0.
2	2.288921	2.138517	2.002169	1.762398	1.430003	1.155479	1.195364	1.461012	1.816472	2.127722	...	1.102389	0.084431	-0.
3	2.304328	2.169369	2.002169	1.802194	1.476131	1.180445	1.195364	1.461012	1.816472	2.148956	...	1.368689	-0.197438	-0.
4	2.535435	2.385330	2.276069	2.080768	1.775967	1.529971	1.546408	1.819431	2.149875	2.446234	...	1.053074	-0.001948	-0.

5 rows × 49 columns

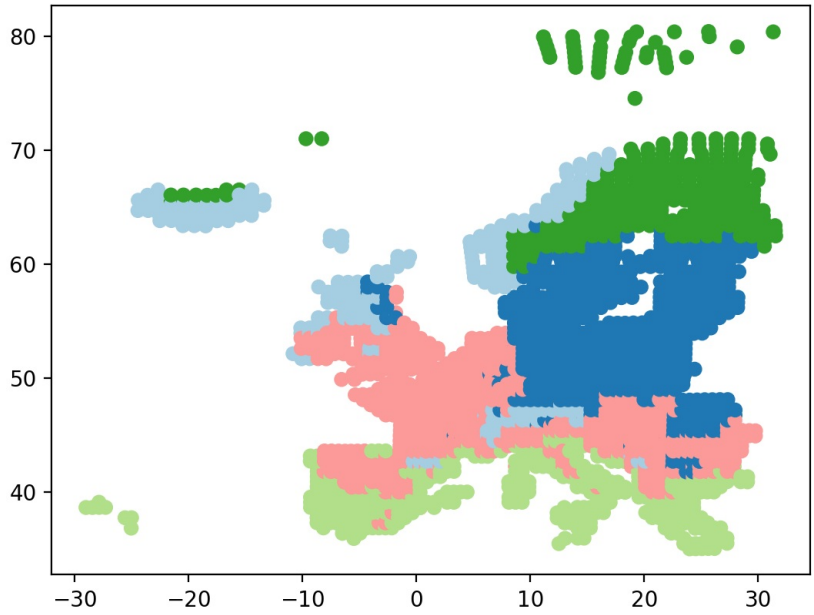
In [96]:
```
coord_cluster = pd.DataFrame({'lat': lat, 'lon':lon, 'cluster':X_clusters})
coord_cluster.head()
```

Out[96]:

	lat	lon	cluster
0	38.61	-29.01	2
1	38.61	-28.44	2
2	38.62	-27.86	2
3	38.62	-27.29	2
4	39.07	-27.87	2

3a

In [97]:
```
# Plot the results to the map: use the cluster labels to give the color to each
# point.
plot_xy(lon, lat, X_clusters)
```

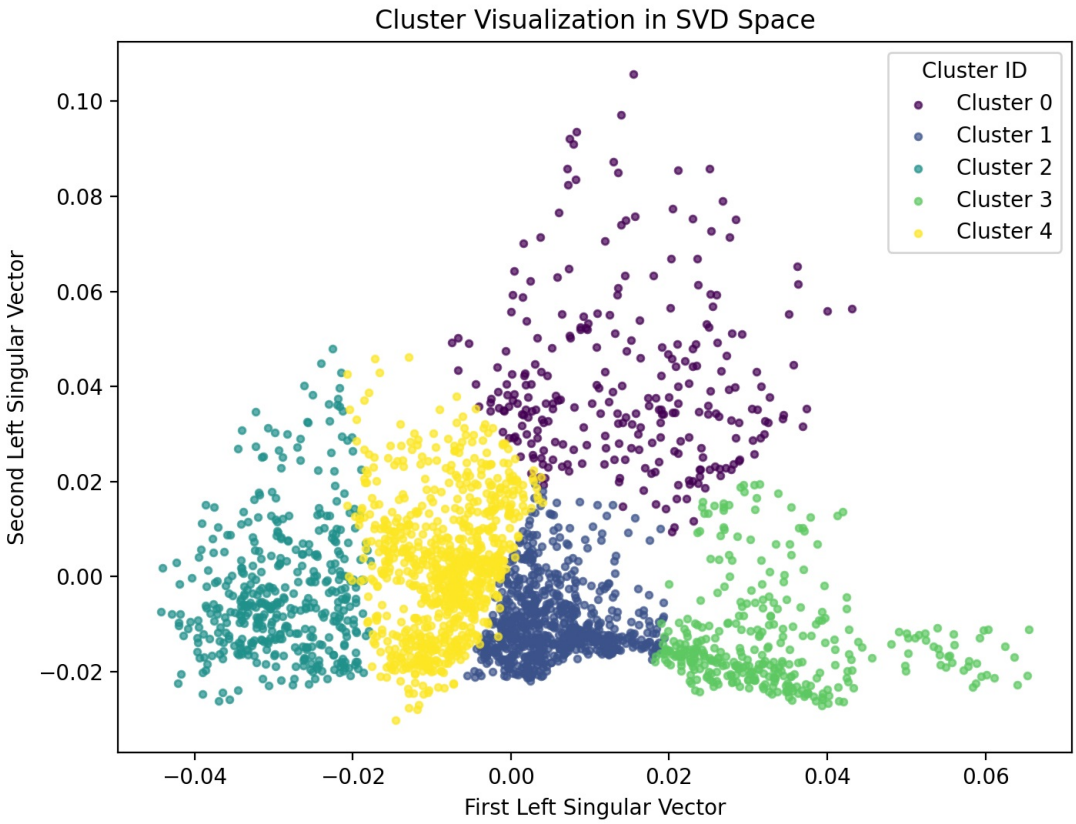


3b

In [98]:
```
X_with_cluster.cluster.unique()
```

Out[98]:
```
array([2, 0, 3, 4, 1], dtype=int32)
```

In [99]:
```
unique_clusters = np.unique(X_clusters)
colors = plt.cm.viridis(np.linspace(0, 1, len(unique_clusters)))

plt.figure(figsize=(8, 6))
for i, cluster_id in enumerate(unique_clusters):
    plt.scatter(U[X_clusters == cluster_id, 0], U[X_clusters == cluster_id, 1],
                color=colors[i], s=10, alpha=0.7, label=f'Cluster {int(cluster_id)}')

plt.xlabel("First Left Singular Vector")
plt.ylabel("Second Left Singular Vector")
plt.title("Cluster Visualization in SVD Space")
plt.legend(title='Cluster ID', loc='upper right')
plt.show()
```



3c

In [100…

```
# Compute the PCA scores, store in Z (of shape N x k)
k = 2 # select the first two components (left singular vectors)
# YOUR PART HERE
Z = U[:, :k] @ np.diag(s[:k]) #pca scores for each example
```

In [101…

```
for k in [1, 2, 3]:
    Z = U[:, :k] @ np.diag(s[:k])

    # cluster and visualize
    Z_clusters = KMeans(5).fit(Z).labels_
    # match clusters as well as possible (try without)
```

```
Z_clusters = match_categories(X_clusters, Z_clusters)
nextplot()
axs = plt.gcf().subplots(1, 2)
plot_xy(lon, lat, X_clusters, axis=axs[0])
axs[0].set_title("Original data")
plot_xy(lon, lat, Z_clusters, axis=axs[1])
axs[1].set_title(f"PCA $(k={k}$)")
```