SOM_Visualization

March 30, 2022

1 SOM Visualization

1.1 Description

This notebook contains code for visualizing the contents of hierarchical data (`*.h5`) files produced by `Sompy_experimentation.ipynb`. The current things it displays are as follows:

- A set of linear-gradient heatmaps
- A set of log-gradient heatmaps
- An interactive interface for browsing through what items are in each cluster
- A "U-matrix" that highlights regions of high euclidean distance gradients, to identify how well-matching the cluster boundaries are

Workflow

1. Enter the particular codebook file you want to use in the codebook selection cell, change the KM_CLUSTERS value to whatever number of clusters you feel is appropriate, and then rerun the notebook.
2. Inspect the heatmaps in the heatmap view cells.
3. Search the cells for materials of interest in the cluster inspector cell.
4. Evaluate the accuracy of the fit of the clusters in the umatrix view cell.

Notes

- The trained SOM has many arrays, one for each parameter it was trained on. This is what is seen in the heatmap views - each heatmap corresponds to one of these arrays.
- The cluster inspector has various features, which shall be listed here.
  - First, each cluster is given its own tab in the inspector. The order of the tabs by color matches the pattern in the colorbar to the side of the cluster map. You can confirm the color of a given cluster with the colored square next to the cluster's name inside of the tab's frame.
  - When a cluster is selected, you may view its statistical information and contained items in the tables in the tab's frame.
  - You can restrict the number of items shown in the tab by entering a **regular expression**(1, 2) that matches the items you wish to see.
  - Once the items have been filtered to your liking, you can hit the "render points" button in order to display all items visible in the table to the cluster map.

[1]:
```
import numpy as np
import pandas as pd
import logging
import ipywidgets as widgets
import matplotlib.pyplot as plt
import tables
import sompy
from sompy.visualization.mapview import View2D
from sompy.sompy import SOMFactory
import skimage
import matplotlib as mpl
from tfprop_sompy.tfprop_vis import render_cluster_borders_to_axes,␣
 ↪dataframe_to_coords, render_points_to_axes
import sklearn.cluster as clust
```

[2]:
```
# Codes from tfprop_sompy packages. Private code from Luna's group.
from tfprop_sompy.jupyter_integration.cluster_inspector import␣
 ↪sort_materials_by_cluster, cluster_tabs,\
                                                                      ␣
 ↪make_cluster_graph
from tfprop_sompy.tfprop_vis import render_posmap_to_axes, kmeans_clust,␣
 ↪render_points_to_axes,\
                                   show_posmap, ViewTFP, dataframe_to_coords,\
                                   clusteringmap_category, UMatrixTFP
```

[3]:
```
# This makes all the loggers stay quiet unless it's important
logging.getLogger().setLevel(logging.WARNING)
```

[4]:
```
# Paste name of file generated by training in Sompy_experimentation
CODEBOOK_FILE = 'som_enamel_data.h5'

# Number of clusters for k-means clustering
KM_CLUSTERS = 6
```

[5]:
```
# Creates necessary pd dataframes for visualization
stored_cb_matrix = pd.read_hdf(CODEBOOK_FILE, 'sm_codebook_matrix')
stored_mapsize = pd.read_hdf(CODEBOOK_FILE, 'sm_codebook_mapsize').values
mats_df = pd.read_hdf(CODEBOOK_FILE, 'sm_data')

# FIXME:
# We do a hack using the "pytables" library in order to extract the information
# For some reason pandas doesn't like to read object series out of h5 files

stored_columns = None
stored_matfamilies = None
with tables.open_file(CODEBOOK_FILE, "r") as store:
    # We normally get byte strings from this
    # The mapping operation turns them all into unicode strings ready for␣
 ↪presentation
```

```
    # Obtaining the column names
    stored_columns = list(map(lambda x: x.decode('utf-8'), store.root.
↪sm_codebook_columns.property_names.read()))

    # Obtaining the material identification
    stored_matfamilies = list(map(lambda x: x, store.root.
↪sm_codebook_matfamilies.material_families.read()))
```

[6]:
```
mats_df
```

[6]:
```
     modulus  hardness  carb   crys    fluo  age tooth_label mammal age_group  \
0      76.07      4.17  0.08  12.21  962.65    6          6j      h         p
1      75.07      4.45  0.09  12.53  962.60    6          6j      h         p
2      74.10      3.91  0.10  12.77  962.61    6          6j      h         p
3      68.91      3.94  0.11  12.95  962.51    6          6j      h         p
4      63.81      3.11  0.12  13.18  962.68    6          6j      h         p
..       …         …     …      …     …       …    …         …
133    64.80      3.47  0.10  13.06  960.71   10          10      o       wad
134    59.40      3.32  0.11  13.35  960.65   10          10      o       wad
135    56.92      2.89  0.12  13.53  960.57   10          10      o       wad
136    57.43      3.09  0.12  13.66  960.50   10          10      o       wad
137    59.56      3.29  0.13  13.83  960.47   10          10      o       wad

    position  depth    kc       b
0      outer   1.00  1.02  258.34
1      outer   0.80  0.93  277.70
2     middle   0.60  1.02  243.59
3     middle   0.40  1.00  244.44
4      inner   0.20  0.92  264.45
..       …      …     …      …
133    outer   0.77  1.02  352.11
134   middle   0.59  1.04  304.30
135   middle   0.39  1.07  305.00
136    inner   0.21  1.14  259.27
137    inner   0.02  1.20  227.67

[138 rows x 13 columns]
```

[7]:
```
# Insert new column with material identification
mats_df["Row"] = stored_matfamilies
```

[8]:
```
# Function located in sompy.py
# Creates a SOM object
sm = SOMFactory.build(mats_df[stored_columns].values,
                mapsize=(*stored_mapsize,),
                normalization="var",
                 initialization="pca",
                 component_names=stored_columns)
```

[9]:
```
# SOM object contains a codebook object
# Setting the matrix to the values of our current data.
sm.codebook.matrix = stored_cb_matrix.values
```

[10]:
```
# Using scikit-learn K-means clustering method to predict cluster fit index for␣
↪our SOM object.
cl_labels = kmeans_clust(sm, KM_CLUSTERS)
```

```
Performing K-means clustering to SOM trained data…
```

[11]:
```
# This returns our function used to "filter"
# our output in order to make differences more exaggerated
def create_knee_function(cutoff: float, middle: float, maxm: float, minm:␣
↪float):
    def new_knee(x: float):
        # [-1,1] value determining closeness to "extremes"
        if x < middle:
            diff = 0.5*(x-middle)/(middle-minm)
        else:
            diff = 0.5*(x-middle)/(maxm-middle)
        # 0 if we're below our cutoff
        if np.abs(diff) < cutoff:
            return middle
        # Linearly interpolate between our middle and the boundary
        else:
            if diff < 0:
                interp_diff = (diff + cutoff)/(0.5 - cutoff)
                return middle + interp_diff*(middle - minm)
            else:
                interp_diff = (diff - cutoff)/(0.5 - cutoff)
                return middle + interp_diff*(maxm - middle)
    return new_knee

class ViewTFP2(View2D):
    """Map viewer override that allows for specifying the normalization␣
↪method"""

    def __init__(self, *args, knee_value=0, stdev_colorscale_coeff=1.0,␣
↪**kwargs):
        super().__init__(*args, **kwargs)
        self.knee_value = knee_value
        self.stdev_colorscale_coeff = stdev_colorscale_coeff

    def _calculate_figure_params(self, som, which_dim, col_sz,
                             width=None, height=None):
    """ Class method in MapView._calculate_figure_params() overrided """
    codebook = som._normalizer.denormalize_by(som.data_raw,
                                                  som.codebook.matrix)

    indtoshow, sV, sH = None, width, height

    if which_dim == 'all':
        dim = som._dim
        row_sz = np.ceil(float(dim) / col_sz)
        msz_row, msz_col = som.codebook.mapsize
        ratio_hitmap = msz_row / float(msz_col)
        ratio_fig = row_sz / float(col_sz)
        indtoshow = np.arange(0, dim).T
        sH, sV = (width, height) or (16, 16*ratio_fig*ratio_hitmap)

    elif type(which_dim) == int:
        dim = 1
        msz_row, msz_col = som.codebook.mapsize
        ratio_hitmap = msz_row / float(msz_col)
        indtoshow = np.zeros(1)
        indtoshow[0] = int(which_dim)
        sH, sV = (width, height) or (16, 16*ratio_hitmap)

    elif type(which_dim) == list:
        max_dim = codebook.shape[1]
        dim = len(which_dim)
        row_sz = np.ceil(float(dim) / col_sz)
        msz_row, msz_col = som.codebook.mapsize
        ratio_hitmap = msz_row / float(msz_col)
        ratio_fig = row_sz / float(col_sz)
        indtoshow = np.asarray(which_dim).T
        sH, sV = (width, height) or (16, 16*ratio_fig*ratio_hitmap)

    no_row_in_plot = dim / col_sz + 1  # 6 is arbitrarily selected
    if no_row_in_plot <= 1:
        no_col_in_plot = dim
    else:
        no_col_in_plot = col_sz

    axis_num = 0

    width = sH
    height = sV

    return (width, height, indtoshow, no_row_in_plot, no_col_in_plot,
            axis_num)
    def prepare(self, *args, **kwargs):
        self._close_fig()
        self._fig = plt.figure(figsize=(self.width, self.height))
        self._fig.suptitle(self.title)
        plt.rc('font', **{'size': self.text_size})

# NOTE: I'm using type rST syntax right now because I don't want to make
# excessive changes to make this use python's formal typing system.
# We may want to move to python's typing system later
    def show(self, som :sompy.sompy.SOM, cl_labels :list, what='codebook',␣
 ↪which_dim='all',
             cmap=None, col_sz=None, desnormalize=False, col_norm=None,␣
 ↪normalizer="linear", savepath="",
             isOutHtmap=True):
        """ Class method in View2D.show() overridden

        There's now an extra parameter, "col_norm", which is used to determine␣
 ↪whether to normalize by
        the median or the mean
        :param som: The self-organizing map to visualize
        :type som: sompy.sompy.SOM
        :param cl_labels: Cluster labels (?)
        :type cl_labels: list
        :param what: unused
        :type what: str
        :param which_dim: Which dimensions to display
        :type which_dim: "all" or int or list
        :param cmap: The color map to use for the plot
        :type cmap: matplotlib.colors.Colormap
        :param col_sz: Number of columns
        :type col_sz: integer
        :param desnormalize: Whether or not to denormalize the codebook
        :type desnormalize: Boolean
        :param col_norm: Determines what "middle value" to use for normalization
        :type col_norm: "median" or "mean"
        :param normalizer: Which normalizer to use
        :type normalizer: "linear" or "log"
        """
        (self.width, self.height, indtoshow, no_row_in_plot, no_col_in_plot,
         axis_num) = \
            self._calculate_figure_params(som, which_dim, col_sz,
                                          width=self.width, height=self.height)
        self.prepare()
        # Mathtext font to sans-serif
        mpl.rcParams['mathtext.fontset'] = 'custom'
        mpl.rcParams['mathtext.rm'] = 'sans\-serif'
        mpl.rcParams['mathtext.cal'] = 'sans\-serif'

        cmap = cmap or plt.get_cmap('RdYlBu_r')

        if not desnormalize:
            codebook = som.codebook.matrix
        else:
            codebook = som._normalizer.denormalize_by(som.data_raw,
                                                      som.codebook.matrix)

        if which_dim == 'all':
            names = som._component_names[0]
        elif type(which_dim) == int:
            names = [som._component_names[0][which_dim]]
        elif type(which_dim) == list:
            names = som._component_names[0][which_dim]

        while axis_num < len(indtoshow):
            axis_num += 1
            ax = plt.subplot(no_row_in_plot, no_col_in_plot, axis_num)
            ind = int(indtoshow[axis_num-1])

            if col_norm == 'median':  # normalized by median
                middle_point = np.median(codebook[:, ind].flatten())
            else: # normalized by mean
                middle_point = np.mean(codebook[:, ind].flatten())

            cb_min = np.min(codebook[:, ind].flatten())
            cb_max = np.max(codebook[:, ind].flatten())

            min_color_scale = middle_point - self.stdev_colorscale_coeff \
                * np.std(codebook[:, ind].flatten())
            max_color_scale = middle_point + self.stdev_colorscale_coeff \
                * np.std(codebook[:, ind].flatten())
            min_color_scale = min_color_scale if min_color_scale >= \
                cb_min else cb_min
            max_color_scale = max_color_scale if max_color_scale <= \
                cb_max else cb_max

            # FIXME: Break this out into less hacked-in code
            # "Middle color" should be mean or median?
            # The min value should be at the bottom of the range
            # The max value should be at the top of the range
            # The "min_color_scale" value should be the absolute bottom of the␣
↪original color range
            # The "max_color_scale" value should be the absolute top of the␣
↪original color range
```

```
            # .5 - .5 * min((middle_point - min_color_scale) / (middle_point -␣
↪cb_min), 1)
            # .5 + .5 * min((middle_point - max_color_scale) / (middle_point -␣
↪cb_max), 1)
            #
            # I probably can reorder this to make more sense
            # FIXME: This code currently biases the colors in a not very useful␣
↪manner
            #cmap_bottom = .5 * (1 - min((middle_point - min_color_scale) /␣
↪(middle_point - cb_min), 1))
            #cmap_top = .5 * (1 + min((middle_point - max_color_scale) /␣
↪(middle_point - cb_max), 1))
            #my_cmap = ListedColormap(cmap(np.linspace(cmap_bottom, cmap_top,␣
↪num=512)))
            my_cmap = cmap
            if normalizer == 'log':
                norm = mpl.colors.
↪SymLogNorm(linthresh=(max_color_scale-min_color_scale)/100, vmin=0.
↪7*min_color_scale,
                                            vmax=max_color_scale,
                                            clip=True)
            else:
                norm = mpl.colors.Normalize(vmin=min_color_scale*0.5,
                                              vmax=max_color_scale,
                                              clip=True)

            mp = codebook[:, ind].reshape(som.codebook.mapsize[0],
                                            som.codebook.mapsize[1])
            # FIXME: Break this out into less hacked-in code
            # Insert the scaling function here
            # as-is this is likely very slow - an immediate improvement would␣
↪come
            # from using something such as "numba" here
            scaler = np.vectorize(create_knee_function(self.knee_value,␣
↪middle_point, np.max(codebook[:, ind].flatten()), np.min(codebook[:, ind].
↪flatten())))
            pl = ax.pcolormesh(scaler(mp.T), norm=norm, cmap=my_cmap)
            ax.set_xlim([0, som.codebook.mapsize[0]])
            ax.set_ylim([0, som.codebook.mapsize[1]])
            ax.set_aspect('equal')
            ax.set_title(names[axis_num - 1],fontsize=22)
            # Disable ticks and tick labels
            disable_ticks = ["labelbottom", "labelleft", "bottom", "left",␣
↪"right", "top"]
            disable_ticks_dict = {a: False for a in disable_ticks}
            # This unpacks the dict into the keyword arguments,
            # setting them all as false without needing to write each one out
            ax.tick_params(**disable_ticks_dict)

            plt.colorbar(pl, ticks=None, shrink=1, fraction=0.046, pad=0.00441)␣
↪# format='%.1f',


            # draw line segment to the border of cluster
            msz = som.codebook.mapsize

            for i in range(len(cl_labels)):
                rect_x = [i // msz[1], i // msz[1],
                          i // msz[1] + 1, i // msz[1] + 1]
                rect_y = [i % msz[1], i % msz[1] + 1,
                          i % msz[1] + 1, i % msz[1]]

                if i % msz[1] + 1 < msz[1]:  # top border
                    if cl_labels[i] != cl_labels[i+1]:
                        ax.plot([rect_x[1], rect_x[2]],
                                [rect_y[1], rect_y[2]], 'k-', lw=4)

                if i + msz[1] < len(cl_labels):  # right border
                    if cl_labels[i] != cl_labels[i+msz[1]]:
                        ax.plot([rect_x[2], rect_x[3]],
                                [rect_y[2], rect_y[3]], 'k-', lw=4)


        # plt.subplots_adjust(wspace=0.1, hspace=0.1)
        plt.tight_layout()
        plt.show()

        # save figure of heat map
        if isOutHtmap:
            print("Saving figure of heat map for all thermofluid prop. to {}...
↪".format(savepath))
            self._fig.savefig(savepath)
```

[12]:

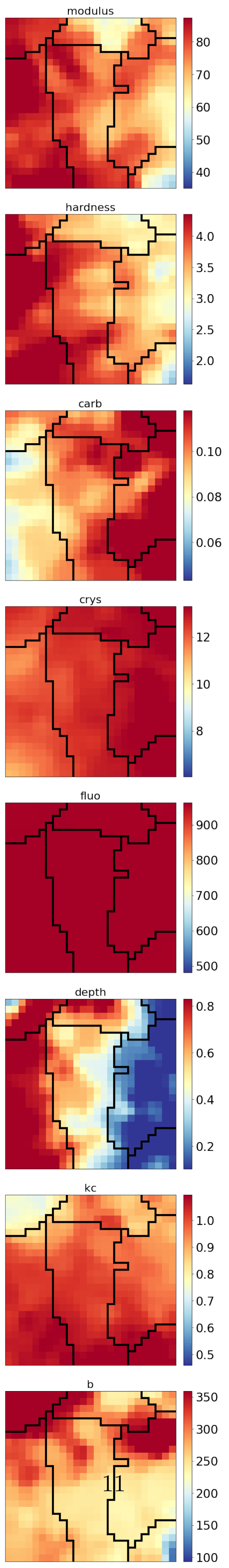
```
# Setting parameters for our heatmap
heatmap_size = (50, 50)
heatmap_col_sz = 1
gauss_alpha = None

cmap = plt.get_cmap('RdYlBu_r')  # set color map using Matplolib function

#Initializing a map viewer imported from mapview.py
viewTFP = ViewTFP(*heatmap_size, '', stdev_colorscale_coeff=1., text_size=20)
viewTFP2 = ViewTFP2(*heatmap_size, '', stdev_colorscale_coeff=1., text_size=24)
```

[13]:
```
# Initializing widget
my_out = widgets.Output()

# No scaling
viewTFP.knee_value = 0.00

# Displaying heatmap through the widget
# Uses matplotlib functions

with my_out:
    print("Linear scaling")
    viewTFP.show(sm, cl_labels, col_sz=heatmap_col_sz,
                          which_dim='all', desnormalize=True, col_norm='mean',
                          cmap=cmap, isOutHtmap=True)

my_out
```

```
Output()
```

[14]:
```
####### Produce Heat Maps #######

# No scaling
viewTFP2.knee_value = 0.00

viewTFP2.show(sm, cl_labels, col_sz=heatmap_col_sz,
                          which_dim='all', desnormalize=True, col_norm='median',
                          cmap=cmap, isOutHtmap=True)

# Dont Use These For This Set Up
# ax1 = cbar.ax
# ax1.set_position([1,0.5,0.5,.1])
```

```
<ipython-input-11-e54ed6dc4a3c>:142: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  ax = plt.subplot(no_row_in_plot, no_col_in_plot, axis_num)
```


modulus
80
70
60
50
40
hardness
4.0
3.5
3.0
2.5
2.0
carb
0.10
0.08
0.06
crys
12
10
8
fluo
900
800
700
600
500
depth
0.8
0.6
0.4
0.2
kc
1.0
0.9
0.8
0.7
0.6
0.5
b
350
300
250
200
150
100

```
Saving figure of heat map for all thermofluid prop. to …
```

```
[15]: my_out = widgets.Output() # Initializing widget
      cmap = plt.get_cmap('RdYlBu_r')  # set color map 'viridis' 'RdYlBu_r'

      # No scaling
      viewTFP.knee_value = 0.0
      with my_out:
          print("Log scaling")
          viewTFP.show(sm, cl_labels, col_sz=heatmap_col_sz,
                           which_dim='all', desnormalize=True, col_norm='mean',
                           cmap=cmap, normalizer="log")
      my_out
```

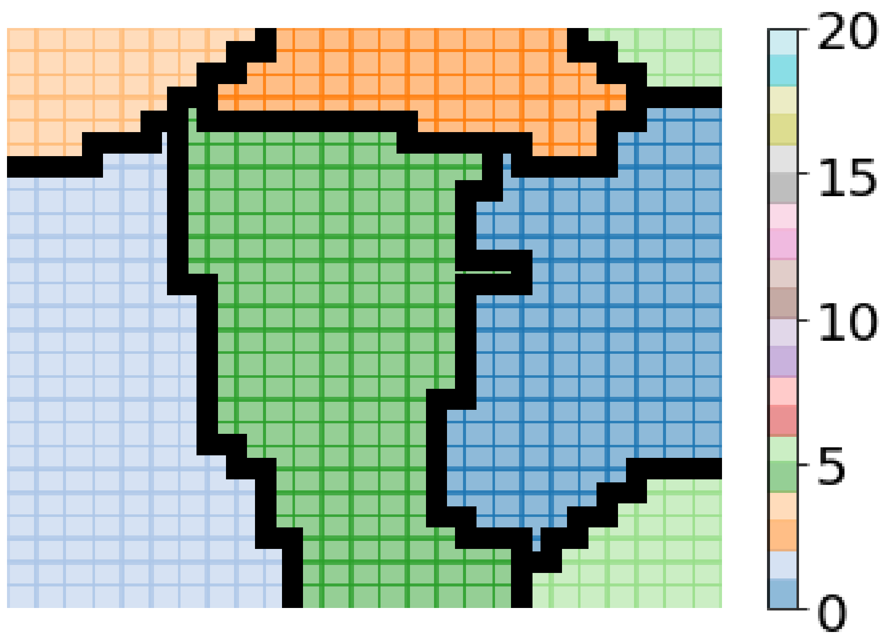
```
Output()
```

```
[16]: mats_df.index
```

```
[16]: Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
                  …
                  128, 129, 130, 131, 132, 133, 134, 135, 136, 137],
                 dtype='int64', length=138)
```

```
[17]: # Creating 2-D list organized by clusters
      # 5 cluster = 5 lists within this list
      clusters_list = sort_materials_by_cluster(sm, mats_df, cl_labels)

      # This makes it so it will display the full lists
      pd.set_option('display.max_rows', 2000)
      pd.set_option('display.max_columns', 50)
      pd.set_option('display.width', 1000)

      # This should be the last statement of the cell, to make it display
      # That, or assign the return value to a variable, and have that variable be the␣
       ↪final expression in a cell
      cluster_tabs(sm, mats_df, clusters_list, cl_labels)
```

```
Tab(children=(HBox(children=(VBox(children=(HBox(children=(Label(value='Cluster␣
↪#0'), HTML(value="<div style='…
```



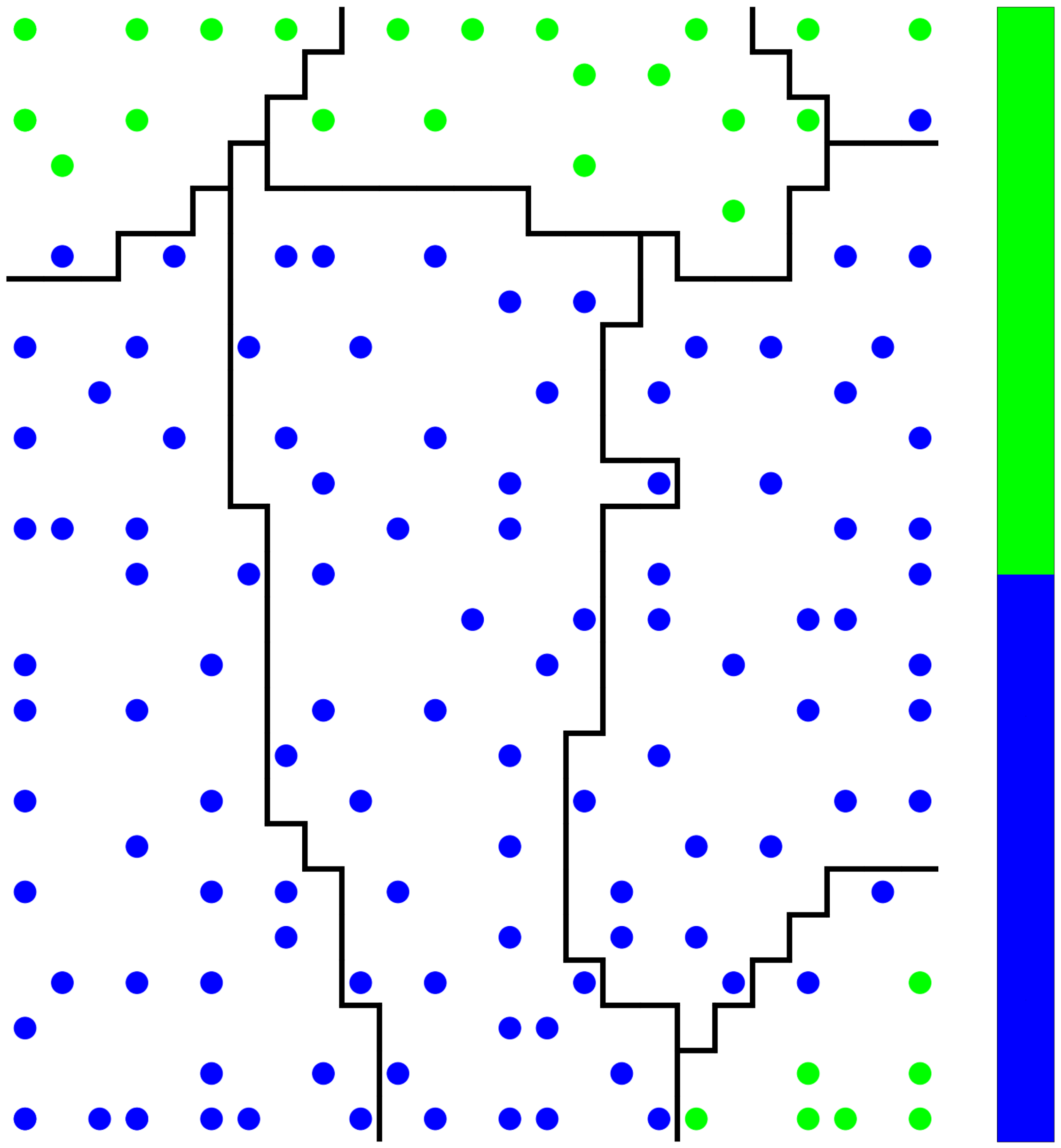
[18]:
```
mats_df["Row"] = None
```

[19]:
```
# cluster number -> color graph
def make_cluster_graph(mysom: sompy.sompy.SOM, cl_labels: np.ndarray):
    Col1=6 #Total figure width
    pl1=2.7 #Drawing area width
    AR=14.63/18.47 #Height/width of figure.

    fig, ax = plt.subplots(figsize=[Col1,Col1*AR])
    #fig, ax = plt.subplots(1, 1)
    n_palette = 20
    cmap = plt.get_cmap("tab20")
    norm = mpl.colors.Normalize(vmin=0, vmax=n_palette, clip=True)
    msz = mysom.codebook.mapsize
    pl = ax.pcolormesh(cl_labels.reshape(msz[0], msz[1]).T % n_palette,
                  cmap=cmap, norm=norm, edgecolors='face',
                  lw=0.5, alpha=0.5)
    render_cluster_borders_to_axes(ax, cl_labels, msz)
    ax.axis('off')
    plt.tight_layout(rect=[0,0,1,1])
    fig.colorbar(pl, ax=ax)
    return fig, ax
```

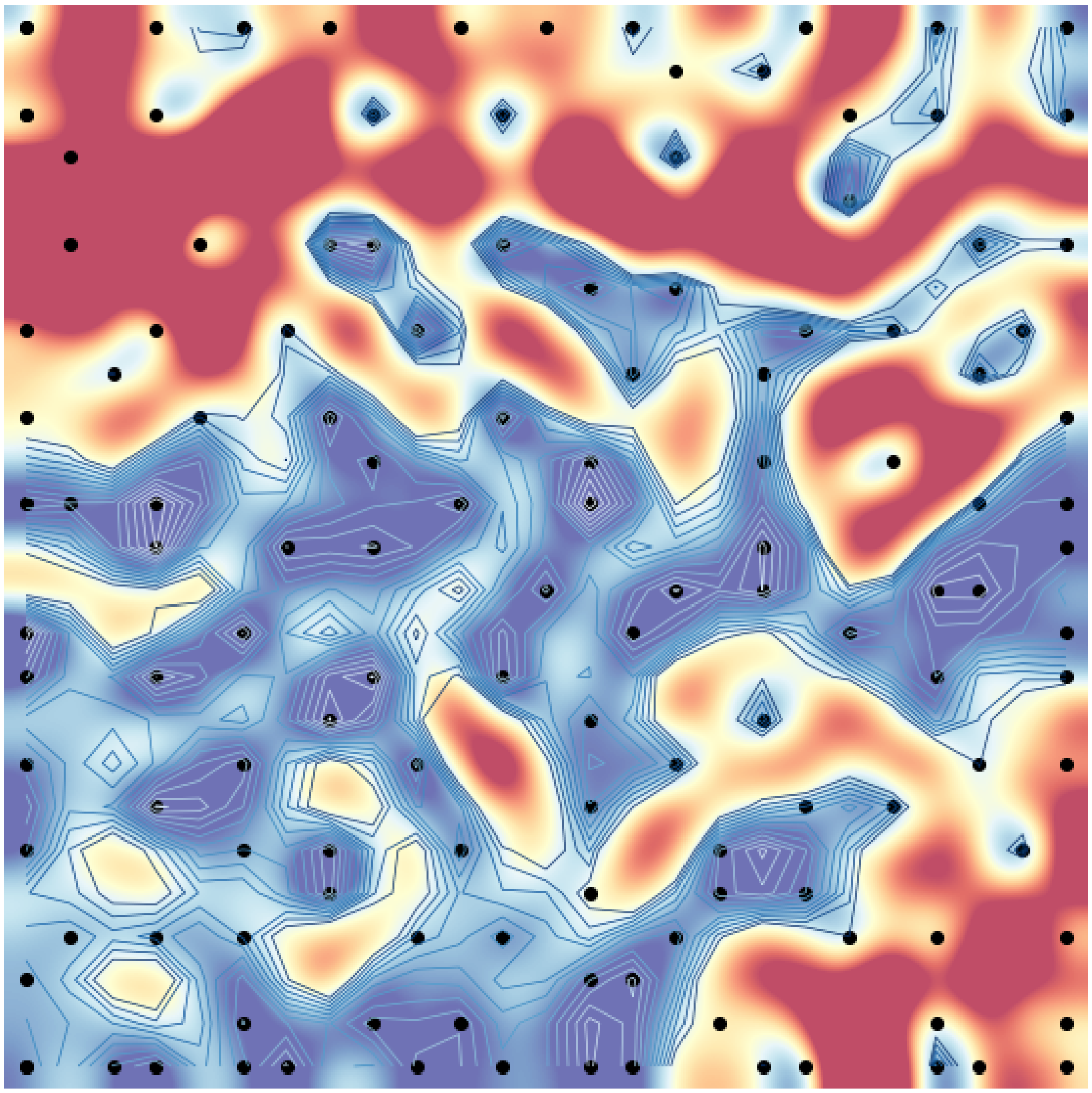
[20]:
```
# This function prints labels on cluster map
```

```
def clusteringmap_category(sm,n_clusters,dataset,colorcategory,labels,␣
↪savepath):
    """
    Description:
    This function is used to output maps that prints colors on dots based
    on their properties
    """
    categories = dataset[colorcategory] #if colorcategory is one col of the␣
↪dataset
    cmap = plt.get_cmap("Greys") #cmap for background
    n_palette = 100000  # number of different colors in this color palette
    color_list = [cmap((i % n_palette)/n_palette) for i in range(n_clusters)]
    msz = sm.codebook.mapsize
    proj = sm.project_data(sm.data_raw)
    coord = sm.bmu_ind_to_xy(proj)

    fig, ax = plt.subplots(1, 1, figsize=(30,30))

    cl_labels = clust.KMeans(n_clusters=n_clusters,random_state=555).
↪fit_predict(sm.codebook.matrix)

    # fill each rectangular unit area with cluster color
    #  and draw line segment to the border of cluster
    norm = mpl.colors.Normalize(vmin=0, vmax=n_palette, clip=True)
    ax.pcolormesh(cl_labels.reshape(msz[0], msz[1]).T % n_palette,
                  cmap=cmap, norm=norm, edgecolors='face',
                  lw=0.5, alpha=0.5)

    ax.scatter(coord[:, 0]+0.5, coord[:, 1]+0.5, c='k', marker='o')
    ax.axis('off')

    categoryname = list(dataset.groupby(colorcategory).count().index)
    categories_int = categories.apply(categoryname.index)

    N = len(categoryname)
    cmap_labels = plt.cm.brg
    # extract all colors from the .jet map
    cmaplist = [cmap_labels(i) for i in range(cmap_labels.N)]
    # create the new map
    cmap_labels = cmap_labels.from_list('Custom cmap', cmaplist, cmap_labels.N)
    # define the bins and normalize
    bounds = np.linspace(0,N,N+1)
    norm_labels = mpl.colors.BoundaryNorm(bounds, cmap_labels.N)

    scat = ax.scatter(coord[:, 0]+0.5, coord[:, 1]+0.5,␣
↪c=categories_int,s=1000,cmap=cmap_labels,norm=norm_labels)
    cbar = plt.colorbar(scat, spacing='proportional',ticks=bounds)
    cbar.ax.get_yaxis().set_ticks([])

    for j, lab in enumerate(categoryname):
        cbar.ax.text(1, (2 * j + 1) / (2*(len(categoryname))), lab, ha='left',␣
↪va='center', fontsize=0)
    cbar.ax.get_yaxis().labelpad = 15
    # cbar.ax.set_ylabel('# of contacts', rotation=270)
    ax.axis('off')


    for label, x, y in zip(labels, coord[:, 0], coord[:, 1]):
        x += 0.2
        y += 0.2
        # "+ 0.1" means shift of label location to upperright direction

        # randomize the location of the label
        #   not to be overwrapped with each other
        x += 0.1 * np.random.randn()
        y += 0.3 * np.random.randn()

        # wrap of label for chemical compound
        #label = str_wrap(label)

        ax.text(x+0.4, y+0.4, label, horizontalalignment='left',␣
↪verticalalignment='bottom',rotation=30, fontsize=12, weight='semibold')
    # cl_labels = som.cluster(n_clusters)
    cl_labels = clust.KMeans(n_clusters = n_clusters, random_state = 555).
↪fit_predict(sm.codebook.matrix)

    for i in range(len(cl_labels)):
        rect_x = [i // msz[1], i // msz[1],
                  i // msz[1] + 1, i // msz[1] + 1]
        rect_y = [i % msz[1], i % msz[1] + 1,
                  i % msz[1] + 1, i % msz[1]]

        if i % msz[1] + 1 < msz[1]:  # top border
            if cl_labels[i] != cl_labels[i+1]:
                ax.plot([rect_x[1], rect_x[2]],
                        [rect_y[1], rect_y[2]], 'k-', lw=8)

        if i + msz[1] < len(cl_labels):  # right border
            if cl_labels[i] != cl_labels[i+msz[1]]:
                ax.plot([rect_x[2], rect_x[3]],
                        [rect_y[2], rect_y[3]], 'k-', lw=8)

    plt.savefig(savepath)
    return cl_labels, ax, fig
```

```
[21]: # Generating another cluster map to overlay material identification

      ############## Figure widths in inches, nature standard ##############

      # fig, ax = make_cluster_graph(sm, cl_labels)

      # Labels material identification on each data points on the cluster map
      # Requires a column for color category
      cl_labels, ax, fig = clusteringmap_category(sm, KM_CLUSTERS, mats_df, "mammal",␣
       ↪mats_df["Row"],'plot.png')

      #ax1 = fig.get_axes()

      #ax1[1].set_position([.5,.5,.5,.5])

      # ax.properties()
```



[22]:
```python
# Initializing size
umat_size = (50, 50)

# Utilizes SOMPY's UMatrixView, which uses matplotlib
umat = UMatrixTFP(*umat_size, 'U-matrix')

# Plots the UMatrix map
# Must set "ifOutUmat" to True in order to output a image file
umat.show(sm, mats_df, mats_df, 'umatrix.png',cmap=cmap, isOutUmat=True)
```



```
Saving figure of U-matrix to umatrix.png…
```

```
[22]: ([],
        array([[0.37317606, 0.39076175, 0.40042121, 0.42511023, 0.44975078,
                0.35097839, 0.32043691, 0.36475934, 0.42360291, 0.30913562,
                0.24243348, 0.52641792, 0.48334425, 0.31473974, 0.44209173,
                0.53833193, 0.5818204 , 0.66123886, 0.99376887, 1.34276046,
                1.41531033, 0.85840399, 0.57019541, 0.63102414, 0.51195197],
               [0.42662793, 0.41189165, 0.43859362, 0.41482191, 0.4609754 ,
                0.47122165, 0.42601603, 0.41656121, 0.42878985, 0.4039293 ,
                0.39200667, 0.53754726, 0.44018139, 0.33043614, 0.46359715,
                0.55255457, 0.53598171, 0.73455124, 0.89999592, 0.91520203,
                1.03860161, 0.83883411, 0.72555258, 0.77373092, 0.76075039],
               [0.35112846, 0.43128235, 0.53232877, 0.44205611, 0.50435585,
                0.54254445, 0.40738645, 0.46144375, 0.42344022, 0.39044725,
                0.54622528, 0.51302932, 0.33044715, 0.32762093, 0.53358436,
                0.65025542, 0.48458573, 0.62345564, 0.88666111, 1.17592573,
                1.24925753, 0.89212393, 0.77647788, 0.8039997 , 0.85402051],
               [0.3144104 , 0.46542526, 0.54734353, 0.42273056, 0.5127573 ,
                0.46867345, 0.27014282, 0.40734972, 0.40860071, 0.27380071,
                0.48078994, 0.49612545, 0.23328278, 0.21517174, 0.39022745,
                0.61785969, 0.57928191, 0.59750005, 0.89521823, 1.0751143 ,
                1.36826434, 0.97337562, 0.52495928, 0.56975826, 0.58548501],
               [0.42103399, 0.44170049, 0.42178242, 0.3930278 , 0.44869413,
                0.38216421, 0.28380008, 0.33310041, 0.40417658, 0.31881213,
                0.36782949, 0.53914873, 0.40723485, 0.3070725 , 0.33424362,
                0.4709035 , 0.71978336, 0.92480643, 0.89130613, 0.58383546,
                0.97046287, 1.09768224, 0.57913521, 0.50811403, 0.46637679],
               [0.36161399, 0.33211299, 0.34325641, 0.43362299, 0.4497005 ,
                0.39260003, 0.33226398, 0.3004502 , 0.4193215 , 0.39598886,
                0.27309396, 0.40818351, 0.42592049, 0.42312905, 0.45765446,
                0.49295075, 0.63662404, 0.7775049 , 0.80240155, 0.68616396,
                0.87972354, 1.35491381, 1.04082858, 0.5287578 , 0.45434754],
               [0.29001177, 0.35191807, 0.48434799, 0.51749408, 0.39884318,
                0.38701766, 0.44767039, 0.4301042 , 0.35937226, 0.39021082,
                0.41116216, 0.3648572 , 0.29342899, 0.41116959, 0.4905932 ,
                0.45945137, 0.46087232, 0.47554091, 0.6561767 , 0.646757  ,
                0.77336479, 1.20161022, 1.21447061, 0.74934136, 0.60469078],
               [0.38948196, 0.37326074, 0.57032443, 0.58111518, 0.30351688,
                0.2937863 , 0.53340908, 0.51658355, 0.25239866, 0.28406101,
                0.46164487, 0.39411355, 0.29994901, 0.36127088, 0.34684854,
                0.30295122, 0.45889889, 0.65356507, 0.52018393, 0.30543866,
                0.56881384, 0.78798479, 0.70693803, 0.71071531, 0.61154597],
               [0.39360456, 0.32232602, 0.40306668, 0.55794169, 0.46760136,
                0.44840467, 0.52137419, 0.45492598, 0.32283879, 0.25208092,
                0.4012182 , 0.38032115, 0.29119629, 0.33634336, 0.325379  ,
                0.40222224, 0.58777587, 0.61354404, 0.40728496, 0.27676377,
                0.59786217, 0.67553081, 0.41250294, 0.60924323, 0.78989265],
               [0.30668097, 0.32950867, 0.37506253, 0.44460183, 0.51618776,
                0.5188758 , 0.39063947, 0.32598491, 0.45940654, 0.45904241,
                0.49566706, 0.43807672, 0.31565223, 0.33034915, 0.37659231,
                0.58472052, 0.53931949, 0.31901018, 0.45853831, 0.53730221,
                0.81616639, 0.80610079, 0.61287401, 0.74875102, 0.83342686],
               [0.3111605 , 0.31401657, 0.38073996, 0.41848707, 0.42979331,
                0.37892445, 0.40016618, 0.57150811, 0.67874326, 0.49550233,
                0.39697953, 0.49918266, 0.39363768, 0.2932814 , 0.43514119,
                0.51608843, 0.50046277, 0.44812924, 0.52957151, 0.5527543 ,
                0.80726874, 0.77492694, 0.64260552, 0.70074044, 0.58770625],
               [0.36559008, 0.38196811, 0.41892894, 0.3821469 , 0.44825928,
                0.48105853, 0.60434173, 0.76474495, 0.543899  , 0.29357153,
                0.30637252, 0.38489502, 0.43757491, 0.42960524, 0.41168225,
                0.30573803, 0.51960444, 0.72528708, 0.49132206, 0.32451518,
                0.58907955, 0.55836027, 0.43170014, 0.60119675, 0.60435638],
               [0.35629731, 0.35632924, 0.41257275, 0.42148373, 0.44952023,
                0.52328714, 0.55355533, 0.50368971, 0.42004192, 0.41032148,
                0.39473079, 0.30729485, 0.36398749, 0.37374119, 0.36305019,
                0.43045754, 0.68135981, 0.63862581, 0.38670101, 0.39686967,
                0.71068049, 0.80005024, 0.61289669, 0.65774095, 0.62358603],
               [0.27149045, 0.26567012, 0.33366916, 0.4608993 , 0.50594703,
                0.44821494, 0.37391688, 0.3490283 , 0.36523112, 0.41185916,
                0.40996146, 0.40513783, 0.36786664, 0.21691024, 0.28453821,
                0.5358725 , 0.58584902, 0.42967088, 0.33422319, 0.50240299,
                1.08126919, 1.02344615, 0.6874896 , 0.57710801, 0.59595313],
               [0.30102867, 0.30180272, 0.28854821, 0.38444102, 0.57538715,
                0.64715012, 0.48993256, 0.35929344, 0.42323519, 0.39168428,
                0.2939959 , 0.33691337, 0.44353873, 0.33653431, 0.4137359 ,
                0.5166364 , 0.38095851, 0.3897126 , 0.39471092, 0.76627735,
                1.06814475, 0.63661645, 0.59310712, 0.50859278, 0.45861184],
               [0.51408005, 0.44980308, 0.39821304, 0.35924363, 0.44536479,
                0.60687348, 0.61450862, 0.41660546, 0.55899193, 0.57610425,
                0.34628765, 0.27034468, 0.4152742 , 0.49158388, 0.60820229,
                0.6181928 , 0.51276732, 0.42341182, 0.3705924 , 0.74606086,
                0.72600668, 0.40196073, 0.54322677, 0.52620426, 0.52906139],
               [0.5919386 , 0.55475899, 0.58371201, 0.47973882, 0.32184656,
                0.33325459, 0.54301969, 0.56386608, 0.53925484, 0.58569274,
                0.47286019, 0.31019346, 0.37514596, 0.45468826, 0.50367107,
                0.55816521, 0.5273753 , 0.46378174, 0.50742139, 0.84496765,
                0.96319601, 0.67980821, 0.63836736, 0.50536857, 0.65041136],
               [0.48695489, 0.56160917, 0.7220856 , 0.57956509, 0.29953137,
                0.26800178, 0.4643526 , 0.60888607, 0.41853824, 0.49415902,
                0.50273996, 0.27395185, 0.2661928 , 0.33751887, 0.36358368,
                0.38771503, 0.3998607 , 0.36862521, 0.59897817, 1.0521555 ,
                1.1160787 , 0.87264943, 0.62161248, 0.45358501, 0.54530169],
               [0.52377734, 0.73469599, 0.7815243 , 0.52683382, 0.35338345,
                0.33170554, 0.42880672, 0.62183553, 0.58054986, 0.51263219,
                0.44016199, 0.38283823, 0.37888464, 0.51546713, 0.61635343,
                0.65545299, 0.48957328, 0.34410604, 0.72153255, 0.97002121,
                0.59240469, 0.61701029, 0.70995311, 0.6402336 , 0.48967669],
               [1.09182016, 1.04678585, 0.80099574, 0.49007731, 0.46489779,
                0.45213129, 0.38345426, 0.54114534, 0.67277273, 0.47510931,
                0.32643292, 0.46916066, 0.67905552, 0.62825163, 0.5821275 ,
                0.86019277, 0.67732979, 0.42690332, 0.75645837, 0.73428277,
                0.34333293, 0.39033664, 0.57576866, 0.78510284, 0.88335113],
               [0.95779482, 0.87155045, 0.87062577, 0.60870174, 0.62224649,
                0.58882301, 0.43483189, 0.60718439, 0.53850283, 0.39046765,
                0.36373332, 0.40089119, 0.65733457, 0.72069426, 0.49031037,
                0.75549837, 0.75362751, 0.43879845, 0.53554723, 0.787674  ,
                0.6430256 , 0.47539896, 0.46869056, 0.57555092, 0.76235243],
               [0.40797112, 0.52723749, 0.70912386, 0.58704336, 0.67446352,
                0.70686952, 0.58575068, 0.59060301, 0.40300656, 0.33263074,
                0.32490723, 0.26295175, 0.36287882, 0.69051949, 0.79754076,
                0.77636557, 0.59137777, 0.52877177, 0.44811296, 0.57869103,
                0.79796128, 0.58695316, 0.43915276, 0.45807081, 0.410507  ],
               [0.56578374, 0.66833743, 0.84250198, 0.80070013, 0.68306256,
                0.56887005, 0.52585501, 0.48670191, 0.47145327, 0.45754967,
                0.33050593, 0.26041086, 0.29941596, 0.43759724, 0.75427234,
                0.74651725, 0.40894753, 0.49934407, 0.53194   , 0.39562645,
                0.6244876 , 0.7746353 , 0.63006698, 0.54207626, 0.57587271],
               [0.68798216, 0.64717158, 0.78557438, 0.83483918, 0.67965616,
                0.44835996, 0.56729208, 0.59251007, 0.53469443, 0.46508798,
                0.35295295, 0.32705262, 0.3151689 , 0.33127889, 0.4459132 ,
                0.6326786 , 0.50590128, 0.42653821, 0.56183568, 0.47607809,
                0.62665737, 0.78220303, 0.59476855, 0.50042913, 0.55797781],
               [0.59398556, 0.57794926, 0.68670189, 0.65964845, 0.75370361,
                0.80869301, 0.76018435, 0.65481557, 0.64793631, 0.45641862,
                0.35550611, 0.38118811, 0.32000201, 0.33031016, 0.3546421 ,
                0.49880329, 0.61870559, 0.64856327, 0.66572734, 0.47028743,
                0.71722405, 0.70616126, 0.40110409, 0.4295862 , 0.39542477]]))
```

[23]:
```
# Calculate quantization error value
sm.calculate_quantization_error()
```

[23]:
```
0.09080619458930875
```

[24]:
```
# Calculate topographic error value
sm.calculate_topographic_error()
```

[24]:
```
0.014492753623188406
```

[ ]:
```

```