Mathematical Tools for Big Data - Assignment 3

Students :

- **Alexandra de Carvalho** (93346)
- **Diogo Pedrosa** (94358)
- **Roshan Poudel** (109806)

Common Steps

Reading and General data analysis

In [1]:

```
import pandas as pd
import numpy as np
import random

random.seed(2022)
# reading data
data = pd.read_csv("data/cash-crops-nepal.csv")
# visualize some data
data.iloc[:5,:]
```

Out[1]:

	Year AD	Year BS	Crop	Area	Production	Yield
0	1984/85	2041/42	OILSEED	127820	84030	657
1	1985/86	2042/43	OILSEED	137920	78390	568
2	1986/87	2043/44	OILSEED	142890	82500	577
3	1987/88	2044/45	OILSEED	151490	94370	623
4	1988/89	2045/46	OILSEED	154860	99190	641

In [2]:

```
data.shape
```

Out[2]:

```
(105, 6)
```

In [3]:

```
# What are the different crops
print(data.iloc[:,2].unique())
```

```
['OILSEED' 'POTATO' 'TOBACCO' 'SUGARCANE' 'JUTE']
```

In [4]:

```
# shuffling data
agri_data = data.iloc[np.random.permutation(len(data))]
trunc_data = agri_data[["Area", "Production", "Yield"]]
trunc_data.iloc[:5,:]
```

Out[4]:

	Area	Production	Yield
65	24910	616580	24752
44	8820	4890	554

	Area	Production	Yield
8	165240	93690	567
57	4283	3809	889
42	8550	6430	752

In [5]:

```
# (custom choice for) normalizing data
trunc_data = trunc_data / trunc_data.max()
trunc_data.iloc[:5,:]
```

Out[5]:

	Area	Production	Yield
65	0.130810	0.259492	0.615461
44	0.046316	0.002058	0.013775
8	0.867725	0.039430	0.014099
57	0.022491	0.001603	0.022105
42	0.044899	0.002706	0.018699

Loading SOM utils

In [6]:

```
from scripts.som_utils import *
```

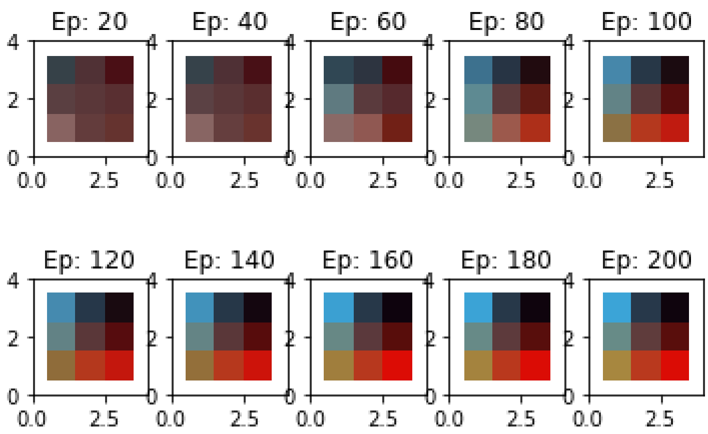
Base SOM (SOM1)

In [7]:

```
from scripts.our_som1 import SOM as SOM_1_base
# som = SOM(x_size, y_size, num_features)
som_1 = SOM_1_base(3,3,3)
joined_df, clustered_df = som_train_predict(som_1, trunc_data, agri_data, num
joined_df.iloc[0:5]
```

```
[3 3]
SOM training epoches 20
neighborhood radius  2.6878753795222865
learning rate  0.009048374180359595
-------------------------------------
SOM training epoches 40
neighborhood radius  2.4082246852806923
learning rate  0.008187307530779819
-------------------------------------
SOM training epoches 60
neighborhood radius  2.157669279974593
learning rate  0.007408182206817179
-------------------------------------
SOM training epoches 80
neighborhood radius  1.9331820449317627
learning rate  0.006703200460356393
-------------------------------------
SOM training epoches 100
neighborhood radius  1.7320508075688772
learning rate  0.006065306597126334
-------------------------------------
SOM training epoches 120
neighborhood radius  1.5518455739153598
learning rate  0.005488116360940264
```

```
-------------------------------------
SOM training epoches 140
neighborhood radius  1.3903891703159093
learning rate  0.004965853037914096
-------------------------------------
SOM training epoches 160
neighborhood radius  1.2457309396155174
learning rate  0.004493289641172216
-------------------------------------
SOM training epoches 180
neighborhood radius  1.1161231740339044
learning rate  0.004065696597405992
-------------------------------------
SOM training epoches 200
neighborhood radius  1.0
learning rate  0.0036787944117144234
-------------------------------------
```

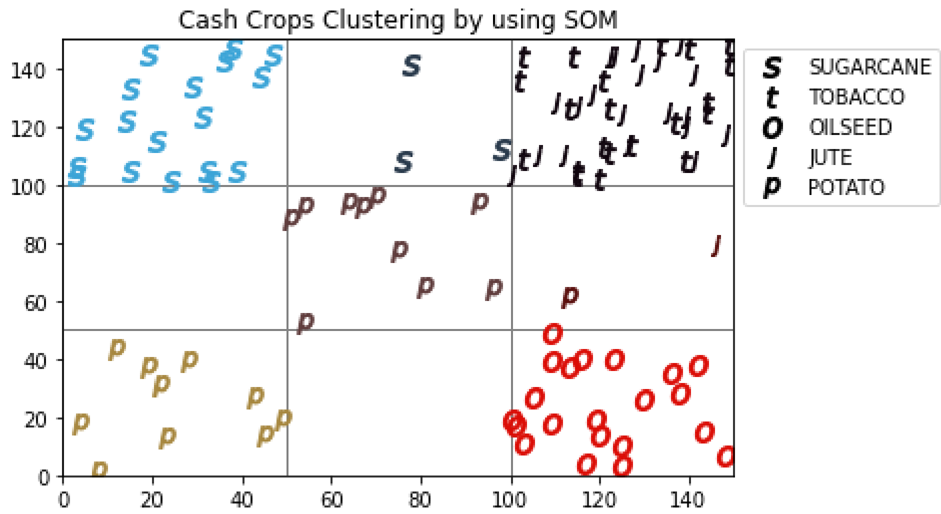


Out[7]:

	Year AD	Year BS	Crop	Area	Production	Yield	Area_norm	Production_norm
65	1986/87	2043/44	SUGARCANE	24910	616580	24752	0.130810	0.259492
44	1986/87	2043/44	TOBACCO	8820	4890	554	0.046316	0.002058
8	1992/93	2049/50	OILSEED	165240	93690	567	0.867725	0.039430
57	1999/2000	2056/57	TOBACCO	4283	3809	889	0.022491	0.001603
42	1984/85	2041/42	TOBACCO	8550	6430	752	0.044899	0.002706

In [8]:

```
visualize_som(som_1, joined_df)
```



In [9]:

```
trunc_data.head()
```

Out[9]:

	Area	Production	Yield
65	0.130810	0.259492	0.615461
44	0.046316	0.002058	0.013775
8	0.867725	0.039430	0.014099
57	0.022491	0.001603	0.022105
42	0.044899	0.002706	0.018699

In [10]:

```
cd = pd.DataFrame(clustered_df['bmu'].apply(lambda x: x[0].tolist()), index =
cd = cd.bmu.apply(list).apply(pd.Series).astype(float)
cd.head()
```

Out[10]:

	0	1	2
65	0.153170	0.220287	0.290279
44	0.059699	0.013780	0.050131
8	0.859671	0.048252	0.019456
57	0.059699	0.013780	0.050131
42	0.059699	0.013780	0.050131

Q1: What is the numeric criteria that you may use to determine if a change in the algorithm produces improvements?

Throughout this assignment, we will be exploring different changes to the proposed algorithm, and their impact in results. Thus, we are in need of a numerical criteria that will allow us to measure results. For this reason, we are going to use the metrics of neighbourhood preservation and trustworthiness. Neighborhood preservation, like the other quality measures, assesses the extent to which neighborhoods present in the input are also present in the projection. The opposite of trustworthiness is how much the neighborhoods in the projection are present in the input [2]. These measure how the projection preserves the neighborhoods present in the input space by ranking the k-nearest neighbors of each sample before and after projection. The implementation of this criteria is in function `neighborhood_preservation_trustworthiness` inside `soms/som_utils.py` file.

We also have implemented `quantization error` as a metric which is the mean euclidean distance between a data sample and its best-matching unit. The lower the quantization error the better SOM is. The implementation of this criteria is in function `quantization_error_test` inside `soms/som_utils.py` file. These metrics were tested by the authors of papers [1] and [2].

In [11]:
```
neighborhood_preservation_trustworthiness(1, trunc_data, cd)
```

Out[11]:
```
(0.9672676837725381, 0.8807023650884301)
```

In [12]:
```
quantization_error_test(trunc_data, cd)
```

Out[12]:
```
0.1460964878053812
```

Q2: Write the version SOM1A, where you change the curve of the learning factor. Did you achieve improvements?

The learning rate controls the size of weight vector. Therefore, chosing its decay function is important. There are many learning rate functions, like the power series implemented in `scripts/our_som1.py` . We changed the learning rate to:

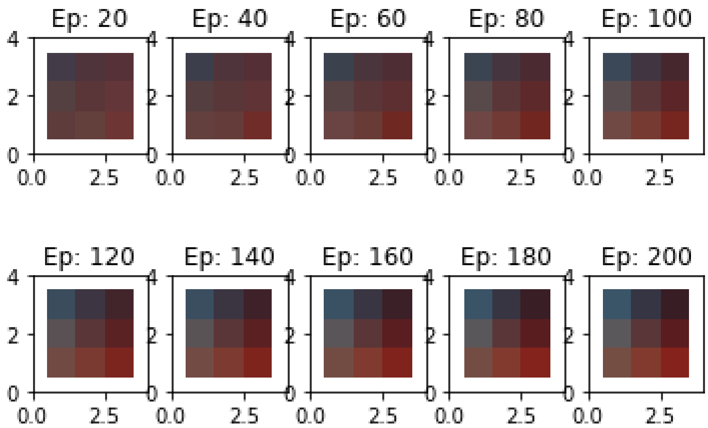
$$learning\_rate = initial\_learning\_rate \times \frac{1}{iteration}$$

We found that this linear learning rate, implemented in `scripts/our_som1_A.py` , did not improves the algorithm performance, as shown by comparing the measurement function results.

In [13]:
```
from scripts.our_som1_A import SOM as SOM_1_A
# som = SOM(x_size, y_size, num_features)
som_1_A = SOM_1_A(3,3,3)
joined_df, clustered_df = som_train_predict(som_1_A, trunc_data, agri_data, r
#joined_df.iloc[0:5]
```

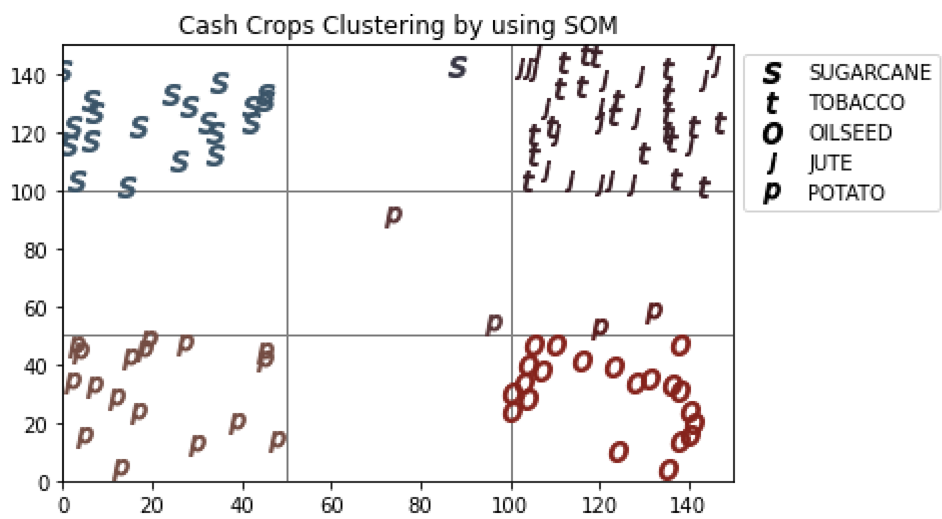
```
SOM training epoches 20
neighborhood radius  2.6878753795222865
learning rate  0.0005
-------------------------------------
SOM training epoches 40
neighborhood radius  2.4082246852806923
learning rate  0.00025
-------------------------------------
SOM training epoches 60
neighborhood radius  2.157669279974593
```

```
learning rate  0.00016666666666666666
-------------------------------------
SOM training epoches 80
neighborhood radius  1.9331820449317627
learning rate  0.000125
-------------------------------------
SOM training epoches 100
neighborhood radius  1.7320508075688772
learning rate  0.0001
-------------------------------------
SOM training epoches 120
neighborhood radius  1.5518455739153598
learning rate  8.333333333333333e-05
-------------------------------------
SOM training epoches 140
neighborhood radius  1.3903891703159093
learning rate  7.142857142857143e-05
-------------------------------------
SOM training epoches 160
neighborhood radius  1.2457309396155174
learning rate  6.25e-05
-------------------------------------
SOM training epoches 180
neighborhood radius  1.1161231740339044
learning rate  5.555555555555556e-05
-------------------------------------
SOM training epoches 200
neighborhood radius  1.0
learning rate  5e-05
-------------------------------------
```



In [14]:

```
visualize_som(som_1_A, joined_df)
```



In [15]:

```
cd = pd.DataFrame(clustered_df['bmu'].apply(lambda x: x[0].tolist()), index =
cd = cd.bmu.apply(list).apply(pd.Series).astype(float)
```

In [16]:

```
neighborhood_preservation_trustworthiness(1, trunc_data, cd)
```

Out[16]:

```
(0.9900138696255201, 0.8622012644682984)
```

In [17]:

```
quantization_error_test(trunc_data, cd)
```

Out[17]:

```
0.4853763903076811
```

Q3: Write the version SOM1B, where you change the curve of the deviation. Did you achieve improvements?

We changed the curve of deviation (decay_radius) in SOM_B, we changed it to :

$$radius\_decay = 0.1$$

$$radius = init\_radius \times e^{-iteration \times radius\_decay}$$

We changed the curve of deviation slighly only as we wanted it to be somewhat similar with original curve of deviation[3].

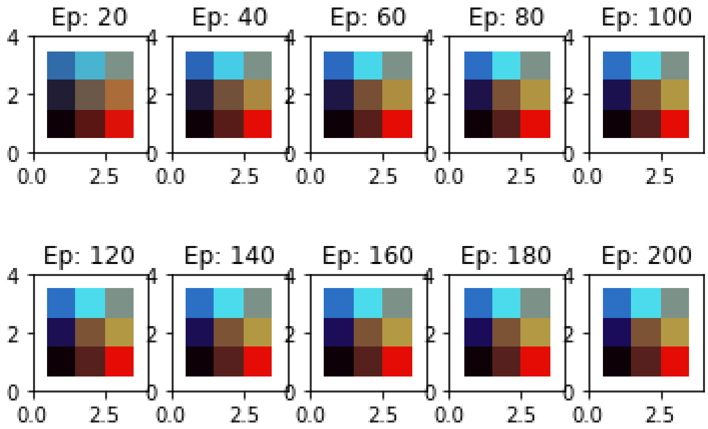
Changing the curve of deviation did bring better results. But they are still more or less comparable and not that different which can be seen in visualizations as well.

In [18]:

```
from scripts.our_som1_B import SOM as SOM_1_B
# som = SOM(x_size, y_size, num_features)
som_1_B = SOM_1_B(3,3,3)
joined_df, clustered_df = som_train_predict(som_1_B, trunc_data, agri_data, r
#joined_df.iloc[0:5]
```

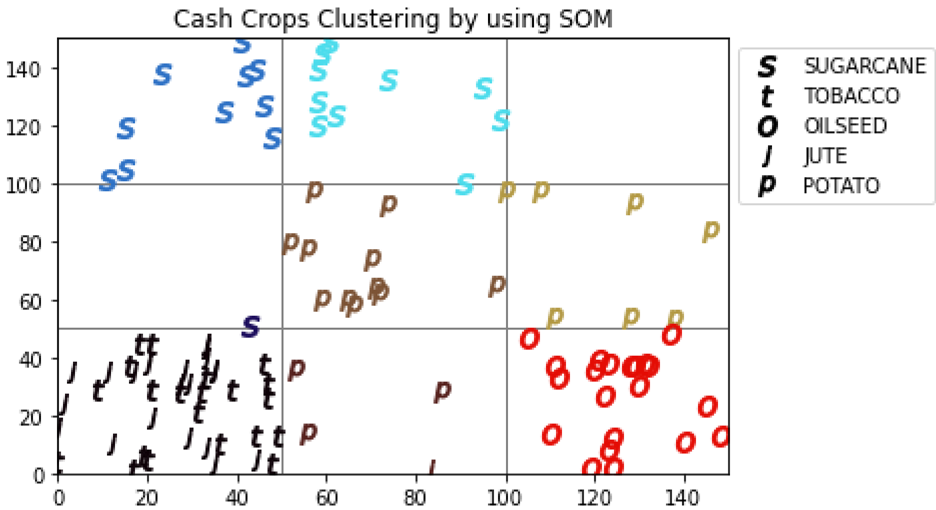
```
SOM training epoches 20
neighborhood radius  0.4060058497098381
learning rate  0.009048374180359595
-------------------------------------
SOM training epoches 40
neighborhood radius  0.05494691666202536
learning rate  0.008187307530779819
-------------------------------------
SOM training epoches 60
neighborhood radius  0.007436256529990755
learning rate  0.007408182206817179
-------------------------------------
SOM training epoches 80
neighborhood radius  0.0010063878837075356
learning rate  0.006703200460356393
-------------------------------------
SOM training epoches 100
neighborhood radius  0.00013619978928745456
learning rate  0.006065306597126334
-------------------------------------
```

```
SOM training epoches 120
neighborhood radius  1.843263705998463e-05
learning rate  0.005488116360940264
-------------------------------------
SOM training epoches 140
neighborhood radius  2.494586157310704e-06
learning rate  0.004965853037914096
-------------------------------------
SOM training epoches 160
neighborhood radius  3.3760552415777734e-07
learning rate  0.004493289641172216
-------------------------------------
SOM training epoches 180
neighborhood radius  4.568993923413789e-08
learning rate  0.004065696597405992
-------------------------------------
SOM training epoches 200
neighborhood radius  6.183460867315673e-09
learning rate  0.0036787944117144234
-------------------------------------
```



In [19]:

```
visualize_som(som_1_B, joined_df)
```



In [20]:

```
cd = pd.DataFrame(clustered_df['bmu'].apply(lambda x: x[0].tolist()), index =
cd = cd.bmu.apply(list).apply(pd.Series).astype(float)

neighborhood_preservation_trustworthiness(1, trunc_data, cd)
```

Out[20]:

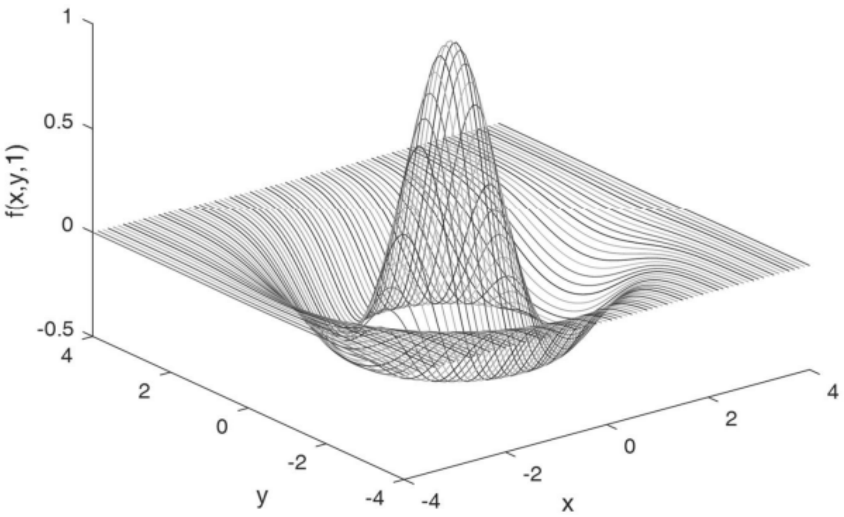
```
(0.7763291724456773, 0.8845573559757789)
```

In [21]:
```
quantization_error_test(trunc_data, cd)
```

Out[21]:
```
0.09592649384500612
```

Q4: Write the version SOM1C, where you change the normal distribution to other distribution of your choice. Did you achieve improvements?

We changed the standard normal distribution to the mexican hat (Ricker wavelet) distribution [4]. The distribution is more or less similar to gaussian distribution but with a divit around the base (like a hat). Mexican hat distribution looks like



We calculate the distribution using the following formula (Read more at [4] Page 12

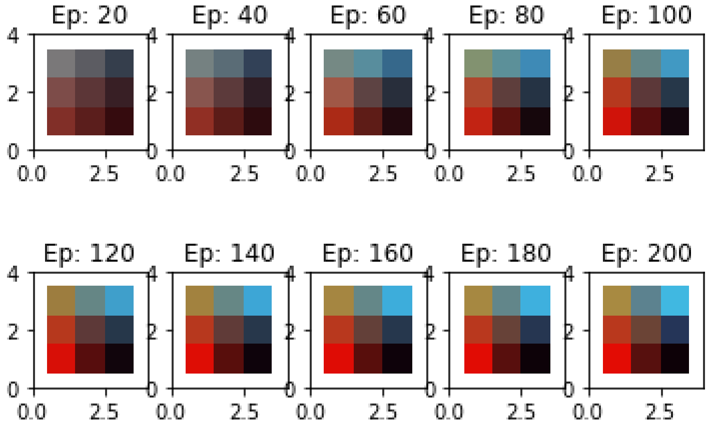
$$f(\mathbf{x}, \mathbf{c}, w) = \left(1 - \frac{\|\mathbf{x}-\mathbf{c}\|^2}{w}\right) e^{-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2w}}$$

But changing to the mexican hat distribution also didn't result in any better results.

In [37]:
```
from scripts.our_som1_C import SOM as SOM_1_C
# som = SOM(x_size, y_size, num_features)
som_1_C = SOM_1_C(3,3,3)
joined_df, clustered_df = som_train_predict(som_1_C, trunc_data, agri_data, r
#joined_df.iloc[0:5]
```

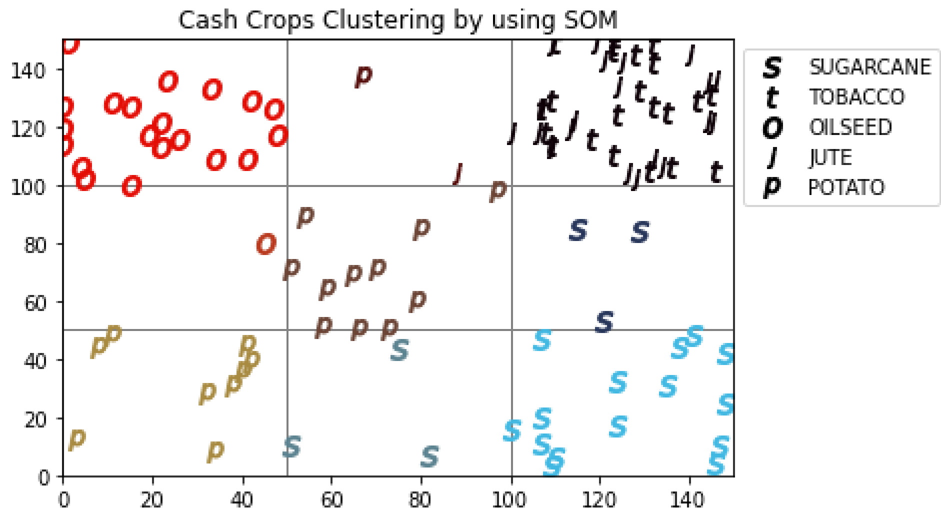
```
SOM training epoches 20
neighborhood radius  2.6878753795222865
learning rate  0.009048374180359595
-------------------------------------
SOM training epoches 40
```

```
neighborhood radius  2.4082246852806923
learning rate  0.008187307530779819
-------------------------------------
SOM training epoches 60
neighborhood radius  2.157669279974593
learning rate  0.007408182206817179
-------------------------------------
SOM training epoches 80
neighborhood radius  1.9331820449317627
learning rate  0.006703200460356393
-------------------------------------
SOM training epoches 100
neighborhood radius  1.7320508075688772
learning rate  0.006065306597126334
-------------------------------------
SOM training epoches 120
neighborhood radius  1.5518455739153598
learning rate  0.005488116360940264
-------------------------------------
SOM training epoches 140
neighborhood radius  1.3903891703159093
learning rate  0.004965853037914096
-------------------------------------
SOM training epoches 160
neighborhood radius  1.2457309396155174
learning rate  0.004493289641172216
-------------------------------------
SOM training epoches 180
neighborhood radius  1.1161231740339044
learning rate  0.004065696597405992
-------------------------------------
SOM training epoches 200
neighborhood radius  1.0
learning rate  0.0036787944117144234
-------------------------------------
```



In [23]:

```
visualize_som(som_1_C, joined_df)
```



In [24]:
```
cd = pd.DataFrame(clustered_df['bmu'].apply(lambda x: x[0].tolist()), index =
cd = cd.bmu.apply(list).apply(pd.Series).astype(float)

neighborhood_preservation_trustworthiness(1, trunc_data, cd)
```

Out[24]:
```
(0.9473878871937125, 0.885568098470865)
```

In [25]:
```
quantization_error_test(trunc_data, cd)
```

Out[25]:
```
0.1230900092047722
```

Q5* : Determine the mathematical conditions that ensure the convergence of equation (3) in page 14 of this slides.

Método do ponto fixo: convergência

Em que condições é que, dada uma aproximação inicial x_0, o método iterativo $x_k = g(x_{k-1}),\ \ k = 1, 2, \ldots$, converge para a raiz α de $f(x) = 0$ no intervalo $I = [a, b]$?

Teorema: Seja $g \in C^1(I)$ e α o único zero de f em I.
Se

$$g(I) \subset I \ \ (\text{i. e., } \forall x \in I,\ g(x) \in I)$$

e

$$0 < M = \max_{x \in I} |g'(x)| < 1,$$

então g possui um único ponto fixo α no intervalo I e a sucessão de aproximações $\{x_k\}_{k\in\mathbb{N}}$ gerada por $x_k = g(x_{k-1}),\ k = 1, 2, \ldots$, converge para α, qualquer que seja a aproximação inicial $x_0 \in I$.

We'll use the point fixed theorem to prove that eq 3 converges.

Proving that equation 3 converges.

$\Rightarrow$ Using Fixed point theorem.

given,

$g(\omega) = \omega_K(t) + \alpha(t) h_{cK}(t) [x(t) - \omega_K(t)]$

$g'(\omega) = 1 - \alpha(t) h_{cK}(t)$

and we know,

$0 < \alpha < 1 \Rightarrow \alpha \in]0,1[$

According to fixed point theorem,

$0 < \max|g'(\omega)| < 1$, g has a fixed point in $\mathbb{R}$

and

$\omega_K = g(\omega_{K-1})$ with $K \in \mathbb{N}$ converges to that unique fixed point $\forall \omega_0$.

So,

we have.

$0 < \max|1 - \alpha(t) h_{cK}(t)| < 1$

$\rightarrow \max|1 - \alpha(t) h_{cK}(t)|$ is achieved when we have $\min(\alpha(t) h_{cK}(t))$.

$\rightarrow$ we know,

$0 < \alpha(t) < 1$ *

$0 < h_{cK}(t) < 1$ *

$\rightarrow$ If we pick minimum value for $\alpha(t)$ and $h_{cK}(t)$, we'll have $\max|1 - \alpha(t) h_{cK}(t)|$.

$\rightarrow$ Since $\alpha(t)$ and $h_{cK}(t)$ are greater than 0 and smaller than 1, This implies $0 < \alpha(t) h_{cK}(t) < 1$

$\Rightarrow 0 < \max|1 - \alpha(t) h_{cK}(t)| < 1$. This is also satisfied.

$\rightarrow$ So, eq 3 converges given the conditions above.

Q6 : As explained in class, SOM can be seen as a Euler integration method for the corresponding ODE. Estimate the absolute error after N epochs.

Euler integration method is a numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. It's basic formula is the last value plus some step times the derivative of the original function. In our SOM, we can see it as the function that updates the nodes' weights in every iteration. Because it is an approximation to the original value, it has going to have an error associated. This can be calculated as the difference between the original value and the estimated value. In order to calculate the original value, we integrate the neighbourhood function after n epochs and solve it against the current radius. This is calculated and returned in the training function of the `scripts/our_som1_6.py` file, along with the weights array. Then, in the `scipts/som_utils.py` the `som_abs_error` calculates the maximum difference between the weights at n epochs and the original value, to present the error.

$$y_1 = y_0 + f\left(t_0, y_0\right)\left(t_1 - t_0\right)$$

Then after N interations to get the error, we simply subtract our estimation from the original value.

$$error = |w(t) - y(t)|$$

We used N as 200 here in the calculations

In [26]:
```
from scripts.our_som1_6 import SOM as SOM_1_6
som_1_6 = SOM_1_6(3,3,3)
n = 200
error = som_abs_error(som_1_6, trunc_data, num_epochs=n, init_learning_rate=0
print(error)
```

```
1.34958949505949
```

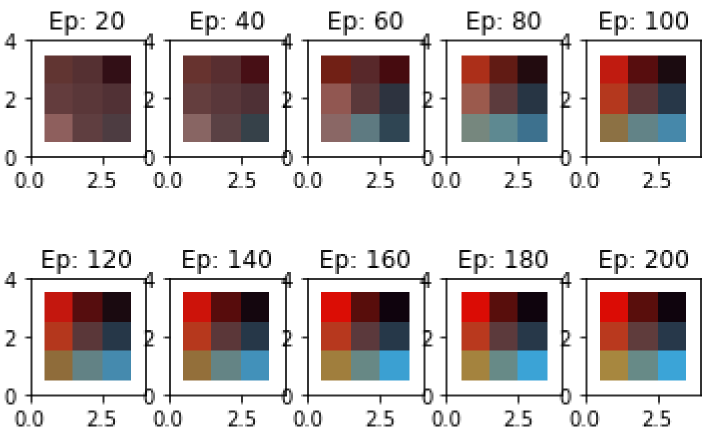
Q7* : How could you change the SOM method to use Runge-Kutta second order method? Is the improvements?

Runge-Kutta method is one other way of approximating values. As before, to the weight it is added the step times the neighbourhood function. Only this time the influence is calculated with more complex parameters. It is also added the step to the power of three. This is done in `scripts/our_som1_7.py` .

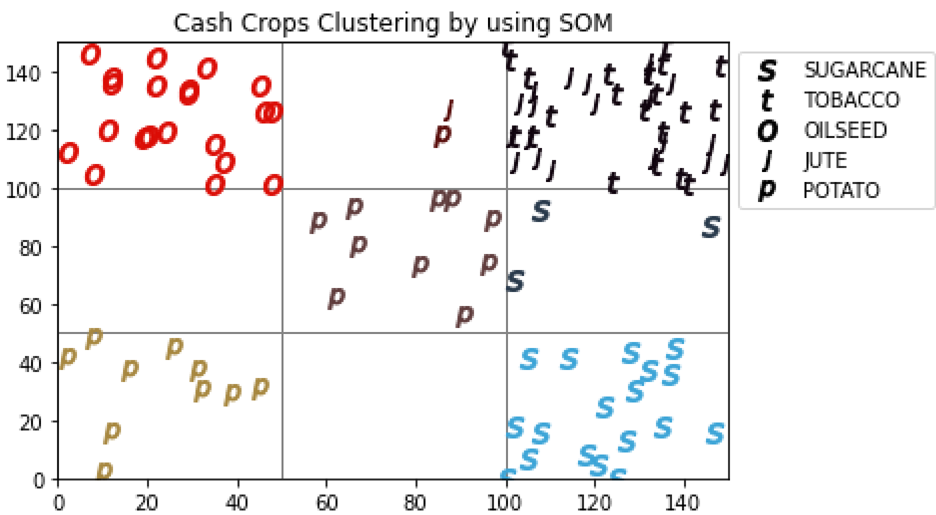
In [27]:
```
from scripts.our_som1_7 import SOM as SOM_1_7
som_1_7 = SOM_1_7(3,3,3)
joined_df, clustered_df = som_train_predict(som_1_7, trunc_data, agri_data, r
```

```
[3 3]
SOM training epoches 20
neighborhood radius  2.6878753795222865
learning rate  0.009048374180359595
-------------------------------------
SOM training epoches 40
neighborhood radius  2.4082246852806923
learning rate  0.008187307530779819
-------------------------------------
SOM training epoches 60
neighborhood radius  2.157669279974593
learning rate  0.007408182206817179
-------------------------------------
SOM training epoches 80
```

```
neighborhood radius  1.9331820449317627
learning rate  0.006703200460356393
-------------------------------------
SOM training epoches 100
neighborhood radius  1.7320508075688772
learning rate  0.006065306597126334
-------------------------------------
SOM training epoches 120
neighborhood radius  1.5518455739153598
learning rate  0.005488116360940264
-------------------------------------
SOM training epoches 140
neighborhood radius  1.3903891703159093
learning rate  0.004965853037914096
-------------------------------------
SOM training epoches 160
neighborhood radius  1.2457309396155174
learning rate  0.004493289641172216
-------------------------------------
SOM training epoches 180
neighborhood radius  1.1161231740339044
learning rate  0.004065696597405992
-------------------------------------
SOM training epoches 200
neighborhood radius  1.0
learning rate  0.0036787944117144234
-------------------------------------
```



In [28]:
```
visualize_som(som_1_C, joined_df)
```



In [29]:
```
cd = pd.DataFrame(clustered_df['bmu'].apply(lambda x: x[0].tolist()), index =
```

```
cd = cd.bmu.apply(list).apply(pd.Series).astype(float)

neighborhood_preservation_trustworthiness(1, trunc_data, cd)
```

Out[29]:

```
(0.9672676837725381, 0.8807023650884301)
```

In [30]:

```
quantization_error_test(trunc_data, cd)
```

Out[30]:

```
0.14603804458101802
```

Q8* : Estimate the absolute error after N epochs by using Q7.

We used the same approch aw with Q6, but Runge-kutta equation.

In [31]:

```
from scripts.our_som1_8 import SOM as SOM_1_8
som_1_8 = SOM_1_8(3,3,3)
n = 200
error = som_abs_error(som_1_8, trunc_data, num_epochs=n, init_learning_rate=0
print(error)
```

```
1.35028495109477
```

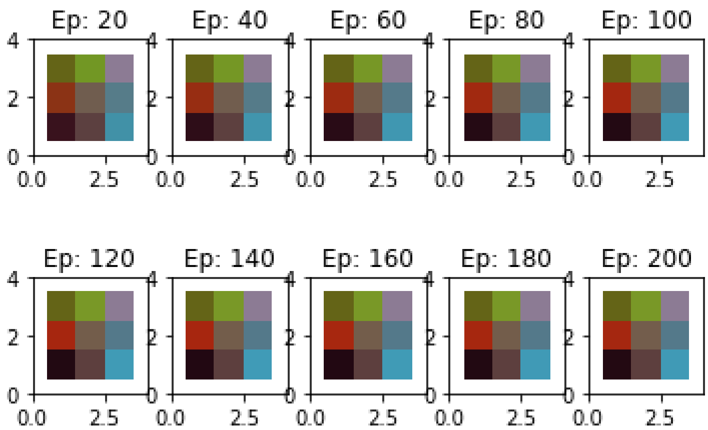
Q9 : How would you combine the answers to Q1-Q8, in order to suggest an improved version?

In [32]:

```
from scripts.our_som1_final import SOM as SOM_1_9
som_1_9 = SOM_1_9(3,3,3)
joined_df, clustered_df = som_train_predict(som_1_9, trunc_data, agri_data, r
```

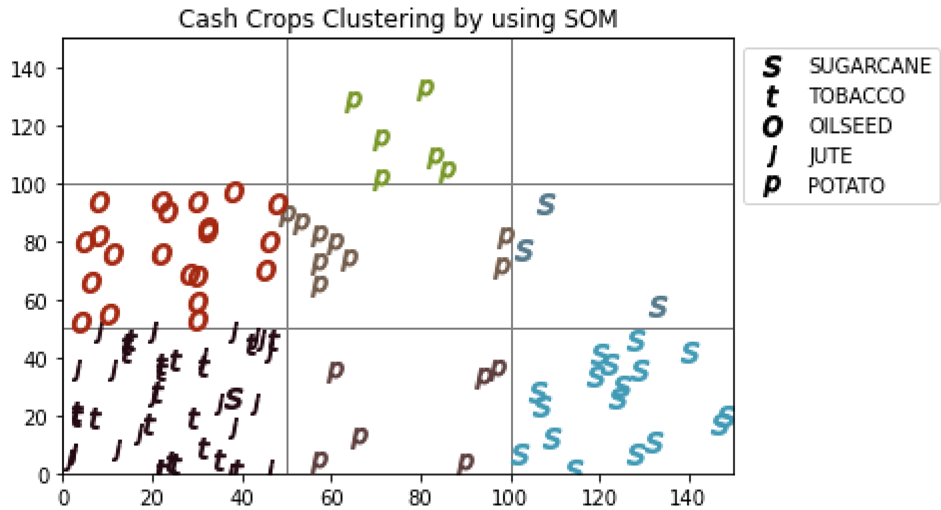
```
[3 3]
SOM training epoches 20
neighborhood radius  0.4060058497098381
learning rate  0.0005
-------------------------------------
SOM training epoches 40
neighborhood radius  0.054946916666202536
learning rate  0.00025
-------------------------------------
SOM training epoches 60
neighborhood radius  0.0074362565299990755
learning rate  0.00016666666666666666
-------------------------------------
SOM training epoches 80
neighborhood radius  0.0010063878837075356
learning rate  0.000125
-------------------------------------
SOM training epoches 100
neighborhood radius  0.00013619978928745456
learning rate  0.0001
```

```
-------------------------------------
SOM training epoches 120
neighborhood radius  1.843263705998463e-05
learning rate  8.333333333333333e-05
-------------------------------------
SOM training epoches 140
neighborhood radius  2.494586157310704e-06
learning rate  7.142857142857143e-05
-------------------------------------
SOM training epoches 160
neighborhood radius  3.3760552415777734e-07
learning rate  6.25e-05
-------------------------------------
SOM training epoches 180
neighborhood radius  4.568993923413789e-08
learning rate  5.555555555555556e-05
-------------------------------------
SOM training epoches 200
neighborhood radius  6.183460867315673e-09
learning rate  5e-05
-------------------------------------
```



In [33]:

```python
visualize_som(som_1_C, joined_df)
```



In [34]:

```python
cd = pd.DataFrame(clustered_df['bmu'].apply(lambda x: x[0].tolist()), index =
cd = cd.bmu.apply(list).apply(pd.Series).astype(float)

neighborhood_preservation_trustworthiness(1, trunc_data, cd)
```

Out[34]:

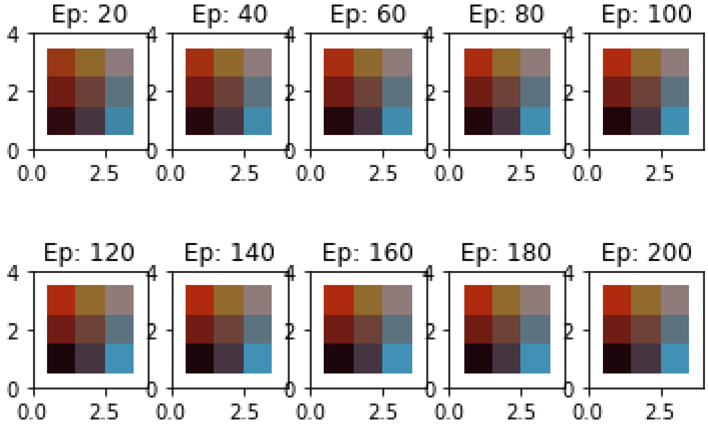
```
(0.660748959778086, 0.8785590449772142)
```

In [35]:
```
quantization_error_test(trunc_data, cd)
```

Out[35]:
```
0.2815332223676341
```

In [36]:
```
n = 200
error = som_abs_error(som_1_9, trunc_data, num_epochs=n, init_learning_rate=0
print(error)
```

```
SOM training epoches 20
neighborhood radius  0.4060058497098381
learning rate  0.0005
-------------------------------------
SOM training epoches 40
neighborhood radius  0.05494691666202536
learning rate  0.00025
-------------------------------------
SOM training epoches 60
neighborhood radius  0.007436256529990755
learning rate  0.00016666666666666666
-------------------------------------
SOM training epoches 80
neighborhood radius  0.0010063878837075356
learning rate  0.000125
-------------------------------------
SOM training epoches 100
neighborhood radius  0.00013619978928745456
learning rate  0.0001
-------------------------------------
SOM training epoches 120
neighborhood radius  1.843263705998463e-05
learning rate  8.333333333333333e-05
-------------------------------------
SOM training epoches 140
neighborhood radius  2.494586157310704e-06
learning rate  7.142857142857143e-05
-------------------------------------
SOM training epoches 160
neighborhood radius  3.3760552415777734e-07
learning rate  6.25e-05
-------------------------------------
SOM training epoches 180
neighborhood radius  4.568993923413789e-08
learning rate  5.555555555555556e-05
-------------------------------------
SOM training epoches 200
neighborhood radius  6.183460867315673e-09
learning rate  5e-05
-------------------------------------
```



```
2.32672218993885
```

Combining all the modifications we had done didn't really yield in better metrics for SOM which is bit strange but a lot more exploring has to be done to get any significant improvement as the initial results were also very good.

References

- **[1]** W. Natita, W. Wiboonsak, and S. Dusadee, “Appropriate Learning Rate and Neighborhood Function of Self-organizing Map (SOM) for Specific Humidity Pattern Classification over Southern Thailand,” 2016, doi: 10.7763/IJMO.2016.V6.504.
- **[2]** G. Breard, “Evaluating Self-Organizing Map Quality Measures as Convergence Criteria,” Open Access Master’s Theses, Jan. 2017, doi: 10.23860/thesis-breard-gregory-2017.
- **[3]** W. Zhang, J. Wang, D. Jin, L. Oreopoulos, and Z. Zhang, “A Deterministic Self-Organizing Map Approach and its Application on Satellite Data based Cloud Type Classification,” arXiv:1808.08315 [cs, stat], Oct. 2018, Accessed: Feb. 15, 2022. [Online]. Available: http://arxiv.org/abs/1808.08315
- **[4]** “Self-organising maps.” [Online]. Available: https://coursepages2.tuni.fi/tiets07/wp-content/uploads/sites/110/2019/01/Neurocomputing3.pdf

In [ ]: