Benedikt Hopf
Alireza Ketabdari

| 1 | 2 | 3 | Σ |
|---|---|---|---|
| | | | |

# Exercise Sheet Nr. 4

(Deadline December 21, 2021)

## Problem 1

a) The answer is, that from a generative model one can generate more data. So the model is a full model of the generative process of the distribution, and by repeating that process one can generate more data. This is possible if one has access to the complete joint distribution and not just a conditional distribution which we would have in a discriminative model (this can only discriminate, i.e do for example classification, but not generate more data).

Data can be generated, by starting at some point, that does not depend on anything (in a Bayesian network that always exists, since it has to be a directed acyclic graph, which always has at least one node which has no incoming edge). The value of that node can be generated by drawing from the associated distribution. Then, which that value given, one can continue which the variables, that now have all dependencies fullfilled and so on, untill all variables have been drawn.

b) The bayesian network is shown in figure 1:

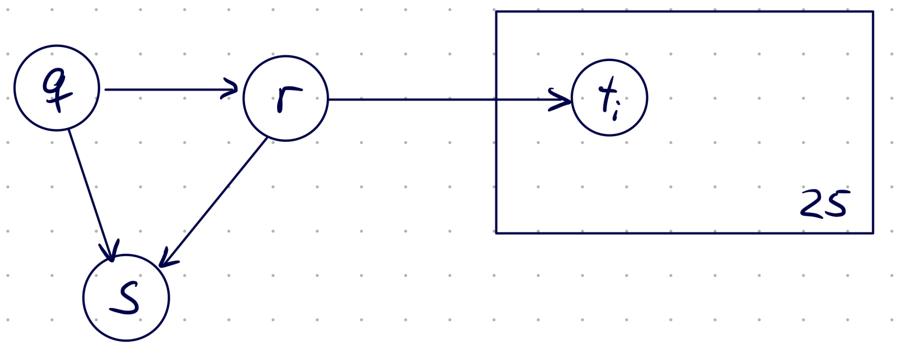


Figure 1: Original Network, as given by the factorization $p(q, r, s, t_1, \dots, t_{25}) = p(q)p(r|q)p(s|q, r) \prod_{i=1}^{25} p(t_i|r)$.

- Given the empty set, since there are 2 ways from $q$ to $s$: direct way from $q$ to $s$ and $q$ to $r$ to $s$ and both are not blocked, they are not D-separated and therefore not independent.
- Even by giving $r$ they are not D-separated because of non blocked direct way from $q$ to $s$. So they are also not independent in that case.
- There are multiple ways to do this. Possibly the most straight forward one is to add a node in between every edge, while keeping the direction of the arrows the same and observing all new nodes. This is shown in figure 2 If observed nodes are not allowed, another way is shown in figure 3. Here the arrows are made to be head-to-head and thus do not need to (indeed must not) be observed. The

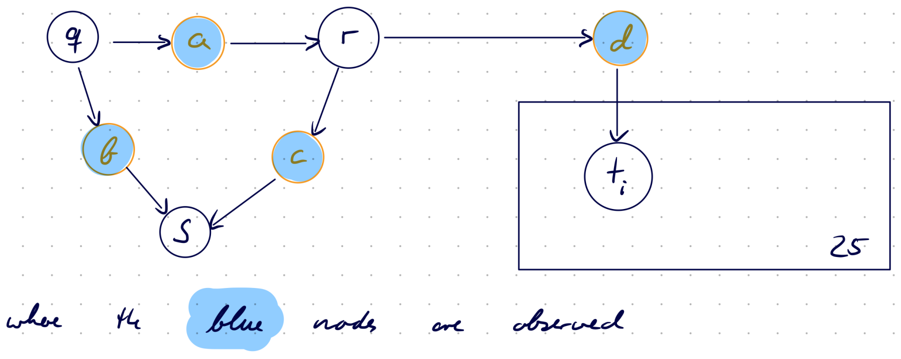


Figure 2: One way to make all the original variables independent. The blue marked variables are observed. Note, that the directions of the arrows stayed the same, which makes this a very intuitive solution. By the first point of d-separation as defined on slide 21 of the script (the arrows meet head-to-tail and the node is observed) this blocks every path, and thus everything is d-separated, so independent.

downside of that is, that the direction of the arrows get a bit messed up from before.

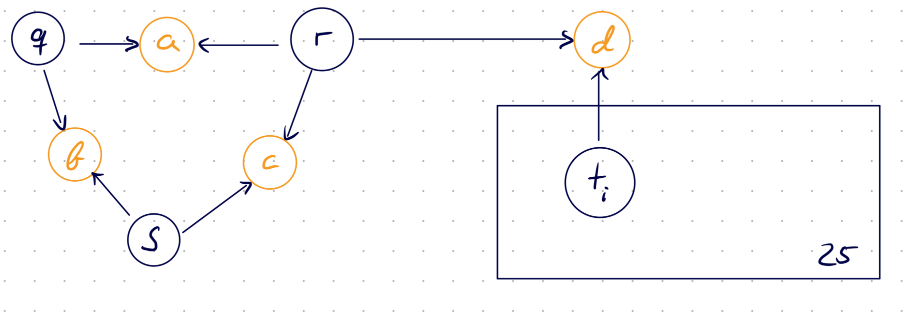


Figure 3: Another way to make all the original variables independent. No variables need to be observed. Note, that the directions of the arrows need to be changes, which makes this a less intuitive solution. By the second point of d-separation as defined on slide 21 of the script (the arrows meet head-to-head and the node is not observed) this blocks every path, and thus everything is d-separated, so independent.

c) a) It is enough if we find just a single path which is not blocked between two nodes . Then, these two nodes are not D-separated and then we don not need to continue in-dependency check. So we only have to check until we find the first not-blocked path. This also means, that is all paths are blocked, we have to check them all.

b) This is true. It follows from the conditions for d-separation as stated on slide 21. These conditions do not say anything about a direction. So is a path is blocked in one direction it is blocked in both diretions and thereore d-separation is also symmetric.

c) The simplest way to violate this condition is when B is a single separated node that does not have any connection to other nodes, on the other hand, A is connected by C and C is the parent and A is the descendant. Thus, A is dependent on C (and C on A) and B is totally independent (D-separated).

d)
- Not independent. They are directly connected. Therefore there is no path that could be blocked and so they cannot be d-separated and therefore also not independent.
- These are independent. All the paths (A-B-E-G, A-C-G, A-C-E-G and A-B-E-C-G) are blocked. This is because all paths from A to G go through B or C with a head-to-tail arrow, so observed means blocked. Therefore all paths are blocked, so we have d-separation and conditional independence.
- Here we do not have conditional independence. F is not given, then it is free to go through it from F (F is head-to-tail), and finally, G (head-to-head) is given, thus, the path is connected to C though G (according to D-separated rules).
- Here we do not have conditional independence. Starting from B we can go as follows:
  - Start at B
  - E (not blocked if we go on to C, since head-to-head and observed)
  - C (not blocked if we go on to G, since tail-to-tail and not observed)
  - G

e) Simpson's paradox is given by the lecture slides as "Any statistical relationship between two variables can be reversed by including additional factors in the analysis."in mathematical terms this means, that we can have

$$P(A|B) > P(A|\neg B)$$

but

$$P(A|B,C) < P(A|\neg B,C)$$

This happens if we look at two groups, with very different outcome. So lets say, the first group has a very large probability (for A) and the other one a very small one. If now need some additional effect (B) that makes the probability for A always smaller. Then we have

$$P(A|B,C) < P(A|\neg B,C)$$

The paradox now happens if $B$ and $C$ tend to happen together and $C$ makes $A$ much more likely. Then when observing $B$ (without knowing $C$) we might see $A$ becoming more likely, but not because $B$ causes that, but with $B$ we often also get $C$ and that causes the probability to increase. (This effect can obviously not happen if $C$ is directly observed.

Let's make a concrete example (this is completely made up and might be wrong, but it should get the idea across): We are looking at cars and bikes and their maximum speeds, depending on what brakes they use. We distinguish between manual brakes and ones with hydraulic support. In general both bikes and cars get slower with a more complex brake (since that adds weight), so we would expect to see a slower speed (lower probability for high speeds) with a hydraulic brake. And that also holds if we condition on cars or bikes. But if we do not condtion on the type of vehicle, we will see many more cars with hydraulic brakes then bikes (since that is basically standard in cars), so the maximum speed of a vehicle (not knowing what kind of vehicle) is expected to be higher if it has hydraulic brakes, then if it does not, simply because in that case it likely is a car which is much faster than a bike.

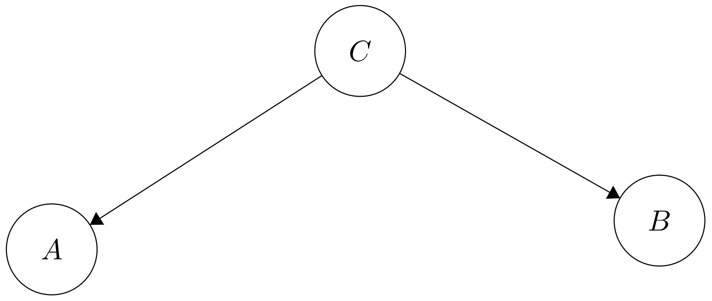
Why can it be problematic?
First, people often expect statistical relationships to be immutable. They often are not. The relationship between two variables might increase, decrease, or even change direction depending on the set of variables being controlled. Moreover, Simpson's paradox reminds researchers that causal inferences, particularly in non-experimental studies, can be hazardous. Uncontrolled and even unobserved variables that would eliminate or reverse the association observed between two variables might exist.
Simpson's paradox can be avoided (but of course not guaranteed) in a study if the most accurate experimental design is used. During the planning stage, the selection of the analysis should include the confounding variables so that the most correct answer to the research question will be gotten when the study is being analyzed. If unequal distribution of data into groups and undetected confounding variables are combined in a study, Simpson's paradox may occur.
Hence, to avoid the wrong conclusion which is in fact different from the correct outcome of the study, the most suitable experimental design should be generated and dispersed between the sample group. And even that does not give full certainty.
Three steps should be considered to avoid it:

- Randomized target and control groups. This avoids bias due to who is in each group.
- Randomized block design: In a randomized block design, the study data are grouped into subgroups according to their similar characteristics.

One can also look at the groups one gets after conditionalizing (in the case of the above example vehicles with and without a hydraulic brake) and try to find similarities between these and explain the result that way (in the case of the example one would find that the groups contain mostly cars and mostly bikes respectively).

f) The difference is that if we have $P(A|B)$ we only know, that $B = b$ happened, but not what else might have happened. In contrast $P(A|do(B))$ assumes, that we have set $B$ ourselves, so we know, that $B = b$ happened because we made it happen and not for any other reason.

Assume we have the following graphical model:



The we can see, that $B$ does not cause $C$. Still we might observe that because $B$ gives us an idea about $C$ and $C$ causes $A$. If we, however, have the ability to do a $do(B)$, we can be sure, that $B$ happened because we made that, so $B$ not does not tell us anything about $C$ and therefore not about $A$ either so we would now not see any correlation and would also not assume causality.

# Problem2

December 16, 2021

## 1 Problem 2

We will do these calculations here in python, in order to avoid having to type them out in a calculator every time and also to avoid mistakes due to that.

[1]:
```
import numpy as np
```

[2]:
```
data_list = [
    0.162,
    0.144,
    0.074,
    0.220,
    0.194,
    0.062,
    0.044,
    0.100
]
p_abc = np.array(data_list).reshape(2,2,2)
a = 0
b = 1
c = 2
```

[3]:
```
print(p_abc)
```

```
[[[0.162 0.144]
  [0.074 0.22 ]]

 [[0.194 0.062]
  [0.044 0.1  ]]]
```

Sanity check:

[4]:
```
print(f"Sum of probabilities, should be 1: {p_abc.sum()}")
```

```
Sum of probabilities, should be 1: 1.0
```

## 1.1 calculate $p(a)$

Now we can easily compute the marginal $p(a)$ by summing over all values where $a == 0$ and $a == 1$.

```
[5]: p_a = p_abc.sum(axis=(c,b), keepdims=True)
     p_a
```

```
[5]: array([[[0.6]],

            [[0.4]]])
```

## 1.2 calculate $p(c|a)$

We can do something similar for the conditional distributions:

We sum (marginalize) over $b$, since we do not care about $b$ in $p(c|a)$ and then normalize for $c$ in order to make it a conditional distribution.

```
[6]: p_c_a = p_abc.sum(axis=b, keepdims=True)
     p_c_a /= p_c_a.sum(axis=c, keepdims=True)
     p_c_a
```

```
[6]: array([[[0.39333333, 0.60666667]],

            [[0.595     , 0.405     ]]])
```

## 1.3 calculate $p(b|a,c)$

This time we must not marginalize over any variable, since $p(b|a,c)$ depends on all variables, but we still normalize over $b$ to make it a probability distribution.

```
[7]: p_b_ac = p_abc.copy()
     p_b_ac /= p_b_ac.sum(axis=b, keepdims=True)
     p_b_ac
```

```
[7]: array([[[0.68644068, 0.3956044 ],
             [0.31355932, 0.6043956 ]],

            [[0.81512605, 0.38271605],
             [0.18487395, 0.61728395]]])
```

Now, that we have the marginal/conditional probability distributions, we can multiply them up and see is we recover $p(a,b,c)$:

## 1.4 recover $p(a,b,c)$

```
[8]: p_abc_candidate = p_a * p_c_a * p_b_ac
```

[9]:
```
print("Original distribution")
print(p_abc)
print("\n\n\n")
print("Recovered distribution")
print(p_abc_candidate)
print("\n")
```

```
Original distribution
[[[0.162 0.144]
  [0.074 0.22 ]]

 [[0.194 0.062]
  [0.044 0.1  ]]]




Recovered distribution
[[[0.162 0.144]
  [0.074 0.22 ]]

 [[0.194 0.062]
  [0.044 0.1  ]]]
```

And finally we do a numerical comparison:

[10]:
```
is_correctly_recovered = np.allclose(p_abc, p_abc_candidate)
print(
    f"The original distribution has{' not' if not is_correctly_recovered else␣
 ↪''} been recovered correctly"
)
```

```
The original distribution has been recovered correctly
```

# Problem 3

December 16, 2021

## 1 Problem 3

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

First we will define the $W^{(i)}$ from the assignment sheet.

[2]:
```
W1 = np.loadtxt("W1.txt")
W2 = np.loadtxt("W2.txt")
k = 2
```

We define $W$ and later also $P$ and $S$ to be 4D tensors, where the first dimension is the lower index, the second is the upper index, and dimensions three and four are the matrix. So e.g. $(W_i^{(j)})_{kl}$ would be `W[i, j, k, l]`.

[3]:
```
W = np.array([[W1, W2]])
print(W)
```

```
[[[[1.  0.5 0.3 0.1 0.1]
   [0.5 1.  0.4 0.1 0.1]
   [0.3 0.4 1.  0.3 0.3]
   [0.1 0.1 0.3 1.  0.5]
   [0.1 0.1 0.3 0.5 1. ]]

  [[1.  0.2 0.5 0.1 0.1]
   [0.2 1.  0.3 0.1 0.1]
   [0.5 0.3 1.  0.3 0.3]
   [0.1 0.1 0.3 1.  0.5]
   [0.1 0.1 0.3 0.5 1. ]]]]
```

[4]:
```
# 4D tesor
def calc_P_from_W(W):
    P = np.zeros_like(W)
    for i, Wstep in enumerate(W):
        for j, Windex in enumerate(Wstep):
            P[i, j] = _calc_P_from_W(Windex) # calculate all matricies
    return P
# 2D matrix
```

```
def _calc_P_from_W(W):
    # np.diag(np.diag(X)) gives a Matrix that has just the diagonal of X and 0␣
 ↪everywhere else
    W_zero_diag = W - np.diag(np.diag(W))
    denominator = 2 * W_zero_diag.sum(axis=-1, keepdims=True)

    W_zero_diag_normalized = W_zero_diag / denominator
    return W_zero_diag_normalized + np.diag(0.5 * np.ones(W.shape[0]))

# 4D tesor
def calc_S_from_W(W, k=k):
    S = np.zeros_like(W)
    for i, Wstep in enumerate(W):
        for j, Windex in enumerate(Wstep):
            S[i, j] = _calc_S_from_W(Windex, k) # calculate all matricies
    return S
# 2D matrix
def _calc_S_from_W(W, k):
    # get neighbour indexes, sorted by similarity and select the k largest
    neighbours = np.argsort(W)[:, -k:]
    # select corresponding neighbour values
    kNN = np.take_along_axis(W, neighbours, axis=-1)
    # calculate the sum (denominatior)
    kNN_sum = kNN.sum(axis=-1)
    S = np.zeros_like(W)

    for i, row in enumerate(W):
        for j, x in enumerate(row):
            if j in neighbours[i]:
                S[i,j] = x / kNN_sum[i]
    return S
```

## 1.1 a)

Now we can easily calculate $P$ and $S$:

[5]:
```
P = calc_P_from_W(W)
S = calc_S_from_W(W)

print("P-matricies: ")
print("P^(1):")
print(P[0, 0])
print()
print("P^(2):")
print(P[0, 1])
print()
print()
print("S-matricies: ")
print("S^(1):")
print(S[0, 0])
print()
print("S^(2):")
print(S[0, 1])
```

```
P-matricies:
P^(1):
[[0.5        0.25       0.15       0.05       0.05      ]
 [0.22727273 0.5        0.18181818 0.04545455 0.04545455]
 [0.11538462 0.15384615 0.5        0.11538462 0.11538462]
 [0.05       0.05       0.15       0.5        0.25      ]
 [0.05       0.05       0.15       0.25       0.5       ]]

P^(2):
[[0.5        0.11111111 0.27777778 0.05555556 0.05555556]
 [0.14285714 0.5        0.21428571 0.07142857 0.07142857]
 [0.17857143 0.10714286 0.5        0.10714286 0.10714286]
 [0.05       0.05       0.15       0.5        0.25      ]
 [0.05       0.05       0.15       0.25       0.5       ]]


S-matricies:
S^(1):
[[0.66666667 0.33333333 0.         0.         0.        ]
 [0.33333333 0.66666667 0.         0.         0.        ]
 [0.         0.28571429 0.71428571 0.         0.        ]
 [0.         0.         0.         0.66666667 0.33333333]
 [0.         0.         0.         0.33333333 0.66666667]]

S^(2):
[[0.66666667 0.         0.33333333 0.         0.        ]
 [0.         0.76923077 0.23076923 0.         0.        ]
 [0.33333333 0.         0.66666667 0.         0.        ]
 [0.         0.         0.         0.66666667 0.33333333]
 [0.         0.         0.         0.33333333 0.66666667]]
```

### 1.2 b)

Now we will define a function, that does one step:

[6]:
```
def do_step(P, S, normalize = False): # This is the version for more than two␣
 ↪matricies, not sure if that was required
    Pt = P[-1]
    St = S[-1]

    Ptplus1 = []
    for i, Si in enumerate(St):
        Ptplus1.append(
            Si @ (
                np.concatenate([ Pt[:i], Pt[i+1:] ]).mean(axis=0) # leave out i␣
↪and average
            ) @ Si.T
        )

    # If requested, normalize as described in the paper
    if normalize:
        for i, p in Ptplus1:
            Ptplus1[i] = _calc_P_from_W(p)

    return np.array(Ptplus1)
```

So now we can do the steps and print the matricies:

[7]:
```
P = calc_P_from_W(W)
S = calc_S_from_W(W)
Pcs = [ #initialize P^(c)
    (P[0, 0] + P[0, 1]) / 2
]
for i in range(2):
    P1, P2 = do_step(P, S)
    P = np.concatenate([P, [[P1, P2]]])
    Pcs.append(
        (P1 + P2) / 2
    )
Pcs = np.array(Pcs)
```

[8]:
```
for i, (Ps, Pc) in enumerate(zip(P, Pcs)):
    print("#"*100)
    print()
    print(f"Matricies #{i}:")
    print("-"*100)
    print(f"P_{i}^(1): ")
    print(Ps[0])

    print("-"*100)
    print(f"P_{i}^(2): ")
    print(Ps[1])

    print("-"*100)
    print(f"P_{i}^(c): ")
    print(Pc)
    print("-"*100)
    print()
print("#"*100)
```

```
################################################################################
####################

Matricies #0:
--------------------------------------------------------------------------------
--------------------
P_0^(1):
[[0.5        0.25       0.15       0.05       0.05      ]
 [0.22727273 0.5        0.18181818 0.04545455 0.04545455]
 [0.11538462 0.15384615 0.5        0.11538462 0.11538462]
 [0.05       0.05       0.15       0.5        0.25      ]
 [0.05       0.05       0.15       0.25       0.5       ]]
--------------------------------------------------------------------------------
--------------------
P_0^(2):
[[0.5        0.11111111 0.27777778 0.05555556 0.05555556]
 [0.14285714 0.5        0.21428571 0.07142857 0.07142857]
 [0.17857143 0.10714286 0.5        0.10714286 0.10714286]
 [0.05       0.05       0.15       0.5        0.25      ]
 [0.05       0.05       0.15       0.25       0.5       ]]
--------------------------------------------------------------------------------
--------------------
P_0^(c):
[[0.5        0.18055556 0.21388889 0.05277778 0.05277778]
 [0.18506494 0.5        0.19805195 0.05844156 0.05844156]
 [0.14697802 0.13049451 0.5        0.11126374 0.11126374]
 [0.05       0.05       0.15       0.5        0.25      ]
 [0.05       0.05       0.15       0.25       0.5       ]]
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

Matricies #1:
--------------------------------------------------------------------------------
--------------------
P_1^(1):
[[0.33421517 0.28747795 0.25207861 0.06084656 0.06084656]
 [0.29805996 0.33421517 0.27399849 0.06613757 0.06613757]
 [0.18537415 0.20238095 0.36151603 0.09693878 0.09693878]
 [0.05       0.05       0.12142857 0.38888889 0.36111111]
 [0.05       0.05       0.12142857 0.36111111 0.38888889]]
```

```
--------------------------------------------------------------------------------
--------------------
P_1^(2):
[[0.33675214 0.22919132 0.3017094  0.07179487 0.07179487]
 [0.21938318 0.38207059 0.23731397 0.06159225 0.06159225]
 [0.29017094 0.23145957 0.33675214 0.09358974 0.09358974]
 [0.08333333 0.07307692 0.11666667 0.38888889 0.36111111]
 [0.08333333 0.07307692 0.11666667 0.36111111 0.38888889]]
--------------------------------------------------------------------------------
--------------------
P_1^(c):
[[0.33548365 0.25833464 0.27689401 0.06632072 0.06632072]
 [0.25872157 0.35814288 0.25565623 0.06386491 0.06386491]
 [0.23777254 0.21692026 0.34913409 0.09526426 0.09526426]
 [0.06666667 0.06153846 0.11904762 0.38888889 0.36111111]
 [0.06666667 0.06153846 0.11904762 0.36111111 0.38888889]]
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

Matricies #2:
--------------------------------------------------------------------------------
--------------------
P_2^(1):
[[0.29180313 0.2859771  0.28021763 0.068394   0.068394  ]
 [0.28270772 0.30690928 0.27944532 0.06499313 0.06499313]
 [0.27146101 0.27297615 0.29866982 0.0844476  0.0844476 ]
 [0.07991453 0.07649573 0.10421245 0.37654321 0.37345679]
 [0.07991453 0.07649573 0.10421245 0.37345679 0.37654321]]
--------------------------------------------------------------------------------
--------------------
P_2^(2):
[[0.28592025 0.26590739 0.287239   0.0728773  0.0728773 ]
 [0.27943531 0.30157719 0.28681508 0.07324554 0.07324554]
 [0.26500418 0.25250591 0.29502054 0.08490804 0.08490804]
 [0.07380952 0.06648352 0.09761905 0.37654321 0.37345679]
 [0.07380952 0.06648352 0.09761905 0.37345679 0.37654321]]
--------------------------------------------------------------------------------
--------------------
P_2^(c):
[[0.28886169 0.27594225 0.28372832 0.07063565 0.07063565]
 [0.28107152 0.30424323 0.2831302  0.06911933 0.06911933]
 [0.26823259 0.26274103 0.29684518 0.08467782 0.08467782]
 [0.07686203 0.07148962 0.10091575 0.37654321 0.37345679]
 [0.07686203 0.07148962 0.10091575 0.37345679 0.37654321]]
--------------------------------------------------------------------------------
```

```
--------------------

##########################################################################################
####################
```

### 1.3 c)

Now all that is left is to implement the stopping criterion and put everything together.

[9]:
```python
def should_stop(Pcs, epsilon=1E-6):
    Pc_t = Pcs[-2]
    Pc_tplus1 = Pcs[-1]

    E = np.linalg.norm(Pc_tplus1 - Pc_t) / np.linalg.norm(Pc_t)
    return E < epsilon, E
```

[10]:
```python
def SNF(W, k, epsilon=1e-6, normalize=False, max_iter=100):
    # calculate P and S

    P = calc_P_from_W(W)
    S = calc_S_from_W(W, k=k)

    # initialize Pc
    Pcs = [
        (P[0, 0] + P[0, 1]) / 2
    ]
    Es = []

    #update
    for i in range(max_iter):
        Pis = do_step(P, S, normalize=normalize)
        P = np.concatenate([P, [Pis]])
        Pcs.append(
            Pis.mean(axis=0)
        )

        # check for convergence
        converged, E = should_stop(Pcs, epsilon=epsilon)
        # save E
        Es.append(E)
        # and stop if converged
        if converged:
            break
    Pcs = np.array(Pcs)
    Es = np.array(Es)

    # return everything, that could be wanted
    return Pcs, Es, P, S
```

```
[11]: output, errors, _, _ = SNF(W, 2) # I have also tried with normalization as␣
      ↪described in the paper, but it is not required in the task and did not help␣
      ↪anyways.
```

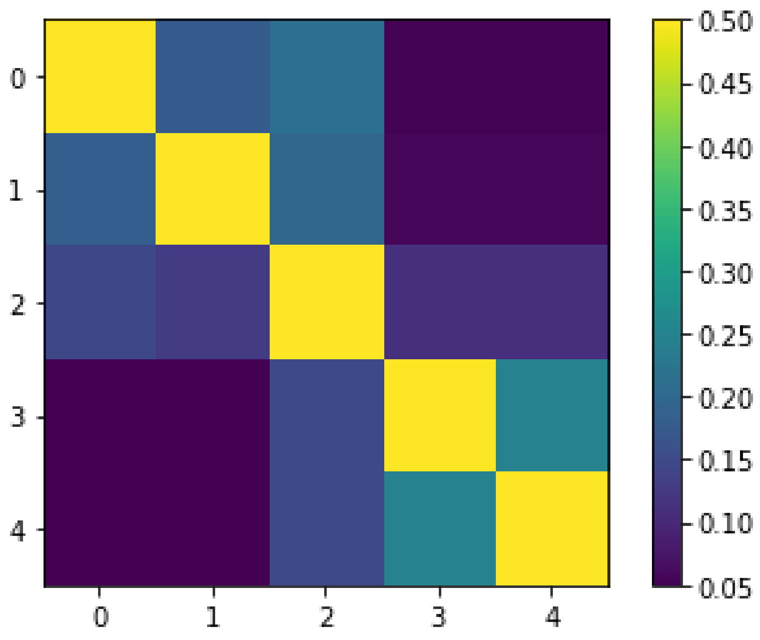
```
[12]: for i, (pc, e) in enumerate(zip(output, [np.inf] + list(errors))):
          print("#"*100)
          print()
          print(f"\033[1m Iteration {i} \033[0m")
          print()
          print("-"*100)
          print(f"P_{i}^(c): ")
          print(pc)
          print("-"*100)
          plt.imshow(pc)
          plt.colorbar()
          plt.show()
          print("-"*100)
          print(f"Error: E = {e}")
          print("-"*100)

          print()
      print("#"*100)
```

```
################################################################################
####################

 Iteration 0

--------------------------------------------------------------------------------
--------------------
P_0^(c):
[[0.5        0.18055556 0.21388889 0.05277778 0.05277778]
 [0.18506494 0.5        0.19805195 0.05844156 0.05844156]
 [0.14697802 0.13049451 0.5        0.11126374 0.11126374]
 [0.05       0.05       0.15       0.5        0.25      ]
 [0.05       0.05       0.15       0.25       0.5       ]]
--------------------------------------------------------------------------------
--------------------
```

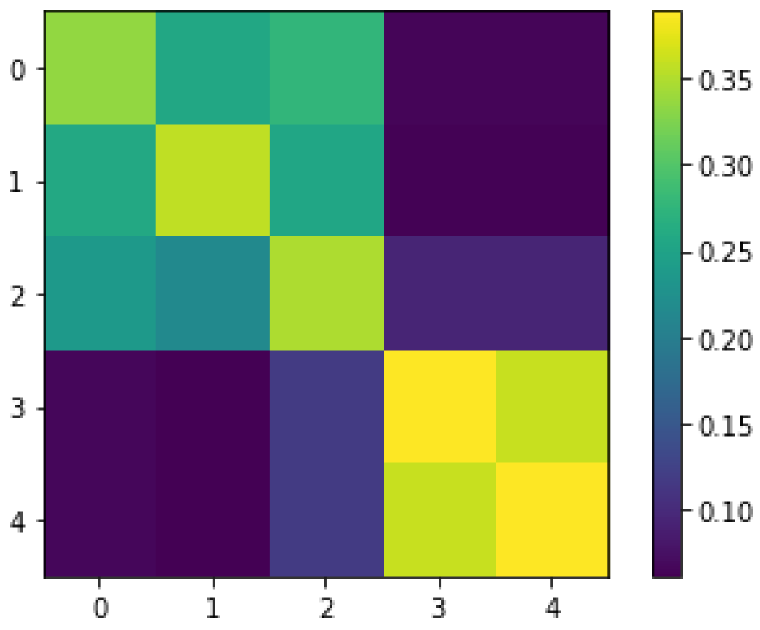


```
--------------------------------------------------------------------------------
--------------------
Error: E = inf
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 1

--------------------------------------------------------------------------------
--------------------
P_1^(c):
[[0.33548365 0.25833464 0.27689401 0.06632072 0.06632072]
 [0.25872157 0.35814288 0.25565623 0.06386491 0.06386491]
 [0.23777254 0.21692026 0.34913409 0.09526426 0.09526426]
 [0.06666667 0.06153846 0.11904762 0.38888889 0.36111111]
 [0.06666667 0.06153846 0.11904762 0.36111111 0.38888889]]
--------------------------------------------------------------------------------
--------------------
```

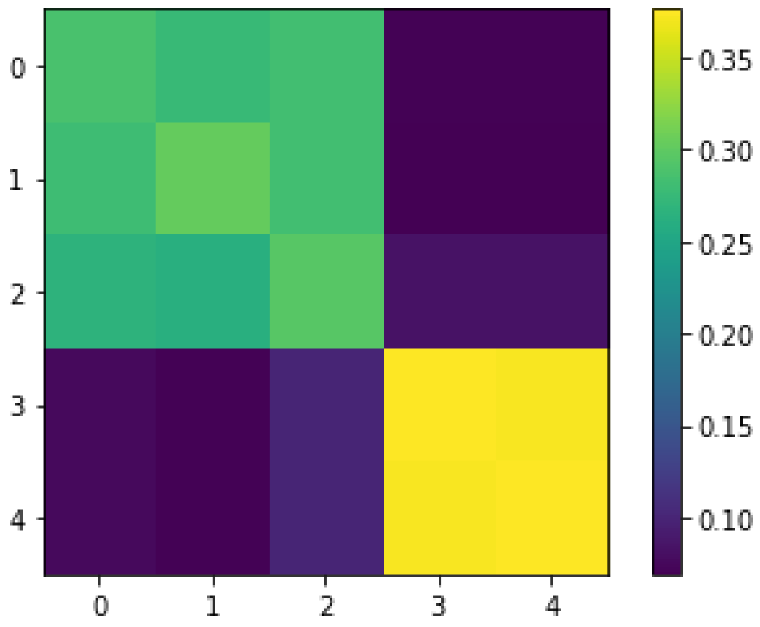


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.30823076923503195
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 2

--------------------------------------------------------------------------------
--------------------
P_2^(c):
[[0.28886169 0.27594225 0.28372832 0.07063565 0.07063565]
 [0.28107152 0.30424323 0.2831302  0.06911933 0.06911933]
 [0.26823259 0.26274103 0.29684518 0.08467782 0.08467782]
 [0.07686203 0.07148962 0.10091575 0.37654321 0.37345679]
 [0.07686203 0.07148962 0.10091575 0.37345679 0.37654321]]
--------------------------------------------------------------------------------
--------------------
```

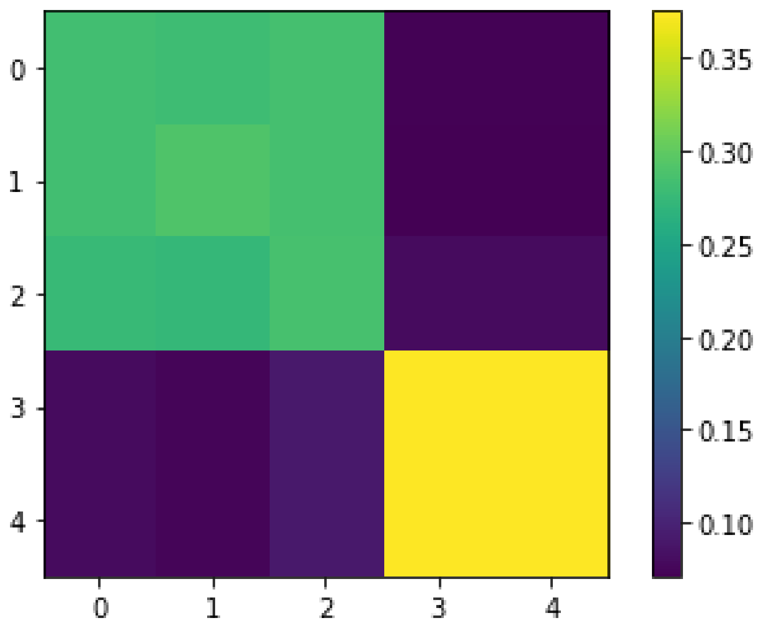


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.10210461245409906
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 3

--------------------------------------------------------------------------------
--------------------
P_3^(c):
[[0.28362129 0.28125912 0.28518008 0.07337262 0.07337262]
 [0.28283126 0.29128104 0.28518095 0.07130271 0.07130271]
 [0.27563059 0.27410647 0.28648235 0.08033615 0.08033615]
 [0.07969068 0.07590871 0.09241816 0.37517147 0.37482853]
 [0.07969068 0.07590871 0.09241816 0.37482853 0.37517147]]
--------------------------------------------------------------------------------
--------------------
```

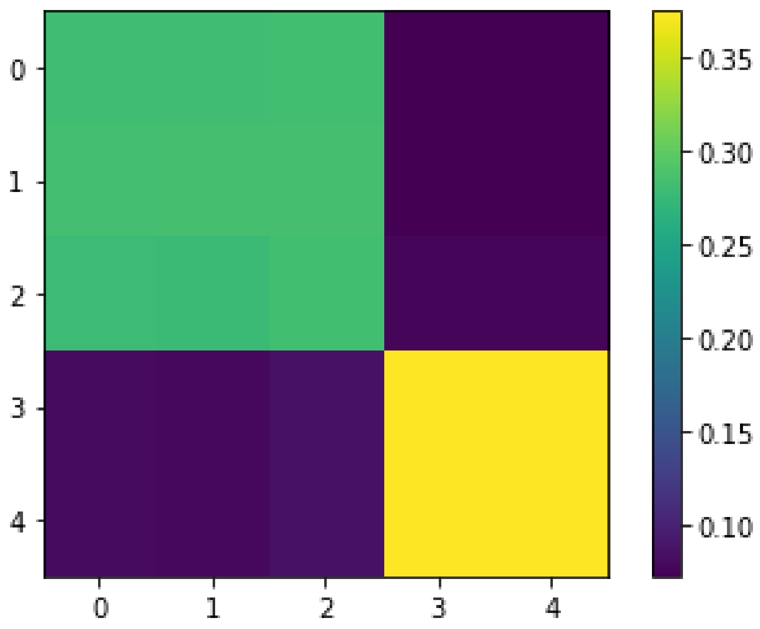


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.02416007977648423
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 4

--------------------------------------------------------------------------------
--------------------
P_4^(c):
[[0.28240852 0.28189187 0.28383356 0.0740915  0.0740915 ]
 [0.28418409 0.28599468 0.28542605 0.07298849 0.07298849]
 [0.28009694 0.27941161 0.28341068 0.07753344 0.07753344]
 [0.08172963 0.07904671 0.0876368  0.37501905 0.37498095]
 [0.08172963 0.07904671 0.0876368  0.37498095 0.37501905]]
--------------------------------------------------------------------------------
--------------------
```

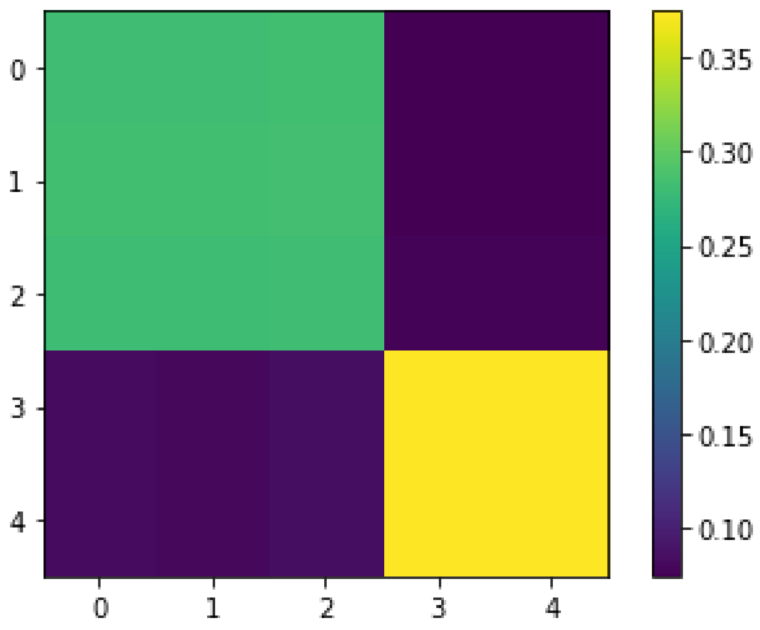


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.011744649087641394
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 5

--------------------------------------------------------------------------------
--------------------
P_5^(c):
[[0.28263458 0.2827536  0.28369413 0.07463152 0.07463152]
 [0.28353285 0.28438268 0.28448865 0.07374775 0.07374775]
 [0.28131186 0.28133    0.28290926 0.07634203 0.07634203]
 [0.08212473 0.08054913 0.08528284 0.37500212 0.37499788]
 [0.08212473 0.08054913 0.08528284 0.37499788 0.37500212]]
--------------------------------------------------------------------------------
--------------------
```

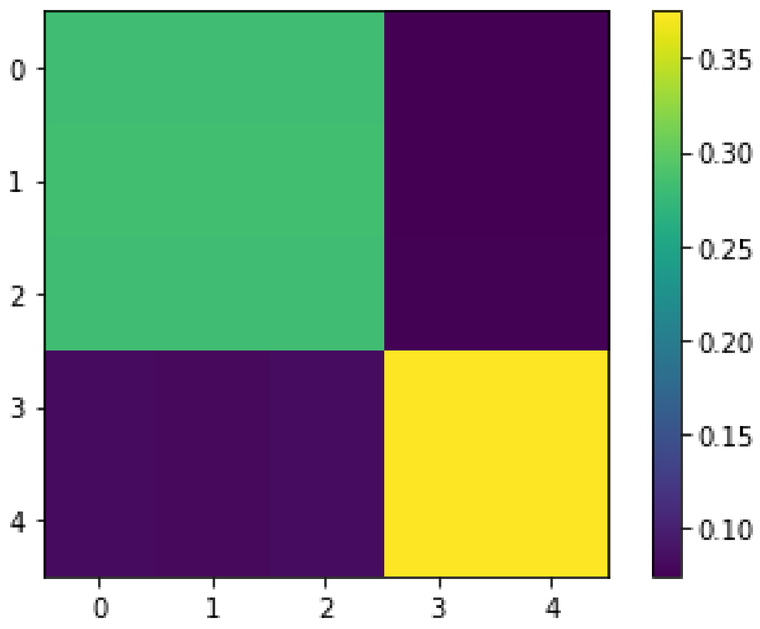


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.004747451072213179
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 6

--------------------------------------------------------------------------------
--------------------
P_6^(c):
[[0.28282901 0.28273859 0.28326392 0.07472709 0.07472709]
 [0.28357053 0.28364779 0.28395632 0.07426883 0.07426883]
 [0.28233883 0.28220018 0.28288688 0.07558869 0.07558869]
 [0.08252054 0.0815087  0.08400203 0.37500024 0.37499976]
 [0.08252054 0.0815087  0.08400203 0.37499976 0.37500024]]
--------------------------------------------------------------------------------
--------------------
```

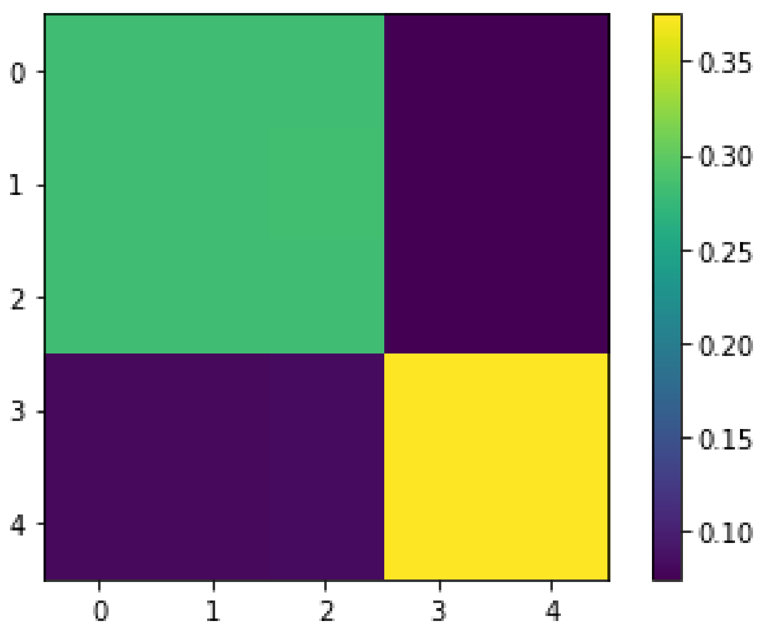


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.002712593804803818
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 7

--------------------------------------------------------------------------------
--------------------
P_7^(c):
[[0.28294498 0.28297979 0.28322979 0.07482968 0.07482968]
 [0.28327404 0.28336235 0.28354285 0.0745147  0.0745147 ]
 [0.282623   0.28264198 0.28294092 0.07525877 0.07525877]
 [0.08254515 0.08198921 0.08334527 0.37500003 0.37499997]
 [0.08254515 0.08198921 0.08334527 0.37499997 0.37500003]]
--------------------------------------------------------------------------------
--------------------
```

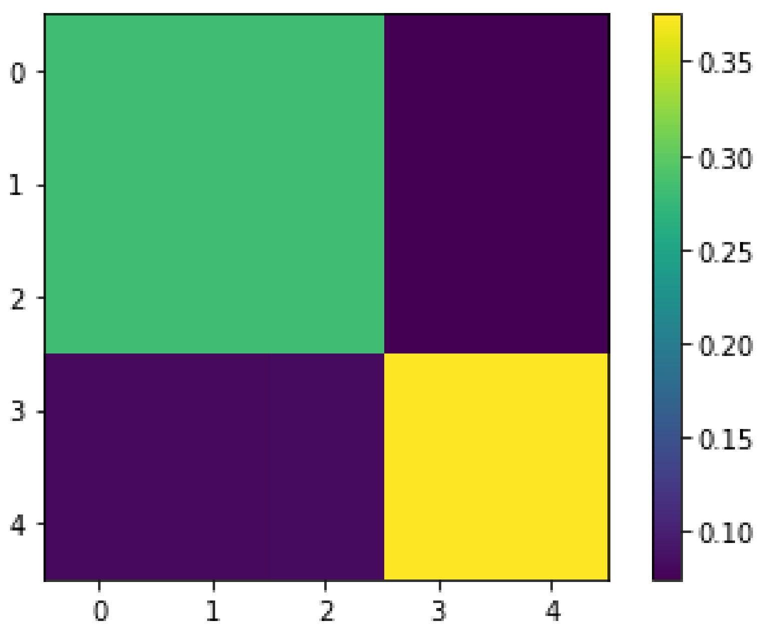


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.0013230059513200314
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 8

--------------------------------------------------------------------------------
--------------------
P_8^(c):
[[0.28301642 0.28297276 0.28312174 0.07483352 0.07483352]
 [0.2832483  0.28321767 0.28334811 0.07467201 0.07467201]
 [0.28288758 0.28283846 0.28300017 0.07505366 0.07505366]
 [0.08261834 0.08227683 0.08299755 0.375      0.375     ]
 [0.08261834 0.08227683 0.08299755 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

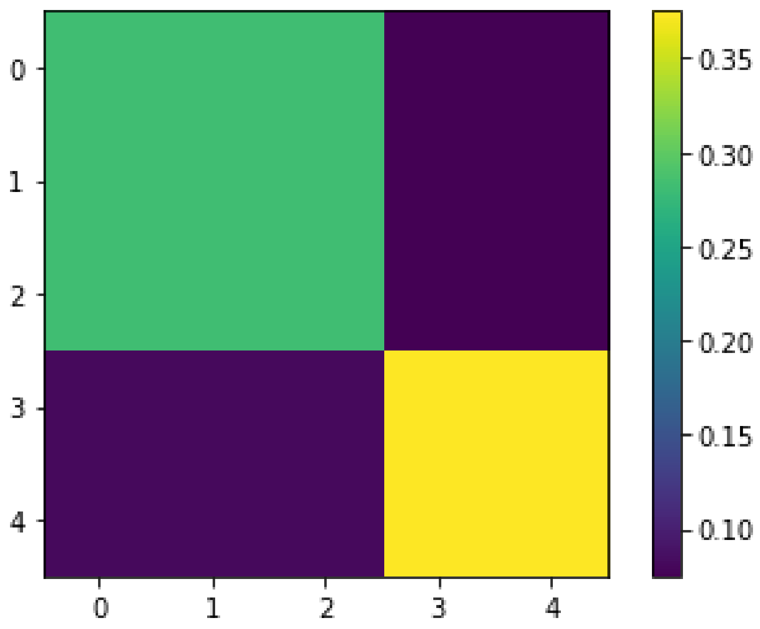


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.0007381780571392802
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 9

--------------------------------------------------------------------------------
--------------------
P_9^(c):
[[0.28304642 0.28304758 0.28311738 0.07485178 0.07485178]
 [0.28314702 0.28315232 0.28321634 0.07474844 0.07474844]
 [0.28295979 0.28295932 0.28303292 0.07496164 0.07496164]
 [0.08260651 0.08242507 0.08281328 0.375      0.375     ]
 [0.08260651 0.08242507 0.08281328 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

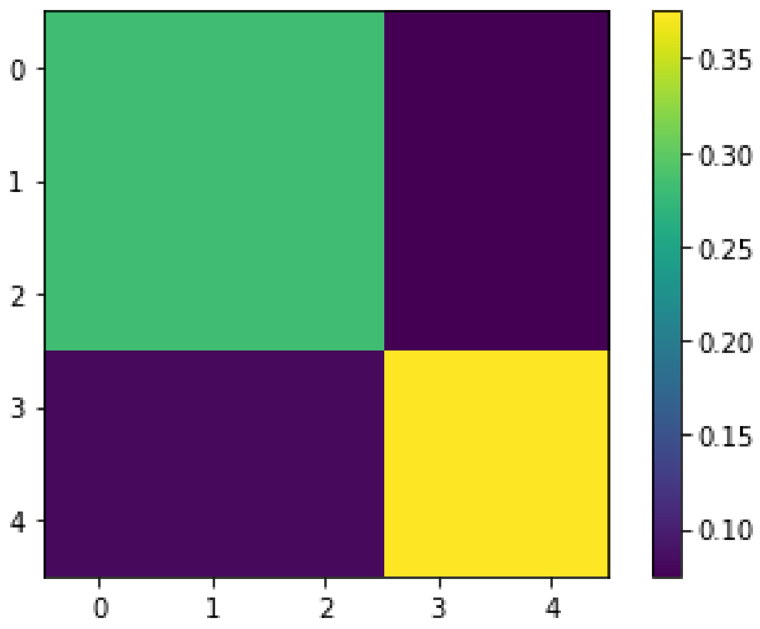


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.0003844320425488355
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 10

--------------------------------------------------------------------------------
--------------------
P_10^(c):
[[0.28306519 0.2830473  0.28309027 0.07484787 0.07484787]
 [0.28313411 0.28311728 0.28315868 0.07479519 0.07479519]
 [0.28302978 0.28301138 0.28305536 0.0749052  0.0749052 ]
 [0.08261889 0.08251012 0.08271782 0.375      0.375     ]
 [0.08261889 0.08251012 0.08271782 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

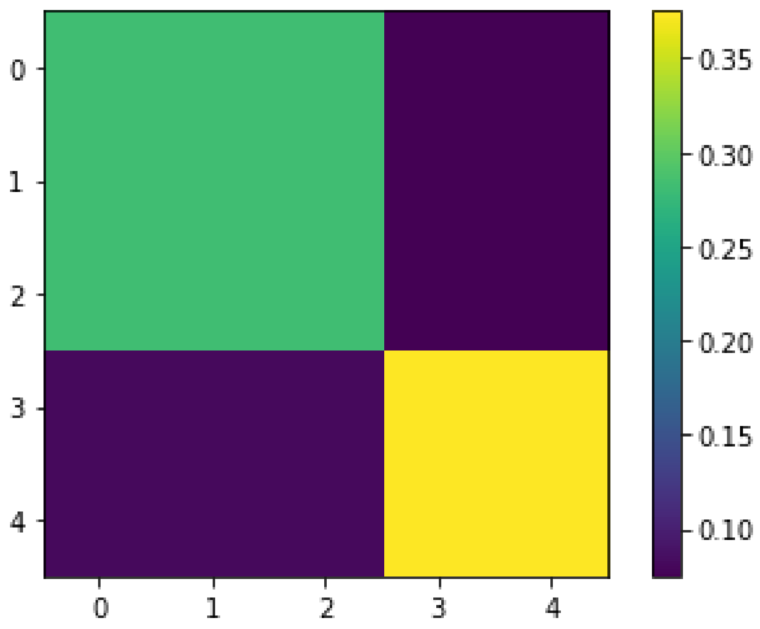


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.00020608992047994139
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 11

--------------------------------------------------------------------------------
--------------------
P_11^(c):
[[0.28307243 0.28307037 0.28309021 0.07485072 0.07485072]
 [0.28310207 0.28310033 0.2831197  0.07481835 0.07481835]
 [0.28304855 0.28304634 0.28306648 0.07487939 0.07487939]
 [0.08261148 0.08255481 0.08266589 0.375      0.375     ]
 [0.08261148 0.08255481 0.08266589 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

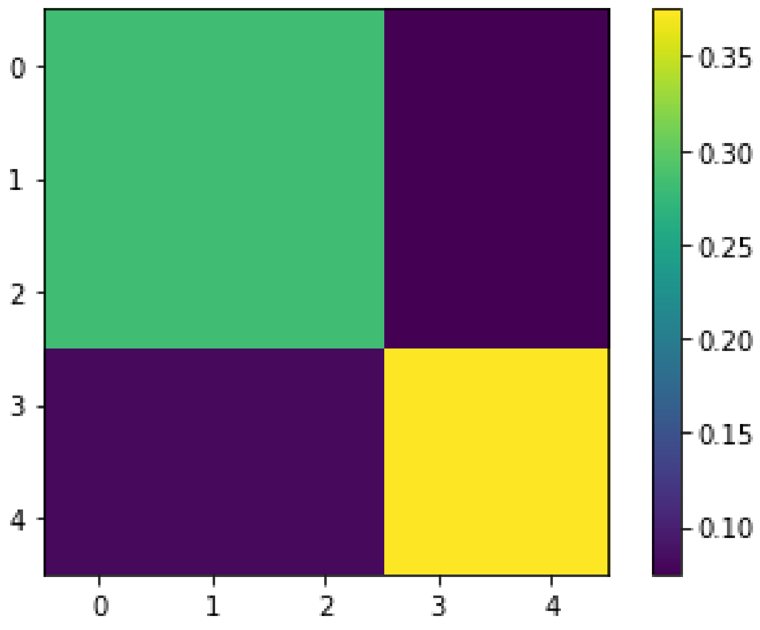


```
--------------------------------------------------------------------------------
--------------------
Error: E = 0.00011199314926600407
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 12

--------------------------------------------------------------------------------
--------------------
P_12^(c):
[[0.28307718 0.28307081 0.28308324 0.07484853 0.07484853]
 [0.28309734 0.28309106 0.28310336 0.07483209 0.07483209]
 [0.28306733 0.28306092 0.28307343 0.07486373 0.07486373]
 [0.0826132  0.08257973 0.08263945 0.375      0.375     ]
 [0.0826132  0.08257973 0.08263945 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

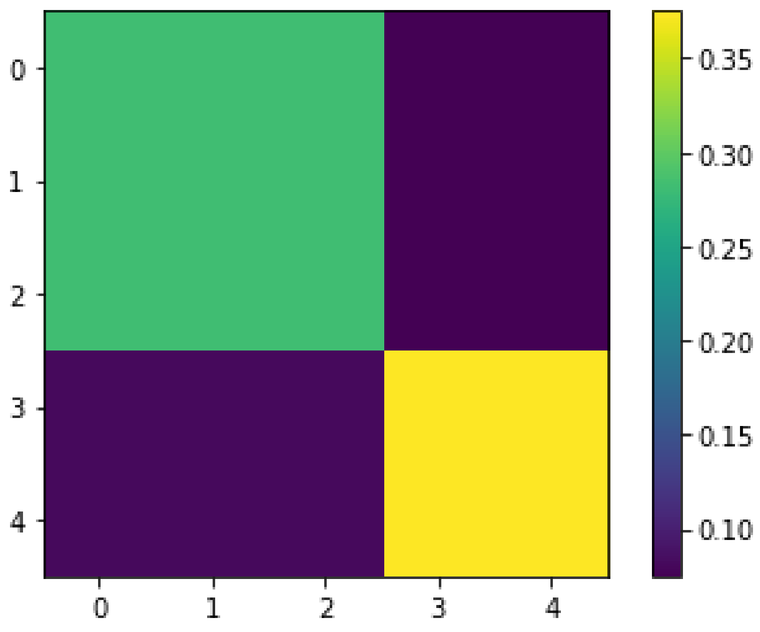


```
--------------------------------------------------------------------------------
--------------------
Error: E = 5.808816317746934e-05
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 13

--------------------------------------------------------------------------------
--------------------
P_13^(c):
[[0.28307893 0.28307779 0.28308347 0.07484885 0.07484885]
 [0.28308755 0.28308645 0.28309208 0.07483899 0.07483899]
 [0.28307228 0.28307114 0.28307683 0.07485645 0.07485645]
 [0.08261022 0.082593   0.08262477 0.375      0.375     ]
 [0.08261022 0.082593   0.08262477 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

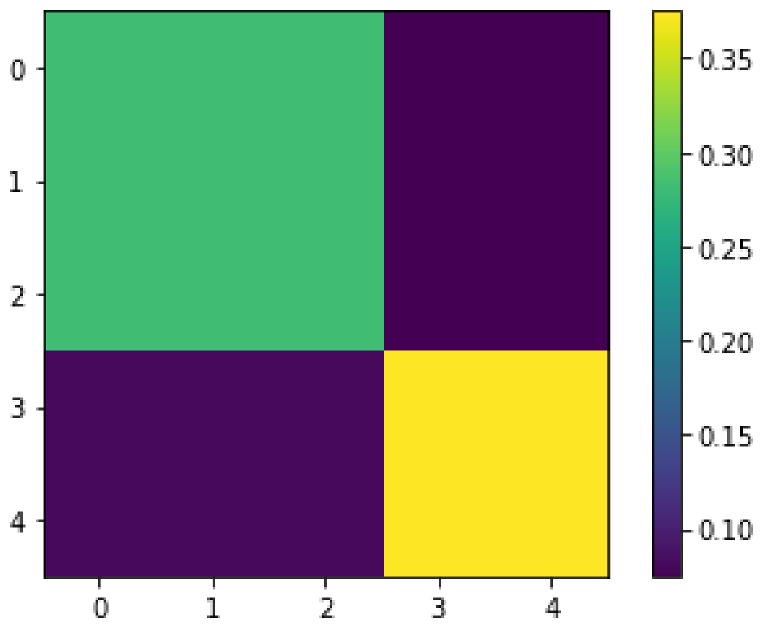


```
--------------------------------------------------------------------------------
--------------------
Error: E = 3.255008690844517e-05
--------------------------------------------------------------------------------
--------------------

################################################################################
####################
```

**Iteration 14**

```
--------------------------------------------------------------------------------
--------------------
P_14^(c):
[[0.28308014 0.28307804 0.28308164 0.07484799 0.07484799]
 [0.28308599 0.28308391 0.28308749 0.07484299 0.07484299]
 [0.28307738 0.28307529 0.28307888 0.07485208 0.07485208]
 [0.08261032 0.08260025 0.08261739 0.375      0.375     ]
 [0.08261032 0.08260025 0.08261739 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

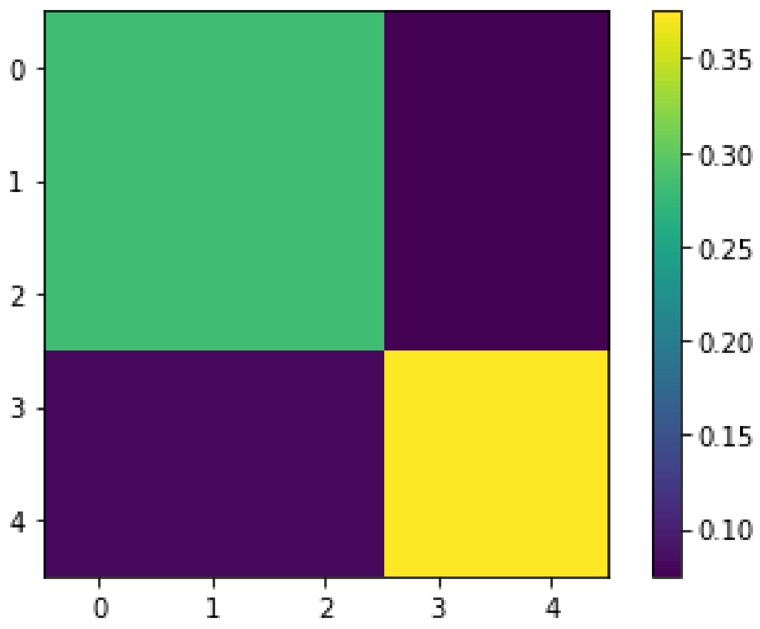


```
--------------------------------------------------------------------------------
--------------------
Error: E = 1.646191710555071e-05
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 15

--------------------------------------------------------------------------------
--------------------
P_15^(c):
[[0.28308057 0.28308013 0.28308175 0.07484797 0.07484797]
 [0.28308307 0.28308262 0.28308424 0.07484503 0.07484503]
 [0.28307871 0.28307826 0.28307989 0.07485002 0.07485002]
 [0.08260929 0.08260415 0.08261323 0.375      0.375     ]
 [0.08260929 0.08260415 0.08261323 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

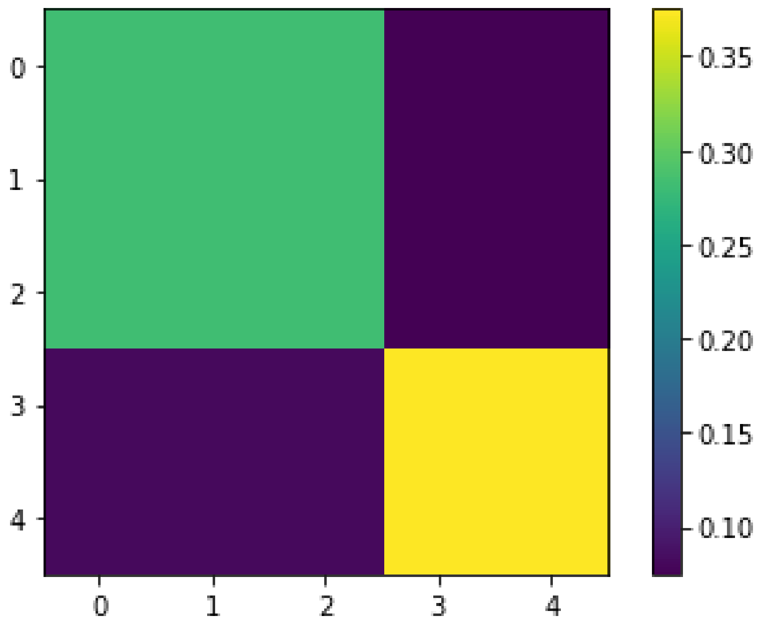


```
----------------------------------------------------------------------------
-------------------
Error: E = 9.431521149778312e-06
----------------------------------------------------------------------------
-------------------

############################################################################
###################

 Iteration 16

----------------------------------------------------------------------------
-------------------
P_16^(c):
[[0.28308088 0.28308023 0.28308126 0.07484768 0.07484768]
 [0.28308258 0.28308192 0.28308295 0.07484619 0.07484619]
 [0.28308011 0.28307945 0.28308049 0.07484879 0.07484879]
 [0.08260923 0.08260625 0.08261116 0.375      0.375     ]
 [0.08260923 0.08260625 0.08261116 0.375      0.375     ]]
----------------------------------------------------------------------------
-------------------
```

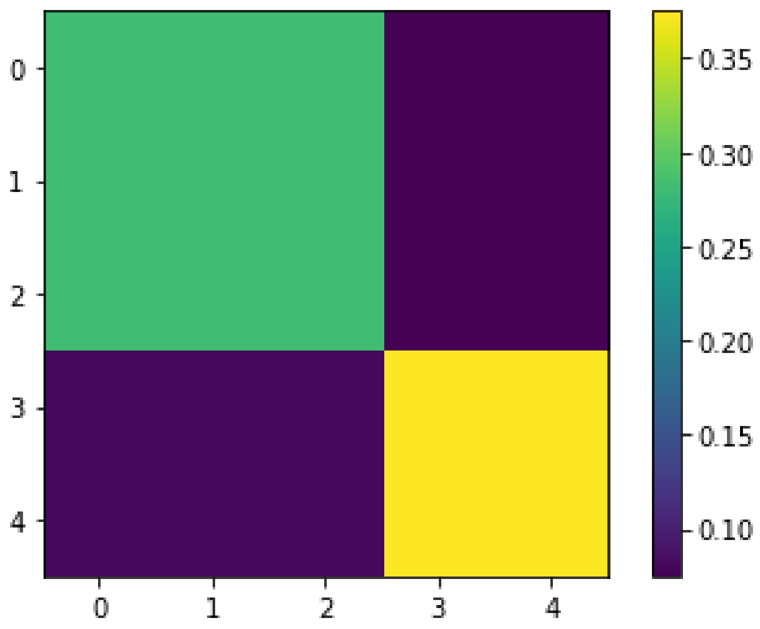


```
----------------------------------------------------------------------------------
--------------------
Error: E = 4.680440454801369e-06
----------------------------------------------------------------------------------
--------------------

##################################################################################
####################

 Iteration 17

----------------------------------------------------------------------------------
--------------------
P_17^(c):
[[0.28308099 0.28308084 0.2830813  0.07484765 0.07484765]
 [0.28308171 0.28308156 0.28308202 0.07484678 0.07484678]
 [0.28308047 0.28308031 0.28308078 0.07484821 0.07484821]
 [0.0826089  0.08260738 0.08260998 0.375      0.375     ]
 [0.0826089  0.08260738 0.08260998 0.375      0.375     ]]
----------------------------------------------------------------------------------
--------------------
```

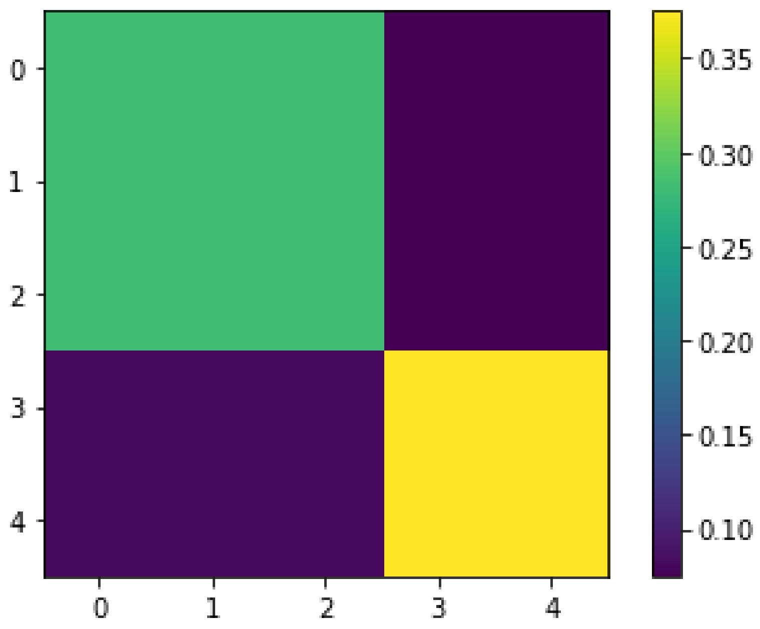


```
--------------------------------------------------------------------------------
--------------------
Error: E = 2.724907137148452e-06
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 18

--------------------------------------------------------------------------------
--------------------
P_18^(c):
[[0.28308107 0.28308087 0.28308117 0.07484756 0.07484756]
 [0.28308156 0.28308136 0.28308166 0.07484712 0.07484712]
 [0.28308085 0.28308065 0.28308095 0.07484786 0.07484786]
 [0.08260886 0.08260799 0.0826094  0.375      0.375     ]
 [0.08260886 0.08260799 0.0826094  0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```

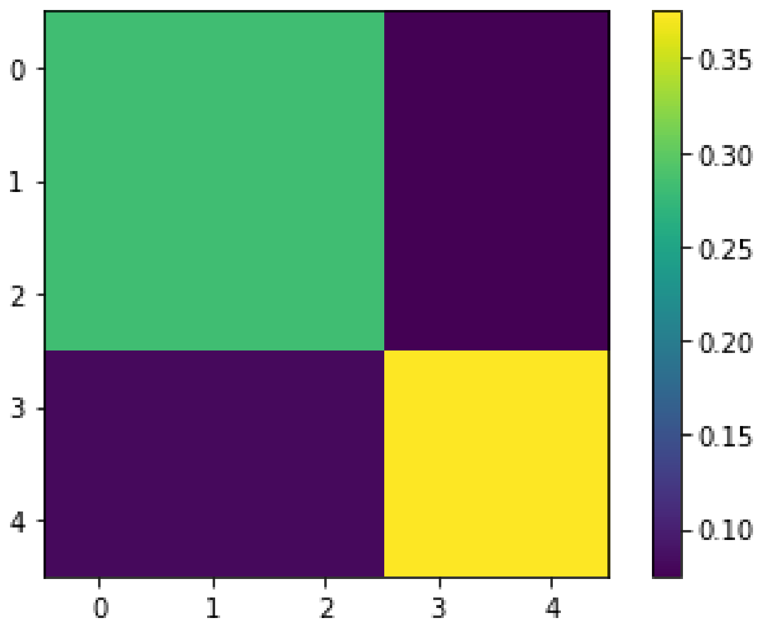


```
--------------------------------------------------------------------------------
--------------------
Error: E = 1.3333189024757566e-06
--------------------------------------------------------------------------------
--------------------

################################################################################
####################

 Iteration 19

--------------------------------------------------------------------------------
--------------------
P_19^(c):
[[0.2830811  0.28308105 0.28308118 0.07484754 0.07484754]
 [0.28308131 0.28308126 0.28308139 0.07484729 0.07484729]
 [0.28308095 0.2830809  0.28308104 0.0748477  0.0748477 ]
 [0.08260876 0.08260832 0.08260906 0.375      0.375     ]
 [0.08260876 0.08260832 0.08260906 0.375      0.375     ]]
--------------------------------------------------------------------------------
--------------------
```



```
--------------------------------------------------------------------------------
--------------------
Error: E = 7.853202634581425e-07
--------------------------------------------------------------------------------
--------------------

################################################################################
####################
```

One can clearly see, that the $P$ matrix still changes after iteration $t > 2$, but at $t = 2$ it is already rather obvious, how the final result is going to look. It takes 19 iterations to reach the convergence threshold, but visually it does not really change anymore after iteration 5.