The Power Iteration and the *PageRank*

1. Implement a function `power_iteration(A,m,p=1)` which takes as input a matrix $A \in \mathbb{R}^{n\times n}$, a maximum iteration number $m \in \mathbb{N}$ and an *optional* parameter p which determines the order of the p-norm and is set to $p = 1$ by default. This function shall then return the m-th iterates x_m and μ_m of the power iteration.

   Hints:

   - You can use a random distribution x_0 as initial guess by calling for example the numpy function

     `x = numpy.random.dirichlet(np.ones(n),size=1).reshape(n)`

     or simply choose

     `x = 1./n * np.ones(n).`

   - For the normalization step use the numpy function

     `numpy.linalg.norm(x, ord=p),`

     which allows the choices $p \in \{1, 2, \infty\}$ (among others).

2. Determine the **PageRank** of the web structure given above. Therefore apply your function `power_iteration(A,m,p=1)` to the PageRank matrix

$$A := P = \alpha P_1 + (1-\alpha)P_2,$$

where

$$P_1 = \begin{array}{cc} & \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \begin{array}{c} 1\\2\\3\\4\\5\\6\\7\\8\\9\\10\\11 \end{array} & \begin{pmatrix} 1 & & & 1/2 & & & & & & & \\ & & 1 & 1/2 & 1/3 & & 1/2 & 1/2 & 1/2 & & \\ & 1 & & & & & & & & & \\ & & & & 1/3 & & & & & & \\ & & & & & 1 & 1/2 & 1/2 & 1/2 & 1 & 1 \\ & & & & 1/3 & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \end{pmatrix} \end{array}, \quad P_2 := \frac{1}{n} e e^T = \left(\frac{1}{n}\right)_{ij}.$$

Play around with the damping factor α. What do you observe?

Hint: For implementing P_1 and P_2, and thus P, you can use this code snippet.

```python
import numpy as np
def P(alpha):
    P_1=np.array([[1,0,0,1.0/2,0,0,0,0,0,0,0,0],
                  [0,0,1.0,1.0/2,1.0/3,0,1.0/2,1.0/2,1.0/2,0,0],
                  [0,1.0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,1.0/3,0,0,0,0,0,0],
                  [0,0,0,0,0,1.0,1.0/2,1.0/2,1.0/2,1.0,1.0],
                  [0,0,0,0,1.0/3,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0]])

    P_2 = (1.0 / 11.0) * np.ones((11,11))

    return alpha * P_1 + (1-alpha) * P_2
```

Solution:

```python
import numpy as np


def power_iteration(A, m, p=1):
    """
    Solves eigenvalue problem via Power Method
    Expects the largerst eigenvalue of A to be scritly larger

    Parameters
    ----------
    A : (n,n) ndarray
        matrix
    m : int
        number of iterations
    p : int or numpy.inf, optional
        specifying the order of the p-Norm used for normalization

    Returns
    -------
    x : (n,1) ndarray
        normalized (with p-Norm) eigenvector for largest eigenvalue
    mu : float
         largest eigenvalue
    """
    n = A.shape[1]
    # x = np.random.dirichlet(np.ones(n), size=1).reshape(n)
    x = 1./n * np.ones(n)
    for k in range(m):
        z = A.dot(x)
        x = z / np.linalg.norm(z, ord=p)
        mu = x.dot(z) / (x.dot(x))
    return x, mu


def P(alpha):
    P_1 = np.array([[1, 0, 0, 1.0/2, 0, 0, 0, 0, 0, 0, 0],
                    [0,0,1.0,1.0/2,1.0/3,0,1.0/2,1.0/2,1.0/2,0,0],
                    [0,1.0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,1.0/3,0,0,0,0,0,0],
                    [0,0,0,0,0,1.0,1.0/2,1.0/2,1.0/2,1.0,1.0],
                    [0,0,0,0,1.0/3,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0]])

    P_2 = (1.0 / 11.0) * np.ones((11, 11))

    return alpha * P_1 + (1-alpha) * P_2


if __name__ == "__main__":
    m = 20
    k = 10
    # run with different damping factors
    for alpha in np.linspace(0, 1, k, endpoint=False):
        print("\n-----------------\nalpha =",
              np.round(alpha, 4),
              "\n-----------------")
```

```
    eigvec, mu_max = power_iteration(P(alpha), m, p=1)
    print("Maximal eigenvalue = ", np.around(mu_max, 10))
    print("\nEigenvector = \n", np.around(eigvec, 5))

    lbdmax_np = np.max(np.linalg.eigvals(P(alpha)))
    print("\nMaximal eigenvalue from numpy = ",
          np.around(np.real(lbdmax_np), 10))
```