Master Informatique, parcours MALIA & MIASHS

Carnets de note Python pour le cours de Network Analysis for Information Retrieval

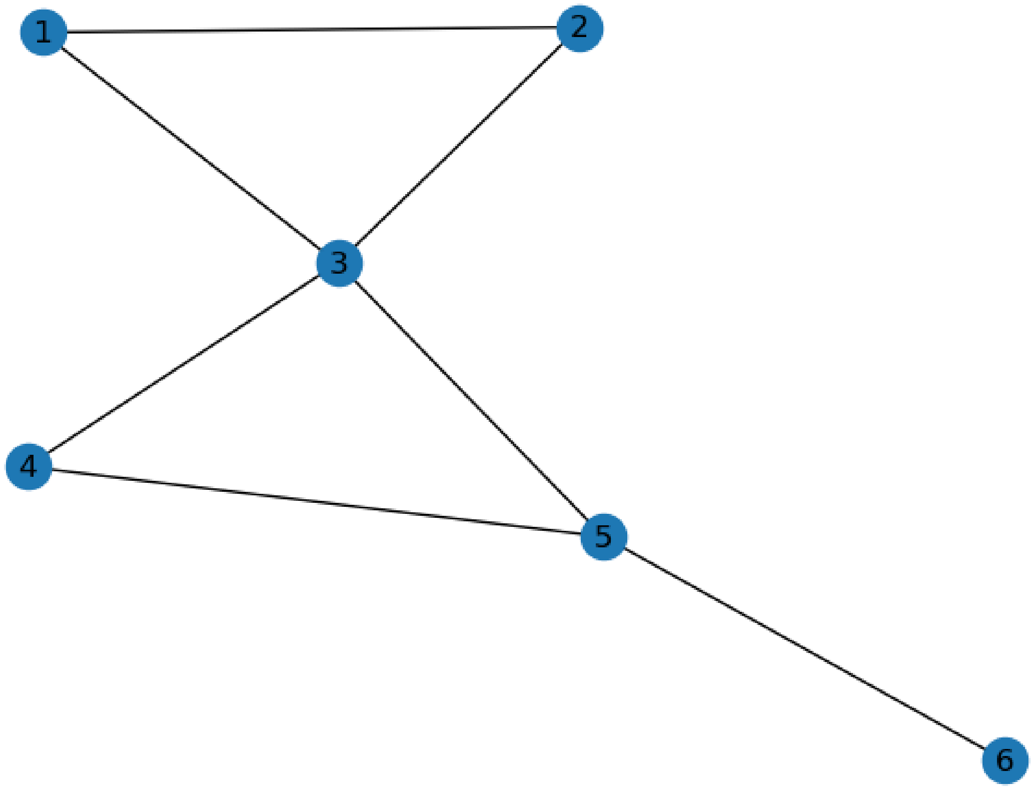
Julien Velcin, laboratoire ERIC, Université Lyon 2

Premières manipulation de graphe

Premiers essais pour vous familiariser avec la librairie Networkx https://networkx.org

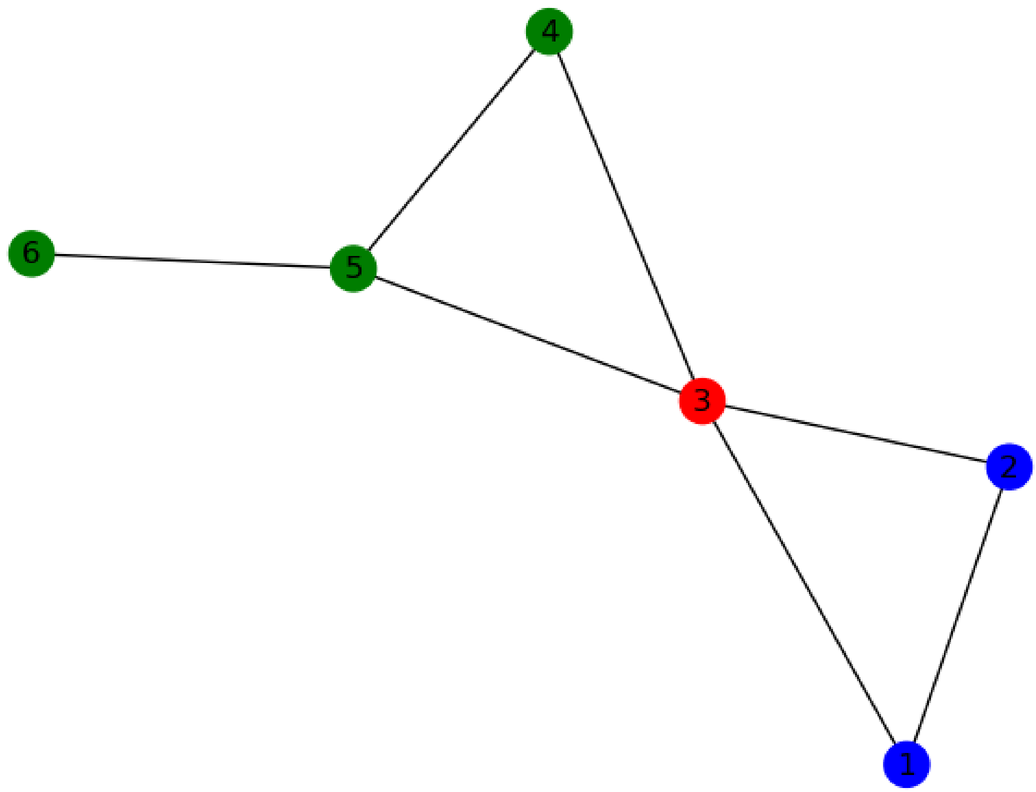
In [2]:
```
import networkx as nx
from scipy.sparse import diags

G = nx.Graph()

G.add_nodes_from([
    (1, {"class": "gr1"}),
    (2, {"class": "gr1"}),
    (3, {"class": "hub"}),
    (4, {"class": "gr2"}),
    (5, {"class": "gr2"}),
    (6, {"class": "gr2"}),
    ])

G.add_edges_from(
    [(1, 2), (1, 3), (2, 3), (3, 4), (3, 5), (4, 5), (5, 6)]
)
```

In [3]:
```
nx.draw(G, with_labels=True)
```



Il est facile de colorer les noeuds via des *color maps*.

In [4]:
```
color_map = []
for node in G:
    if G.nodes[node]["class"] =="gr1":
        color_map.append('blue')
    elif G.nodes[node]["class"] =="gr2":
        color_map.append('green')
    else:
        color_map.append('red')
nx.draw(G, node_color=color_map, with_labels=True)
```



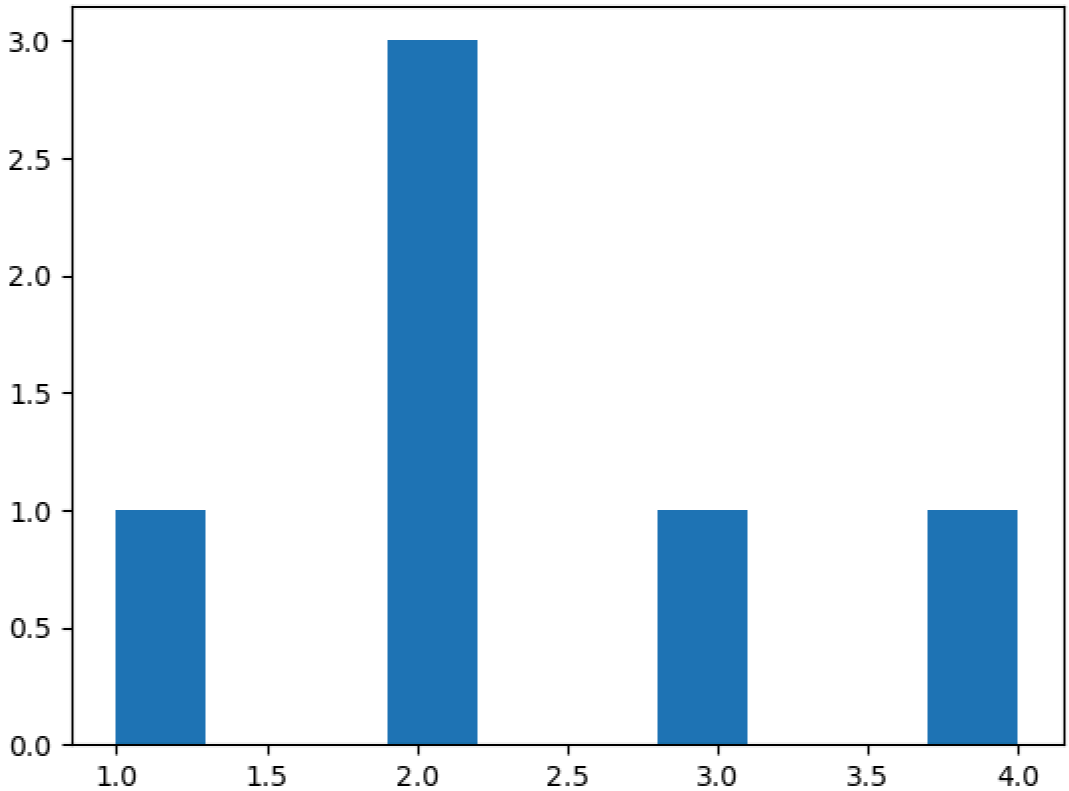
In [5]:
```
# récupérer les noeuds adjacents
list(G.adj[1])
```

Out[5]: [2, 3]

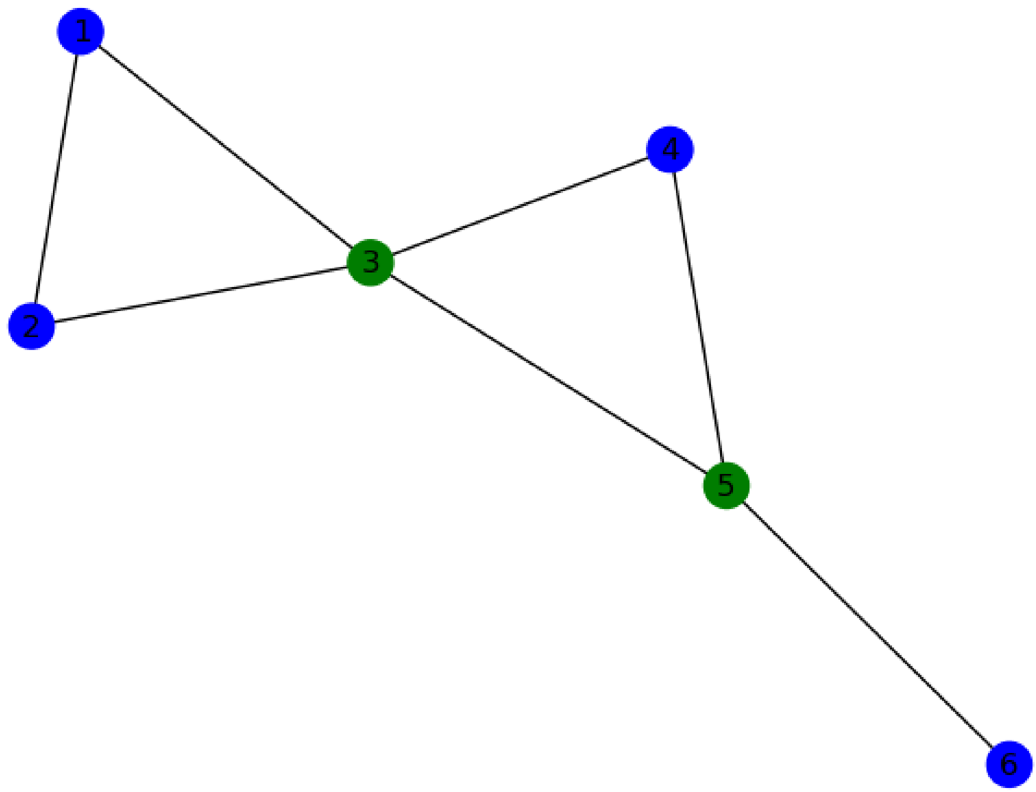
In [6]:
```
# degrés des noeuds
G.degree()
```

Out[6]: DegreeView({1: 2, 2: 2, 3: 4, 4: 2, 5: 3, 6: 1})

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
d_degree = dict(G.degree())
n, bins, patches = plt.hist(d_degree.values())
plt.show()
```



In [8]:
```
color_map = []
for node in G:
    if d_degree[node]<3:
        color_map.append('blue')
    else:
        color_map.append('green')
nx.draw(G, node_color=color_map, with_labels=True)
```



In [9]:
```
G.adj
```

Out[9]:
```
AdjacencyView({1: {2: {}, 3: {}}, 2: {1: {}, 3: {}}, 3: {1: {}, 2: {}, 4: {}, 5:
{}}, 4: {3: {}, 5: {}}, 5: {3: {}, 4: {}, 6: {}}, 6: {5: {}}})
```

Pour des calculs efficaces, il peut être judicieux de passer des structures *scipy* :

In [10]:
```
A = nx.to_scipy_sparse_array(G)
```

In [11]:
```
A.todense()
```

Out[11]:
```
array([[0, 1, 1, 0, 0, 0],
       [1, 0, 1, 0, 0, 0],
       [1, 1, 0, 1, 1, 0],
       [0, 0, 1, 0, 1, 0],
       [0, 0, 1, 1, 0, 1],
       [0, 0, 0, 0, 1, 0]])
```

In [12]:
```
A2 = A@A.transpose()
A2.todense()
```

Out[12]:
```
array([[2, 1, 1, 1, 1, 0],
       [1, 2, 1, 1, 1, 0],
       [1, 1, 4, 1, 1, 1],
       [1, 1, 1, 2, 1, 1],
       [1, 1, 1, 1, 3, 0],
       [0, 0, 1, 1, 0, 1]])
```

In [13]:
```
# tous les chemins de longueur 1 ou 2
(A+A2).todense()
```

Out[13]:
```
array([[2, 2, 2, 1, 1, 0],
       [2, 2, 2, 1, 1, 0],
       [2, 2, 4, 2, 2, 1],
       [1, 1, 2, 2, 2, 1],
       [1, 1, 2, 2, 3, 1],
       [0, 0, 1, 1, 1, 1]])
```

In [14]:
```
A3 = A@A2.transpose()
A3.todense()
```

Out[14]:
```
array([[2, 3, 5, 2, 2, 1],
       [3, 2, 5, 2, 2, 1],
       [5, 5, 4, 5, 6, 1],
       [2, 2, 5, 2, 4, 1],
       [2, 2, 6, 4, 2, 3],
       [1, 1, 1, 1, 3, 0]])
```

In [15]:
```
# tous les chemins de longueur 1, 2 ou 3
(A+A2+A3).todense()
```

Out[15]:
```
array([[4, 5, 7, 3, 3, 1],
       [5, 4, 7, 3, 3, 1],
       [7, 7, 8, 7, 8, 2],
       [3, 3, 7, 4, 6, 2],
       [3, 3, 8, 6, 5, 4],
       [1, 1, 2, 2, 4, 1]])
```

In [16]:
```
# calcul du PPR (q_u dans le cours)

GA = nx.google_matrix(G)
print(GA)
```

```
[[0.025      0.45       0.45       0.025      0.025      0.025     ]
 [0.45       0.025      0.45       0.025      0.025      0.025     ]
 [0.2375     0.2375     0.025      0.2375     0.2375     0.025     ]
 [0.025      0.025      0.45       0.025      0.45       0.025     ]
 [0.025      0.025      0.30833333 0.30833333 0.025      0.30833333]
 [0.025      0.025      0.025      0.025      0.875      0.025     ]]
```

```
/var/folders/44/_q8kssp12vb3ks1rlb59jm6m0000gp/T/ipykernel_17641/2859332506.py:3:
tureWarning: google_matrix will return an np.ndarray instead of a np.matrix in
NetworkX version 3.0.
  GA = nx.google_matrix(G)
```

In [17]:
```
K = 20
A_K = A
for i in range(K):
    A_K = A_K@A.transpose()
```

Calculons la matrice de transition (ou matrice aléatoire, ou matrice probabiliste) :

In [31]:
```
A_tran = A.transpose()
print(A_tran.todense())
```

```
norm = A_tran.sum(axis=0)
norm_adjusted = np.array([n if n>0 else 1 for n in norm])
print(norm)
P = A_tran/norm
P.todense()
```

```
[[0 1 1 0 0 0]
 [1 0 1 0 0 0]
 [1 1 0 1 1 0]
 [0 0 1 0 1 0]
 [0 0 1 1 0 1]
 [0 0 0 0 1 0]]
[2 2 4 2 3 1]
```

Out[31]:
```
array([[0.        , 0.5       , 0.25      , 0.        , 0.        ,
         0.        ],
        [0.5       , 0.        , 0.25      , 0.        , 0.        ,
         0.        ],
        [0.5       , 0.5       , 0.        , 0.5       , 0.33333333,
         0.        ],
        [0.        , 0.        , 0.25      , 0.        , 0.33333333,
         0.        ],
        [0.        , 0.        , 0.25      , 0.5       , 0.        ,
         1.        ],
        [0.        , 0.        , 0.        , 0.        , 0.33333333,
         0.        ]])
```

La matrice de transition est la base des mesures de centralité spectrales (cf. ci-dessous)

Illustration : décomposition spectrale

Cas sans boucle ni multi-arcs

In [19]:
```
A_d = A.todense()
print(A_d)
```

```
[[0 1 1 0 0 0]
 [1 0 1 0 0 0]
 [1 1 0 1 1 0]
 [0 0 1 0 1 0]
 [0 0 1 1 0 1]
 [0 0 0 0 1 0]]
```

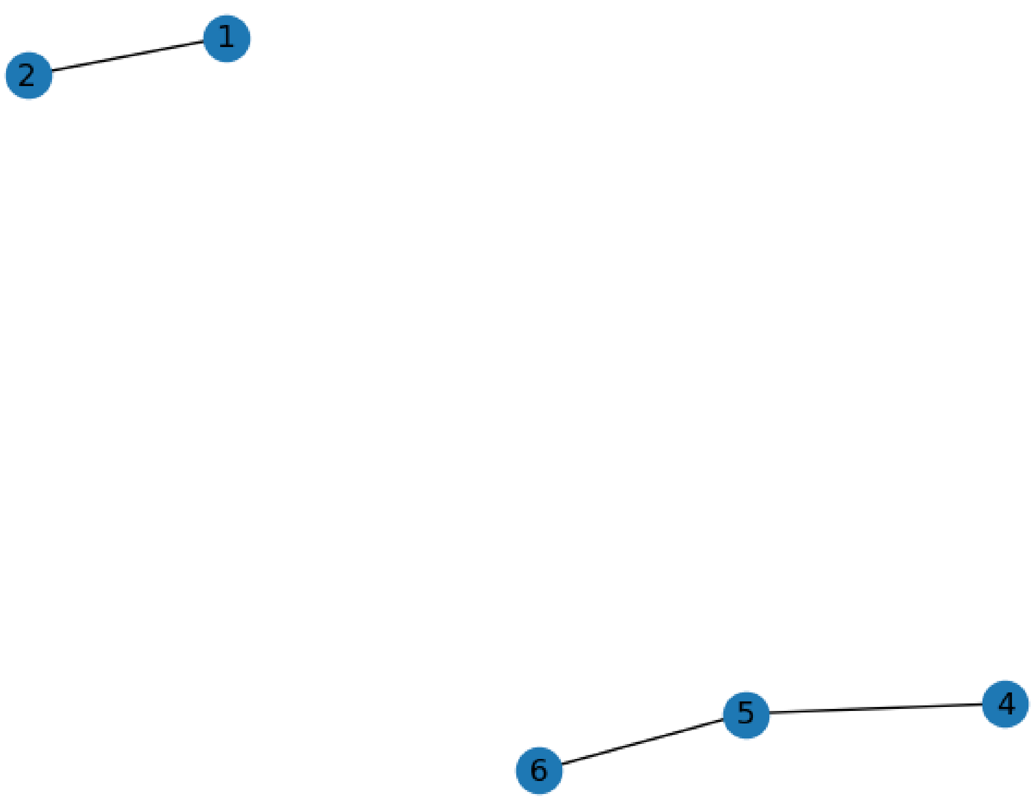
In [20]:
```
# matrice diagonale des degrés
D_d = np.diag([v for k,v in G.degree()])

print(D_d)
```

```
[[2 0 0 0 0 0]
 [0 2 0 0 0 0]
 [0 0 4 0 0 0]
 [0 0 0 2 0 0]
 [0 0 0 0 3 0]
 [0 0 0 0 0 1]]
```

In [21]:
```
# simple laplacien
L_d = D_d - A_d
print(L_d)
```

```
[[ 2 -1 -1  0  0  0]
 [-1  2 -1  0  0  0]
 [-1 -1  4 -1 -1  0]
 [ 0  0 -1  2 -1  0]
 [ 0  0 -1 -1  3 -1]
 [ 0  0  0  0 -1  1]]
```

In [22]:
```
#A_2comp = A_d.copy()
#A_2comp[2,:] = 0
#A_2comp[:,2] = 0
#G_2comp = nx.Graph(A_2comp)

G_2comp = G.copy()
G_2comp.remove_node(3)

nx.draw(G_2comp, with_labels=True)
```



In [23]:
```
A_2comp = nx.to_scipy_sparse_array(G_2comp).todense()
D_2comp = np.diag([v for k,v in G_2comp.degree()])
print(D_2comp)
```

```
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 2 0]
 [0 0 0 0 1]]
```

In [24]:
```
L_2comp = D_2comp - A_2comp
```

In [25]:
```
print(L_2comp)
```

```
[[ 1 -1  0  0  0]
 [-1  1  0  0  0]
 [ 0  0  1 -1  0]
 [ 0  0 -1  2 -1]
 [ 0  0  0 -1  1]]
```

In [26]:
```
from numpy import linalg as LA
eigenvalues, eigenvectors = LA.eig(L_2comp)
```

In [27]:
```
print(eigenvalues)
print(eigenvectors)
```

```
[ 2.000000e+00  0.000000e+00  3.000000e+00  1.000000e+00 -6.172564e-17]
[[ 7.07106781e-01  7.07106781e-01  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [-7.07106781e-01  7.07106781e-01  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00 -4.08248290e-01 -7.07106781e-01
   5.77350269e-01]
 [ 0.00000000e+00  0.00000000e+00  8.16496581e-01  3.06694063e-16
   5.77350269e-01]
 [ 0.00000000e+00  0.00000000e+00 -4.08248290e-01  7.07106781e-01
   5.77350269e-01]]
```

Eigen centrality

On va utiliser la méthode d'itération puissance (*power iteration*):

e(t+1) = P*e(t)

In [28]:
```
A = nx.to_scipy_sparse_array(G)
A.todense()
```

Out[28]:
```
array([[0, 1, 1, 0, 0, 0],
       [1, 0, 1, 0, 0, 0],
       [1, 1, 0, 1, 1, 0],
       [0, 0, 1, 0, 1, 0],
       [0, 0, 1, 1, 0, 1],
       [0, 0, 0, 0, 1, 0]])
```

In [33]:
```
P.todense()
```

```
Out[33]: array([[0.        , 0.5       , 0.25      , 0.        , 0.        ,
          0.        ],
         [0.5       , 0.        , 0.25      , 0.        , 0.        ,
          0.        ],
         [0.5       , 0.5       , 0.        , 0.5       , 0.33333333,
          0.        ],
         [0.        , 0.        , 0.25      , 0.        , 0.33333333,
          0.        ],
         [0.        , 0.        , 0.25      , 0.5       , 0.        ,
          1.        ],
         [0.        , 0.        , 0.        , 0.        , 0.33333333,
          0.        ]])
```

```
In [34]: e_0 = np.ones(6)
         print(e_0)
```

```
[1. 1. 1. 1. 1. 1.]
```

```
In [35]: e_1 = P@e_0
         print(e_1)
```

```
[0.75       0.75       1.83333333 0.58333333 1.75       0.33333333]
```

```
In [36]: e_2 = P@e_1
         print(e_2)
```

```
[0.83333333 0.83333333 1.625      1.04166667 1.08333333 0.58333333]
```

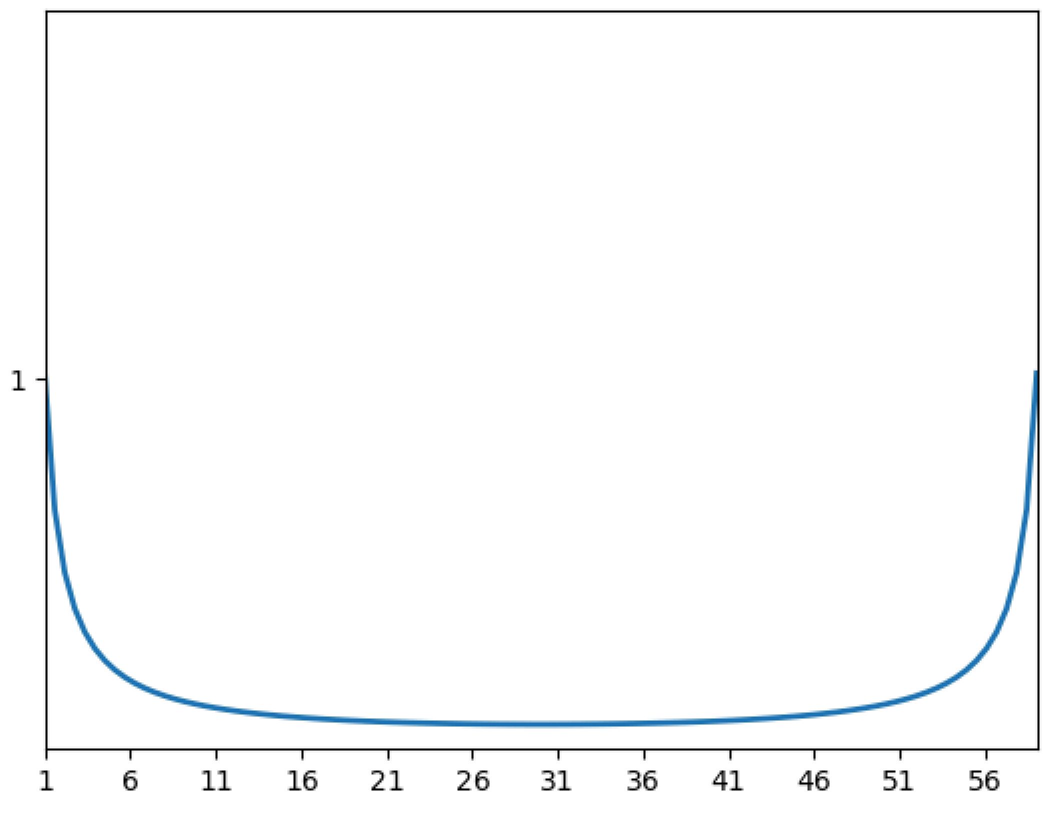
```
In [37]: e = e_0
         for i in range(20):
             print(e)
             e = P@e
```

```
[1. 1. 1. 1. 1. 1.]
[0.75       0.75       1.83333333 0.58333333 1.75       0.33333333]
[0.83333333 0.83333333 1.625      1.04166667 1.08333333 0.58333333]
[0.82291667 0.82291667 1.71527778 0.76736111 1.51041667 0.36111111]
[0.84027778 0.84027778 1.71006944 0.93229167 1.17361111 0.50347222]
[0.84765625 0.84765625 1.69762731 0.81872106 1.39713542 0.3912037 ]
[0.84823495 0.84823495 1.72272859 0.89011863 1.22497106 0.46571181]
[0.85479962 0.85479962 1.70161796 0.83900584 1.34145327 0.40832369]
[0.8528043  0.8528043  1.72145363 0.87255558 1.2532311  0.44715109]
[0.85676556 0.85676556 1.70682579 0.84810711 1.31379229 0.4177437 ]
[0.85508923 0.85508923 1.71874987 0.86463721 1.2685037  0.43793076]
[0.85723208 0.85723208 1.7102424  0.85252204 1.29993684 0.42283457]
[0.85617664 0.85617664 1.71680538 0.86087288 1.27665618 0.43331228]
[0.85728966 0.85728966 1.71216514 0.85475341 1.29295006 0.42555206]
[0.85668612 0.85668612 1.71564972 0.85902464 1.28097005 0.43098335]
[0.85725549 0.85725549 1.71318845 0.85590245 1.2894081  0.42699002]
[0.85692486 0.85692486 1.71500941 0.85809981 1.28323835 0.4298027 ]
[0.85721478 0.85721478 1.71372088 0.85649847 1.28760496 0.42774612]
[0.85703761 0.85703761 1.71466567 0.85763187 1.28442557 0.42920165]
[0.85718522 0.85718522 1.71399541 0.85680828 1.28668401 0.42814186]
```

On observer bien la convergence de la chaîne de Markov.

Miscellanées

In [50]:
```
# affichage du facteur pour le RatioCut

import matplotlib.pyplot as plt

x = np.linspace(1, 59, 100)
y = (1/x) + (1/(60-x))

fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

ax.set(xlim=(1, 59), xticks=np.arange(1, 59,5),
       ylim=(0, 2), yticks=np.arange(1, 2))

plt.show()
```



In [ ]: