In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import random
```

Ex.VII.1 : Write a code to simulate consensus dynamics on a network, and verify that the dynamics asymptotically converges towards the state $x_* = 1^\top x0/n$.

In [2]:

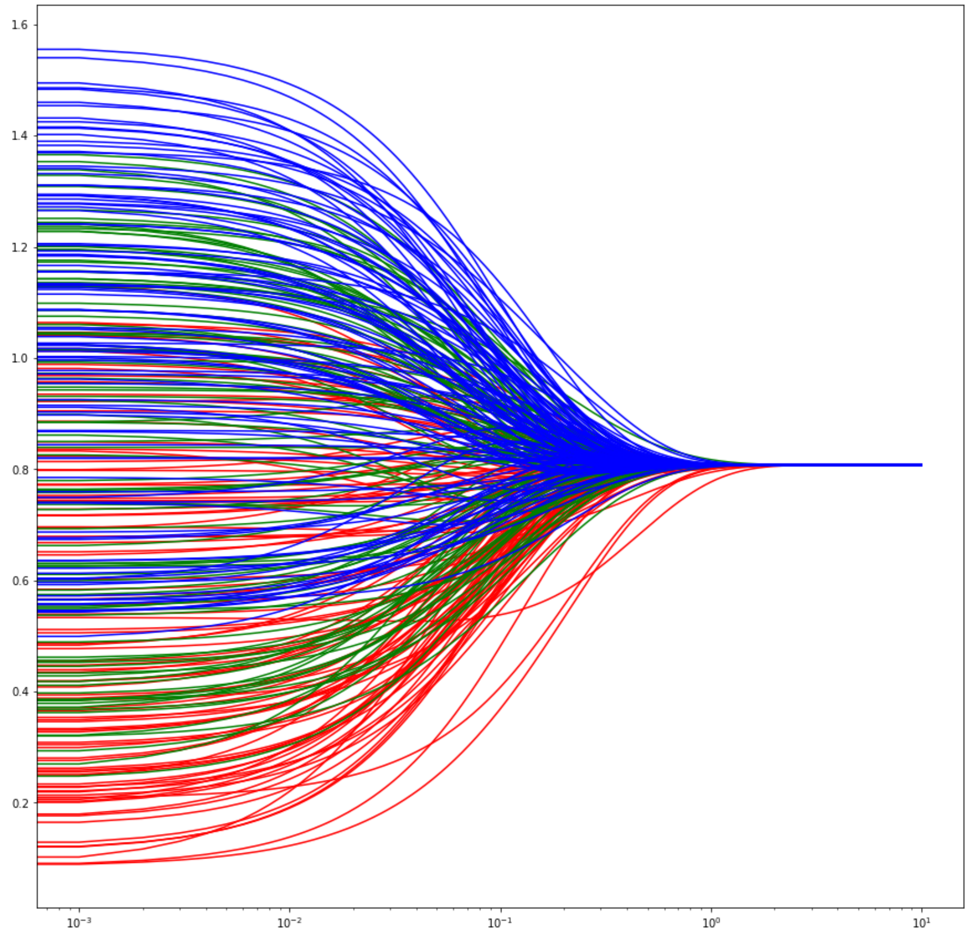
```
# Ex.VII.1

G = nx.karate_club_graph ()
n = len (G.nodes())
x0 = np.random.rand (1, n)

L = nx.laplacian_matrix (G).todense ()

x = x0
dt = 0.0001
t = 0.0
while t < 100:
    x = x - dt * np.dot (x, L)
    t += dt

print (x)
print (np.sum (x0) / n)
```

```
[[0.60301162 0.60301162 0.60301162 0.60301162 0.60301162 0.
60301162
  0.60301162 0.60301162 0.60301162 0.60301162 0.60301162 0.
60301162
  0.60301162 0.60301162 0.60301162 0.60301162 0.60301162 0.
60301162
  0.60301162 0.60301162 0.60301162 0.60301162 0.60301162 0.
60301162
  0.60301162 0.60301162 0.60301162 0.60301162 0.60301162 0.
60301162
  0.60301162 0.60301162 0.60301162 0.60301162]]
0.6030116174982025
```

Ex.VII.2 : Write a code to reproduce the numerical results of Figure 23 in the lecture notes

In [3]:

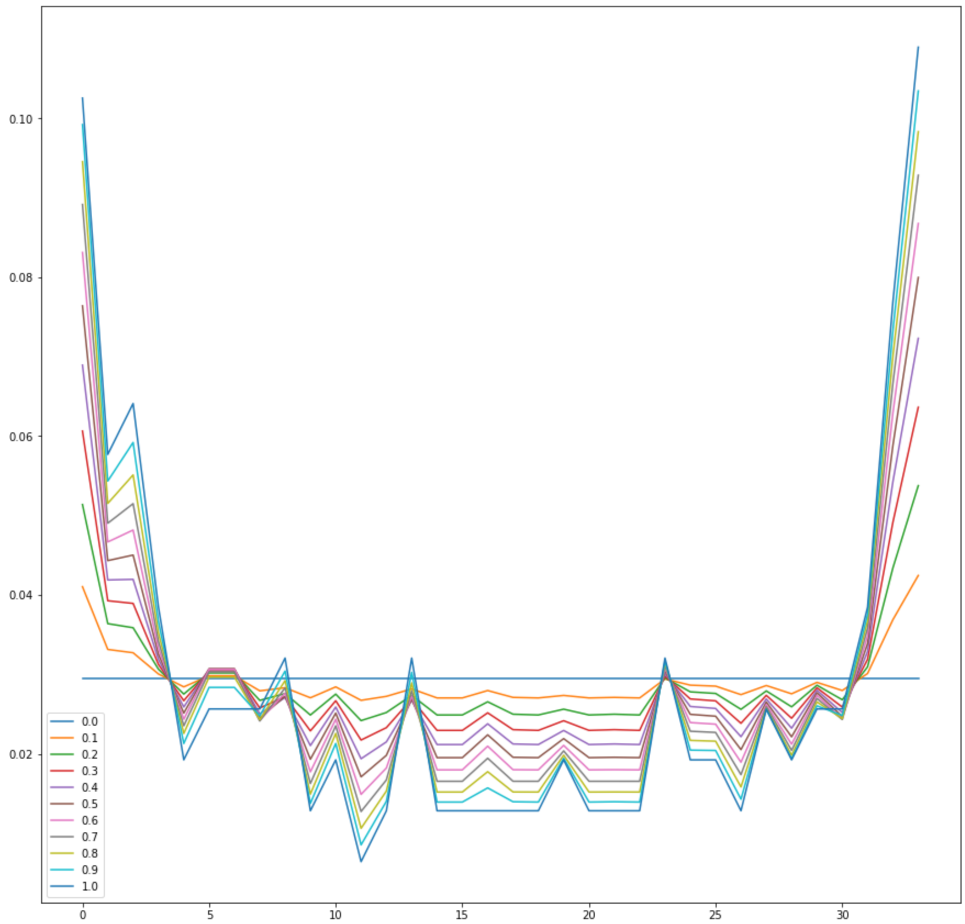
```
# Ex.VII.2

n = 300
G = nx.Graph ()
G.add_nodes_from (range (n))

dynamics_dic = {}
for i in range (n):
    dynamics_dic[i] = []
    for j in range (n):
        p = 0.02
        if i / 100 == j / 100:
            p = 0.8
        if random.random () < p:
            G.add_edge (i, j)

x0 = []
for i in range (n):
    x0.append (random.random () + (i / 100) * 0.2 )
L = nx.laplacian_matrix (G).todense ()


x = np.array (x0)
dt = 0.001
t = 0.0
times = []
while t < 10:
    times.append (t)
    for i in range (n):
        dynamics_dic[i].append (x.item(i))
    x = x - dt * np.dot (x, L)
    t += dt

plt.figure (figsize=(15,15))
for i in range (100):
    plt.plot (times, dynamics_dic[i], 'r')
for i in range (100, 200):
    plt.plot (times, dynamics_dic[i], 'g')
for i in range (200, 300):
    plt.plot (times, dynamics_dic[i], 'b')
plt.xscale ('log')
plt.show ()
```



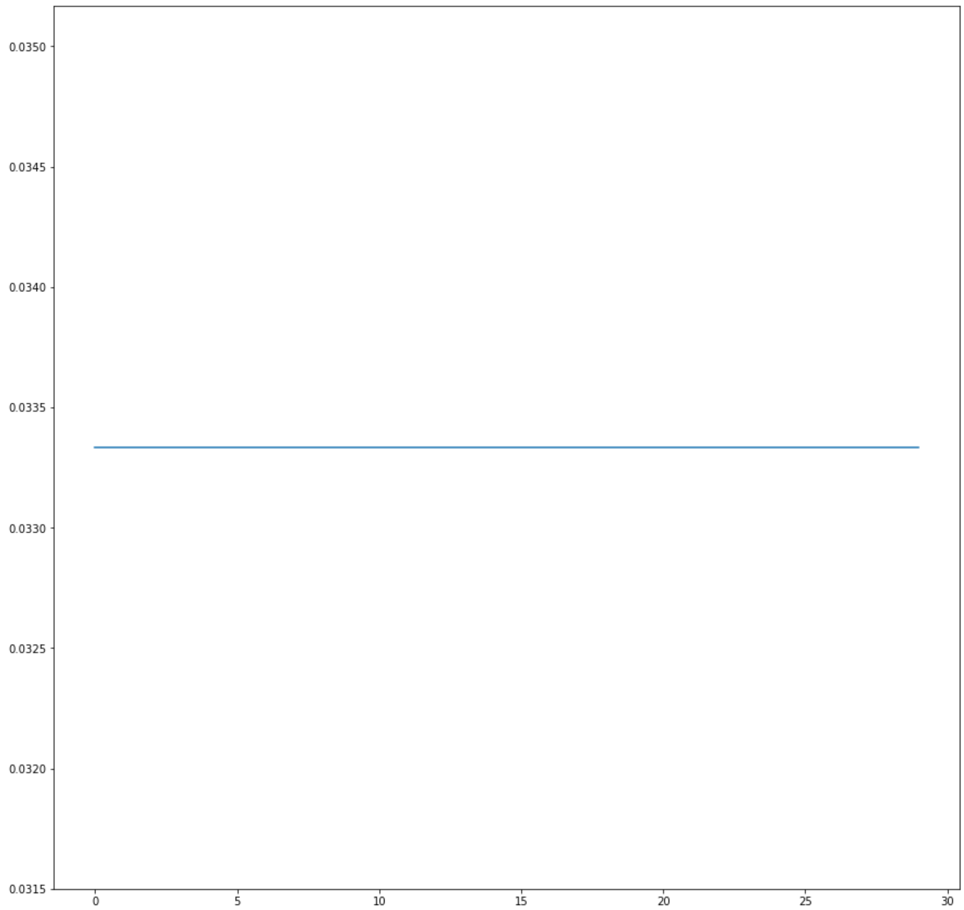
Ex.VIII.1: Write your own code to calculate the Pagerank of a directed network. Test on an example the dependency of Pagerank on the teleportation coefficient. Test your code on an undirected, connected, regular network and comment the results.

In [4]:

```
# ex. VIII.1


def pageRank (G, alpha):
    A = nx.adjacency_matrix (G)
    A = A.todense ()
    row_sums = np.sum (A, axis=1)
    T = A / (1.0 * row_sums)

    n = len (G.nodes())
    u = np.ones ((1, n)) / n

    p = np.ones ((1, n)) / n
    for i in range (1000):
        p = alpha * np.dot (p, T) + (1 - alpha) * u

    return p

plt.figure (figsize=(15,15))
G = nx.karate_club_graph ()
for alpha in np.arange (0, 1.00001, 0.1):
    p = pageRank (G, alpha)
    plt.plot ([p.item (ci) for ci in range (len (G.nodes()))], label=str
plt.legend ()
plt.show ()


plt.figure (figsize=(15,15))
G = nx.random_regular_graph (5, 30)
p = pageRank (G, 0.85)
plt.plot ([p.item (ci) for ci in range (len (G.nodes()))])
plt.show ()
```



0.10
0.08
0.06
0.04
0.02
0
5
10
15
20
25
30
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0



Ex.VIII.2: By performing stochastic simulations of an ensemble of random walkers on a graph, verify numerically Kac's formula.

In [5]:

```
# ex. VIII.2

G = nx.karate_club_graph ()
n = len (G.nodes ())

def step (G, node):
    return random.choice (list(G.neighbors (node)))

def retTime (G, node):
    steps = 0
    nnode = node
    while (steps == 0) or (nnode != node):
        nnode = step (G, nnode)
        steps += 1
    return steps

repeat_num = 10000
m = [0.0] * n
for i in range (n):
    for j in range (repeat_num):
        m[i] += retTime (G, i)
    m[i] /= repeat_num

A = nx.adjacency_matrix (G)
A = A.todense ()
row_sums = np.sum (A, axis=1)
T = A / (1.0 * row_sums)

p = np.ones ((1, n)) / n
for i in range (10000):
    p = np.dot (p, T)

for i in range (n):
    print (p.item (i) * m[i] - 1)
```

```
0.017241025641022256
0.0015153846153812545
0.0017692307692274145
-0.009684615384618733
-0.009261538461541652
0.008846153846150617
-0.022489743589746647
0.007789743589740272
0.017833333333329815
-0.007946153846157156
0.0007557692307660346
-0.003782051282054666
```

```
-0.010326923076926398
0.00808974358974024
0.007996153846150378
0.010166666666663327
0.011424358974355675
-0.011444871794875189
-0.006388461538464951
0.011780769230765875
-0.006326923076926394
-0.0002512820512854397
-0.00829230769231104
0.01418910256409922
0.0016673076923043872
-0.010996153846157264
-0.011337179487182825
0.0015256410256376807
0.005319230769227357
0.008787179487176111
0.0028769230769196685
0.0005038461538429129
0.019307692307688917
-0.016953205128208526
```

Ex IX.1: Propose and justify a generalization of Markov stability Eq. 207 in the case of directed networks. SOLUTION: See page 15 of the slides of Week 7.

Ex IX.2: Read ”Comparing clusterings - an information based distance, M Meila, 2017”, and implement numerically a method to compare different partitions.

In [ ]:

```
# Ex.IX.2

from math import log

def compareCD (partition1, partition2):
    p1 = {}
    n = len (partition1.keys())
    if len (partition1.keys()) != n:
        return -1
    h1 = 0.0
    for com in set(partition1.values()) :
        p1[com] = set ([nodes for nodes in partition1.keys() if partitio
        pk = 1.0 * len (p1[com]) / n
        h1 -= log (pk) * pk

    p2 = {}
    h2 = 0.0
    for com in set(partition2.values()) :
        p2[com] = set ([nodes for nodes in partition2.keys() if partitio
        pk = 1.0 * len (p2[com]) / n
        h2 -= log (pk) * pk


    I = 0.0
    for c1 in set(partition1.values()) :
        for c2 in set(partition2.values()) :
            pk1 = 1.0 * len (p1[c1]) / n
            pk2 = 1.0 * len (p2[c2]) / n
            c1c2 = len (p1[c1].intersection (p2[c2]))
            if c1c2 > 0:
                pkk = 1.0 * c1c2 / n
                I += pkk * log (pkk / pk1 / pk2)

    return h1 + h2 - 2 * I
```