PageRank实验内容与要求

**第一个部分：从概率转移矩阵计算得到pagerank值**

1. 实验目的

* 熟悉、理解和掌握PageRank算法结构，能够利用编程实现概率转移矩阵计算得到每个节点对应的rank值，并在此基础上理解采集器陷阱；
* 熟悉、理解和掌握避免采集器陷阱的“抽税”算法，并能在已经实现的PageRank程序上进行相应的修改得到相应的“抽税”算法程序；
* 更进一步，理解PageRank算法的收敛性分析（此为补充内容）。

1. 实验工具和环境

最好使用Python 3版本，利用其他程序语言实现也可以。

1. 算法过程

* 输入：概率转移矩阵（M），最大迭代次数（max\_iterations），收敛阈值（min\_epsilon）
* 初始化page\_rank值（可以使用向量储存，）：page\_rank[i]=1/n其中n是图中节点个数
* 当迭代次数小于最大迭代次数，且前后所有page\_rank值差的绝对值的和小于阈值，则：新的page\_rank值计算结果为前一个page\_rank向量左乘概率转移矩阵

否则，跳出循环

* 输出每一个节点对应的page\_rank值。

1. 实验步骤（也可以不完全按照此步骤执行，理解上面算法过程并将其实现即可）
2. 创建一个python项目文件，并添加一个python格式的文件
3. 该程序用到numpy包，在文件头输入“import numpy”
4. 定义一个pagerank迭代器（可命名为PRIterator）的类，并在类中定义变量最大迭代次数，收敛阈值以及概率转移矩阵，并将前两个设置默认值为100,0.00001
5. 在类中只需要定义一个page\_rank的方法即可，方法主要实现算法的第二步和第三步，初始化代码如下：

graph\_size = numpy.size(self.matrix,0) #得到图节点个数  
page\_rank = numpy.ones([graph\_size, 1])/graph\_size #初始化各页面被访问概率

循环过程代码如下：

flag = False  
**for** i **in** range(self.max\_iterations):  
 old\_page\_rank = page\_rank  
 page\_rank = numpy.dot(self.matrix, page\_rank)  
 change = numpy.abs(page\_rank - old\_page\_rank)  
   
 **if** numpy.sum(change) < self.min\_spsilon:  
 flag = True  
 **break**

1. 利用print函数输出结果
2. 分别输入课本例5.1、例5.3和例5.4中的概率转移矩阵，并调用类中的方法，看输出结果

**第二个部分：采用“抽税”算法**

1. “抽税”算法过程（划线部分为与之前算法不同部分）：

* 输入：概率转移矩阵（M），最大迭代次数（max\_iterations），收敛阈值（min\_epsilon），beta常数
* 初始化page\_rank值（可以使用向量储存）：page\_rank[i]=1/n其中n是图中节点个数
* 当迭代次数小于最大迭代次数，且前后所有page\_rank值差的绝对值的和小于阈值，则：page\_rank = beta\*M\*pagerank + （1-beta）\***e**/n

否则，跳出循环

* 输出每一个节点对应的page\_rank值。

1. 实验步骤（主要提及需要修改的地方）：
2. 在类中需要加入一个beta变量
3. 初始化时候，添加一个**e**向量，代码如下

e = numpy.ones([graph\_size, 1])

1. 循环语句中需要修改迭代式，代码如下：

page\_rank = self.beta \* numpy.dot(self.matrix, page\_rank) + (1-self.beta) \* e/graph\_size

1. 重新输入例5.1、例5.3和例5.4中的概率转移矩阵，看输出结果