Google PageRank notes

It says "There is an additional weight of 0.85(1/n) connecting Pi to Pj provided that Pi links to Pj and that Pi links to n pages total.", really it should say "Pi links to n>0 pages total"

It appears their matrices are column-major indexed, P[i][j] means the ith column and the jth row, which I have not seen before. Actually now I may have remembered from an optimization class that this is Matlab style.

The stationary distribution for the internet on page 4 makes sense: P1 is the lowest rank since it has no inbound links, P2 is less highly ranked than P4 since they both have inbound links but P2's inbound link is from a page with two outbound links which makes it weigh less, and P3 is the highest rank since it has two inbound links.

It looks like there is a mistake in the general Pagerank matrix formula, the first $\bar{v_n}$ should be $\bar{v_2}$.

Exercises

Exercise 7.1.

(a) rank(P1) should be less than rank(P2), because P1 has no inbound links, while P2 has an inbound link.

(b) $[0.3508, 0.6491]$

(c) There is no dispartity, however I did think that the difference in rank would be larger. The reason why the difference is smaller than I was expecting is because I did not account for part of the full diagram being invisible, which is P2's directed arrows to itself and to P1, each with weight 1/2. This is why P1's weight was not very small, because P2 is important and one of P2's two links points to P1.

Exercise 7.2

(a) rank(P3)=rank(p5) are the smallest, since they have no inbound links while the rest do, so they receive no additional weight while the others do. P6 has three inbound links, more than the rest, so it has the highest rank. P4 has two inbound links, and it is referred to by the top-ranked P6, so it has the second highest rank. P1 is more highly ranked than P2 since they both have a single inbound link but P2's is from a website with no inbound links while P1's is from P2 which has more than one inbound link, making P1 having the third highest rank and P2 having the fourth highest rank.

(b) $[0.1114, 0.1352, 0.0731, 0.3393, 0.0731, 0.2681]$

(c) There are two disparities. First, P4 is more highly ranked than P6, against what I thought. It's probably because it is referred to by a highly-ranked page, and that outweights the additional link that P6 has. Second, P2 is ranked more highly than P1, against what I thought. I don't understand this one, I thought I had a good explanation.

How about an exercise to show that the generalized PageRank matrix is a regular transition matrix?

Exercise 7.3

(a) This helps https://www.math.umd.edu/~immortal/MATH401/book/ch_markov_chains.pdf. It's the previous chapter of the book. $[[0, 1], [1, 0]]$ is not regular since there are zeros, and for $\bar{x}_0 = [1, 0], T^k\bar{x}_0$ is $[1, 0]$ for even powers and $[0, 1]$ for odd powers of k, meaning $T^k\bar{x}_0$ does not converge.

(b) Yes, consider this internet: $a_1 \leftrightsquigarrow a_2 \leftrightsquigarrow \ldots \leftrightsquigarrow a_{1000} \rightsquigarrow a_{1001} \leftrightsquigarrow a_{1002}$. There is a whole rather connected internet in the first 1000 pages, it's just that the 1001st page does not connect to the left internet page. Unfortunately, the probability of eventually encountering the 1001st page is 1, thus the limiting distribution is $[0, 0, ..., 0, \frac{1}{2}, \frac{1}{2}]$. This gives a non-regular T because there is no direct link from e.g.a_1 to a_3.

Exercise 7.4

(a) I am going to guess first and then calculate. P3 and P4 are lowest and equal, then P5, then P2, then P1, then P6.

It is $[0.2454, 0.1765, 0.0954, 0.0954, 0.1359, 0.2515]$.

Ok my intuitive ordering was right.

Exercise 7.5

Numerical convergence happens after the twelfth step.

```
def iterations_until_numerical_convergence(
    pr_matrix:np.array,vector:np.array
)->int:
    n=len(vector)
    steps=0
    prev=vector
    vector=pr_matrix@vector
    decimals_equal=min([
        get_decimal_places_equal(prev,vector)
        for i in range(n)
    ])
    while decimals_equal<4:
        prev=vector
        vector=pr_matrix@vector
        decimals_equal=min([
            get_decimal_places_equal(prev[i],vector[i])
            for i in range(n)
        ])
        steps+=1
        print(f'{steps=}\n{prev=}, \n{vector=},')
        print(f'{get_decimal_places_equal(prev,vector)=}\n')
    return steps

iterations_until_numerical_convergence(
    e7_5_matrix,
    np.array([1,0,0,0,0,0],dtype=float).T
)
```

Exercise 7.6

What is $\bar{v_n}$, since each page i has n outbound links, then each entry j equals $1/n$ since i has an outbound link to j. Then $T = \frac{0.15}{n}[\text{all entries ones}] + 0.85[\bar{v_1}\bar{v_2}..\bar{v_n}] = \frac{0.15}{n}[\text{all entries ones}] + 0.85[\text{all entries } \frac{1}{n}] = 0.15[\text{all entries } \frac{1}{n}] + 0.85[\text{all entries } \frac{1}{n}] = [\text{all entries } \frac{1}{n}]$. Indeed each page has pagerank $\frac{1}{n}$. The vector with all entries equal and adding to 1 is the vector with each entry equal to $\frac{1}{n}$, too. Now check the eigenequality: $[\text{all entries } \frac{1}{n}]_{n\times n} \cdot [\text{all entries } \frac{1}{n}]_{n\times 1} =$ a vector where each entry is the sum of n terms, all $\frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$, so a vector of all $\frac{1}{n}$, equal to the original vector being multiplied, hence it is an eigenvector with eigenvalue 1.

Exercise 7.7

(a) Say p is the fraction of weight put on the constant matrix and $1 - p$ is the other fraction of the weight, which is put on the second matrix. Typically $p = .15$. If $p = 1$ then all pages have the same Pagerank, which is not generally true, so to some extent, varying p does affect ranking. However, for all other values of p, the order should not be affected. It's because if you think of each entry in the final matrix as a function of p, they are all nonnegative linear functions of p, and hence as p is varied, each point on every line is in the same order, except for when $p = 1$ and they are all equal. Since the orders between the entries stays the same, intuitively the eigenvector should stay the same – this is the loosest statement of the argument, everything up to this last sentence was demonstrated to be true.

(b)

```python
def make_pr_matrix_exercise_7_7(
    edges:List[Tuple[int,int]],
    n_vertices:int,
    p:float#Fraction of weight to put on the constant matrix t
)->List[List[int]]:
    part1=np.array(
        [
            [1 for _ in range(n_vertices)]
            for _ in range(n_vertices)
        ]
        ,dtype=float
    )
    part2=np.array(
        [
            [0 for _ in range(n_vertices)]
            for _ in range(n_vertices)
        ]
        ,dtype=float
    )
    out_degrees={v:0 for v in range(n_vertices)}
    for (v1,_) in edges:
        out_degrees[v1]+=1
    for (v1,v2) in edges:
        part2[v2,v1]=1/out_degrees[v1]
    for v,out_degree in out_degrees.items():
        if out_degree==0:
            part2[:,v]=1/n_vertices
    return p/n_vertices*part1+(1-p)*part2

def order(li:List[float])->List[int]:
    li_sorted=sorted(li)
    return [
        li_sorted.index(li[ix])
        for ix in range(len(li))
    ]

e7_7_eigvecs=[get_probability_eigenvector(
    make_pr_matrix_exercise_7_7(
    one_indexed_vertices_to_zero_indexed([
        (1,2),
        (1,3),
        (2,3),
        (3,1)
    ])
    ,3
    ,p
)) for p in [.85,.15,.6,.4]]

print([order(li) for li in e7_7_eigvecs])
#Gives:
#[[1, 0, 2], [1, 0, 2], [1, 0, 2], [1, 0, 2]]
```

The orders are all the same.

(c)

```python
print(get_probability_eigenvector(
    make_pr_matrix_exercise_7_7(
    one_indexed_vertices_to_zero_indexed([
        (1,2),
        (1,3),
        (2,3),
        (3,1)
    ])
    ,3
    ,1
)))

#Gives:
#[0.3333, 0.3333, 0.3333]
```

As I predicted while thinking through part (a), all the pages have the same pagerank, so they are all number 1.

Exercise 7.8

It is $[0.1042, 0.0584, 0.1408, 0.0954, 0.1891, 0.0327, 0.1536, 0.0327, 0.0833, 0.1098]$. I had a feeling 5 would be the highest, indeed it is.

Exercise 7.9

Having outbound links on your page does not affect your Google Pagerank because a page i's column in T is not affected by the number of outbound links, only the other pages are affected. Actually this is not a good argument, because think of the following two moral counterarguments.

1. A page's Google Pagerank is the limit of the fraction of time spent on that page given the transition matrix T so maybe if the outbound links of a node are changed then it makes a surfer more or less likely to end up on a page that has a high or low chance of circling back to the original page, thereby overall increasing or decreasing the Pagerank of the page.
2. Consider the internet being a cycle of n pages, then removing the outbound link of one page. Then, instead of the probability eigenvector being $\frac{1}{n}$, it will increase the long-term fraction of time spent on the page with the outbound link removed. Unless $n = 1$, this affects the the Pagerank of each page.

In fact, counter-argument 2 is true, and the question is false. When a link is removed from page 3 to page 1 in a 3-cycle, the Pageranks become $[0.1844, 0.3412, 0.4744]$.

Exercise 7.10

```python
def line(n:int)->List[List[float]]:
    return make_pr_matrix([
            (z,z+1)
            for z in range(n)
            if z!=n-1
        ]
        ,n
    )
```

There are only three types of entries (i, j):

1. $i < n \to T[i, i+1] = 0.15 \cdot \frac{1}{n} + 0.85$
2. $i = n \to \forall j, T[i, j] = \frac{1}{n}$
3. Otherwise, $T[i, j] = 0.15 \cdot \frac{1}{n}$

Exercise 7.11

That is what the second matrix does. The first matrix makes Pagerank's T more reasonable by removing the assumption that a surfer can only follow links, and having them go to a random page 15% of the time, whether or not there is a link to that page from the current page. Indeed, in reality a surfer can type any URL into their browser.

Appendix for tool source code

```python
def one_indexed_vertices_to_zero_indexed(
    edges:List[Tuple[int,int]]
)->List[Tuple[int,int]]:
    return [(v1-1,v2-1) for (v1,v2) in edges]

def make_pr_matrix(
    edges:List[Tuple[int,int]],n_vertices:int
)->List[List[int]]:
    part1=np.array(
        [
            [1 for _ in range(n_vertices)]
            for _ in range(n_vertices)
        ]
        ,dtype=float
    )
    part2=np.array(
        [
            [0 for _ in range(n_vertices)]
            for _ in range(n_vertices)
        ]
        ,dtype=float
    )
    out_degrees={v:0 for v in range(n_vertices)}
    for (v1,_) in edges:
        out_degrees[v1]+=1
    for (v1,v2) in edges:
        part2[v2,v1]=1/out_degrees[v1]
    for v,out_degree in out_degrees.items():
        if out_degree==0:
            part2[:,v]=1/n_vertices
    return .15/n_vertices*part1+.85*part2

def get_probability_eigenvector(
    pr_matrix:List[List[float]]
):
    eigenpairs=np.linalg.eig(pr_matrix)
    ix=[
        i for i in range(len(eigenpairs.eigenvalues))
        if np.isclose(eigenpairs.eigenvalues.tolist()[i],1)
    ][0]
    return [
        round(float(x.real),4)
        for x in
        eigenpairs.eigenvectors[:,ix]
        /sum(eigenpairs.eigenvectors[:,ix])
    ]

#Example.
internet1_pr_matrix=make_pr_matrix(
    one_indexed_vertices_to_zero_indexed([
        (1,2),
        (1,3),
        (2,3),
        (3,4),
        (4,3)
    ])
    ,4
)
print(internet1_pr_matrix)
#Gives:
# array([[0.0375, 0.0375, 0.0375, 0.0375],
#        [0.4625, 0.0375, 0.0375, 0.0375],
#        [0.4625, 0.8875, 0.0375, 0.8875],
#        [0.0375, 0.0375, 0.8875, 0.0375]])
print(get_probability_eigenvector(
    internet1_pr_matrix
))
#Gives:
# [0.03749999999999999,
#  0.05343749999999988,
#  0.47111486486486487,
#  0.4379476351351353]
#You can also go straight to the probability eigenvector:
get_probability_eigenvector(
    make_pr_matrix(
        one_indexed_vertices_to_zero_indexed([
            (2,1),
            (3,2),
            (4,5),
            (4,6),
            (5,6)
        ])
        ,6
    )
)
```