

Page Rank

Ryan Greenup

October 15, 2020

Contents

1	Introduction	2
1.1	The PageRank Method	2
1.2	Power Walk and the Random Surfer	3
1.3	Stability and Convergence	3
2	Implementing PageRank Generally	3
2.1	Definitions	4
2.1.1	Notation	5
2.2	Random Surfer Model	6
2.2.1	Issues	6
2.2.2	Markov Chains	7
2.2.3	Limitations	10
2.3	Power walk	10
3	Sparse Matrices	10
3.1	Solving the Stationary Distribution	11
4	Implementing the Models	11
4.1	Implementing the Random Surfer	11
4.1.1	Small Graph, Ordinary Matrices	11
4.1.2	Large Graph, Sparse Matrices using CRS	19
4.2	Power Walk Method	24
4.2.1	Introduction	24
4.2.2	Ordinary Matrices	24
4.2.3	Sparse Matrices	26
5	Creating a Package	33
6	Investigating the Second EigenValue	34
6.1	ER Graphs Plotting Various Values	34
6.1.1	Erdos Reny Game	34
6.1.2	Correlation Plot	34
6.1.3	Density of Adjacency Matrix	37
6.1.4	Trace of Transition Probability Matrix	41
6.1.5	Conclusion	45

6.2	Model the log transformed data using a linear regression or log(-x) regression	47
6.2.1	Change the colour of each model by using $\text{pivot}_{\text{longer}}$	47
6.3	Import wikipedia data	47
6.4	Look at the Trace of the Matrix as a comparison point	47
6.5	Use BA Graphs	47
7	Simulating the Structure of the Web	47
8	Relating the Power Walk to the Random Surfer	48
8.1	Introduction	48
8.2	Value of [1st Term]	48
8.3	Value of {2nd Term}	49
8.4	Equate the Power Walk to the Random Surfer	50
8.5	Conclusion	51
8.6	The Second Eigenvalue	51
8.6.1	The Random Surfer	51
8.6.2	Power Walk	51
9	Appendix	54
9.1	Graph Diagrams	54
10	my to do list	54
10.1	Look at the Trace of the Matrix as a comparison point	54
10.2	Use BA Graphs	54
10.3	TODO Diamater	54
10.4	Improving the Performance of Page Rank	56

1 Introduction

Any collection of interconnected information can form a network structure, consider for example citations, webpages, wikis, encyclopedias and interpersonal relationships. The analysis of these networks can be used to draw insights about the behaviour of such networks.

One important form of analysis is *network centrality*, a concept concerned with the measure of the importance, popularity and relevance of a node. In a relatively small graph, visualised in such a way so as to minimise the overlapping of edges, a general expectation would be that the centrality score would be correlated with geometric-centrality, this is demonstrated in figure 5 where the 2nd vertex has the highest *PageRank* score and is geometrically very central.

1.1 The PageRank Method

There are multiple ways to measure network centrality but this report is concerned with the *PageRank* method, this method asserts that the centrality of a vertex can be measured by the frequency of incidence with that vertex during a random walk.

This approach only makes sense if the random walk can:

1. Traverse the entire network
2. Escape dead ends on a directed graph

and so the *PageRank* method involves:

- Altering a corresponding transition probability matrix such that it corresponds to a stochastic primitive *Markov Chain*.
- Considering the stationary distribution of this new graph.

1.2 Power Walk and the Random Surfer

The typical method to adjust the transition probability matrix is the *Random Surfer*, introduced by Page and Brin in 1998 [20] as a distinguishing feature of the *Google* search engine, this approach essentially introduces some probability of teleporting to other nodes during a random walk, this is illustrated in figure 4.

A shortcoming of this approach is that it assumes all edges are positively weighted. This means that the model treats any link as an endorsement of the destination node, this may not necessarily always be true (consider for example burned-in advertisements or negative reviews). In the past attributing weights to links was not particularly feasible, recent developments in sentiment analysis has however made this possible meaning that this limitation is more significant.

The *Power Walk* approach, introduced by Park and Simoff in 2013 [24] is an alternative way to create a transition probability matrix that is defined for real weighted edges and could be used with sentiment analysis to more effectively measure network centrality.

These individual approaches are discussed in more detail at 2 .

1.3 Stability and Convergence

The rate at which the algorithm for *PageRank* converges to a solution and the stability of that solution can both be measured by the second eigenvalue of the corresponding transition probability matrix (The details of this are discussed at 6).

It is not clear how the second eigenvalue is related to the method parameters of the *Power Walk* algorithm [24, §3.4] and this report aims to:

1. Implement methods to perform *PageRank* analysis using:
 - (a) The *Random Surfer* model
 - (b) The *Power Walk* model
2. Investigate the Relationship between the parameters of the *Power Walk* transition probability matrix and the second eigenvalue

2 Implementing PageRank Generally

A graph can be expressed as an adjacency matrix **A**:

$$A_{i,j} \in \{0, 1\}$$

Where each element of the matrix indicates whether or not travel from vertex j to vertex i is possible with a value of 1. ¹

¹Some authors define an adjacency matrix transposed (see e.g. [AdjacencyMatrix2020a, 1, 22]) this unfortunately includes the *igraph* library [12] but that convention will not be followed in this paper

During a random walk the probability of arriving at vertex j from vertex i can similarly be described as an element of a transition probability matrix $\mathbf{T}_{i,j}$, this matrix can be described by the following relationship²:

$$\mathbf{T} = \mathbf{A}\mathbf{D}_{\mathbf{A}}^{-1} : \quad (1)$$

$$\mathbf{D}_{\mathbf{A}} = \text{diag}(\vec{1}\mathbf{A}) \quad (2)$$

The value of \mathbf{D} is such that under matrix multiplication \mathbf{A} will have columns that sum to 1 (i.e. a *column stochastic matrix*, see § 2.1), for a reducible or non-stochastic graph the definition of \mathbf{D} would need to be adjusted to achieve this, this is discussed below

During the random walk, the running tally of frequencies, at the i^{th} step of the walk, can be described by a vector \vec{p} , this vector can be determined for each step by matrix multiplication:

$$p_{i+1}\vec{p} = \mathbf{T}\vec{p}_i \quad (3)$$

This relationship is a linear recurrence relation, more importantly however it is a *Markov Chain* [19, §4.4].

Finding the Stationary point for this relationship will give a frequency distribution for the nodes and a metric to measure the centrality of vertices.

2.1 Definitions

The following definitions are used in this report³ :

Markov Chains are discrete mathematical model such that future values depend only on current values [10, §1.5]

Stochastic Matrices contain only positive values where each column sums to 1 [19, 10] (i.e. \mathbf{T} is stochastic $\iff \vec{1}\mathbf{T} = \vec{1}$)

- some authors use rows (see e.g. [19, §15.3]), in this paper columns will be used, i.e. columns will add to one and an entry $\mathbf{A}_{i,j} \neq 0$ will indicate that travel is permitted from vertex j to vertex i .
 - *Column Stochastic* and *Row Stochastic* can be used to more clearly distinguish between which type of stochastic matrix is being used.
- Many programming languages return *unit-eigenvectors* \vec{x} such that $||\vec{x}|| = 1$ as opposed to $\text{sum}(\vec{x}) = 1$, so when solving for a stationary vector it can be necessary to perform $\vec{p} \leftarrow \frac{\vec{p}}{\sum \vec{p}}$

Irreducible graphs have a path from from any given vertex to another vertex. [19, §15.2]

Ergodic graphs are irreducible graphs with further constraints outside the scope of this report (see e.g. [23, 8])

²In this paper $\vec{1}$ refers to a vector containing only values of 1, the size of which should be clear from the context

³see generally [19, Ch. 15] for further reading

- It is a necessary but not a sufficient condition of ergodic graphs that all vertices be reachable from any other vertices (see [25] for a counter example.)

Primitive Matrices are non-negative irreducible matrices that have only one eigenvalue on the unit circle.

- If a matrix is primitive it will approach a limit under exponentiation [19, §15.2]

Transition Probability Matrix is a stochastic matrix where each column is a vector of probabilities such that $T_{i,j}$ represents the probability of travelling from vertex j to vertex i during a random walk.

- Some Authors consider the transpose (see e.g. [19]).

Aperiodic Markov chains are markov chains with an irreducible and primitive transition probability matrix.

- If the transition probability matrix is irreducible and imprimitive it is said to be a periodic Markov chain.

Regular Markov Chains are regular irreducible and aperiodic.

Sparse Matrices contain a majority of elements with values equal to 0 [19, §4.2]

Sparse Iterating the

PageRank A process of measuring graph centrality by using a random walk algorithm and measuring the most frequent node

- In the literature (see e.g. [14, 19]) the Random Surfer model is usually used to refer to the introduction of a probability of travelling to any other node, this is discussed in CROSSREF

2.1.1 Notation

- **A**
 - Is the adjacency matrix of a graph such that $A_{i,j} = 1$ Indicates travel from j to i is possible.
- **T**
 - Indicates the probability of a movement during a random walk, such that $T_{i,j}$ is equal to the probability of travelling $j \rightarrow i$ during a random walk.
- $D_A = \text{diag}(\vec{1}A)$
- $D_A^{-1} = \begin{cases} 0, & [D_A]_i = 0 \\ \left[\frac{1}{D_A} \right], & [D_A]_i \neq 0 \end{cases}$
 - A diagonal scaling matrix such that $T = AD_A^{-1}$, the piecewise definition is such that D_A^{-1} is still defined even if A is a reducible graph.
 - * Where D^{-1} is a matrix such that multiplication with which scales each column of A to 1.
 - * $D_A^{-1} = \vec{1}D_A^{-1} = \frac{1}{\vec{1}D_A}$ for some stochastic matrix A
- n

- Refers to the number of vertices in a graph, $n = \text{nrow}(\mathbf{A}) = \text{ncol}(\mathbf{A})$
 - $\mathbf{E}_{i,j} = \frac{1}{n}$
 - A matrix of size $n \times n$ representing the background probability of jumping to any vertex of a graph.
 - $\vec{1}$
 - a vector of length n containing only the value 1.
 - * The convention that a vector behaves as a vertical $n \times 1$ matrix will be used here.
 - * Some authors use \mathbf{e} , see e.g. [19]
 - $\mathbf{J} = \vec{1} \cdot \vec{1}^T \iff \mathbf{J}_{i,j} = 1$
 - A completely dense $n \times n$ matrix containing only 1
 - α
 - The probability of teleporting from one vertex to another during a random walk.
 - * In the literature α is often referred to as a damping factor (see e.g. [5, 7, 11, 16, 6])
- or a smoothing constant (see e.g [17]).

2.2 Random Surfer Model

2.2.1 Issues

The approach in 2 has the following issues

1. Convergence of (3)
 - (a) Will this relationship converge or diverge?
 - (b) How quickly will it converge?
 - (c) Will it converge uniquely?
2. Reducible graphs
 - (a) If it is not possible to perform a random walk across an entire graph for all initial conditions, this approach doesn't have a clear analogue.
3. Cycles
 - (a) A graph that is cyclical may not converge uniquely
 - i. Consider for example the graph $A \rightarrow B$.

2.2.2 Markov Chains

The relationship in (3) is a *Markov Chain* and it is known that the power method will converge: ⁴

- for a stochastic irreducible markov chain [10, §1.5.5],
- regardless of the initial condition of the process for an *aperiodic* Markov chain [19, §4.4]

Stochastic If a vertex had a 0 outdegree the corresponding column sum for the adjacency matrix describing that graph would also be zero and the matrix non-stochastic, this could occur in the context of a random walk where a link to a page with no outgoing links was followed (e.g. an image), this would be the end of the walk.

So to ensure that (3) will converge, the probability transition matrix must be made stochastic, to achieve this a uniform probability of teleporting from a dead end to any other vertex can be introduced:

$$S = T + \frac{\vec{a} \cdot \vec{1}^T}{n} \quad (4)$$

This however would not be sufficient to ensure that (3) would converge, in addition the transition probability matrix must be made irreducible and aperiodic (i.e. primitive). [19]

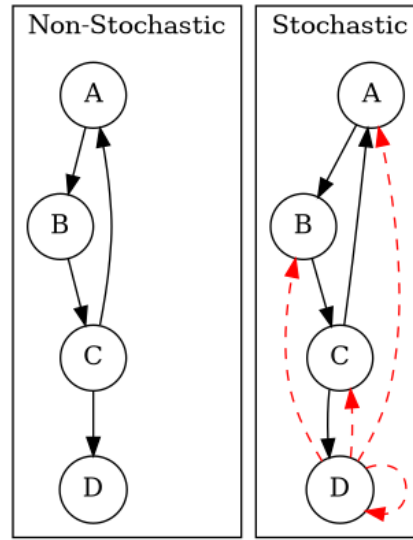


Figure 1: *D* is a *dangling node*, a dead end during a random walk, the corresponding probability transition matrix (**T**) is hence non-stochastic (and also reducible), Introducing some probability of teleporting from a dead end to any other vertex as per (4) (denoted in red) will cause **T** to be stochastic.

Irreducible A graph that allows travel from any given vertex to any other vertex is said to be irreducible [19], see for example figure 2, this is important in the context of a random walk because only in an irreducible graph can all vertexes be reached from any initial condition.

⁴A *Markov Chain* is simply any process that evolves depending on it's current condition, it's interesting to note however that the theory of *Markov Chains* is not mentioned in any of the original papers by page and brin [19, §4.4]

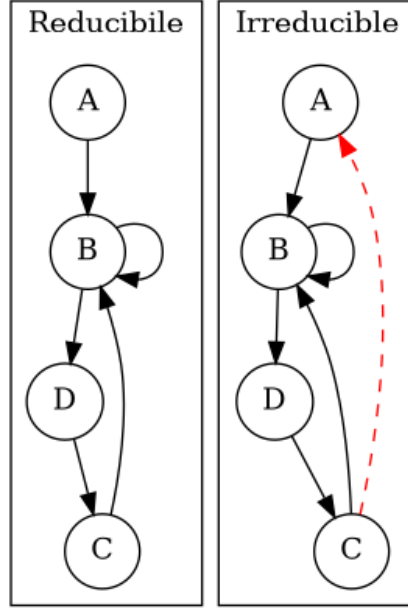


Figure 2: Example of a reducible graph, observe that although C is not a dead end as discussed in 2.2.2 , there is no way to travel from C to A , by adding an edge such an edge in the resulting graph is irreducible. The resulting graph is also aperiodic (due to the loop on B) and stochastic, so there will be a stationary distribution corresponding to (3).

Aperiodic An aperiodic graph has only one eigenvalue that lies on the unit circle, this is important because $\lim_{k \rightarrow \infty} \left(\frac{\mathbf{A}^k}{r} \right)$ exists for a non-negative irreducible matrix \mathbf{A} if and only if \mathbf{A} is aperiodic. A graph that is a periodic can be made aperiodic by interlinking nodes ⁵

The Fix To ensure that the transition probability matrix is primitive (i.e. irreducible and aperiodic) as well as stochastic, instead of introducing the possibility to teleport out of dead ends, introduce a probability of teleporting to any node at any time (α), this approach is known as the *Random Surfer* model and the transition probability matrix is given by [20] :

$$\mathbf{S} = \alpha \mathbf{T} + \frac{(1 - \alpha)}{n} \mathbf{J} \quad (5)$$

This matrix is primitive and stochastic and so will converge (it is also unfortunately completely dense, see 3.1 [19, §4.5]).

The relation ship in (3) can now be re expressed as:

$$p_{i+1} \vec{p}_i \rightarrow \mathbf{T} \vec{p}_i \quad (6)$$

⁵ Actually it would be sufficient to merely link one vertex to itself [19, §15.2] but this isn't very illustrative or helpful in this context

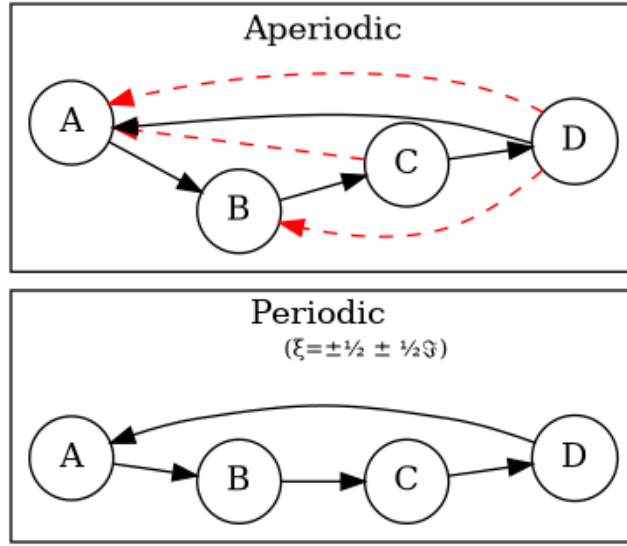


Figure 3: A periodic graph with all eigenvalues on the unit circle $\xi = \frac{\sqrt{2}}{2} e^{\frac{\pi i}{4} k}$, by adding in extra edges the graph is now aperiodic, this does not represent the random surfer model, which would in theory connect every vertex but with some probability.

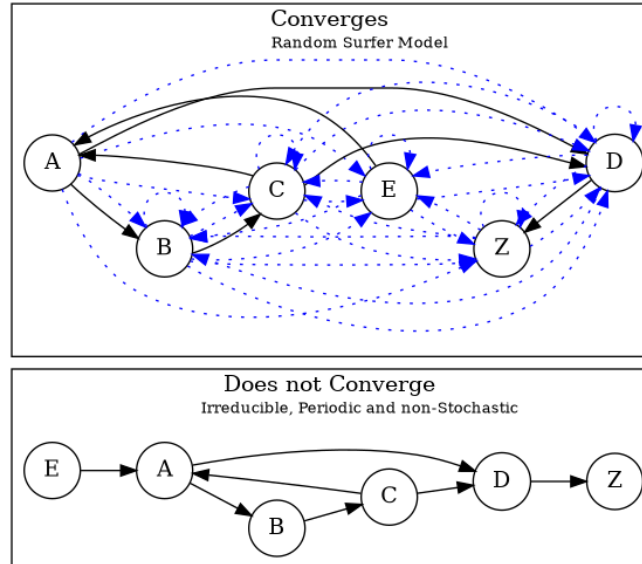


Figure 4: A graph that is aperiodic, reducible and non-stochastic, by applying the random surfer model (5) blue *teleportation* edges are introduced, these may be followed with a probability of $1 - \alpha$

2.2.3 Limitations

The *Random Surfer Model* can only consider positively weighted edges, it cannot take into account negatively weighted edges. This limitation is increasingly important as techniques of sentiment analysis are developed which could indicate that links promote aversion rather than endorsement (e.g. a negative review or an inappropriate advertisement).

2.3 Power walk

The *Power Walk* method is an alternative approach to develop a probability transition matrix to use in place of (3).

Let the probability of travelling to a non-adjacent vertex be some value x and β be the ratio of probability between following an edge or teleporting to another vertex.

This transition probability matrix would be such that the probability of travelling some vertex $j \rightarrow i$ would be :

$$\mathbf{W}_{i,j} = x\beta^{\mathbf{A}_{i,j}} \quad (7)$$

Where \mathbf{W} denotes the power walk probability transition matrix.

Where probability of travelling to any given vertex must be 1 and so:

$$1 = \sum_{j=1}^n [x\beta^{\mathbf{A}_{i,j}}] \quad (8)$$

$$\Rightarrow x = \left(\sum_{j=1}^n \beta^{\mathbf{A}_{i,j}} \right)^{-1} \quad (9)$$

Substituting the value of x from (9) into (7) gives the probability as:

$$\mathbf{W}_{i,j} = \frac{\beta^{\mathbf{A}_{i,j}}}{\sum_{i=j}^n [\beta^{\mathbf{A}_{i,j}}]} \quad (10)$$

In this model all vertices are interconnected by some probability of jumping to another vertex, so much like the random surfer model (5) discussed at 2.2.2 \mathbf{W} will be a primitive stochastic matrix and so if \mathbf{W} was used in place of \mathbf{T} in (3) a solution would exist.

3 Sparse Matrices

Most Adjacency matrices resulting from webpages and analogous networks result in sparse adjacency matrices (see figure 10), this is a good thing because it requires far less computational resources to work with a sparse matrix than a dense matrix [19, §4.2] .

Sparse matrices can be expressed in alternative forms so as to reduce the memory footprint associated with that matrix, one such method is the *Compressed Row Storage* method, this involves listing the elements as a table as in (11) and (12).

This is implemented in **R** with the **Matrix** package [**batesMatrixSparseDense2019a**] .

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \pi \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (11)$$

Row Index	Col Index	Value
1	1	1
3	2	ϕ
4	5	π

(12)

3.1 Solving the Stationary Distribution

The relationship in (3) ⁶ is equivalent to the eigenvalue problem, where $\vec{p} = \lim_{i \rightarrow \infty} (\vec{p}_i)$ is the eigenvector ⁷ \vec{x} that corresponds to the eigenvalue $\xi = 1$:

$$\vec{p}(1) = \mathbf{S}\vec{p} \quad (13)$$

Solving eigenvectors for large matrices can be very resource intensive and so this approach isn't suitable for analysing large networks.

Upon iteration (3) will converge to stable stationary point, as discussed in 2.2.2, this approach is known as the power method [21] and is what in practice must be implemented to solve the stationary distribution of (6) and (3).

As mentioned in 2.2.2 and 2.3, the *Random Surfer* and *Power Walk* transition probability matrices are completely dense, that means applying the power method will not be able to take advantage of using sparse matrix algorithms.

With some effort however it is possible to express the algorithms in such a way that only involves sparse matrices.

4 Implementing the Models

To Implement the models, first they'll be implemented using an ordinary matrix and then improved to work with sparse matrices and algorithms, the implementation has been performed with **R** and the preamble is provided in listings 1

..

4.1 Implementing the Random Surfer

4.1.1 Small Graph, Ordinary Matrices

Example Graph Consider the following graph:

⁶This assumes that the transition probability matrix is stochastic and primitive as it would be for **S** and **W**

⁷More accurately the eigenvector specifically scaled specifically to 1, so it would be more correct to say the eigenvector $\sum \vec{x}$

```

1  if (require("pacman")) {
2      library(pacman)
3  }else{
4      install.packages("pacman")
5      library(pacman)
6  }
7
8  pacman::p_load(tidyverse, Matrix, igraph,
   ↪ plotly, mise, docstring, mise, corrplot,
   ↪ latex2exp)
9  # options(scipen=20) # Resist Scientific
   ↪ Notation

```

Listing 1: Implemented Packages used in this report

```

1  g1 <- igraph::graph.formula(
2      1++2, 1+-8, 1+-5,
3      2+-5, 2+-7, 2+-8, 2+-6, 2+-9,
4      3++4, 3+-5, 3+-6, 3+-9, 3+-10,
5      4+-9, 4+-10, 4+-5,
6      5+-8, 6+-8, 7+-8)
7  plot(g1)

```

Listing 2: Produce exemplar graph in figure 5

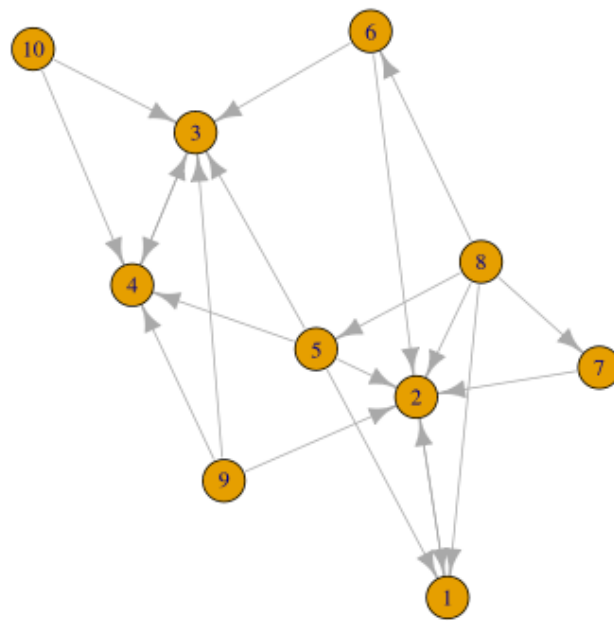


Figure 5: Exemplar graph for *PageRank* examples, produced in listing 2

Adjacency Matrix The adjacency Matrix is given by:

```
1 A <- igraph::get.adjacency(g1, names = TRUE,  
  ↪ sparse = FALSE)  
2  
3 ## igraph gives back the transpose  
4 (A <- t(A))
```

Listing 3: Return the Adjacency Matrix corresponding to figure 5

```
      1 2 8 5 7 6 9 3 4 10  
1 0 1 1 1 0 0 0 0 0  
2 1 0 1 1 1 1 1 0 0  
8 0 0 0 0 0 0 0 0 0  
5 0 0 1 0 0 0 0 0 0  
7 0 0 1 0 0 0 0 0 0  
6 0 0 1 0 0 0 0 0 0  
9 0 0 0 0 0 0 0 0 0  
3 0 0 0 1 0 1 1 0 1  
4 0 0 0 1 0 0 1 1 0  
10 0 0 0 0 0 0 0 0 0
```

```
      1 2 8 5 7 6 9 3 4 10  
1 0 1 1 1 0 0 0 0 0  
2 1 0 1 1 1 1 1 0 0  
8 0 0 0 0 0 0 0 0 0  
5 0 0 1 0 0 0 0 0 0  
7 0 0 1 0 0 0 0 0 0  
6 0 0 1 0 0 0 0 0 0  
9 0 0 0 0 0 0 0 0 0  
3 0 0 0 1 0 1 1 0 1  
4 0 0 0 1 0 0 1 1 0  
10 0 0 0 0 0 0 0 0 0
```

Probability Transition Matrix The probability transition matrix is such that each column of the initial state distribution (i.e. the transposed adjacency matrix) is scaled to 1.

if **A** had vertices with a 0 out-degree, the relationship in (1) would not work, instead columns that sum to 0 would need to be left while all other columns be divided by the column sum to get **T**. An alternative approach using sparse matrices will be presented below and in this case there exists corresponding **T** that is stochastic and so it is sufficient to use the relationship at (1), this is shown in listing 4.

```
1 (T <- A %*% diag(1/colSums(A)))
```

Listing 4: Solve the Transition Probability Matrix by scaling each column to 1 using matrix multiplication.

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
1	0	1	0.2	0.25	0	0.0	0.0000000	0	0	0.0
2	1	0	0.2	0.25	1	0.5	0.3333333	0	0	0.0
8	0	0	0.0	0.00	0	0.0	0.0000000	0	0	0.0
5	0	0	0.2	0.00	0	0.0	0.0000000	0	0	0.0
7	0	0	0.2	0.00	0	0.0	0.0000000	0	0	0.0
6	0	0	0.2	0.00	0	0.0	0.0000000	0	0	0.0
9	0	0	0.0	0.00	0	0.0	0.0000000	0	0	0.0
3	0	0	0.0	0.25	0	0.5	0.3333333	0	1	0.5
4	0	0	0.0	0.25	0	0.0	0.3333333	1	0	0.5
10	0	0	0.0	0.00	0	0.0	0.0000000	0	0	0.0

1. Create a Function

```

1 adj_to_probTrans <- function(A) {
2   A %*% diag(1/colSums(A))
3 }
4
5 (T <- adj_to_probTrans(A)) %>% round(2)

```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
1	0	1	0.2	0.25	0	0.0	0.00	0	0	0.0
2	1	0	0.2	0.25	1	0.5	0.33	0	0	0.0
8	0	0	0.0	0.00	0	0.0	0.00	0	0	0.0
5	0	0	0.2	0.00	0	0.0	0.00	0	0	0.0
7	0	0	0.2	0.00	0	0.0	0.00	0	0	0.0
6	0	0	0.2	0.00	0	0.0	0.00	0	0	0.0
9	0	0	0.0	0.00	0	0.0	0.00	0	0	0.0
3	0	0	0.0	0.25	0	0.5	0.33	0	1	0.5
4	0	0	0.0	0.25	0	0.0	0.33	1	0	0.5
10	0	0	0.0	0.00	0	0.0	0.00	0	0	0.0

##	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
## 1	0	1	0	0	0.25	0.0	0	0.2	0.00	0.0
## 2	1	0	0	0	0.25	0.5	1	0.2	0.33	0.0
## 3	0	0	0	1	0.25	0.5	0	0.0	0.33	0.5
## 4	0	0	1	0	0.25	0.0	0	0.0	0.33	0.5
## 5	0	0	0	0	0.00	0.0	0	0.2	0.00	0.0
## 6	0	0	0	0	0.00	0.0	0	0.2	0.00	0.0
## 7	0	0	0	0	0.00	0.0	0	0.2	0.00	0.0
## 8	0	0	0	0	0.00	0.0	0	0.0	0.00	0.0
## 9	0	0	0	0	0.00	0.0	0	0.0	0.00	0.0
## 10	0	0	0	0	0.00	0.0	0	0.0	0.00	0.0

Page Rank Random Surfer Recall from 2.2.2 the following variables of the *Random Surfer* model:

$$\mathbf{B} = \alpha \mathbf{T} + (1 - \alpha) \mathbf{B} : \quad (14)$$

$$(15)$$

$$\mathbf{B} = \begin{bmatrix} \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix} \quad (16)$$

$$n = ||V|| \quad (17)$$

$$\alpha \in [0, 1] \quad (18)$$

These are assigned to R variables in listing 5.

```
1 B <- matrix(rep(1/nrow(T), length.out =
  ↪ nrow(T)**2), nrow = nrow(T))
2 l <- 0.8123456789
3
4 (S <- l*T+(1-l)*B) %>% round(2)
```

Listing 5: Assign Random Surfer Variables, observe the unique value given to l, this will be relevant later.

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
1	0.02	0.83	0.18	0.22	0.02	0.02	0.02	0.02	0.02	0.02
2	0.83	0.02	0.18	0.22	0.83	0.42	0.29	0.02	0.02	0.02
8	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
5	0.02	0.02	0.18	0.02	0.02	0.02	0.02	0.02	0.02	0.02
7	0.02	0.02	0.18	0.02	0.02	0.02	0.02	0.02	0.02	0.02
6	0.02	0.02	0.18	0.02	0.02	0.02	0.02	0.02	0.02	0.02
9	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
3	0.02	0.02	0.02	0.22	0.02	0.42	0.29	0.02	0.83	0.42
4	0.02	0.02	0.02	0.22	0.02	0.02	0.29	0.83	0.02	0.42
10	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02

Eigen Value Method The eigenvector corresponding to the the eigenvalue of 1 will be the stationary point, this is shown in listing 6

```
1 print(eigen(S, symmetric = FALSE, only.values =
  ↪ TRUE)$values, 9)
2 print(eigen(S, symmetric = FALSE)$vectors, 3)
```

Listing 6: Solve the Eigen vectors and Eigen values of the transition probability matrix corresponding to the graph.


```

[1] 1.00000000e+00+0.00000000e+00i -8.12345679e-01+0.00000000e+00i
[3] 8.12345679e-01+0.00000000e+00i -8.12345679e-01+0.00000000e+00i
[5] 5.81488197e-10+0.00000000e+00i -5.81487610e-10+0.00000000e+00i
[7] -6.74980227e-16+0.00000000e+00i 3.21036747e-17+0.00000000e+00i
[9] 1.34928172e-18+1.1137323e-17i 1.34928172e-18-1.1137323e-17i
      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.4873+0i -7.07e-01+0i 5.00e-01+0i -2.07e-03+0i -6.74e-01+0i
[2,] 0.5268+0i 7.07e-01+0i 5.00e-01+0i 2.07e-03+0i -9.62e-02+0i
[3,] 0.0424+0i 9.09e-18+0i -3.50e-17+0i -5.05e-17+0i 1.38e-09+0i
[4,] 0.0493+0i -1.25e-18+0i -1.65e-16+0i 4.25e-17+0i 3.85e-01+0i
[5,] 0.0493+0i -8.30e-18+0i -3.75e-17+0i 3.71e-17+0i 3.85e-01+0i
[6,] 0.0493+0i -8.30e-18+0i -3.75e-17+0i 9.76e-18+0i 3.85e-01+0i
[7,] 0.0424+0i -1.32e-18+0i -3.50e-17+0i 1.60e-17+0i -3.01e-08+0i
[8,] 0.4915+0i -2.98e-03+0i -5.00e-01+0i -7.07e-01+0i -9.62e-02+0i
[9,] 0.4804+0i 2.98e-03+0i -5.00e-01+0i 7.07e-01+0i -2.89e-01+0i
[10,] 0.0424+0i 5.57e-18+0i -3.77e-17+0i 3.14e-18+0i -3.24e-08+0i
      [,6]      [,7]      [,8]      [,9]
[1,] 6.74e-01+0i 6.53e-01+0i -2.15e-01+0i -2.00e-01+1.53e-01i
[2,] 9.62e-02+0i 1.09e-01+0i -1.96e-01+0i -1.59e-01+0.00e+00i
[3,] 1.38e-09+0i 1.42e-15+0i -2.84e-16+0i -6.73e-17+1.32e-16i
[4,] -3.85e-01+0i -4.37e-01+0i 7.85e-01+0i 6.37e-01+0.00e+00i
[5,] -3.85e-01+0i -3.56e-01+0i 2.81e-01+0i 2.84e-02-1.63e-01i
[6,] -3.85e-01+0i -3.58e-01+0i -3.68e-01+0i 4.84e-02-2.68e-01i
[7,] -3.01e-08+0i -2.63e-02+0i -2.34e-01+0i -3.47e-02+4.29e-01i
[8,] 9.62e-02+0i 1.32e-01+0i -6.40e-02+0i -1.09e-01-2.84e-01i
[9,] 2.89e-01+0i 3.11e-01+0i 1.20e-01+0i -1.34e-01-1.50e-01i
[10,] -3.24e-08+0i -2.82e-02+0i -1.08e-01+0i -7.64e-02+2.83e-01i
      [,10]
[1,] -2.00e-01-1.53e-01i
[2,] -1.59e-01-0.00e+00i
[3,] -6.73e-17-1.32e-16i
[4,] 6.37e-01+0.00e+00i
[5,] 2.84e-02+1.63e-01i
[6,] 4.84e-02+2.68e-01i
[7,] -3.47e-02-4.29e-01i
[8,] -1.09e-01+2.84e-01i
[9,] -1.34e-01+1.50e-01i
[10,] -7.64e-02-2.83e-01i

```

So in this case the stationary point corresponds to the eigenvector given by:

$$\langle -0.49, -0.53, -0.49, -0.48, -0.05, -0.05, -0.05, -0.04, -0.04, -0.04 \rangle$$

this can be verified by using identity (13):

$$1\vec{p} = S\vec{p}$$

```

1 (p      <- eigen(S)$values[1] *
  ↪ eigen(S)$vectors[,1]) %>% Re() %>%
  ↪ round(2)

```

```
[1] 0.49 0.53 0.04 0.05 0.05 0.05 0.04 0.49 0.48 0.04
```

```

1 (p_new <- S %*% p) %>% Re() %>% as.vector()
  ↪ %>% round(2)

```

```
[1] 0.49 0.53 0.04 0.05 0.05 0.05 0.04 0.49 0.48 0.04
```

However this vector does not sum to 1 so the scale should be adjusted (for probabilities the vector should sum to 1):

```

1 (p_new <- p_new/sum(p_new)) %>% Re() %>%
  ↪ as.vector() %>% round(2)

```

```
[1] 0.22 0.23 0.02 0.02 0.02 0.02 0.02 0.22 0.21 0.02
```

Power Value Method Using the power method should give the same result as the eigenvalue method, again but for scale:

```

1 p_new <- p_new *123456789
2
3 while (sum(round(p, 9) != round(p_new, 9))) {
4     (p      <- p_new)
5     (p_new <- S %*% p)
6 }
7
8 round(Re(p_new), 2) %>% as.vector()

```

```
[1] 26602900 28759738 2316720 2693115 2693115 2693115 2316720 26834105
[9] 26230539 2316720
```

If scaled to 1 the same value will be returned:

```

1 (p_new <- p_new/sum(p_new)) %>% Re %>%
  ↪ as.vector() %>% round(2)

```

```
[1] 0.22 0.23 0.02 0.02 0.02 0.02 0.02 0.22 0.21 0.02
```

Scaling If the initial state sums to 1, then the scale of the stationary vector will also sum to 1, so this isn't in practice an issue for the power method:

```

1  p      <- c(1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
2  p_new <- S %*% p
3
4  while (sum(round(p, 9) != round(p_new, 9))) {
5      (p      <- p_new)
6      (p_new <- S %*% p)
7  }
8
9  cbind(p_new, p)

```

```

      [,1]      [,2]
1 0.21548349 0.21548349
2 0.23295388 0.23295388
8 0.01876543 0.01876543
5 0.02181424 0.02181424
7 0.02181424 0.02181424
6 0.02181424 0.02181424
9 0.01876543 0.01876543
3 0.21735625 0.21735625
4 0.21246737 0.21246737
10 0.01876543 0.01876543

```

```

##      [,1]      [,2]
## 1 0.2129185 0.2129185
## 2 0.2313481 0.2313481
## 3 0.2156444 0.2156444
## 4 0.2104889 0.2104889
## 5 0.0232000 0.0232000
## 6 0.0232000 0.0232000
## 7 0.0232000 0.0232000
## 8 0.0200000 0.0200000
## 9 0.0200000 0.0200000
## 10 0.0200000 0.0200000

```

4.1.2 Large Graph, Sparse Matrices using CRS

Creating the Probability Transition Matrix Implementing the page rank method on a larger graph requires the use of more efficient form of matrix storage as discussed at 3

A sparse matrix can be created using the following syntax, which will return a matrix of the class dgCMatrix:

```

1 library(Matrix)
2 ## Create Example Matrix
3 n <- 20
4 m <- 10^6
5 i <- sample(1:m, size = n); j <- sample(1:m,
  ↪ size = n); x <- rpois(n, lambda = 90)
6 A <- sparseMatrix(i, j, x = x, dims = c(m, m))
7
8 summary(A)

```

1000000 x 1000000 sparse Matrix of class "dgCMatrix", with 20 entries

	i	j	x
1	832961	14530	77
2	410264	57606	97
3	782033	111998	86
4	82383	176945	93
5	110039	239517	103
6	713327	249015	98
7	3377	387382	87
8	183673	466594	90
9	459326	509037	98
10	360156	554024	91
11	697837	573216	106
12	460554	582729	80
13	353957	654474	87
14	941579	683010	108
15	955791	763690	104
16	726278	790608	85
17	317527	867693	90
18	71267	949427	81
19	126551	992218	96
20	723320	992960	84

As before in section 4.1.1, the probability transition matrix can be found by:

1. Creating adjacency matrix

- (a) Transposing as necessary such that $A_{i,j} \neq 0$ indicates that j is connected to i by a directed edge.

2. Scaling the columns to one

To implement this for a sparseMatrix of the class dgCMatrix, the same technique of multiplying by a diagonalised matrix as in (2) may be implemented, using sparse matrices has the advantage however that only non-zero elements will be operated on, meaning that columns that some to zero can still be

used to create a probability transition matrix ⁸ practice an error however to create this new matrix, a new `sparseMatrix` will need to be created using the properties of the original matrix, this can be done like so:

```

1  sparse_diag <- function(mat) {
2
3    ## Get the Dimensions
4    n <- nrow(mat)
5
6    ## Make a Diagonal Matrix of Column Sums
7    D <- sparseMatrix(i = 1:n, j = 1:n, x =
8      ↪ colSums(mat), dims = c(n,n))
9
10   ## Throw away explicit Zeroes
11   D <- drop0(D)
12
13   ## Inverse the Values
14   D@x <- 1/D@x
15
16   ## Return the Diagonal Matrix
17   return(D)
18 }
```

Applying this to the previously created sparse matrix:

```

1  D <- sparse_diag(t(A))
2  summary(D)
```

1000000 x 1000000 sparse Matrix of class "dgCMatrix", with 20 entries

	i	j	x
1	3377	3377	0.011494253
2	71267	71267	0.012345679
3	82383	82383	0.010752688
4	110039	110039	0.009708738
5	126551	126551	0.010416667
6	183673	183673	0.011111111
7	317527	317527	0.011111111
8	353957	353957	0.011494253
9	360156	360156	0.010989011
10	410264	410264	0.010309278
11	459326	459326	0.010204082
12	460554	460554	0.012500000

⁸ Although this matrix may still have columns that sum to zero and will hence be non-stochastic

```

13 697837 697837 0.009433962
14 713327 713327 0.010204082
15 723320 723320 0.011904762
16 726278 726278 0.011764706
17 782033 782033 0.011627907
18 832961 832961 0.012987013
19 941579 941579 0.009259259
20 955791 955791 0.009615385

```

and hence the probability transition matrix may be implemented by performing matrix multiplication accordingly:

```
summary((T <- t(A) %*% D))
```

```
1000000 x 1000000 sparse Matrix of class "dgCMatrix", with 20 entries
```

```

      i      j x
1  387382  3377 1
2  949427  71267 1
3  176945  82383 1
4  239517 110039 1
5  992218 126551 1
6  466594 183673 1
7  867693 317527 1
8  654474 353957 1
9  554024 360156 1
10 57606 410264 1
11 509037 459326 1
12 582729 460554 1
13 573216 697837 1
14 249015 713327 1
15 992960 723320 1
16 790608 726278 1
17 111998 782033 1
18 14530 832961 1
19 683010 941579 1
20 763690 955791 1

```

Solving the Random Surfer via the Power Method Solving the eigenvalues for such a large matrix will not be feasible, instead the power method will need to be used to find the stationary point.

However, creating a matrix of background probabilities (denoted by B in section 4.1.1) will not be feasible, it would simply be too large, instead some algebra can be used to reduce B from a matrix into a vector containing only $\frac{1-\alpha}{N}$.

The power method is given by:

$$\vec{p} = S\vec{p} \quad (19)$$

where:

$$S = \alpha \mathbf{T} + (1 - \alpha) \mathbf{B} \quad (20)$$

$$\vec{p} = (\alpha \mathbf{T} + (1 - \alpha) \mathbf{B}) \vec{p} \quad (21)$$

$$= \alpha \mathbf{T} \vec{p} + (1 - \alpha) \mathbf{B} \vec{p} \quad (22)$$

Let $\mathbf{F} = \mathbf{B} \vec{p}$, consider the value of \mathbf{F} :

$$\mathbf{F} = \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix} \begin{bmatrix} \vec{p}_1 \\ \vec{p}_2 \\ \vdots \\ \vec{p}_m \end{bmatrix} \quad (23)$$

$$= \begin{bmatrix} (\sum_{i=0}^m [p_i]) \times \frac{1}{N} \\ (\sum_{i=0}^m [p_i]) \times \frac{1}{N} \\ \vdots \\ (\sum_{i=0}^m [p_i]) \times \frac{1}{N} \end{bmatrix} \quad (24)$$

Probabilities sum to 1 and hence: (25)

$$= \begin{bmatrix} \frac{1}{N} \\ \frac{1}{N} \\ \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{bmatrix} \quad (26)$$

So instead the power method can be implemented by performing an algorithm that involves only sparse matrices:

```

1  ## Find Stationary point of random surfer
2  N      <- nrow(A)
3  alpha <- 0.85
4  F      <- rep((1-alpha)/N, nrow(A))  ## A nx1
   ↪  vector of (1-alpha)/N
5
6  ## Solve using the power method
7  p      <- rep(0, length.out = ncol(T)); p[1] <- 1
8  p_new <- alpha*T %*% p + F
9
10 ## use a Counter to debug
11 i <- 0
12 while (sum(round(p, 9) != round(p_new, 9))) {
13     p      <- p_new
14     p_new <- alpha*T %*% p + F
15     (i <- i+1) %>% print()
16 }
17
18 p %>% head() %>% print()
```

```

[1] 1
[1] 2
6 x 1 Matrix of class "dgeMatrix"
      [,1]
[1,] 1.5e-07
[2,] 1.5e-07
[3,] 1.5e-07
[4,] 1.5e-07
[5,] 1.5e-07
[6,] 1.5e-07

```

4.2 Power Walk Method

4.2.1 Introduction

Recall from 2.3 that the power walk is given by:

$$\mathbf{T} = \mathbf{B}\mathbf{D}_B^{-1}$$

where:

- $\mathbf{B} = \beta^{\mathbf{A}}$
 - $x\beta^1$ probability of following an edge of weight 1
 - $x\beta^0$ probability of following an edge of weight 0
 - $x\beta^{-1}$ probability of following an edge of weight -
- $\mathbf{D}_B = \text{colsums}(\mathbf{B})$

\mathbf{A} The Adjacency Matrix

4.2.2 Ordinary Matrices

Implementing the Power walk using ordinary matrices is very similar to the *Random Surfer* model be done pretty much the same as it is with the random surfer, but doing it with Sparse Matrices is a bit trickier.

Create the Adjacency Matrix


```

1  A <- igraph::get.adjacency(g1, names = TRUE,
  ↪ sparse = FALSE)
2
3  ## * Function to create Prob Trans Mat
4  adj_to_probTrans <- function(A, beta) {
5      B      <- A
6      B      <- beta^A          # Element Wise
  ↪      exponentiation
7      D      <- diag(colSums(B)) # B is completely
  ↪      dense so D  0
8      D_in   <- solve(D)        # Solve returns
  ↪      inverse of matrix
9      W      <- B %*% D_in
10
11     return(as.matrix(W))
12 }
13
14 beta <- 0.867
15 (W <- adj_to_probTrans(A, beta = )) %>% round(2)

```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
1	0.10	0.09	0.1	0.10	0.10	0.10	0.1	0.11	0.11	0.1
2	0.09	0.11	0.1	0.10	0.10	0.10	0.1	0.11	0.11	0.1
8	0.09	0.09	0.1	0.09	0.09	0.09	0.1	0.11	0.11	0.1
5	0.09	0.09	0.1	0.10	0.10	0.10	0.1	0.09	0.09	0.1
7	0.10	0.09	0.1	0.10	0.10	0.10	0.1	0.11	0.11	0.1
6	0.10	0.09	0.1	0.10	0.10	0.10	0.1	0.09	0.11	0.1
9	0.10	0.09	0.1	0.10	0.10	0.10	0.1	0.09	0.09	0.1
3	0.10	0.11	0.1	0.10	0.10	0.10	0.1	0.11	0.09	0.1
4	0.10	0.11	0.1	0.10	0.10	0.10	0.1	0.09	0.11	0.1
10	0.10	0.11	0.1	0.10	0.10	0.10	0.1	0.09	0.09	0.1

Look at the Eigenvalues:

```

1  eigen(W, only.values = TRUE)$values %>% round(9)
2  eigen(W)$vectors/sum(eigen(W)$vectors)

```

[1]	1.000000000+0.000000000i	0.014269902+0.000000000i		
[3]	-0.014148391+0.000000000i	0.014147087+0.000000000i		
[5]	0.007672842+0.004095136i	0.007672842-0.004095136i		
[7]	0.000000000+0.000000000i	0.000000000+0.000000000i		
[9]	0.000000000+0.000000000i	0.000000000+0.000000000i		
	[,1]	[,2]	[,3]	[,4]
[1,]	0.10153165+0i	5.107247e-02+0i	0.073531664+0i	0.009918277+0i
[2,]	0.10159353+0i	-1.161249e-01+0i	0.071987451+0i	-0.009531974+0i

```

[3,] 0.09609664+0i -2.162636e-01+0i 0.198568750+0i 0.141245296+0i
[4,] 0.09725145+0i 6.794340e-02+0i -0.012230606+0i -0.001148014+0i
[5,] 0.10153165+0i 5.107247e-02+0i 0.073531664+0i 0.009918277+0i
[6,] 0.10008449+0i 1.115133e-01+0i -0.005625969+0i -0.156796770+0i
[7,] 0.09865794+0i 1.175228e-01+0i -0.084225633+0i 0.008563891+0i
[8,] 0.10157348+0i -6.053608e-02+0i -0.078607240+0i 0.165540590+0i
[9,] 0.10155286+0i -6.104664e-03+0i -0.079165209+0i -0.166535117+0i
[10,] 0.10012631+0i -9.522175e-05+0i -0.157764873+0i -0.001174456+0i
      [,5]      [,6]      [,7]
[1,] 0.00633946+0.04208220i 0.00633946-0.04208220i 3.014602e-16+0i
[2,] 0.00757768+0.03910216i 0.00757768-0.03910216i 1.909248e-16+0i
[3,] 0.22697603+0.00000000i 0.22697603+0.00000000i 3.985744e-02+0i
[4,] -0.11628681-0.11808928i -0.11628681+0.11808928i -2.471407e-01+0i
[5,] 0.00633946+0.04208220i 0.00633946-0.04208220i 7.520823e-02+0i
[6,] -0.03494625-0.01031801i -0.03494625+0.01031801i 1.719325e-01+0i
[7,] -0.07581902-0.06371153i -0.07581902+0.06371153i 6.131013e-03+0i
[8,] 0.00717270+0.04008639i 0.00717270-0.04008639i 5.526770e-17+0i
[9,] 0.00675977+0.04107970i 0.00675977-0.04107970i 1.105354e-16+0i
[10,] -0.03411300-0.01231382i -0.03411300+0.01231382i -4.598845e-02+0i
      [,8]      [,9]      [,10]
[1,] -1.791605e-17+0i -4.365749e-17+0i 1.179767e-17+0i
[2,] -7.334385e-17+0i -8.731498e-17+0i -5.190977e-17+0i
[3,] -1.241234e-01+0i -1.401965e-01+0i -8.894098e-02+0i
[4,] 1.691000e-01+0i 1.687523e-01+0i 1.041947e-01+0i
[5,] -2.144546e-01+0i 2.715852e-02+0i 3.085359e-02+0i
[6,] 4.535455e-02+0i -1.959109e-01+0i -1.350483e-01+0i
[7,] 7.398187e-02+0i 3.163948e-02+0i -1.260060e-01+0i
[8,] 8.062225e-17+0i 3.638124e-17+0i 5.898837e-18+0i
[9,] 2.687408e-17+0i 3.638124e-17+0i 5.662884e-17+0i
[10,] 5.014155e-02+0i 1.085570e-01+0i 2.149470e-01+0i

```

Unlike the *Random Surfer* Model in listing 6 at 4.1.1 the relationship between the second eigenvalue and the model parameters is not as clear, this provides that the

Use the power method

```

1  ## * Power Method
2  p      <- rep(0, nrow(W))
3  p[1] <- 1
4  p_new   <- rep(0, nrow(W))
5  p_new[2] <- 1
6
7  while (sum(round(p, 9) != round(p_new, 9))) {
8      (p      <- p_new)
9      (p_new <- W %*% p)
10 }
11
12
13 p %>% as.vector()

```

```

[1] 0.10153165 0.10159353 0.09609664 0.09725145 0.10153165 0.10008449
[7] 0.09865794 0.10157348 0.10155286 0.10012631

```

4.2.3 Sparse Matrices

Theory; Simplifying Power Walk to be solved with Sparse Matrices The Random Surfer model is:

$$\mathbf{S} = \alpha \mathbf{T} + \mathbf{F}$$

where:

- \mathbf{T}

- is an $i \times j$ matrix that describes the probability of travelling from vertex j to i
 - * This is transpose from the way that igraph produces an adjacency matrix.

- $\mathbf{F} = \begin{bmatrix} \frac{1}{n} \\ \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{bmatrix}$

Interpreting the transition probability matrix in this way is such that $\mathbf{T} = \mathbf{A} \mathbf{D}_A^{-1}$ under the following conditions:

- No column of \mathbf{A} sums to zero
 - If this does happen the question arises how to deal with \mathbf{D}_A^{-1}
 - * I've been doing $\mathbf{D}_{\mathbf{A},i,j}^T := \text{diag}\left(\frac{1}{\text{colsums}(\mathbf{A})}\right)$ and then replacing any 0 on the diagonal with 1.
 - What is done in the paper is to make another matrix \mathbf{Z} that is filled with 0, if a column sum of \mathbf{A} adds to zero then that column in \mathbf{Z} becomes $\frac{1}{n}$
 - * This has the effect of making each row identical
 - * The probability of going from an orphaned vertex to any other vertex would hence be $\frac{1}{n}$

- * The idea with this method is then to use $D_{(\mathbf{A}+\mathbf{Z})}^{-1}$ this will be consistent with the *Random Surfer* the method using \mathbf{F} in $[[\#eq:sparse-RS]]$ (4.2.3)

where each row is identical that is a 0

The way to deal with the *Power Walk* is more or less the same.
observe that:

$$\left(\mathbf{B} = \beta \mathbf{A} \right) \wedge (\mathbf{A}_{i,j}) \in \mathbb{R} \implies |\mathbf{B}_{i,j}| > 0 \quad \forall i, j > n \in \mathbb{Z}^+ \quad (27)$$

Be mindful that the use of exponentiation in (27) is not an element wise exponentiation and not an actual matrix exponential.

So if I have:

- $\mathbf{O}_{i,j} := 0, \quad \forall i, j \leq n \in \mathbb{Z}^+$
- \vec{p}_i as the state distribution, being a vector of length n

Then It can be shown (see (4.2.3) at 4.2.3):

$$\mathbf{O} \mathbf{D}_{\mathbf{B}}^{-1} \vec{p}_i = (\vec{\delta}^T \vec{p}_i) \vec{1} \quad (28)$$

$$= \text{repeat} \left(\vec{p} \bullet \vec{\delta}^T, n \right) \quad (29)$$

$$(30)$$

where:

- $\vec{\delta}_i = \frac{1}{\text{colsums}(\mathbf{B})}$
– A vector...(n × 1 matrix)

$\vec{1}$ is a vector containing all 1's

- A vector...(n × 1 matrix)

$\vec{\delta}^T$ refers to the transpoxe of $\vec{\delta}$ (1 × n matrix)

$\vec{\delta}^T \vec{p}_i$ is some number (because it's a dot product)

This means we can do:

$$\vec{p}_{i+1} = \mathbf{T}_{\text{pw}} \vec{p}_i \quad (31)$$

$$= \mathbf{B} \mathbf{D}_{\mathbf{B}}^{-1} \vec{p}_i \quad (32)$$

$$= (\mathbf{B} - \mathbf{O} + \mathbf{O}) \mathbf{D}_{\mathbf{B}}^{-1} \vec{p}_i \quad (33)$$

$$= \left((\mathbf{B} - \mathbf{O}) \mathbf{D}_{\mathbf{B}}^{-1} + \mathbf{O} \mathbf{D}_{\mathbf{B}}^{-1} \right) \vec{p}_i \quad (34)$$

$$= (\mathbf{B} - \mathbf{O}) \mathbf{D}_{\mathbf{B}}^{-1} \vec{p}_i + \mathbf{O} \mathbf{D}_{\mathbf{B}}^{-1} \vec{p}_i \quad (35)$$

$$= (\mathbf{B} - \mathbf{O}) \mathbf{D}_{\mathbf{B}}^{-1} \vec{p}_i + \vec{1} (\vec{\delta}^T \vec{p}_i) \quad (36)$$

$$= (\mathbf{B} - \mathbf{O}) \mathbf{D}_{\mathbf{B}}^{-1} \vec{p}_i + \text{rep}(\vec{\delta}^T \vec{p}_i) \quad (37)$$

where:

Let $(\mathbf{B} - \mathbf{O}) = \mathbf{B}_O$:

$$\vec{p_{i+1}} = \mathbf{B}_O \mathbf{D}_B^{-1} \vec{p_i} + \text{rep}(\vec{\delta^T p_i})$$

Now solve D_B^{-1} in terms of \mathbf{B}_O :

$$\mathbf{B}_O = (\mathbf{B} - \mathbf{O}) \quad (38)$$

$$\mathbf{B} = \mathbf{B}_O + \mathbf{O} \quad (39)$$

If we have δ_B as the column sums of \mathbf{B} :

$$\delta_B^{-1} = \vec{1} \mathbf{B} \quad (40)$$

$$= \vec{1} (\mathbf{B}_O + \mathbf{O}) \quad (41)$$

$$= \vec{1} \mathbf{B}_O + \vec{1} \mathbf{O} \quad (42)$$

$$= \vec{1} \mathbf{B}_O + \langle n, n, n, \dots, n \rangle \quad (43)$$

$$= \vec{1} \mathbf{B}_O + \vec{1} n \quad (44)$$

$$\delta_B = 1 / (\text{colSums}(\mathbf{B}_O) + n) \quad (45)$$

Then if we have $D_B = \text{diag}(\delta_B)$:

$$\begin{aligned} D_B^{-1} &= \text{diag}(\delta_B^{-1}) \\ &= \text{diag}(\text{ColSums}(\mathbf{B}_O) + n)^{-1} \end{aligned}$$

And so the the power method can be implemented using sparse matrices:

$$p_{i+1}^{\vec{}} = \mathbf{B}_O \text{diag}(\vec{1} \mathbf{B}_O + \vec{1} n) \vec{p_i} + \vec{1} \delta^T \vec{p_i} \quad (46)$$

in terms of \mathbf{R} :

```

1  p_new <- Bo %*% diag(colSums(B)+n) %*% p +
   ↪ rep(t( ) %*% p, n)
2
3  # It would also be possible to sum the
   ↪ element-wise product
4  (t( ) %*% p) == sum( * p)
5
6  # Because R treats vectors the same as a nX1
   ↪ matrix we could also
7  # perform the dot product of the two vectors,
   ↪ meaning the following
8  # would be true in R but not true generally
9
10 (t( ) %*% p) == ( %*% p)
```

Solving the Background Probability In this case a vertical single column matrix will represent a vector and \otimes will represent the outer product (i.e. the *Kronecker Product*):

Define $\vec{\delta}$ as the column sums of

$$\begin{aligned}\vec{\delta} &= \text{colsum}(\mathbf{B})^{-1} \\ &= \frac{1}{\vec{1}^T \mathbf{B}}\end{aligned}$$

Then we have:

$$\begin{aligned}\mathbf{OD}_{\mathbf{B}}^{-1} \vec{p}_i &= \begin{pmatrix} 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \dots \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} \frac{1}{\delta_1} & 0 & 0 & \dots \\ 0 & \frac{1}{\delta_2} & 0 & \dots \\ 0 & 0 & \frac{1}{\delta_{13}} & \dots \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} p_{i,1} \\ p_{i,2} \\ p_{i,3} \\ \vdots \end{pmatrix} \\ &= \begin{pmatrix} \frac{p_{i,1}}{\delta_1} + \frac{p_{i,2}}{\delta_2} + \frac{p_{i,3}}{\delta_3} & \dots \\ \frac{p_{i,1}}{\delta_1} + \frac{p_{i,2}}{\delta_2} + \frac{p_{i,3}}{\delta_3} & \dots \\ \frac{p_{i,1}}{\delta_1} + \frac{p_{i,2}}{\delta_2} + \frac{p_{i,3}}{\delta_3} & \dots \\ \vdots & \ddots \end{pmatrix} \\ &= \begin{pmatrix} \sum_{k=1}^n [p_{i,k} \delta_i] \\ \sum_{k=1}^n [p_{i,k} \delta_i] \\ \sum_{k=1}^n [p_{i,k} \delta_i] \\ \vdots \end{pmatrix} \\ &= \begin{pmatrix} \vec{\delta}^T \vec{p}_i \\ \vec{\delta}^T \vec{p}_i \\ \vec{\delta}^T \vec{p}_i \\ \vdots \end{pmatrix} \\ &= \vec{\delta}^T \vec{p}_i \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{pmatrix} \\ &= (\vec{\delta}^T \vec{p}_i) \vec{1} \\ &= \text{repeat}(\vec{\delta}^T \vec{p}_i, n)\end{aligned}$$

Observe also that If we let $\vec{\delta}$ and p_i be 1 dimensional vectors, this can also be expressed as a dot product:

Matrices	Vectors
$\vec{\delta}^T \vec{p}_i$	$\vec{\delta}^T \vec{p}_i$

Practical; Implementing the Power Walk on Sparse Matrices

Inspect the newly created matrix and create constants

Setup

1. Define function to create DiagonalsSparse Diagonal Function

Unlike the Random Surfer model the diagonal scaling matrix will always be given by $\mathbf{D}_B^{-1} = \mathbf{B} \text{ diag} \left(\frac{1}{\mathbf{1}\mathbf{B}} \right)$ because $\beta^{\mathbf{A}_{i,j}} \neq 0 \quad \forall \mathbf{A}_{i,j}$, this is convenient but in any case the `sparse_diag` function in listing ?? will still work.

Power Walk

1. Define B

```

1  A      <- Matrix::Matrix(A, sparse = TRUE)
2  B      <- A
3  B@x    <- ~(A@x)
4  B      <- A
5  B      <- ^A
6
7  Bo     <- A
8
9  # These two approaches are equivalent
10 Bo@x   <- ~(A@x) -1 # This in theory
    ↪ would be faster
11 # Bo     <- ~(A) -1
12 # Bo     <- drop0(Bo)
13
14
15  n <- nrow(A)
```

```

10 x 10 sparse Matrix of class "dgCMatrix"
[[ suppressing 10 column names '1', '2', '8' ... ]]
```

```

1  . 1 . . . . . . . .
2  1 . . . . . . . .
8  1 1 . 1 1 1 . . . .
5  1 1 . . . . . 1 1 .
7  . 1 . . . . . . . .
6  . 1 . . . . . 1 . .
9  . 1 . . . . . 1 1 .
3  . . . . . . . 1 .
4  . . . . . . . 1 . .
10 . . . . . . . 1 1 .
```

```

1  print(round(B, 2))
```

```

10 x 10 Matrix of class "dgeMatrix"
  1    2 8    5    7    6 9    3    4 10
```

```

1  1.00 0.87 1 1.00 1.00 1.00 1 1.00 1.00 1
2  0.87 1.00 1 1.00 1.00 1.00 1 1.00 1.00 1
8  0.87 0.87 1 0.87 0.87 0.87 1 1.00 1.00 1
5  0.87 0.87 1 1.00 1.00 1.00 1 0.87 0.87 1
7  1.00 0.87 1 1.00 1.00 1.00 1 1.00 1.00 1
6  1.00 0.87 1 1.00 1.00 1.00 1 0.87 1.00 1
9  1.00 0.87 1 1.00 1.00 1.00 1 0.87 0.87 1
3  1.00 1.00 1 1.00 1.00 1.00 1 1.00 0.87 1
4  1.00 1.00 1 1.00 1.00 1.00 1 0.87 1.00 1
10 1.00 1.00 1 1.00 1.00 1.00 1 0.87 0.87 1

```

```
1 print(Bo,2)
```

```

10 x 10 sparse Matrix of class "dgCMatrix"
[[ suppressing 10 column names '1', '2', '8' ... ]]

```

```

1  .      -0.13 . .      .      .      .      .      .
2 -0.13 .      . .      .      .      .      .      .
8 -0.13 -0.13 . -0.13 -0.13 -0.13 . .      .      .
5 -0.13 -0.13 . .      .      .      . -0.13 -0.13 .
7  .      -0.13 . .      .      .      .      .      .
6  .      -0.13 . .      .      .      . -0.13 .      .
9  .      -0.13 . .      .      .      . -0.13 -0.13 .
3  .      .      . .      .      .      .      . -0.13 .
4  .      .      . .      .      .      . -0.13 .      .
10 .      .      . .      .      .      . -0.13 -0.13 .

```

2. Solve the Scaling Matrix We don't need to worry about any terms of $\delta_B = \text{colsums}(B_o) + n$ being 0:

```
1 (B <- 1/(colSums(Bo)+n))
```

```

      1      2      8      5      7      6      9      3
0.1041558 0.1086720 0.1000000 0.1013479 0.1013479 0.1013479 0.1000000 0.1071237
      4      10
0.1056189 0.1000000

```

```
1 (B <- 1/(colSums(B)))
```

```

      1      2      8      5      7      6      9      3
0.1041558 0.1086720 0.1000000 0.1013479 0.1013479 0.1013479 0.1000000 0.1071237
      4      10
0.1056189 0.1000000

```


3. Find the Transition Probability Matrix

```
1 DB <- diag( B)
2 ## ** Create the Transition Probability
  ↳ Matrix
3 ## Create the Trans Prob Mat using Power
  ↳ Walk
4 T <- Bo %*% DB
```

4. Implement the Loop

```
1 ## ** Implement the Power Walk
2 ## *** Set Initial Values
3 p_new <- rep(1/n, n) # Uniform
4 p <- rep(0, n) # Zero
5 <- 10^(-6)
6 ## *** Implement the Loop
7
8 while (sum(abs(p_new - p)) > ) {
9   (p <- as.vector(p_new)) # P should
  ↳ remain a vector
10  sum(p <- as.vector(p_new)) # P should
  ↳ remain a vector
11  p_new <- T %*% p + rep(t( B) %*% p, n)
12 }
13 ## ** Report the Values
14 print(paste("The stationary point is"))
15 print(p)
```

```
[1] "The stationary point is"
[1] 0.10153165 0.10159353 0.09609664 0.09725146 0.10153165 0.10008449
[7] 0.09865795 0.10157347 0.10155286 0.10012631
```

5 Creating a Package

In order to investigate the effect of the model parameters on the second Eigenvalue it will be necessary to use these functions, in order to document and work with them in a modular way they were placed into an **R** package and made available on *GitHub* [fn: <https://github.com/RyanGreenup/PageRank>], to load this package use the devtools library as shown in listing .

```
Loading required package: usethis
Loading required package: PageRank
```

```
Attaching package: 'PageRank'
```

```

1 library(devtools)
2 library(Matrix)
3 library(tidyverse) # Maybe, TODO check if this
  ↳ is used, I don't think it is
4
5 if (require("PageRank")) {
6   library(PageRank)
7 }else{
8   devtools::install_github("ryangreenup/PageRank")
  ↳ Rank")
9   library(PageRank)
10  }

```

Listing 8: Load the *PageRank* package which consists of the functions from 4

6 Investigating the Second EigenValue

6.1 ER Graphs Plotting Various Values

6.1.1 Erdos Reny Game

The *Erdos Renyi* game creates a graph by creating every single edge with some constant probability. [15, IgraphManualPagesa] Sampling these graphs will provide a broad picture for the overall behaviour between the method parameters of the *Power Walk* and the second eigenvalue.

6.1.2 Correlation Plot

By looping over many random graphs for a variety of probabilities a data set can be constructed and a correlation plot generated. To implement this a data frame of input values was constructed in listing 9, a function that builds a data frame with the second eigenvalue, density, determinant and trace was constructed in listing ?? and finally a correlation plot was generated in listing 10 shown in figure .

```

1 # Generate Constants
2 n      <- 20
3 p      <- 1:n/n
4 beta   <- 1:n/n
5 beta   <- runif(n)*100
6 #sz     <- 1:n/n+10
7 sz     <- (1:n/n)*100+10
8 input_var <- expand.grid("n" = n, "p" = p,
  ↳ "beta" = beta, "size" = sz)
9
10 # Print out a sample of all the rows
11 input_var[sample(1:nrow(input_var), 6),]

```

Listing 9: A data frame consisting of input variables to be used to generate *Erdos Renyi* graphs.

	n	p	beta	size
7154	20	0.70	0.5872766	100
4103	20	0.15	82.8545866	65
133	20	0.65	64.6700020	15
3887	20	0.35	76.8201442	60
1725	20	0.25	64.6700020	35
6071	20	0.55	26.0913073	90

$\text{mean}(\mathbf{A}), |\mathbf{A}|, \text{tr}(\mathbf{A})$) corresponding to the *Power Walk* method using the PageRank package discussed at 5.

```

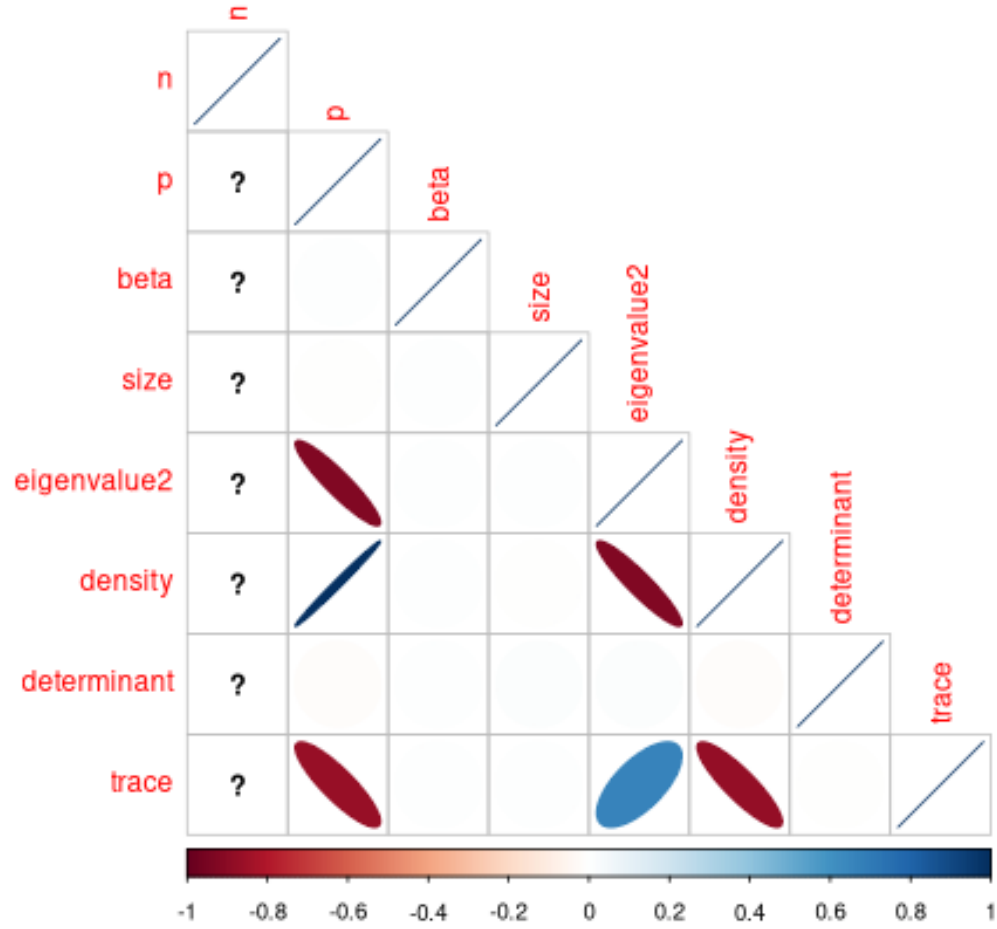
1  random_graph <- function(n, p, beta, size) {
2      g1 <- igraph::erdos.renyi.game(n = sz, p)
3      A <- igraph::get.adjacency(g1) # Row to
      ↪ column
4      A <- Matrix::t(A)
5
6      A_dens <- mean(A)
7      T      <-
      ↪ PageRank::power_walk_prob_trans(A)
8      T_tr   <- sum(diag(T))
9      e2     <- eigen(T, only.values =
      ↪ TRUE)$values[2] # R orders by
      ↪ descending magnitude
10     A_det  <- det(A)
11     T_det  <- det(T)
12     return(c(abs(e2), A_dens, T_det, T_tr)) #
      ↪ A_det and T_tr are uncorrelated
13 }
```

```

1 filename <- "erdosData.rds"
2
3 if (file.exists(filename)) {
4
5     data <- readRDS(filename)
6
7 } else {
8
9     # Loop over the data
10    nc <- length(random_graph(1, 1, 1, 1))
11    Y <- matrix(ncol = nc, nrow = nrow(input_var))
12
13    for (i in 1:nrow(input_var)) {
14        X <- as.vector(input_var[i,])
15        Y[i,] <- random_graph(X$n, X$p, X$beta,
16                               ↪ X$size)
17    }
18
19    ## Remove the Oi component
20    if (sum(abs(Y) != abs(Re(Y))) == 0) {
21        Y <- Re(Y)
22    }
23
24    ## Clean up the data frame
25    Y <- as.data.frame(Y); colnames(Y) <-
26        ↪ c("eigenvalue2", "density", "determinant",
27           ↪ "trace")
28    data <- cbind(input_var, Y)
29    data <- data[data$density!=0,]
30
31    ## Save the data
32    saveRDS(data, filename)
33
34    corplot(cor(data), method = "ellipse", type =
35            ↪ "lower")

```

Listing 10: Produce a correlation plot Created from a dataframe constructed from the values assigned in listing 9 by using the function defined in listing ??, see figure .



6.1.3 Density of Adjacency Matrix

There appears to be a strong negative correlation between the eigenvalue and the density of the adjacency matrix.

This relationship is plotted in listing 11 and figure 6.

The relationship appears almost linear and so the data is log transformed and modelled against that in listing 12 with a corresponding plot generated in listing 13 and shown in figure 13 revealing a concave down relationship. The quartic model fits the data well and has the lowest MSE , however the logarithmic model is visually a good fit, significantly simpler and still has a low MSE , for this reason the logarithmic model will be used.

The coefficients of the logarithmic model, shown in listing 12, imply the following relationship:

$$\xi_2 = \left(1 - \frac{\sum_{i=1}^n \sum_{j=1}^n \mathbf{A}_{i,j}}{n^2} \right)^{0.6} \cdot e^{-0.48} \pm \Delta \quad (47)$$

The maximum residual is given as 0.37 and so $\Delta = 0.4$ would provide a good indication for the value of the second eigenvalue by considering only the interconnectivity of the adjacency matrix.

This suggests that a more interlinked network will converge faster when using the *Power Walk* method.

```

1  ggplot(data) +
2  geom_point(mapping = aes(x = density, y =
   ↪ eigenvalue2, size = beta, color = size )) +
3  scale_size_continuous(range = c(0.1,1)) +
4  labs(x = "Density of Adjacency Matrix", y =
   ↪ TeX("$\\xi_2$ of $T_{PW}$"), title =
   ↪ TeX("$\\xi_2$ of $T_{PW}$ given the
   ↪ Density of the Adjacency Matrix") ) +
5  guides(size = FALSE, col = FALSE)

```

Listing 11: Create a Plot of ξ_2 given Adjacency Matrix, see plot in figure 6

MSE Linear	MSE Quadratic	MSE Quartic	MSE Logarithmic
0.078	0.031	0.013	0.023
(Intercept) log(1 - density)			
-0.4798300	0.6738175		
[1] "Max Residual" "0.371075096908937"			

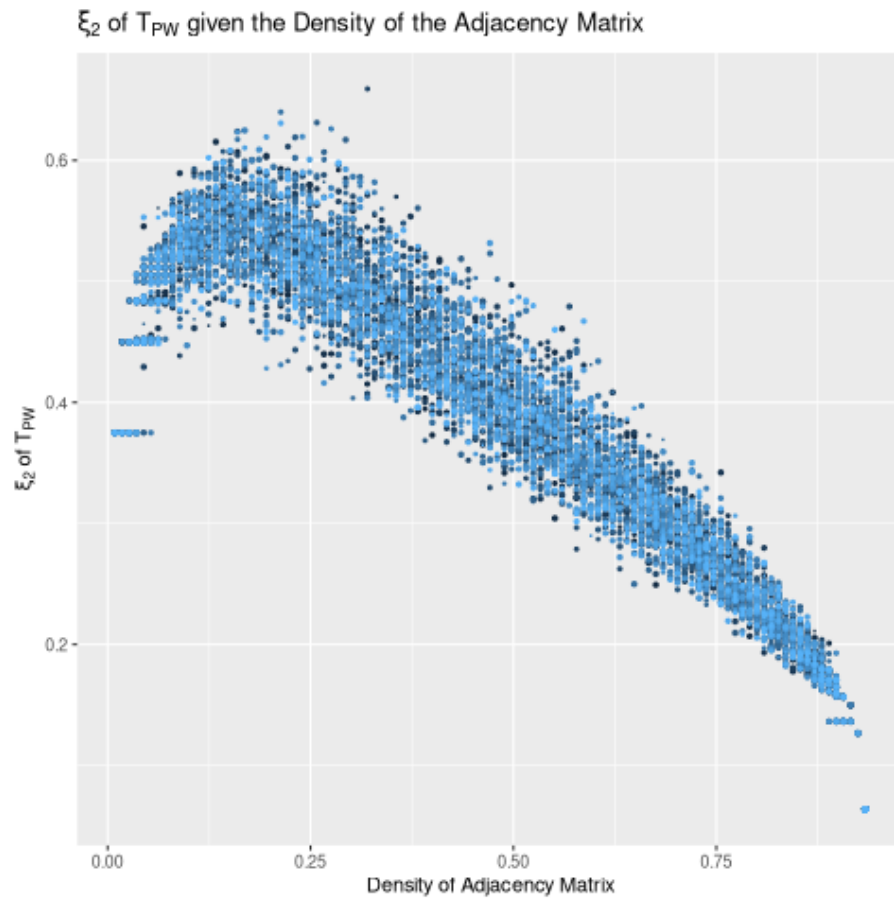


Figure 6: Plot of ξ_2 given Adjacency Matrix, see listing 11

```

1  mod_x1 <- lm(log(eigenvalue2) ~ poly(density,
   ↪ 1), data = data)
2  data$x1 <- predict(mod_x1)
3
4  mod_x2 <- lm(log(eigenvalue2) ~ poly(density,
   ↪ 2), data = data)
5  data$x2 <- predict(mod_x2)
6
7  mod_x4 <- lm(log(eigenvalue2) ~ poly(density,
   ↪ 4), data = data)
8  data$x4 <- predict(mod_x4)
9
10 mod_xl <- lm(log(eigenvalue2) ~ log(1-density),
   ↪ data = data)
11 data$xl <- predict(mod_xl)
12
13 mod_df <- data
14 mod_df_long <- pivot_longer(mod_df, cols = c(x1,
   ↪ x2, x4, xl), names_to = "Model_Type",
   ↪ values_to = "eigenvalue2_mod")
15 mod_df_long$eigenvalue2_log <-
   ↪ log(mod_df_long$eigenvalue2)
16
17
18 print(c("MSE Linear" = mean(mod_x1$residuals^2),
19       "MSE Quadratic" =
   ↪ mean(mod_x2$residuals^2),
20       "MSE Quartic" = mean(mod_x4$residuals^2),
21       "MSE Logarithmic" =
   ↪ mean(mod_xl$residuals^2)
22       ), 2)
23 cat("\n")
24 print(mod_xl$coefficients)
25 cat("\n")
26 print(c("Max Residual", max(mod_xl$residuals)))

```

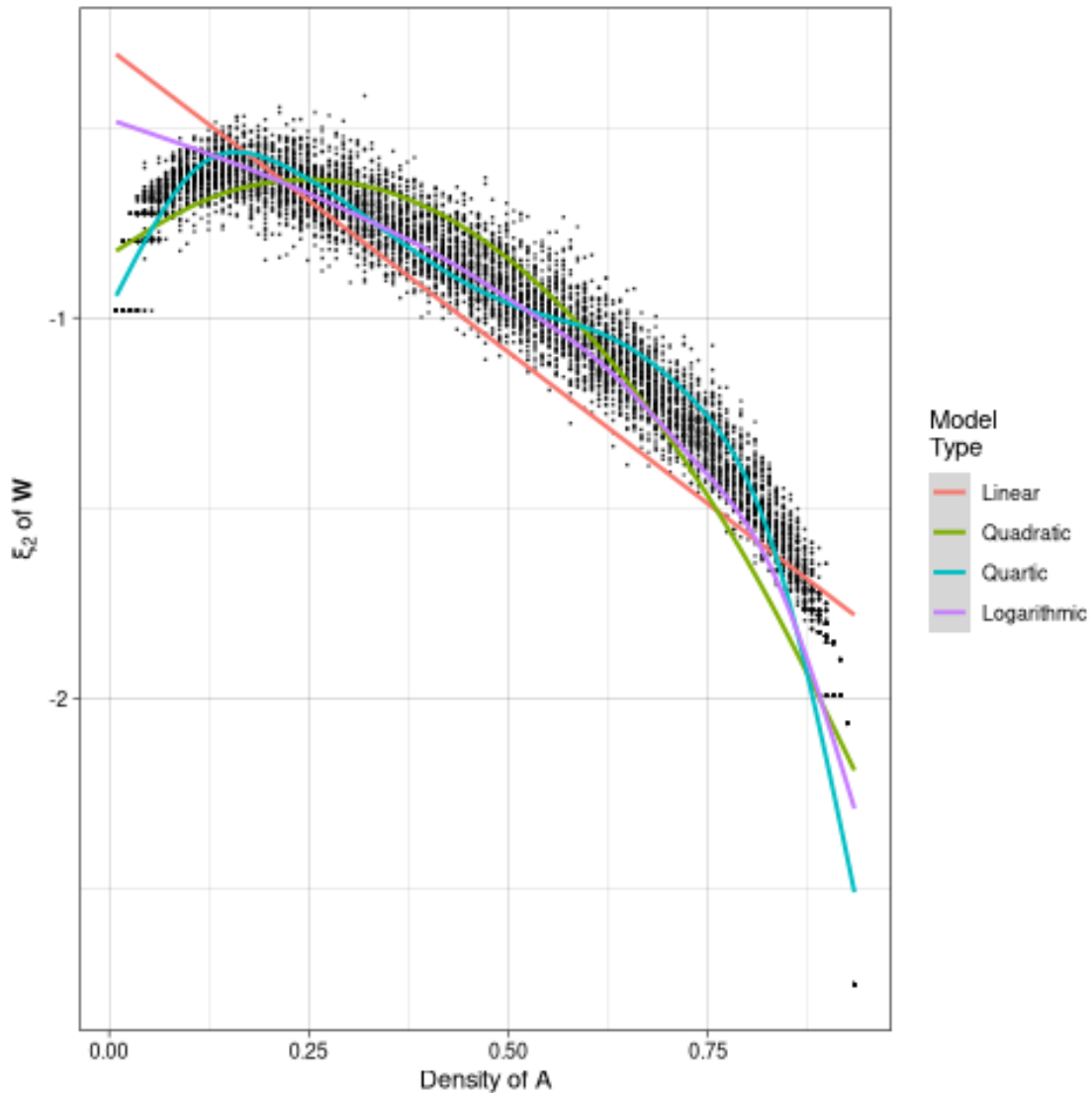
Listing 12: Fit Models to log transformed Density Comparison


```

1 ggplot(mod_df_long, aes(x = density)) +
2   geom_point(aes(y = eigenvalue2_log), fill =
3     ↪ "lightblue", col = "black", size = 0.1,
4     ↪ alpha = 0.2) +
5   geom_smooth(aes(y = eigenvalue2_mod, col =
6     ↪ Model_Type), size = 0.9) +
7   labs(col = c("Model \nType")) +
8   scale_color_discrete(labels = c("Linear",
9     ↪ "Quadratic", "Quartic", "Logarithmic")) +
10  labs(x = "Density of A", y = TeX("$ \\xi_2 $"
11    ↪ of $\\mathbf{W}$")) +
12  theme_linedraw()

```

Listing 13: Plot of Log Transformed ξ_2 against density of Adjacency matrix, using the *Power Walk* algorithm applied to graphs randomly generated with the *Erdos-Renyi* game.



6.1.4 Trace of Transition Probability Matrix

The correlation plot suggests that there is some positive relationship between the trace of the transition probability matrix and the second eigenvalue, these values are plotted in listing 15 and figure 7, this relationship appears to be heteroskedastic and so it is log transformed in listing 14 and figure 8. This plot is still appears to have a non constant variance but this could be due to less data corresponding to lower trace values.

The plot suggests an exponential or hyperbolic model may be a good fit, this is performed in listing 16, ?? and 9. The hyperbolic model appears to be a reasonable fit for trace values less than half, giving the following relationship:

$$\xi_2 \approx \exp\left(\frac{0.2}{\text{tr}(\mathbf{T})}\right) \quad (48)$$

$$\approx \exp\left(\frac{0.2}{\text{tr}(\mathbf{B}\mathbf{D}_\mathbf{B}^{-1})}\right) \quad (49)$$

```
1 ggplot(data, aes(x = trace , y =
  ↳ log(eigenvalue2))) +
2   geom_point(mapping = aes(size = size, color =
  ↳ p, shape = factor(n))) +
3   # stat_smooth() +
4   scale_size_continuous(range = c(0.1,1.5)) +
5   labs(x = "Trace of Transition Matrix", y =
  ↳ TeX("$\\log\\left( \\xi_2 \\right)$ of
  ↳ \\mathbf{W}"))
6   labs(x = "Trace of Transition Matrix", y =
  ↳ TeX("$\\log\\left( \\xi_2 \\right)$ of
  ↳ \\mathbf{W}"))
```

Listing 14: Plot ξ_2 against the trace of the matrix of the *Power Walk* Transition Probability Matrix, see figure 8

```
1 ggplot(data, aes(x = trace , y = eigenvalue2)) +
2   geom_point(mapping = aes(size = size, color =
  ↳ p, shape = factor(n))) +
3   # stat_smooth() +
4   scale_size_continuous(range = c(0.1,1.5)) +
5   labs(x = "Trace of Transition Matrix", y =
  ↳ TeX("$ \\xi_2 $ of $\\mathbf{W}$"))
```

Listing 15: Plot ξ_2 against the trace of the matrix of the *Power Walk* Transition Probability Matrix

MSE Hyperbolic MSE Logarithmic

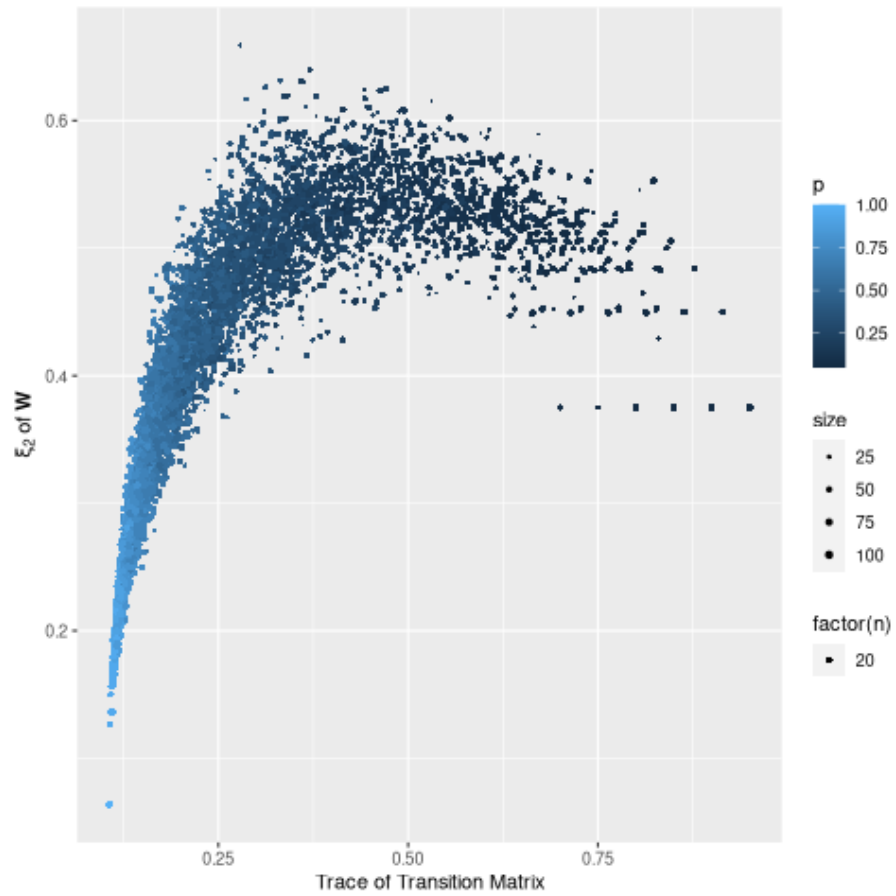


Figure 7: Plot of ξ_2 against the trace of the *Power Walk* probability transition matrix

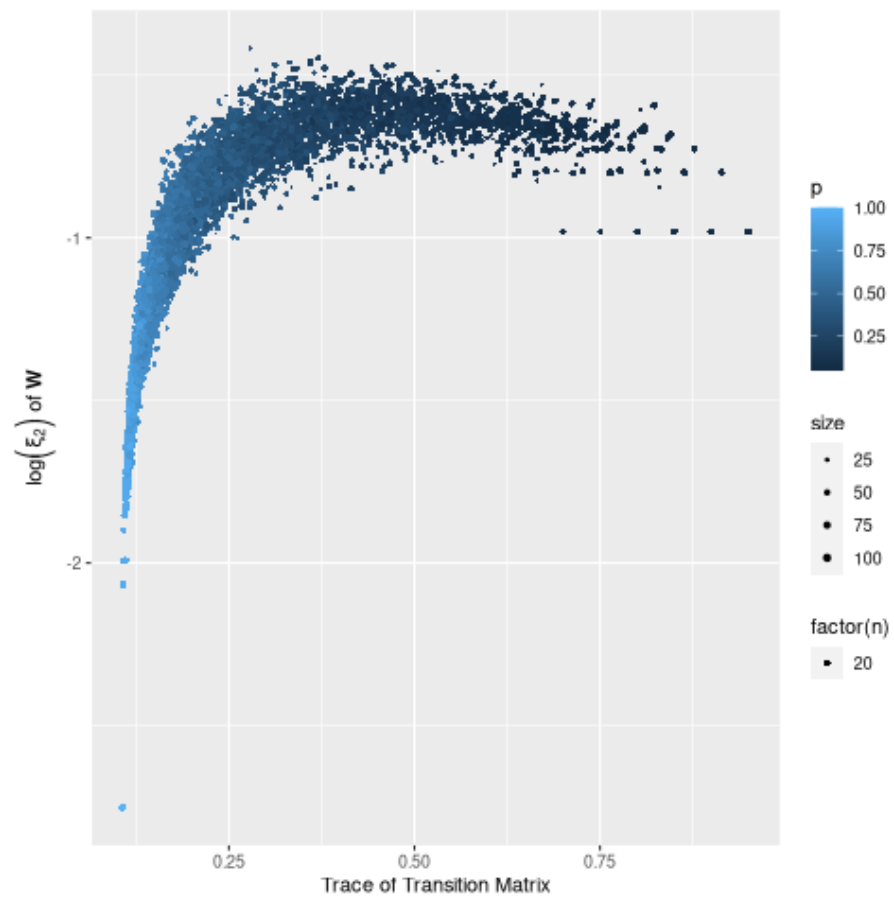


Figure 8: Log transformed plot of the trace of the *Power Walk* probability transition matrix

```

1  mod_df <- data
2
3  mod_hyp <- lm(log(eigenvalue2) ~ 0 +
  ↪ I(trace^(-1)), data = data)
4  mod_df$hyp <- predict(mod_hyp)
5
6  mod_log <- lm(log(eigenvalue2) ~ 0 + log(trace),
  ↪ data = data)
7  mod_df$log <- predict(mod_log)
8
9
10 mod_df_long <- pivot_longer(mod_df, cols =
  ↪ c(hyp, log), names_to = "Model_Type",
  ↪ values_to = "eigenvalue2_mod")
11 mod_df_long$eigenvalue2_log <-
  ↪ log(mod_df_long$eigenvalue2)
12
13
14 print(c("MSE Hyperbolic" =
  ↪ mean(mod_hyp$residuals^2),
15       "MSE Logarithmic" =
  ↪ mean(mod_log$residuals^2)), 2)
16 cat("\n")
17 print(summary(mod_hyp)$coefficients)

```

Listing 16: Fit a Hyperbolic and Logarithmic model to the data, observe that a 0 intercept is set to fix the intercept as it would be expected that a 0 trace would correspond to a 0 eigenvector, the hyperbolic model has a slightly lower mean *MSE*.

0.081

0.127

	Estimate	Std. Error	t value	Pr(> t)
I(trace ⁻¹)	-0.1992157	0.0005471191	-364.1176	0

```

1 ggplot(mod_df_long, aes(x = trace)) +
2   geom_point(shape = 23, aes(y =
   ↪ eigenvalue2_log), fill = "lightblue", col
   ↪ = "black", size = 0.7, alpha = 0.4) +
3   geom_line(aes(y = eigenvalue2_mod, col =
   ↪ Model_Type), size = 1) +
4   labs(col = c("Model \nType")) +
5   scale_color_manual(labels = c("Hyperbolic",
   ↪ "Logarithmic"),
6                           values = c("indianred",
   ↪ "royalblue")) +
7   labs(x = "Trace of Transition Matrix", y =
   ↪ TeX("$\\log\\left( \\xi_2 \\right)$ of
   ↪ \\mathbf{W}"), title = TeX("Models Fitted
   ↪ to Logarithmically scaled $\\xi_{2}$ given
   ↪ Matrix Trace")) +
8   theme_linedraw()

```

6.1.5 Conclusion

The *Erdos Renyi* game produces a wide variety of all possible graphs, this provides two insights into the value of the second eigenvalue of the probability transition matrix corresponding to the power walk method:

$$\xi_2 \approx \exp\left(\frac{0.2}{\text{tr}(\mathbf{B}\mathbf{D}_\mathbf{B}^{-1})}\right) \quad (50)$$

$$\xi_2 = \left(1 - \frac{\sum_{i=1}^n \sum_{j=1}^n \mathbf{A}_{i,j}}{n^2}\right)^{0.6} \cdot e^{-0.48} \pm 0.4 \quad (51)$$

These can be used to evaluate broadly the value of ξ_2 and in turn the rate of convergence of the *Power Walk* method corresponding to a given graph given only the method parameters and the adjacency matrix.

These are not however insightful of any direct relationships between the method parameters and ξ_2 , this will be considered at 8 by trying to find a relationship between the *Power Walk* and the *Random Surfer* models.

6.2 Model the log transformed data using a linear regression or log(-x) regression

6.2.1 Change the colour of each model by using `pivot_longer`

6.3 Import wikipedia data

- Import the wikipedia data

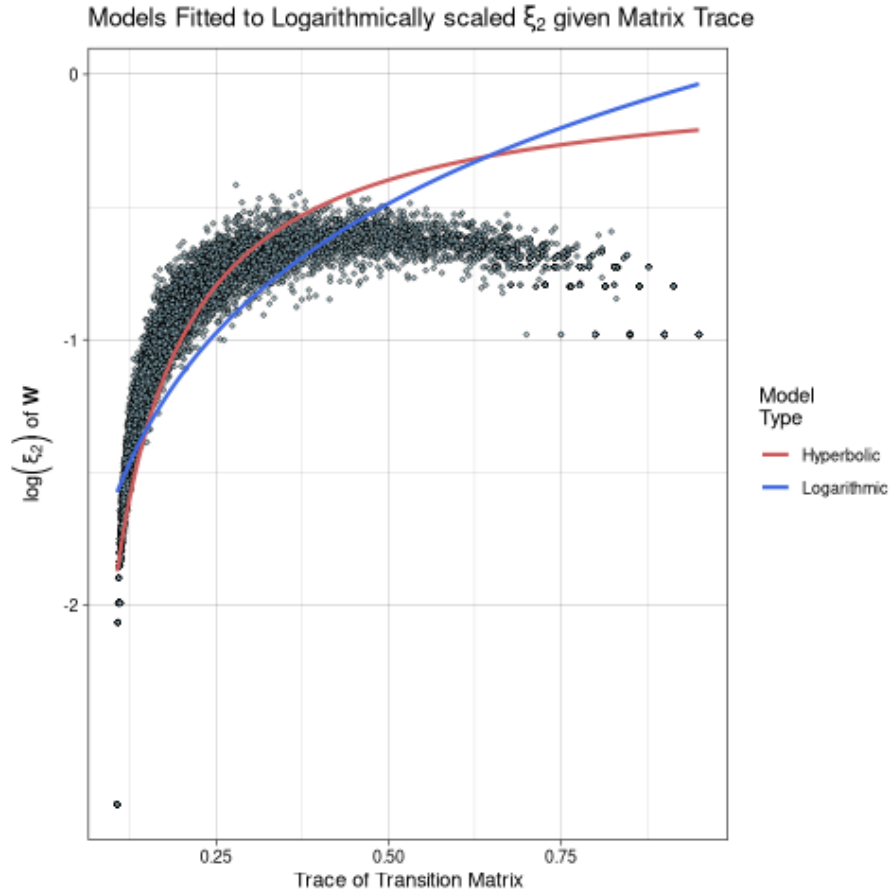


Figure 9: Plot of the second eigenvalue logarithmically scaled across the trace of a corresponding probability transition matrix created using the *Power Walk* method. The graphs were randomly generated using the *Erdos Renyi* game.

- Measure the density
- Use the density to guess the p of the game
 - Justify the witht the scatterplot matrix
- Measure the affect of different β values on λ_2 for graphs ov various sizes given that p value.
 - Or atleast a range within that prob
 - use a *Barabassi-Albert* Random Graph through the `igraph`::

6.4 Look at the Trace of the Matrix as a comparison point

6.5 Use BA Graphs

7 Simulating the Structure of the Web

A graph of the internet is *scale free*, this means that the number of nodes of a graph (n), having j edges is given by [19, §10.7.2]:

$$n \propto j^{-k}, \quad \exists k \in \mathbb{R} \quad (52)$$

The *Erdos Renyi* game is a random network, a superior approach to model the web is to use a scale free networks [3] such as the Barabasi-Albert graph [4]

Maybe I should look at the most appropriate way to simulate social network links, one possibility is [this paper](#) [27].

Actually there is a data set available [13], I should just analyse that, see [how it was done in Visual Analytics as a reminder](#).

8 Relating the Power Walk to the Random Surfer

8.1 Introduction

These are notes relating to [24, §3.3], probably won't put this in the report, just arbitrary notes

So if a term in the Power Walk can be related to α in the random surfer, which is in turn ξ_2 , I'll be able to understand it better. ⁹

Consider the equation:

$$\begin{aligned} \mathbf{T} &= \mathbf{B} \mathbf{D}_{\mathbf{B}}^{-1} \\ &= (\mathbf{B} + \mathbf{O} - \mathbf{O}) \mathbf{D}_{\mathbf{B}}^{-1} \end{aligned}$$

Break this into to terms so that we can simplify it a bit:

$$\mathbf{T} = \left[(\mathbf{B} - \mathbf{O}) \mathbf{D}_{\mathbf{B}}^{-1} \right] + \left\{ \mathbf{O} \mathbf{D}_{\mathbf{B}}^{-1} \right\}$$

⁹Although I'm not quite sure why α is ξ_2 either

8.2 Value of [1st Term]

Observe that for all $\forall i, j \in \mathbb{Z}^+$:

$$\begin{aligned} \mathbf{A}_{i,j} &\in \{0, 1\} \\ \implies \mathbf{B}^{\mathbf{A}_{i,j}} &\in \{\beta^0, \beta^1\} \\ &= \{1, \beta\} \\ \implies \beta \mathbf{A} &= \{1, \beta\} \end{aligned}$$

Using this property we get the following

$$\begin{aligned} \mathbf{B}_{i,j} - \mathbf{O}_{i,j} &= (\beta^{\mathbf{A}_{i,j}} - 1) = \begin{cases} 0, & \mathbf{A}_{i,j} = 0 \\ \beta - 1, & \mathbf{A}_{i,j} = 1 \end{cases} \\ (\beta - 1) \mathbf{A}_{i,j} &= \begin{cases} 0, & \mathbf{A}_{i,j} = 0 \\ \beta - 1, & \mathbf{A}_{i,j} = 1 \end{cases} \end{aligned}$$

This means we have

$$\mathbf{A} \in \{0, 1\} \forall i, j \implies \mathbf{B}_{i,j} - \mathbf{O}_{i,j} = (\beta - 1) \mathbf{A}_{i,j}$$

$$\begin{aligned} \mathbf{B} &= (\mathbf{B} + \mathbf{O} - \mathbf{O}) \\ &= (\mathbf{B} - 1) \end{aligned}$$

8.3 Value of {2nd Term}

$$\begin{aligned} \mathbf{O} \mathbf{D}_{\mathbf{B}}^{-1} &= \begin{pmatrix} 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \dots \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} \frac{1}{\delta_1} & 1 & 1 & \dots \\ 1 & \frac{1}{\delta_2} & 1 & \dots \\ 1 & 1 & \frac{1}{\delta_3} & \dots \\ \vdots & & & \ddots \end{pmatrix} \\ &= n \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \dots \\ \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \dots \\ \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \dots \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} \frac{1}{\delta_1} & 1 & 1 & \dots \\ 1 & \frac{1}{\delta_2} & 1 & \dots \\ 1 & 1 & \frac{1}{\delta_3} & \dots \\ \vdots & & & \ddots \end{pmatrix} \\ &= n \mathbf{E} \mathbf{D}_{\mathbf{B}}^{-1} \end{aligned}$$

where the following definitions hold ($\forall i, j \in \mathbb{Z}^+$):

- $\mathbf{E}_{i,j} = \frac{1}{n}$
- $\mathbf{D}_{\mathbf{B},k,k}^{-1} = \frac{1}{\delta_k}$

- The value of δ is value that each term in a column must be divided by to become zero, in the case of the power walk that is just $\frac{1}{\text{colSums}(\mathbf{B})} = \vec{1}\mathbf{B}$, but if there were zeros in a column, it would be necessary to swap out the 0s for 1s and then sum in order to prevent a division by zero issue and because the 0s should be left.
- $\mathbf{A} \in \{0, 1\} \forall i, j$ is the unweighted adjacency matrix of the relevant graph.

putting this all together we can do the following:

$$\begin{aligned}\mathbf{T} &= \mathbf{B}\mathbf{D}_\mathbf{B}^{-1} \\ &= (\mathbf{B} + \mathbf{O} - \mathbf{O}) \mathbf{D}_\mathbf{B}^{-1} \\ &= (\mathbf{B} - \mathbf{O}) \mathbf{D}_\mathbf{B}^{-1} + \mathbf{O}\mathbf{D}_\mathbf{B}^{-1}\end{aligned}$$

From above:

$$\begin{aligned}&= (\beta - 1) \mathbf{A}_{i,j} + n\mathbf{E}\mathbf{D}_\mathbf{B}^{-1} \\ &= \mathbf{A}_{i,j} (\beta - 1) + n\mathbf{E}\mathbf{D}_\mathbf{B}^{-1}\end{aligned}$$

because $\mathbf{D}\mathbf{D}^{-1} = \mathbf{I}$ we can multiply one side through:

$$= \mathbf{D}_\mathbf{A}\mathbf{D}_\mathbf{A}^{-1}\mathbf{A}_{i,j} (\beta - 1) + n\mathbf{E}\mathbf{D}_\mathbf{B}^{-1}$$

But the next step requires showing that:

$$(\beta - 1) \mathbf{D}_\mathbf{A}\mathbf{D}_\mathbf{A}^{-1} = \mathbf{I} - n\mathbf{D}_\mathbf{B}^{-1}$$

8.4 Equate the Power Walk to the Random Surfer

Define the matrix $\mathbf{D}_\mathbf{M}$:

$$\mathbf{D}_\mathbf{M} = \text{diag}(\text{colSum}(\mathbf{M})) = \text{diag}(\vec{1}\mathbf{M}) \quad (53)$$

To scale each column of that matrix to 1, each column will need to be divided by the column sum, unless the column is already zero, this needs to be done to turn an adjacency matrix into a matrix of probabilities:

$$\mathbf{D}_\mathbf{A}^{-1} : [\mathbf{D}_\mathbf{A}^{-1}]_i = \begin{cases} 0, & [\mathbf{D}_\mathbf{A}]_i = 0 \\ \left[\frac{1}{\mathbf{D}_\mathbf{A}}\right], & [\mathbf{D}_\mathbf{A}]_i \neq 0 \end{cases} \quad (54)$$

In the case of the power walk $\mathbf{B} = \beta^\mathbf{A} \neq 0$ so it is sufficient:

$$\mathbf{D}_B^{-1} = \frac{1}{\text{diag}(\vec{1}(\beta^A))} \quad (55)$$

Recall that the *power walk* gives a transition probability matrix:

Power Walk

$$\mathbf{T} = \boxed{\mathbf{A}\mathbf{D}_A^{-1}} \mathbf{D}_A (\beta - 1) \mathbf{D}_B^{-1} + \boxed{\mathbf{E}} n \mathbf{D}_B^{-1} \quad (56)$$

Random Surfer

$$\mathbf{T} = \alpha \boxed{\mathbf{A}\mathbf{D}_A^{-1}} + (1 - \alpha) \boxed{\mathbf{E}} \quad (57)$$

So these are equivalent when:

$$\mathbf{D}_A (\beta - 1) \mathbf{D}_B^{-1} = \mathbf{I} \alpha \quad (58)$$

$$\begin{aligned} \vec{1}(1 - \alpha) &= -n \mathbf{D}_B^{-1} \\ \implies \vec{1} \alpha &= \vec{1} - n \mathbf{D}_B^{-1} \end{aligned} \quad (59)$$

Hence we have:

$$\mathbf{D}_A (\beta - 1) \mathbf{D}_B^{-1} = \vec{1} \alpha = \mathbf{I} - n \mathbf{D}_B^{-1} \quad (60)$$

Solving for β with (58):

$$\beta = \frac{1 - \Theta}{\Theta} \quad (61)$$

$$(62)$$

where: ¹⁰

$$\bullet \Theta = \mathbf{D}_A \mathbf{D}_B^{-1}$$

but we can't really do this so instead:

$$\beta \mathbf{1}_{[n,n]} = (1 - \Theta) \Theta^{-1}$$

If β is set accordingly then by (60):

¹⁰NOTE: Similar to a sigmoid function, which is a solution to $p \propto p(1 - p)$, I wonder if this provides a connection to the exponential nature of the power walk ‘erdos.renyi’ ‘erdos.renyi’

$$\begin{aligned} \mathbf{A} (\beta - 1) \mathbf{D}_{\mathbf{B}}^{-1} &= \alpha = \mathbf{I} - n\mathbf{D}_{\mathbf{B}}^{-1} \\ \implies \mathbf{A} (\beta - 1) \mathbf{D}_{\mathbf{B}}^{-1} &= \mathbf{I} - n\mathbf{D}_{\mathbf{B}}^{-1} \end{aligned} \quad (63)$$

And setting $\Gamma = \mathbf{I} - n\mathbf{D}_{\mathbf{B}}^{-1}$ from (59) and putting in (56) we have:

$$\begin{aligned} \mathbf{T} &= \boxed{\mathbf{A}\mathbf{D}_{\mathbf{A}}^{-1}} \mathbf{D}_{\mathbf{A}} (\beta - 1) \mathbf{D}_{\mathbf{B}}^{-1} + \boxed{\mathbf{E}} n\mathbf{D}_{\mathbf{B}}^{-1} \\ \mathbf{T} &= \Gamma \boxed{\mathbf{A}\mathbf{D}_{\mathbf{A}}^{-1}} + (1 - \Gamma) \boxed{\mathbf{E}} \\ \mathbf{T} &= \Gamma \mathbf{A}\mathbf{D}_{\mathbf{A}}^{-1} + (1 - \Gamma) \mathbf{E} \end{aligned} \quad (64)$$

Where \mathbf{E} is square matrix of $\frac{1}{n}$ as in (16) (23)

8.5 Conclusion

So when the adjacency matrix is stictly boolean, the power walk is equivalent to the random surfer.

8.6 The Second Eigenvalue

8.6.1 The Random Surfer

The Second eigenvalue ξ_2 of the Power Surfer is less than α ([See 3.2; Stability and Concvergence, of proposal](#)).

8.6.2 Power Walk

Because the Power Walk relates to the random surfer as demonstrated in section , what can be said about ξ_2

Applying this to Power Walk Let $\Lambda_{(2)}(\mathbf{T}) = \lambda_2$ return the second value of a transition, probability Matrix, then observe that:

$$\Lambda_{(2)}(\mathbf{T}_{\text{RS}}) \leq |\alpha| \implies \Lambda_{(2)}(\mathbf{T}_{\text{PW}}) \leq \left| \frac{\alpha - \mathbf{D}_{\mathbf{a}}\mathbf{D}_{\mathbf{B}}^{-1}}{\mathbf{D}_{\mathbf{A}}\mathbf{D}_{\mathbf{B}}^{-1}} \right| \quad (65)$$

where:

- $\lambda_{(2)}(\mathbf{T})$ refers to the transition probability matrix of the power walk and random surfer appoaches as indicated.

My attempt

$$\beta \mathbf{1}_{[n,n]} = \frac{1 - \Theta}{\Theta} \quad (66)$$

$$(67)$$

where:

- $\Theta = \mathbf{D}_A \mathbf{D}_B^{-1}$

So I thought maybe if I could find a value of β that satisfied (66) then I could show circumstances under which $|\xi_2| < \alpha$.

Seemingly it's only satisfied where $\beta = 1$ though, using this simulation:

```

1  g1 <- igraph::erdos.renyi.game(n = 9, 0.2)
2  A <- igraph::get.adjacency(g1) # Row to column
3  A <- t(A)
4  # plot(g1)
5
6  ## * Finding beta values to behave like Random
   ↪ Surfer
7  beta <- 10
8  B <- beta^A
9
10 DA <-
   ↪ PageRank::create_sparse_diag_sc_inv_mat(A)
11 DB_inv <-
   ↪ PageRank::create_sparse_diag_scaling_mat(B)
12
13 THETA <- DA %*% DB_inv
14
15 THETA <- function(A, beta) {
16   B <- beta^A
17   DA <-
   ↪ PageRank::create_sparse_diag_sc_inv_mat(A)
18   DB_inv <-
   ↪ PageRank::create_sparse_diag_scaling_mat(B)
19   return(DA %*% DB_inv)
20 }
21
22 THETA_inv <- function(A, beta) {
23   B <- beta^A
24   DB <-
   ↪ PageRank::create_sparse_diag_sc_inv_mat(B)
25   DA_inv <-
   ↪ PageRank::create_sparse_diag_scaling_mat(A)
26   return(DA %*% DB_inv)
27 }
28
29 beta_func <- function(A, beta) {
30   return(1-THETA(A, beta^A) %*% THETA_inv(A,
   ↪ beta^A))
31 }
32
33 THETA(A, 10) %*% THETA_inv(A, 10)
34
35
36 eta <- 10^-6
37 beta <- 1.01
38 while (mean(beta*matrix(1, nrow(A), ncol(A)) -
   ↪ beta_func(A, beta)) > eta) {
39   beta <- beta + 0.01
40   print(beta)
41   print(diag(beta_func(A, beta)))
42   print(beta*matrix(1, nrow(A), ncol(A)))
43   print(beta_func(A, beta))
44   # Sys.sleep(0.1)

```

9 Appendix

```
1 library(Matrix)
2 library(igraph)
3 n <- 200
4 m <- 5
5 power <- 1
6 g <- igraph::sample_pa(n = n, power = power, m =
  ↪ m, directed = FALSE)
7 plot(g)
8 A <- t(get.adjacency(g))
9 plot(A)
10 image(A)
11
12
13 # Create a Plotting Region
14 par(pty = "s", mai = c(0.1, 0.1, 0.4, 0.1))
15
16
17 # create the image
18
19 title=paste0("Undirected Barabassi Albert Graph
  ↪ with parameters:\n Power = ", power, "; size
  ↪ = ", n, "; Edges/step = ", round(m))
20 image(A, axes = FALSE, frame.plot = TRUE, main =
  ↪ title, xlab = "", ylab = "", )
```

Listing 17: **R** code to produce an image illustrating the density of a simulated Barabasi-Albert graph, the *Barabasi-Albert* graph is a good analouge for the link structure of the internet [19, 3, 4] see the output in figure 10

9.1 Graph Diagrams

Graph Diagrams shown in 2.2.2 where produced using DOT (see [26, 9]).

10 my to do list

10.1 Look at the Trace of the Matrix as a comparison point

10.2 Use BA Graphs

**

10.3 TODO Diamater

Diamater of the web sounds like a fun read [2]

**Undirected Barabassi Albert Graph with parameters:
Power = 1; size = 200; Edges/step = 5**

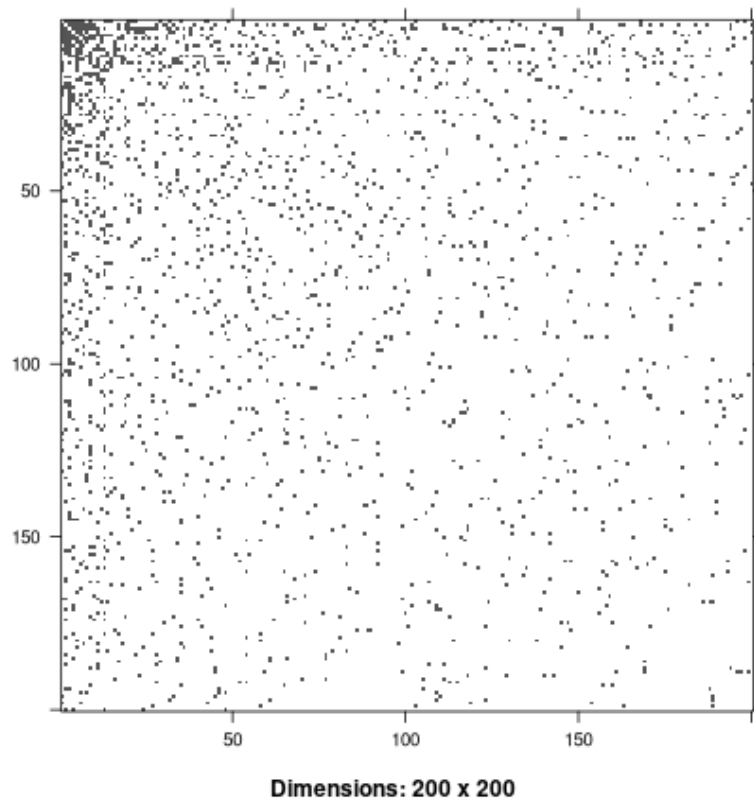


Figure 10: Plot of the adjacency matrix corresponding to a Barabassi-Albert (i.e. *Scale Free*) Graph produced by listing 17, observe the matrix is quite sparse.

10.4 Improving the Performance of Page Rank

This:

Another approach involves reordering the problem and taking advantage of the fact that the transition probability matrix is sparse in order to produce a new algorithm which cannot perform worse than the *power method* but has been shown to improve the rate of convergence in certain cases. [18].

There was also a book that I downloaded that mentioned it
Accelerating the Computation of Page Rank [19]

References

- [1] *Adjacency Matrix*. In: *Wikipedia*. Sept. 20, 2020. URL: https://en.wikipedia.org/w/index.php?title=Adjacency_matrix&oldid=979433676 (visited on 10/10/2020) (cit. on p. 3).
- [2] Réka Albert, Hawoong Jeong, and Albert-László Barabási. “Diameter of the World-Wide Web”. In: *Nature* 401.6749 (Sept. 1999), pp. 130–131. ISSN: 0028-0836, 1476-4687. DOI: [10.1038/43601](https://doi.org/10.1038/43601). URL: <http://www.nature.com/articles/43601> (visited on 10/11/2020) (cit. on p. 54).
- [3] Albert-László Barabási. “The Physics of the Web”. In: *Phys. World* 14.7 (July 2001), pp. 33–38. ISSN: 0953-8585, 2058-7058. DOI: [10.1088/2058-7058/14/7/32](https://doi.org/10.1088/2058-7058/14/7/32). URL: <https://iopscience.iop.org/article/10.1088/2058-7058/14/7/32> (visited on 10/11/2020) (cit. on pp. 47, 54).
- [4] Albert-László Barabási, Réka Albert, and Hawoong Jeong. “Scale-Free Characteristics of Random Networks: The Topology of the World-Wide Web”. In: *Physica A: Statistical Mechanics and its Applications* 281.1-4 (June 2000), pp. 69–77. ISSN: 03784371. DOI: [10.1016/S0378-4371\(00\)00018-2](https://doi.org/10.1016/S0378-4371(00)00018-2). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0378437100000182> (visited on 10/11/2020) (cit. on pp. 47, 54).
- [5] Joost Berkhout and Bernd F. Heidergott. “Ranking Nodes in General Networks: A Markov Multi-Chain Approach”. In: *Discrete Event Dyn Syst* 28.1 (Mar. 1, 2018). The choice of damping factor of Google’s page rank might have a large impact on the values given to vertices. This suggests an approach that uses structural network dynamics to provide an appropriate score distribution. The method implemented is not something I have come yet to understand, but it could be very interesting to see:
 - how it relates to the power walk method
 - whether or not it could offer insights into the convergence and stability of the power walk method
 - Whether or not the method would be compatible with negatively weighted edges., pp. 3–33. ISSN: 1573-7594. DOI: [10.1007/s10626-017-0248-7](https://doi.org/10.1007/s10626-017-0248-7). URL: <https://doi.org/10.1007/s10626-017-0248-7> (visited on 08/19/2020) (cit. on p. 6).
- [6] Monica Bianchini, Marco Gori, and Franco Scarselli. “Inside PageRank”. In: *ACM Trans. Inter. Tech.* 5.1 (Feb. 1, 2005). This is a discussion on the stability, complexity and critical role of parameters involved in the computation., pp. 92–128. ISSN: 15335399. DOI: [10.1145/1052934.1052938](https://doi.org/10.1145/1052934.1052938). URL: <http://portal.acm.org/citation.cfm?doid=1052934.1052938> (visited on 08/18/2020) (cit. on p. 6).

- [7] Michael Brinkmeier. “PageRank Revisited”. In: *ACM Transactions on Internet Technology* 6.3 (Aug. 2006), pp. 282–301. ISSN: 15335399. DOI: [10.1145/1151087.1151090](https://doi.org/10.1145/1151087.1151090). URL: <http://ezproxy.uws.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=iih&AN=22173011&site=ehost-live&scope=site> (visited on 08/19/2020) (cit. on p. 6).
- [8] Mufa Chen. *Eigenvalues, Inequalities, and Ergodic Theory*. Probability and Its Applications. London: Springer, 2005. 228 pp. ISBN: 978-1-85233-868-8 (cit. on p. 4).
- [9] DOT (Graph Description Language). In: *Wikipedia*. June 11, 2020. URL: [https://en.wikipedia.org/w/index.php?title=DOT_\(graph_description_language\)&oldid=961944797](https://en.wikipedia.org/w/index.php?title=DOT_(graph_description_language)&oldid=961944797) (visited on 10/09/2020) (cit. on p. 54).
- [10] François Fouss, Marco Saerens, and Masashi Shimbo. *Algorithms and Models for Network Data and Link Analysis*. New York, NY: Cambridge University Press, 2016. 521 pp. ISBN: 978-1-107-12577-3 (cit. on p. 4, 7).
- [11] Hwai-Hui Fu, Dennis K. J. Lin, and Hsien-Tang Tsai. “Damping Factor in Google Page Ranking”. In: *Applied Stochastic Models in Business and Industry* 22.5-6 (2006), pp. 431–444. ISSN: 1526-4025. DOI: [10.1002/asmb.656](https://doi.org/10.1002/asmb.656). URL: <http://onlinelibrary.wiley.com/doi/abs/10.1002/asmb.656> (visited on 08/19/2020) (cit. on p. 6).
- [12] Gabor Csardi et al. *Igraph R Manual Pages*. May 9, 2019. URL: https://igraph.org/r/doc/as_adjacency_matrix.html (visited on 08/19/2020) (cit. on p. 3).
- [13] Andrea Garritano. *Wikipedia Article Networks*. Dec. 2019. URL: <https://kaggle.com/andreagarritano/wikipedia-article-networks> (visited on 10/03/2020) (cit. on p. 47).
- [14] Pankaj Gupta et al. “WTF: The Who to Follow Service at Twitter”. In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW ’13. New York, NY, USA: Association for Computing Machinery, May 13, 2013, pp. 505–514. ISBN: 978-1-4503-2035-1. DOI: [10.1145/2488388.2488433](https://doi.org/10.1145/2488388.2488433). URL: <http://doi.org/10.1145/2488388.2488433> (visited on 10/09/2020) (cit. on p. 5).
- [15] *Igraph R Manual Pages*. URL: https://igraph.org/r/doc/sample_pa.html (visited on 10/06/2020) (cit. on p. 34).
- [16] Sepandar Kamvar, Taher Haveliwala, and Gene Golub. “Adaptive Methods for the Computation of PageRank”. In: *Linear Algebra and its Applications*. Special Issue on the Conference on the Numerical Solution of Markov Chains 2003 386 (July 15, 2004), pp. 51–65. ISSN: 0024-3795. DOI: [10.1016/j.laa.2003.12.008](https://doi.org/10.1016/j.laa.2003.12.008). URL: <http://www.sciencedirect.com/science/article/pii/S0024379504000023> (visited on 08/19/2020) (cit. on p. 6).
- [17] Moshe Koppel and Nadav Schweitzer. “Measuring Direct and Indirect Authorial Influence in Historical Corpora”. In: *Journal of the Association for Information Science and Technology* 65.10 (2014), pp. 2138–2144. ISSN: 2330-1643. DOI: [10.1002/asi.23118](https://doi.org/10.1002/asi.23118). URL: <https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.23118> (visited on 08/21/2020) (cit. on p. 6).
- [18] Amy N. Langville and Carl D. Meyer. “A Reordering for the PageRank Problem”. In: *SIAM Journal on Scientific Computing; Philadelphia* 27.6 (2006), p. 9. ISSN: 10648275. DOI: [http://dx.doi.org.ezproxy.uws.edu.au/10.1137/040607551](https://doi.org/10.1137/040607551). URL: <http://search.proquest.com/docview/921138313/abstract/24AFC1417CF6412BPQ/1> (visited on 08/19/2020) (cit. on p. 56).
- [19] Amy N. Langville and Carl D. Meyer. *Google’s PageRank and beyond: The Science of Search Engine Rankings*. Neuaufl. Princeton: Princeton Univ. Press, 2012. 224 pp. ISBN: 978-0-691-15266-0 (cit. on pp. 4–8, 10, 47, 54, 56).

- [20] Larry Page and Sergey Brin. “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Computer Networks and ISDN Systems* 30.1-7 (Apr. 1, 1998), pp. 107–117. ISSN: 0169-7552. DOI: [10.1016/S0169-7552\(98\)00110-X](https://doi.org/10.1016/S0169-7552(98)00110-X). URL: <http://www.sciencedirect.com/science/article/pii/S016975529800110X> (visited on 08/19/2020) (cit. on pp. 3, 8).
- [21] Ron Larson and Bruce H. Edwards. *Elementary Linear Algebra*. 2nd ed. Includes index. Lexington, Mass: D.C. Heath, 1991. 592 pp. ISBN: 978-0-669-24592-9 (cit. on p. 11).
- [22] George Meghabghab and Abraham Kandel. *Search Engines, Link Analysis, and User’s Web Behavior: A Unifying Web Mining Approach*. Studies in Computational Intelligence v. 99. Berlin: Springer, 2008. 269 pp. ISBN: 978-3-540-77468-6 978-3-540-77469-3 (cit. on p. 3).
- [23] Nathanael Ackerman, Cameron Freer, Alex Kruckman, and Rehana Patel. *Properly Ergodic Structures*. Oct. 25, 2017. URL: <https://math.mit.edu/~freer/papers/properly-ergodic-structures.pdf> (visited on 10/10/2020) (cit. on p. 4).
- [24] Laurence A. F. Park and Simeon Simoff. “Power Walk: Revisiting the Random Surfer”. In: *Proceedings of the 18th Australasian Document Computing Symposium*. ADCS ’13. Brisbane, Queensland, Australia: Association for Computing Machinery, Dec. 5, 2013, pp. 50–57. ISBN: 978-1-4503-2524-0. DOI: [10.1145/2537734.2537749](https://doi.org/10.1145/2537734.2537749). URL: <http://doi.org/10.1145/2537734.2537749> (visited on 07/31/2020) (cit. on pp. 3, 48).
- [25] saz. *Probability Theory - Is This Graph Ergodic?* URL: <https://math.stackexchange.com/questions/1327283/is-this-graph-ergodic> (visited on 10/10/2020) (cit. on p. 5).
- [26] *The DOT Language*. URL: <https://graphviz.org/doc/info/lang.html> (visited on 10/09/2020) (cit. on p. 54).
- [27] Rui Zeng et al. “A Practical Simulation Method for Social Networks”. In: 144 (2013), p. 8 (cit. on p. 47).