Investigating the Second Eigen Value of the Power Walk Page Rank Method

Ryan Greenup

October 25, 2020

Contents

1 Introduction

Any collection of interconnected information can form a network structure, this can include for example citation networks, web-pages, wikis, power grids, wiring diagrams, encyclopedias and interpersonal relationships. The analysis of these networks can be used to draw insights about the behavior 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, visualized in such a way so as to minimize 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 node has the highest *PageRank* score [1] 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 node can be measured by the frequency of traversal during a random walk.

This approach relies on the assumption that the random walk can:

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

and so the *PageRank* method involves modifying a graph in some way to address these assumptions, specifically by modifying the corresponding probability transition matrix to be stochastic and primitive.

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 [23] 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.

[1] See listing 5 for the *Random Surfer* solution

The *Power Walk* approach, introduced by Park and Simoff in 2013 [28] 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 § 3 .

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[2].

It is not clear how the second eigenvalue is related to the method parameters of the *Power Walk* algorithm [28, §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 Definitions use in this report

The following definitions are used throughout this report [3]:

Markov Chains are discrete mathematical models such that future values depend only on current values [14, §1.5], this captures the concept of a random walk because the next destination depends only on the current location.

Stochastic Matrices contain only positive values where each column sums to 1 [22, 14] (i.e. $\mathbf{T}$ is stochastic $\iff \vec{1}\mathbf{T} = \vec{1}$)

- some authors use rows (see e.g. [22, §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 node j to node 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 $\texttt{sum}\left(\vec{\texttt{x}}\right) = 1$, so when solving for a stationary vector it can be necessary to perform $\vec{\texttt{p}} \leftarrow \frac{\vec{\texttt{p}}}{\sum \vec{p}}$

Irreducible graphs have a path from from any given node to another node. [22, §15.2]

Ergodic graphs are irreducible graphs with further constraints outside the scope of this report (see e.g. [26, 10])

[2] This is discussed in § 7

[3] see generally [22, Ch. 15] for further reading

- It is a necessary but not a sufficient condition of ergodic graphs that all nodes be reachable from any other nodes (see [30] 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 [22, §15.2], hence the significance of this concept.

Transition Probability Matrix is a stochastic matrix where each column is a vector of probabilities such that $\mathbf{T}_{i,j}$ represents the probability of traveling from node j to node i during a random walk.

- Some Authors consider the transpose (see e.g. [22]).

Aperiodic Markov chains have 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 irreducible and aperiodic.

Sparse Matrices contain a majority of elements with values equal to 0 [22, §4.2]

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. [17, 22]) the term *Random Surfer* is usually used to refer specifically to the smoothing algorithm shown in (5), but *PageRank* refers to the entire concept including the *Random Surfer*. In this report the term *PageRank* will be used generally to denote the concept of measuring node centrality by the frequency of node traversal during a random walk and the models *Random Surfer* or *Power Walk* will be denoted specifically where necessary.

2.1 Notation

- $\mathbf{A}$
  - Is the *adjacency matrix* of a graph such that $\mathbf{A}_{i,j} = 1$ indicates travel from j to i is possible.
- $\mathbf{T}$
  - Is the *transition probability matrix* of a graph, this matrix indicates the probability of a movement during a random walk, such that $\mathbf{T}_{i,j}$ is equal to the probability of traveling $j \to i$ during a random walk.
- n
  - Refers to the number of nodes in a graph, $n = \texttt{nrow}\left(\mathbf{A}\right) = \texttt{ncol}\left(\mathbf{A}\right)$
- $\mathbf{D}_{\mathbf{A}} = \operatorname{diag}\left(\vec{1}\mathbf{A}\right)$
- $\mathbf{D}_{\mathbf{A}}^{-1} = \begin{cases} 0, & [\mathbf{D}_{\mathbf{A}}]_i = 0 \\ \left[\frac{1}{\mathbf{D}_{\mathbf{A}}}\right], & [\mathbf{D}_{\mathbf{A}}]_i \neq 0 \end{cases}$
  - A diagonal scaling matrix such that $\mathbf{T} = \mathbf{A}\mathbf{D}_{\mathbf{A}}^{-1}$, the piece-wise definition is such that $\mathbf{D}_{\mathbf{A}}^{-1}$ is still defined even if $\mathbf{A}$ is a reducible graph.

  - Where $\mathbf{D}^{-1}$ is a matrix such that multiplication with which scales each column of $\mathbf{A}$ to 1.
  - $\mathbf{D}_{\mathbf{A}}^{-1} = \vec{1}\mathbf{D}_{\mathbf{A}}^{-1} = \frac{1}{\vec{1}\mathbf{D}_{\mathbf{A}}}$ for some stochastic matrix $\mathbf{A}$

- $\mathbf{E}_{i,j} = \frac{1}{n}$
  - A matrix of size $n \times n$ representing the background probability of jumping to any node of a graph.
- $\vec{1}$
  - a vector of length n containing only the value 1, this size of which should be clear from the context.
    - 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. [22]
- $\mathbf{J} = \vec{1} \cdot \vec{1}^{\mathrm{T}} \iff \mathbf{J}_{i,j} = 1$
  - A completely dense $n \times n$ matrix containing only 1
- ξ_n
  - The n^{th} largest eigenvalue of a transition probability matrix $\mathbf{T}$, the use of λ has been avoided because some authors use λ to represent damping factor α, given that α=ξ_2 for certain graphs, this can be very ambiguous.
- α
  - A probability such that $1-\alpha$ represents the probability of teleporting from one node to another during a random walk, see 4.
    - In the literature α is often referred to as a damping factor (see e.g. [5, 8, 15, 20, 6]) or a smoothing constant (see e.g [21]).

Part I

Implementing PageRank

3 Mathematics of Page Rank

[11, §1.1.7]

3.1 The Stationary Distribution of a Probability Transition Matrix

A graph can be expressed as an adjacency matrix $\mathbf{A}$:

$$\mathbf{A}_{i,j} \in \{0, 1\}$$

Where each element of the matrix indicates whether or not travel from node j to node i is possible with a value of 1. [4]

During a random walk on a graph the probability of arriving at node j from node 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 relationship:

$$\mathbf{T} = \mathbf{A}\mathbf{D}_{\mathbf{A}}^{-1} \tag{1}$$

The value of $\mathbf{D}_{\mathbf{A}}^{-1}$ is such that under matrix multiplication $\mathbf{A}$ will have columns that sum to 1 [5], this matrix is the *transition probability matrix* $\mathbf{T}$.

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

$$\vec{p}_{i+1} = \mathbf{T}\vec{p}_i \tag{2}$$

This relationship is a linear recurrence relation, more generally however it is a *Markov Chain* [22, §4.4] and Finding the Stationary point for this relationship will give a frequency distribution for the nodes corresponding to the random walk and thus a metric to measure the centrality of nodes.

3.1.1 The Stationary Distribution of a Markov Chain

The stationary distribution (p) of a Markov Chain is a vector that sums to 1 and does not change upon further iteration [11, §1.1.7], for example in the case of (2), if:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

[4] Some authors define an adjacency matrix transposed (see e.g. [1, 25]) this unfortunately includes the `igraph` library [16] but that convention will not be followed in this paper

[5] such a matrix is said to be a *column stochastic matrix*, for a reducible or non-stochastic graph the definition of $\mathbf{D}_{\mathbf{A}}^{-1}$ needs to piece-wise, as shown in § 2

the corresponding stationary distribution[6] would be $p = \langle \frac{1}{2}, \frac{1}{2} \rangle$. and if (2) converges to a value generally the following relationship holds (see § 3.2):

$$p = \lim_{i \to \infty} (p_i)$$

This value represents the probability of a random walk being present on that node at any given iteration and so ascribing this value to a centrality metric is quite intuitive. For example, the node most commonly traversed during a random walk will have the highest corresponding value in p and is hence considered the most central of the graph. This is a particularly appropriate framework for measruing the centrality of interlinked documents, such as pages on the web[7], because navigation involves following links between pages.

3.2 Random Surfer Model

3.2.1 Problems with the Stationary Distribution

The approach to measuring centrality using the stationary distribution in § 3.1 has the following issues

1. Convergence of (2)
   (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 the resulting frequencies of node traversal are not meaningful.
3. Cycles
   (a) A graph that is cyclical may not converge uniquely
      i. Consider for example the graph $(A) \longleftrightarrow (B)$ or taking a directed edge into a closed loop.

3.2.2 Markov Chains

The relationship in (2) is a *Markov Chain* [8] and it is known that the relationship will converge to a value when iterated:

- for a stochastic irreducible markov chain [14, §1.5.5],
- regardless of the initial condition of the process for an *aperiodic* Markov chain [22, §4.4]

[6] Also known as the *Stationary Point*

[7] Although web pages are a classical example of interlinked documents, a lot of modern software use a wiki-like approach that also exhibits this structure (e.g. *Joplin*, *Notable*, *DokuWiki*, `org-mode`, *Markdown* etc.)

[8] A *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 [22, §4.4]

and so these concepts will be explored in order to address the issues with (2).

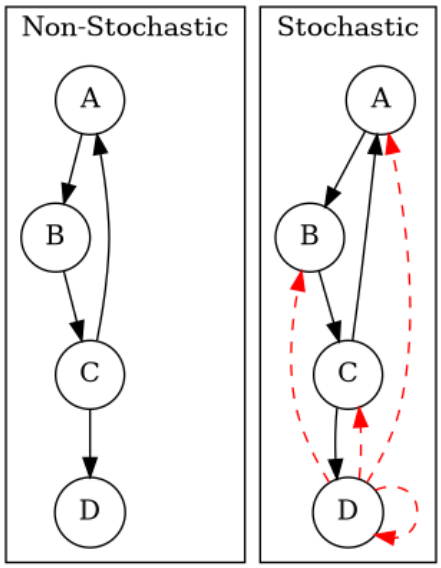

Figure 1: D is a *dangling node*, a dead end during a random walk that acts as a *rank sink*, the corresponding probability transition matrix $(\mathbf{T})$ is hence non-stochastic (and also reducible), Introducing some probability of teleporting from a dead end to any other node as per (3) (denoted in red) will cause $\mathbf{T}$ to be stochastic.

3.2.2 .1 Stochastic If some node had a 0 out-degree 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 node with no outgoing links was followed, as in D in Figure 1, such a node acts as a *rank sink* [22, §4.3] which is a node that accumulates a higher *PageRank* at each iteration and would represent the end of the walk. This could occur in the context of the web by following a hyperlink to a destination with no outgoing links (e.g. a PDF, image or text file would be a destination with no outgoing links).

So to ensure that (2) 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 node could be introduced:

$$\mathrm{S} = \mathrm{T} + \frac{\vec{a} \cdot \vec{1}^{\mathrm{T}}}{n} \tag{3}$$

$$a_i = \begin{cases} 1, & \deg(V_i) = 0 \\ 0, & \deg(V_i) \neq 0 \end{cases} \tag{4}$$

This however would not be sufficient to ensure that (2) would converge, in addition the transition probability matrix must be made irreducible and aperiodic (i.e. primitive). [22]

3.2.2 .2 Irreducible A graph that allows travel from any given node to any other node is said to be irreducible [22], see for example figure 2, this is important in the context of a random walk because only in an irreducible graph can all nodes be reached from any initial condition.

3.2.2 .3 Aperiodic An aperiodic graph has a corresponding transition probability matrix $(\mathbf{A})$ with only one eigenvalue that lies on the unit circle, this is important because $\lim_{k\to\infty}\left(\frac{\mathbf{A}}{r}^{k}\right)$ exists for a non-negative irreducible matrix $\mathbf{A}$ if and only if $\mathbf{A}$ is aperiodic. A graph that is periodic can be made aperiodic by interlinking nodes [9] as shown in Figure 3.

[9] Actually it would be sufficient to merely link one node to itself [22, §15.2] but this isn't very illustrative (or helpful in this context because the graph may still be reducible or non-stochastic)

3.2.2 .4 The Fix

To ensure that the transition probability matrix is primitive (i.e. irreducible and aperiodic) as well as stochastic, instead of merely introducing the possibility to teleport out of dead ends, some probability of teleporting to any node at any time can be introduced $(1-\alpha)$, this would make travel between any nodes possible during the random would as illustrated in Figure 4. This approach is known as the *Random Surfer* model and the corresponding transition probability matrix is given by [23] :

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

This matrix is primitive and stochastic and so will converge [22, §4.5], it is also unfortunately completely dense, making it resource intensive to work with (see § 4.1).

Using this the relation ship in (2) can now be re expressed as:

$$\vec{p_{i+1}} \to \mathbf{S}\vec{p_i} \tag{6}$$

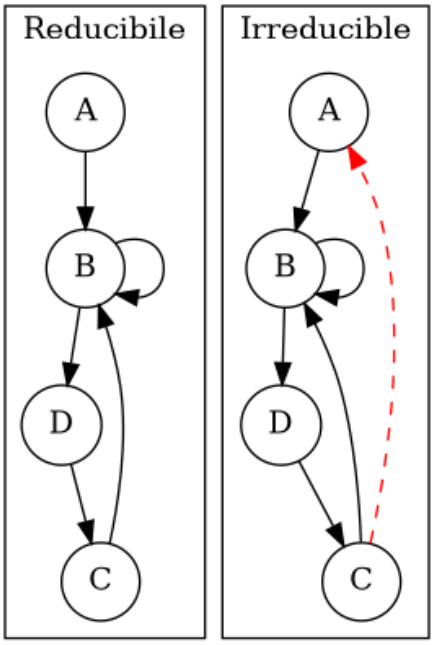

Figure 2: Example of a reducible graph, observe that although C is not a dead end as discussed in § 3.2.2 .1, there is no way to travel from C to A and the graph is hence reducible, by adding an edge to connect $C \to A$ travel is permitted to any node from all nodes and hence the resulting graph is reducible. The resulting graph is also aperiodic (due to the loop on B) and stochastic, so there will be a stationary distribution corresponding to (2).

3.2.3 Limitations

The *Random Surfer* Model can only consider positively weighted edges, it cannot take into account negatively weighted edges which might indicate that links promote aversion rather than endorsement.

3.3 Power walk

The *Power Walk* method is an alternative approach to develop a probability transition matrix to use in place of $\mathbf{T}$ in (2) (and $\mathbf{S}$ in (6)), it presents the benefit of being able to use real-weighted edges (as opposed to strictly positive).

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

This transition probability matrix $(\mathbf{W})$ would be such that the probability of traveling to some node $j \to i$ would be [28]:

$$\mathbf{W}_{i,j} = x\beta^{\mathbf{A_{i,j}}} \tag{7}$$

The random walk is constrained to the graph and so the probability of traveling to one of the nodes generally is 1, hence:

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

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

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

(10)

In this model all nodes are interconnected by some probability of jumping to another node, so much like the random surfer model (5) discussed at 3.2.2 .4 $\mathbf{W}$ will be a primitive stochastic matrix and so if $\mathbf{W}$ was substituted with $\mathbf{T}$ in (2) a solution would exist.

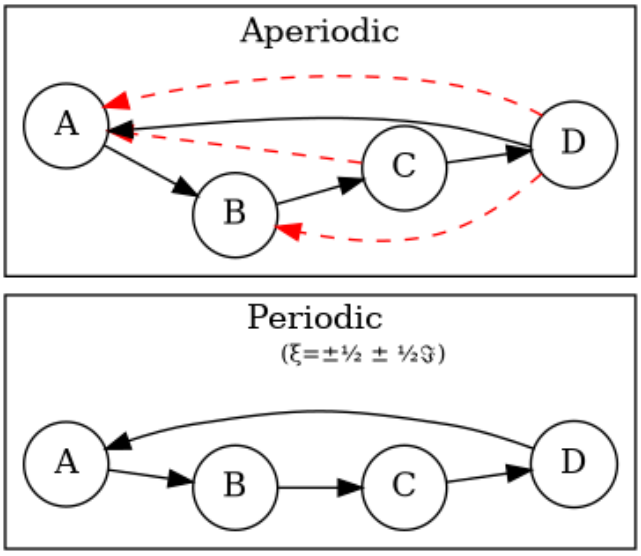

Figure 3: A graph that is periodic with all eigenvalues on the unit circle $\xi = \frac{\sqrt{2}}{2} e^{\frac{\pi i}{4} k}$, by adding in extra edges the graph is now aperiodic (this does not represent the random surfer or power walk models, which would in theory connect every node with some probability)

4 Sparse Matrices

Most Adjacency matrices resulting from web-pages and analogous networks result in sparse adjacency matrices (see figure 16), this is a good thing because it requires far less computational resources to work with a sparse matrix than a dense matrix [22, §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 *Compressed Column Storage*, this involves listing only the non-zero elements and using a *pointer* vector, this as is shown in (11) and table 1. The *Value* column of table 1 contains the non-zero values of the matrix following column-major order [10] and the *Row Index* column indicates which row the value corresponds to. The *Pointer* column is such that the i^{th} entry gives the

[10] *Column-Major Order* means that the values are transcribed top-to-bottom, left-to-right and is the approach that ***R***, *Julia* and *Fortran* take to matrices and vectors, as opposed to *Row-Major Order* which is used by ***C*** and *Python*, this often means that paying mind to the direction of operations when using nested loops to iterate over matrices can have a distinct effect on performance.

index of the *Values* column that corresponds to the first non-zero element in the i^{th} row of the matrix, for example the 3^{rd} row has ϕ as the first non-zero value, this is the 2^{nd} entry in the values column and so 2 is the corresponding value [19]. This is implemented in ***R*** with the `Matrix` package [13].

Table 1: Compressed Column Storage corresponding to (11)

Value	***Row Index***	***Pointer***
1	1	1
ϕ	3	3
7	2	2
13	5	5
2	3	4
π	4	

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 \\ 0 & \phi & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & \pi \\ 0 & 0 & 13 & 0 & 0 \end{bmatrix} \tag{11}$$

4.1 Solving the Stationary Distribution

The relationship in (2) [11] is equivalent to the eigenvalue value problem, where $\vec{p} = \lim_{i\to\infty}(\vec{p_i})$ is the eigenvector [12] $\vec{x}$ that corresponds to the eigenvalue $\xi = 1$:

$$\vec{p}(1) = \mathbf{S}\vec{p} \tag{12}$$

Solving eigenvectors for large matrices can be very resource intensive and so this approach isn't suitable for analyzing large networks, it is however an appropriate method to check against and will be implemented in this report for that purpose.

Upon iteration (2) will converge to a stable stationary point if $\mathbf{T}$ is given by (5) or (7), as discussed in § 3.2.2 .4. This approach of iterating the relationship until it converges to the stationary point is known

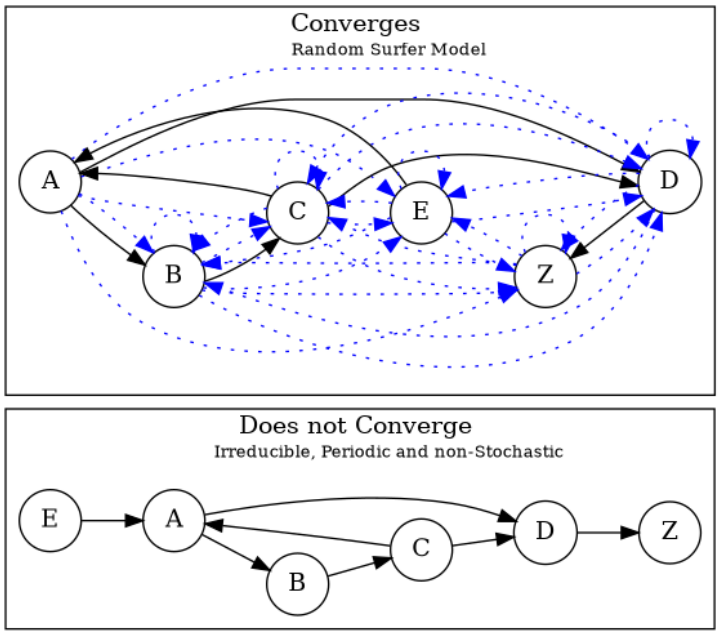

Figure 4: A graph that is aperiodic, reducible and non-stochastic, by applying the random surfer model (5) blue *teleportation* edges are introduced, these may be followed with a probability of $1 - \alpha$

[11] This assumes that the transition probability matrix $\mathbf{T}$ is stochastic and primitive as it would be for $\mathbf{S}$ and $\mathbf{W}$

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

as the power method [24] and
is what in practice must be implemented to solve the stationary point due to how resource intensive it is to solve eigenvalues for large matrices.

As mentioned in §§ 3.2.2 .4 and 3.3 , 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.

5 Implementing the Models

To Implement the models via the power method, first they'll be implemented using an ordinary matrix and then improved to work with sparse matrices and algorithms, the results can be verified against ξ_1.

The implementation has been performed with ***R*** and the preamble is provided in listing 29, an exemplar graph was created in listing 1 and shown in figure 5, and the corresponding adjacency matrix provided in listing 2 (for the sake of comparison this graph was reproduced from [28]).

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

Listing 1: Produce exemplar graph in figure 5

```
        A <- igraph::get.adjacency(g1, names = TRUE, sparse = FALSE)

        ## igraph gives back the transpose
        (A <- t(A))
```

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

Listing 2: Return the Adjacency Matrix corresponding to figure 5

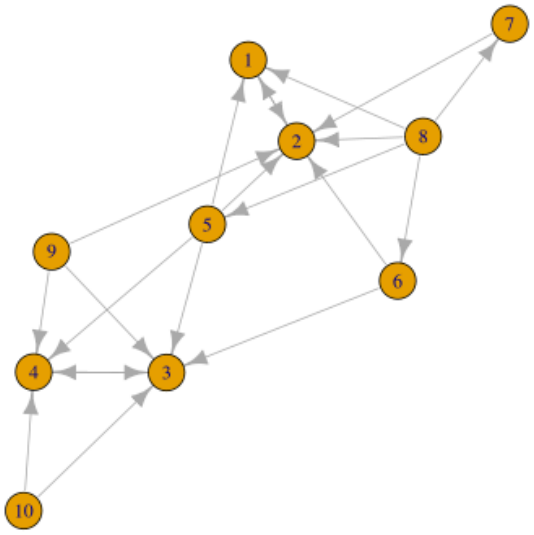

Figure 5: Exemplar graph for *PageRank* examples, produced in listing 1

5.1 Implementing the Random Surfer

5.1.1 Ordinary Matrices

Before implementing the *Random Surfer* model on a sparse matrix, it will be implemented for ordinary matrix objects.

5.1.1 .1 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 $\mathbf{A}$ had nodes with a 0 out-degree, the relationship in (1) would not work, instead columns that sum to 0 would need to remain while all other columns be divided by the column sum to get $\mathbf{T}$. An alternative approach, that addresses this issue using sparse matrices will be presented below at § 5.1.2 but in this case there exists corresponding $\mathbf{T}$ that is stochastic and so it is sufficient to use the relationship at (1), this is shown in listing 3.

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

```
      [,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
```

Listing 3: Solve the Transition Probability Matrix by scaling each column to 1 using matrix multiplication.

```
adj_to_probTrans <- function(A) {
    A %*% diag(1/colSums(A))
}

(T <- adj_to_probTrans(A)) %>% round(2)
```

```
##    [,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
```

5.1.1 .2 Page Rank Random Surfer

Recall from 3.2.2 .4 the following variables of the *Random Surfer* model:

$$\mathbf{B} = \alpha\mathbf{T} + (1-\alpha)\,\mathbf{B}: \tag{13}$$

$$\tag{14}$$

$$\mathbf{B} = \begin{bmatrix} \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix} \tag{15}$$

$$n = ||V|| \tag{16}$$

$$\alpha \in [0,1] \tag{17}$$

These are assigned to ***R*** variables in listing 4.

```
B <- matrix(rep(1/nrow(T), length.out = nrow(T)**2), nrow =
↪  nrow(T))
l <- 0.8123456789

(S <- l*T+(1-l)*B) %>% round(2)
```

```
      [,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
```

Listing 4: Assign Random Surfer Variables, observe the unique value given to `l`, this will be relevant later.

Eigen Value Method The eigenvector corresponding to the the eigenvalue of 1 will be the stationary point, this is shown in listing 5

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 (12):

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

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

```
[1]  1.00000000e+00+0.0000000e+00i -8.12345679e-01+0.0000000e+00i
[3]  8.12345679e-01+0.0000000e+00i -8.12345679e-01+0.0000000e+00i
[5]  5.81488197e-10+0.0000000e+00i -5.81487610e-10+0.0000000e+00i
[7] -6.74980227e-16+0.0000000e+00i  3.21036747e-17+0.0000000e+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
```

Listing 5: Solve the Eigen vectors and Eigen values of the transition probability matrix corresponding to the graph.

```
(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
```

```
(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):

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

```
p_new <- p_new *123456789

while (sum(round(p, 9) != round(p_new, 9))) {
    (p     <- p_new)
    (p_new <- S %*% p)
}

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:

```
(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:

```r
p     <- c(1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
p_new <- S %*% p

while (sum(round(p, 9) != round(p_new, 9))) {
    (p     <- p_new)
    (p_new <- S %*% p)
}

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
```

5.1.2 Sparse Matrices

5.1.2 .1 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 4

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

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

summary(A)
```

```
1000000 x 1000000 sparse Matrix of class "dgCMatrix", with 20 entries
        i      j   x
1  141572  65888  92
2  547799  69135  87
3  368656 123865  87
4  881320 129763 111
5   53637 154979  92
6  193808 192238 100
7  467415 260074  86
8   28105 276311  79
9  481097 316591 102
10 559159 319674  93
11 927895 322174  77
12 562619 372818  82
13 123000 391022  80
14  75909 417462  70
15 309593 457917  78
16 434992 521070 101
17 617821 769436  93
18 673173 811478 103
19 860284 841473 104
20 734100 852938  83
```

As before in section 5.1.1 .1, the probability transition matrix can be found by:

1. Creating adjacency matrix
   (a) Transposing as necessary such that $\mathbf{A}_{i,j} \neq 0$ 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 diagonalized matrix as in (1) may be implemented, using sparse matrices has the advantage however that only non-zero elements will be operated on, meaning that columns that sum to zero can still be used to

create a probability transition matrix [13]

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:

```
sparse_diag <- function(mat) {

## Get the Dimensions
n <- nrow(mat)

## Make a Diagonal Matrix of Column Sums
D <- sparseMatrix(i = 1:n, j = 1:n, x = colSums(mat), dims =
↪  c(n,n))

## Throw away explicit Zeroes
D <- drop0(D)

## Inverse the Values
D@x <- 1/D@x

## Return the Diagonal Matrix
return(D)
}
```

Listing 6: A function that takes in a column $\rightarrow$ row adjacency matrix ($\mathbf{A}$) and returns a diagonal matrix ($\mathbf{D}_{\mathbf{A}}^{-1}$) such that $\vec{1}\mathbf{A}\mathbf{D}_{\mathbf{A}}^{-1} = \vec{1}$

Applying this to the previously created sparse matrix:

[13] Although this matrix will still have columns that sum to zero and will hence be non-stochastic

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

```
1000000 x 1000000 sparse Matrix of class "dgCMatrix", with 20 entries
        i      j           x
1   28105  28105 0.012658228
2   53637  53637 0.010869565
3   75909  75909 0.014285714
4  123000 123000 0.012500000
5  141572 141572 0.010869565
6  193808 193808 0.010000000
7  309593 309593 0.012820513
8  368656 368656 0.011494253
9  434992 434992 0.009900990
10 467415 467415 0.011627907
11 481097 481097 0.009803922
12 547799 547799 0.011494253
13 559159 559159 0.010752688
14 562619 562619 0.012195122
15 617821 617821 0.010752688
16 673173 673173 0.009708738
17 734100 734100 0.012048193
18 860284 860284 0.009615385
19 881320 881320 0.009009009
20 927895 927895 0.012987013
```

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  276311  28105 1
2  154979  53637 1
3  417462  75909 1
4  391022 123000 1
5   65888 141572 1
6  192238 193808 1
7  457917 309593 1
8  123865 368656 1
9  521070 434992 1
10 260074 467415 1
11 316591 481097 1
12  69135 547799 1
13 319674 559159 1
14 372818 562619 1
15 769436 617821 1
16 811478 673173 1
17 852938 734100 1
18 841473 860284 1
19 129763 881320 1
20 322174 927895 1
```

5.1.2 .2 Solving the Random Surfer via the Power Method

Solving eigenvalues for large matrices is not 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 5.1.1 .2) will make $\mathbf{S}$ non sparse, instead some algebra can be used to reduce $\mathbf{B}$ from a matrix into a vector containing only $\frac{1-\alpha}{N}$. The power method is given by:

$$\vec{p} = \mathbf{S}\vec{p} \tag{18}$$

where:

$$\mathbf{S} = \alpha\mathbf{T} + (1-\alpha)\,\mathbf{B} \tag{19}$$

$$\vec{p} = (\alpha\mathbf{T} + (1-\alpha)\,\mathbf{B})\,\vec{p} \tag{20}$$

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

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

$$\mathbf{F} = \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix} \begin{bmatrix} \vec{p_1} \\ \vec{p_2} \\ \vdots \\ \vec{p_m} \end{bmatrix} \tag{22}$$

$$= \begin{bmatrix} (\sum_{i=0}^{m} [p_i]) \times \frac{1}{N} \\ (\sum_{i=0}^{m} [p_i]) \times \frac{1}{N} \\ \vdots \\ (\sum_{i=0}^{m} [p_i]) \times \frac{1}{N} \end{bmatrix} \tag{23}$$

Probabilities sum to 1 and hence: (24)

$$= \begin{bmatrix} \frac{1}{N} \\ \frac{1}{N} \\ \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{bmatrix} \tag{25}$$

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

```r
## Find Stationary point of random surfer
N     <- nrow(A)
alpha <- 0.85
F     <- rep((1-alpha)/N, nrow(A))  ## A nx1 vector of (1-alpha)/N

## Solve using the power method
p     <- rep(0, length.out = ncol(T)); p[1] <- 1
p_new <- alpha*T %*% p + F

## use a Counter to debug
i <- 0
while (sum(round(p, 9) != round(p_new, 9))) {
    p     <- p_new
    p_new <- alpha*T %*% p + F
    (i <- i+1) %>% print()
}

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
```

5.2 Power Walk Method

Recall from 3.3 that the power walk is given by:

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

5.2.1 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

```
    A <- igraph::get.adjacency(g1, names = TRUE, sparse = FALSE)

    ## * Function to create Prob Trans Mat
    adj_to_probTrans <- function(A, beta) {
        B     <- A
        B     <- beta^A           # Element Wise exponentiation
        D     <- diag(colSums(B)) # B is completely dense so D   0
        D_in  <- solve(D)         # Solve returns inverse of matrix
        W     <- B %*% D_in

        return(as.matrix(W))
    }

    beta <-   <- 0.867
    (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
```

Inspect the Eigenvalues:

```
eigen(W, only.values = TRUE)$values %>% round(9)
eigen(W)$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
```

Observe that, unlike the *Random Surfer* Model in listing 5 at § 5.1.1 .2, the relationship between the second eigenvalue and the model parameters is not immediately clear.

```r
## * Power Method
p     <- rep(0, nrow(W))
p[1] <- 1
p_new     <- rep(0, nrow(W))
p_new[2]     <- 1

while (sum(round(p, 9) != round(p_new, 9))) {
    (p     <- p_new)
    (p_new <- W %*% p)
}


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
```

5.2.2 Sparse Matrices

5.2.2 .1 Theory; Simplifying Power Walk to be solved with Sparse Matrices

Modifying the Power Walk method to work with sparse matrices is very similar to the approach implemented for the random surfer in § 5.1.2 .2, observe that all elements of $\mathbf{B}$ are postive values: [14]

$$\left(\mathbf{B} = \beta^{\mathbf{A}}\right) \wedge \left(\mathbf{A}_{i,j}\right) \in \mathbb{R} \implies |\mathbf{B}_{i,j}| > 0 \quad \forall i, j > n \in \mathbb{Z}^{+} \tag{26}$$

Define $\mathbf{O}$ to be a completely dense matrix of 0:

- $\mathbf{O}_{i,j} \coloneqq 0, \quad \forall i, j \leq n$

It can be shown (see (48) at § 5.2.2 .1):

$$\mathbf{O}\mathbf{D}_{\mathbf{B}}^{-1}\vec{p}_i = \left(\vec{\delta}^{\mathrm{T}}\vec{p}_i\right)\vec{1} \tag{27}$$

$$= \texttt{repeat}\left(\vec{\delta} \bullet \vec{p},\ \texttt{n}\right) \tag{28}$$

where:

- $\vec{\delta} = \left(\vec{1}\mathbf{B}\right)$ Represents a scaling vector and is equal to the diagonal entries of $\mathbf{D}_{\mathbf{B}}^{-1}$
- $\vec{\delta^{\mathrm{T}}}\vec{p_i}$ is equal to some real value.

[14] Here the use of exponentiation in (26) is used to denote an element wise exponentiation and not an actual matrix exponential.

This means we can do:

$$\begin{aligned}
\overrightarrow{p_{i+1}} &= \mathbf{T}_{\mathrm{pw}}\overrightarrow{p_i} && (29)\\
&= \mathbf{B}\mathbf{D}_{\mathbf{B}}^{-1}\overrightarrow{p_i} && (30)\\
&= (\mathbf{B}-\mathbf{O}+\mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1}\overrightarrow{p_i} && (31)\\
&= \left((\mathbf{B}-\mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1}+\mathbf{O}\mathbf{D}_{\mathbf{B}}^{-1}\right)\overrightarrow{p_i} && (32)\\
&= (\mathbf{B}-\mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1}\overrightarrow{p_i}+\mathbf{O}\mathbf{D}_{\mathbf{B}}^{-1}\overrightarrow{p_i} && (33)\\
&= (\mathbf{B}-\mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1}\overrightarrow{p_i}+\vec{1}(\overrightarrow{\delta^{\mathrm{T}}}\overrightarrow{p_i}) && (34)\\
&= (\mathbf{B}-\mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1}\overrightarrow{p_i}+\mathtt{rep}(\overrightarrow{\delta^{\mathrm{T}}}\overrightarrow{p_i}) && (35)
\end{aligned}$$

If we let $(\mathbf{B}-\mathbf{O})=\mathbf{B_O}$:

$$\overrightarrow{p_{i+1}} \quad = \mathbf{B_O}\mathbf{D}_{\mathbf{B}}^{-1}\overrightarrow{p_i}+\mathtt{rep}(\overrightarrow{\delta^{\mathrm{T}}}\overrightarrow{p_i})$$

Now solve $\mathbf{D}_B^{-1}$ in terms of $\mathbf{B_O}$:

$$\begin{aligned}
\mathbf{B_O} &= (\mathbf{B}-\mathbf{O}) && (36)\\
\mathbf{B} &= \mathbf{B_O}+\mathbf{O} && (37)
\end{aligned}$$

If we have $\delta_{\mathbf{B}}$ as the column sums of $\mathbf{B}$

$$\begin{aligned}
\delta_{\mathbf{B}}^{-1} &= \vec{1}\mathbf{B} && (38)\\
&= \vec{1}\,(\mathbf{B_O}+\mathbf{O}) && (39)\\
&= \vec{1}\mathbf{B_O}+\vec{1}\mathbf{O} && (40)\\
&= \vec{1}\mathbf{B_O}+\langle n,n,n,...n\rangle && (41)\\
&= \vec{1}\mathbf{B_O}+\vec{1}n && (42)\\
\delta_{\mathbf{B}} &= 1/(\mathtt{colSums}(\mathbf{B_O})+\mathtt{n}) && (43)
\end{aligned}$$

Then if we have $\mathbf{D_B}=\mathtt{diag}\,(\delta_{\mathbf{B}})$

$$\begin{aligned}
\mathbf{D}_{\mathbf{B}}^{-1} &= \mathrm{diag}\left(\delta_{\mathbf{B}}^{-1}\right) && (44)\\
&= \mathtt{diag}\,(\mathtt{ColSums}\,(\mathbf{B_0})+n)^{-1} && (45)
\end{aligned}$$

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

$$\vec{p_{i+1}} = \mathrm{B_O}\ \mathrm{diag}\left(\vec{1}\mathbf{B_O} + \vec{1}n\right)\vec{p_i} + \vec{1}\vec{\delta}^{\mathrm{T}}\vec{p_i} \tag{46}$$

in terms of ***R***:

```
p_new <- Bo %*% diag(colSums(B)+n) %*% p + rep(t( ) %*% p, n)

# It would also be possible to sum the element-wise product
(t( ) %*% p) == sum( * p)

# Because R treats vectors the same as a nX1 matrix we could also
# perform the dot product of the two vectors, meaning the following
# would be true in R but not true generally

(t( ) %*% p) == ( %*% p)
```

Solving the Background Probability To show the identity in (27) define $\vec{\delta}$, as before, as the inverse of the column sums of $\mathbf{B}$:

$$\vec{\delta} = \texttt{colSums}(\mathbf{B})^{-1} = \vec{1}\mathbf{B}\boldsymbol{B}$$

Then we have:

$$\begin{aligned}
\mathbf{OD}_{\mathbf{B}}^{-1}\overrightarrow{p_i} &= \begin{pmatrix} 1 & 1 & 1 & \\ 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} \frac{1}{\delta_1} & 0 & 0 & \\ 0 & \frac{1}{\delta_2} & 0 & \dots \\ 0 & 0 & \frac{1}{\delta_{13}} & \\ & \vdots & & \ddots \end{pmatrix} \begin{pmatrix} p_{i,1} \\ p_{i,2} \\ p_{i,3} \\ \vdots \end{pmatrix} \\
&= \begin{pmatrix} \frac{p_{i,1}}{\delta 1} & + & \frac{p_{i,2}}{\delta_2} & + & \frac{p_{i,3}}{\delta_3} & \\ \frac{p_{i,1}}{\delta 1} & + & \frac{p_{i,2}}{\delta_2} & + & \frac{p_{i,3}}{\delta_3} & \dots \\ \frac{p_{i,1}}{\delta 1} & + & \frac{p_{i,2}}{\delta_2} & + & \frac{p_{i,3}}{\delta_3} & \\ & & \vdots & & & \ddots \end{pmatrix} \\
&= \begin{pmatrix} \sum_{k=1}^{n}[p_{i,k}\delta_i] \\ \sum_{k=1}^{n}[p_{i,k}\delta_i] \\ \sum_{k=1}^{n}[p_{i,k}\delta_i] \\ \vdots \end{pmatrix} \\
&= \begin{pmatrix} \overrightarrow{\delta^{\mathbf{T}}}\overrightarrow{p_i} \\ \overrightarrow{\delta^{\mathbf{T}}}\vec{p}_i \\ \overrightarrow{\delta^{\mathbf{T}}}\vec{p}_i \\ \vdots \end{pmatrix} \\
&= \overrightarrow{\delta^{\mathbf{T}}}\overrightarrow{p_i} \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{pmatrix} \\
&= (\overrightarrow{\delta^{\mathbf{T}}}\overrightarrow{p_i})\vec{1} \qquad (47) \\
&= \texttt{repeat}(\overrightarrow{\delta}\overrightarrow{p_i}, \texttt{n}) \qquad (48)
\end{aligned}$$

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
$\vec{\delta^{\mathrm{T}}}\vec{p_i}$	$\vec{\delta} \bullet \vec{p_i}$

5.2.2 .2 Practical; Implementing the Power Walk on Sparse Matrices

The Power Walk method, using the theory from § 5.2.2 .1, can be implemented thusly:

1. Define a function to create a diagonal sparse matrix.

   Unlike the Random Surfer model the diagonal scaling matrix will always be given by $\mathbf{D}_B^{-1} = \mathbf{B}\ \text{diag}\left(\frac{1}{\vec{1}\mathbf{B}}\right)$ 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 6 is more general and has already been implemented.

1. Define the matrix $\mathbf{B}$

```
    A       <- Matrix::Matrix(A, sparse = TRUE)
    B       <- A
    B@x     <-  ^(A@x)
    B       <- A
    B        <-  ^A

    Bo      <- A

    # These two approaches are equivalent
    Bo@x    <-  ^(A@x) -1    # This in theory would be faster
    # Bo      <-  ^(A) -1
    # Bo      <- drop0(Bo)
    n <- nrow(A)

    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
```

```
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

   Observe that there is no need to worry about any terms of $\delta_{\mathbf{B}} = \texttt{colSums}\,(\texttt{B\_o}) + \texttt{n}$ being 0:

```
( 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
```

```
( 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

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

4. Implement the Loop

```r
    ## ** Implement the Power Walk
    ## *** Set Initial Values
    p_new  <- rep(1/n, n)  # Uniform
    p      <- rep(0, n)    # Zero
           <- 10^(-6)
    ## *** Implement the Loop

    while (sum(abs(p_new - p)) >  ) {
        (p <- as.vector(p_new)) # P should remain a vector
        sum(p <- as.vector(p_new)) # P should remain a vector
        p_new  <- T %*% p + rep(t( B) %*% p, n)
    }
    ## ** Report the Values
    print(paste("The stationary point is"))
    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
```

6 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* [15], to load this package use the `devtools` library as shown in listing .

```
        library(devtools)
        library(Matrix)
        library(tidyverse) # Maybe, TODO check if this is used, I don't
        ↪  think it is

        if (require("PageRank")) {
            library(PageRank)
            }else{
            devtools::install_github("ryangreenup/PageRank")
            library(PageRank)
            }
```

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

Attaching package: 'PageRank'
```

Listing 7: Load the *PageRank* package which consists of the functions from 5

[15] https://github.com/RyanGreenup/PageRank

Part II

Investigating ξ_2

7 The Second Eigen Value

7.1 Convergence

The second eigenvalue (ξ_2) refers to the second largest[16] eigenvalue of the probability transition matrix for some graph. The further $|\xi_2|$ is from the value 1 faster the *PageRank* method will converge to the stationary point [9], this can be seen by diagonalising the matrix $\mathbf{T}$ and observing that a smaller value of ξ_2 implies all other eigenvalues, other than the first, are smaller still and hence Λ will reduce to a rank 1 matrix quicker if $|\xi_2|$ is smaller:

$$\vec{p} = \lim_{k\to\infty}\left[\mathbf{T}^k\vec{p}_0\right] \tag{49}$$

$$= \lim_{k\to\infty}\left[\left(\mathbf{V}\Lambda\mathbf{V}^{-1}\right)^k\right]\vec{p_0} \tag{50}$$

$$= \mathbf{V}\lim_{k\to\infty}\begin{bmatrix}\xi_1^k & 0 & 0 & \\ 0 & \xi_2^k & 0 & \cdots \\ 0 & 0 & \xi_3^k & \\ & \vdots & & \ddots\end{bmatrix}\mathbf{V}^{-1}\vec{p}_0 \tag{51}$$

7.2 Stability

The first eigenvalue of $\mathbf{T}$ will be 1 and so the smaller $|\xi_2|$ is, the further apart the first and second eigenvalus are. The greater the distance between the first and second eigenvalue is, the more stable the stationary point will be to perturbations in the corresponding graph [27].

7.3 Discussion

With respect to the Random Surfer model, it has been shown that ξ_2 is bounded above by the smoothing constant α and if the corresponding graph has more than 1 closed subgraph is equal to α [18]. It has also been shown that the power method will always converge $\forall\alpha < 1$ [6] and that an α closer to the value of 1 does not necessarily correspond to a more meaningful ranking [7], hence, given the upper bound of $\xi_2 \leq \alpha$, the value of α can be tuned away from 1 in order to improve the convergence and stability of the *PageRank*.

Being able to determine whether or not any such a relationships exists with the β value for the Power Walk method is important because it can be used determine how to modify the parameters in favour of a more stable solution that can be solved more quickly.

In order to investigate this a variety of graphs will be simulated to observe the behaviour of the method parameters and ξ_2.

8 Erdos Renyi Graphs

[16] with respect to magnitude

8.1 Introduction

The *Erdos Renyi* game, first published in 1959 [29] creates a graph by assuming that the number of nodes is constant and the probability of interlinking these nodes is equal. This approach does not produce graphs consistent with networks such as the web (see 9) or wikis, however, Sampling these graphs will provide a broader picture for the overall behavior of ξ_2 over a broad range of graphs with respect to the parameters of the *Power Walk* method.

8.2 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 input values were specified in listing 8, a function that builds a data frame with the second eigenvalue, density, determinant and trace was constructed in listing 9, a function to map this over the Cartesian product of the input variables was created in listing 10 and finally a correlation plot was generated in listing 11 shown in figure 6. The correlation plot in figure 6 considers the Spearman correlation coefficient, which is concerned with a monotone relationship between the variables, this is appropriate here as the complex relationship between the variables is likely non-linear.

```
        # Generate Constants
        p         <- seq(from = 0.01, to = 0.99, length.out = 5)
        beta      <- seq(from = 1   , to = 20  , length.out = 20)
        size      <- seq(from = 100 , to = 1000, length.out = 5) %>%
        ↪  rev() # Big First
        input_var <- expand.grid("p" = p , "beta" = beta, "size" = size)

        # Print out a sample of all the rows
        input_var[sample(1:nrow(input_var), 6),]
```

```
              p beta size
237 0.255    8  550
13  0.500    3 1000
438 0.500    8  100
456 0.010   12  100
209 0.745    2  550
384 0.745   17  325
```

Listing 8: A data frame consisting of input variables to be used to generate *Erdos Renyi* graphs.

The correlation plot shows a moderate relationship between size, the uniform probability of linking two nodes (p) and the size of the graph with ξ_2. It's worth noting that $p = \text{mean}(\mathbf{A})$.

To inspect these relationships more closely a scatter-plot matrix was produced in figure 12 and shown in figure 7. The matrix indicates a negative relationship between ξ_2 with p and size and a positive relationship between β and ξ_2, these relationships also indicate some degree of interaction.

There also appears to be some relationship between β and the trace of the matrix, this won't be considered for the moment because the trace is not a parameter of the power walk method.

```r
random_graph <- function(p, beta, size) {
    g1      <- igraph::erdos.renyi.game(n = size, p)
    A       <- igraph::get.adjacency(g1) # Row to column
    A       <- Matrix::t(A)

#    A_dens <- mean(A) # Very Slow, equal to p
    T       <- PageRank::power_walk_prob_trans(A, beta = beta)
    tr      <- sum(diag(T))
    e2      <- eigen(T, only.values = TRUE)$values[2] # R orders by
    ↪  descending magnitude
    return(c(abs(e2), tr))
}
```

Listing 9: return ξ_2 from a randomly produced Erdos Renyi game

```r
    sim_graphs <- function(filename, p, beta, size) {

        input_var <- expand.grid("p" = p , "beta" = beta, "size" =
        ↪  size)

        nc    <- length(random_graph(1, 1, 1))
        Y     <- matrix(ncol = nc, nrow = nrow(input_var))
        for (i in 1:nrow(input_var)) {
            X     <- as.vector(input_var[i,])
            Y[i,] <-  random_graph(X$p, X$beta, X$size)
            print(i/nrow(input_var))
        }
        if (sum(abs(Y) != abs(Re(Y))) == 0) {
            Y     <- Re(Y)
        }
        Y     <- as.data.frame(Y); colnames(Y) <- c("eigenvalue2",
        ↪  "trace")
        data <- cbind(input_var, Y)
        saveRDS(data, file = filename)
        return(data)
    }
```

Listing 10: A function to return a data-frame of simulated graphs using the `random_graph` function in listing 9.

```r
filename <- "resources/erdosData.rds"

if (file.exists(filename)) {

    data <- readRDS(filename)

} else {
    data <- sim_graphs(filename, p, beta, size)

}
cormat = cor(data, method = 'spearman')
rownames(cormat) <- colnames(cormat) <- c("Prob\nEdges", " ",
↪  "Size", " ", "Trace")
corrplot(cormat, method = "ellipse", type = "lower")
```

Listing 11: Produce a correlation plot Created from a data-frame constructed from the values assigned in listing 8 by using the function defined in listing 9, see figure 6.

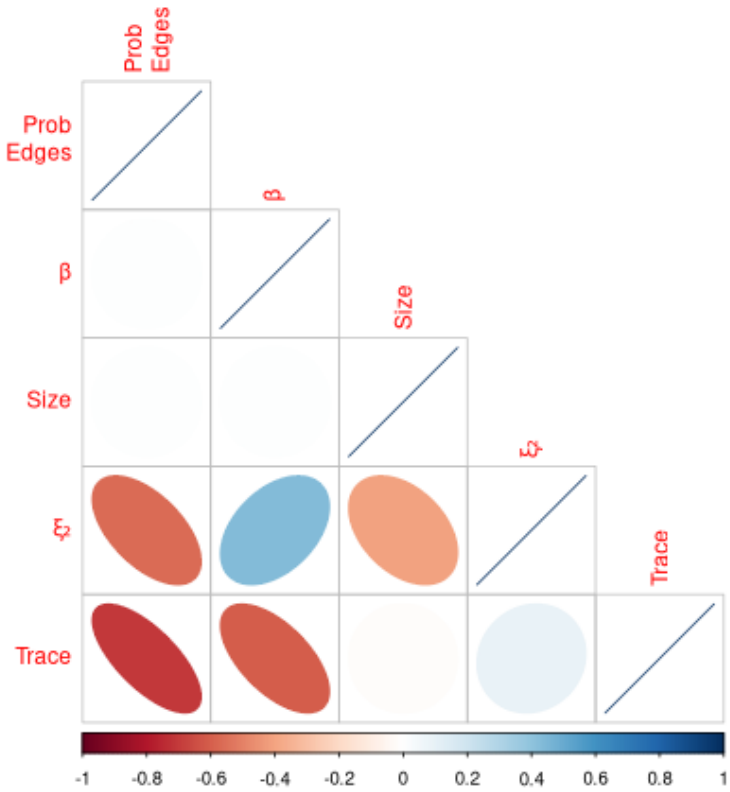

Figure 6: Correlation plot of parameters of *Power Walk* parameters and graph properties.

```
plot(data, labels = c("Mean\nEdges", " ", "Size", " ", "Trace"))
```

Listing 12: Plot Model Diagnostics for data corresponding to graphs, see figure 7

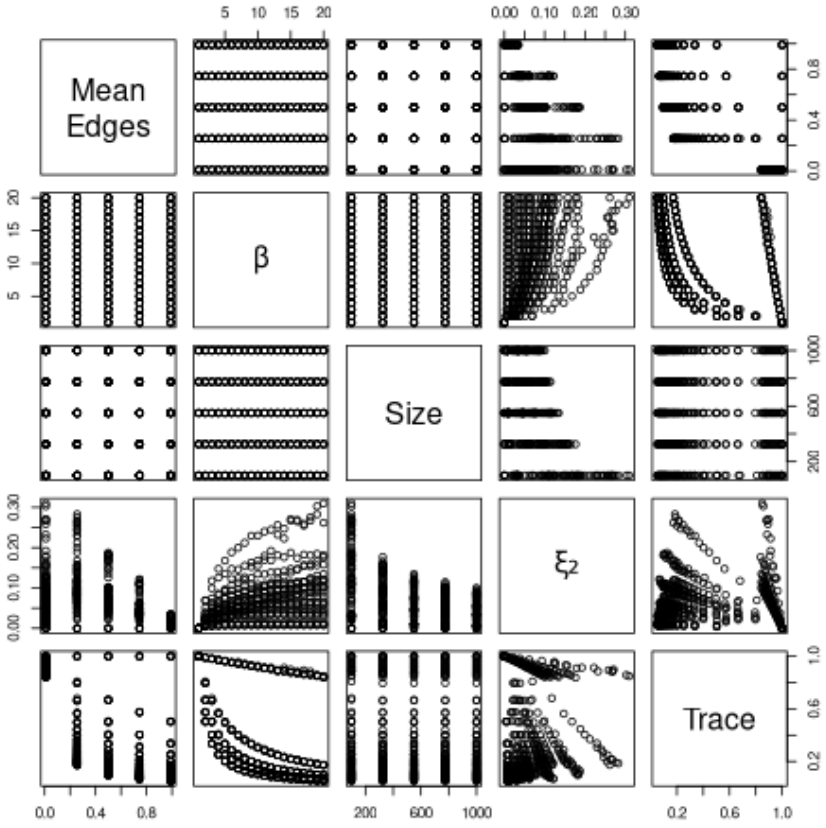

Figure 7: Plot of the Density of the Adjacency Matrix, Beta Value of the Power Walk method with ξ_2 represented by the vertical z axis.

By looking at the ξ_2 value for a variety of p values with a constant size a pattern may be emerge, a corresponding data set is produced in 13, a plot produced in listing 14 and the corresponding plot shown in figure **??**.

```
        filename <- "resources/erdosData_constant_size.rds"

        if (file.exists(filename)) {
            data2 <- readRDS(filename)
        } else {
            p    <- seq(from = 0.01, to = 0.5, length.out = 5   )
            beta <- seq(from = 1   , to = 10  , length.out = 1000)
            size <- 1000

            data2 <- sim_graphs(filename, p, beta, size)
        }
```

Listing 13: Produce a data frame of graphs corresponding to a constant size and link density.

```
        ggplot(data2, mapping = aes(col = factor(p), x = beta, y =
        ↪  eigenvalue2)) +
        geom_point(size = 0.5) +
        stat_smooth() +
        scale_size_continuous(range = c(0.1,1)) +
        labs(x = "Beta", y = TeX("Second Eigenvalue"), title =
        ↪  TeX("Second Eigenvalue given Matrix Density for 100 nodes")
        ↪  ) +
        guides(col = guide_legend("Link Density"))  +
        theme_bw()
```

Listing 14: Produce a plot of ξ_2 for a constant size and a few link densities.

This resulting trends are well defined and positive, the data appears to have a concave down curvature and non constant variance, a square root transform is applied to the in listing 15 and shown in figure 9.

The root transform did not significantly normalize the variance or make the relationship much more linear. The noise would result from using p as a probability in forming the graphs as opposed to strictly enforcing the link density, this variance should be constant so this transform is not desirable.

Both plots in figures 9 **??** appear to indicate a more linear relationship between ξ_2 and β for smaller link densities and β values.

In order to determine whether or not this relationship persists over many sizes a data set containing the relationship between ξ_2 and β was produced for sizes ranging from 100 to 1000 nodes in listing 16, a corresponding plot was produced in listing 17 and shown in figure 10.

This relationship appears to have a slight curvature, this curvature is very slight so a root transform is more appropriate than a log transform, this is implemented in listing 18 and shown in 11.

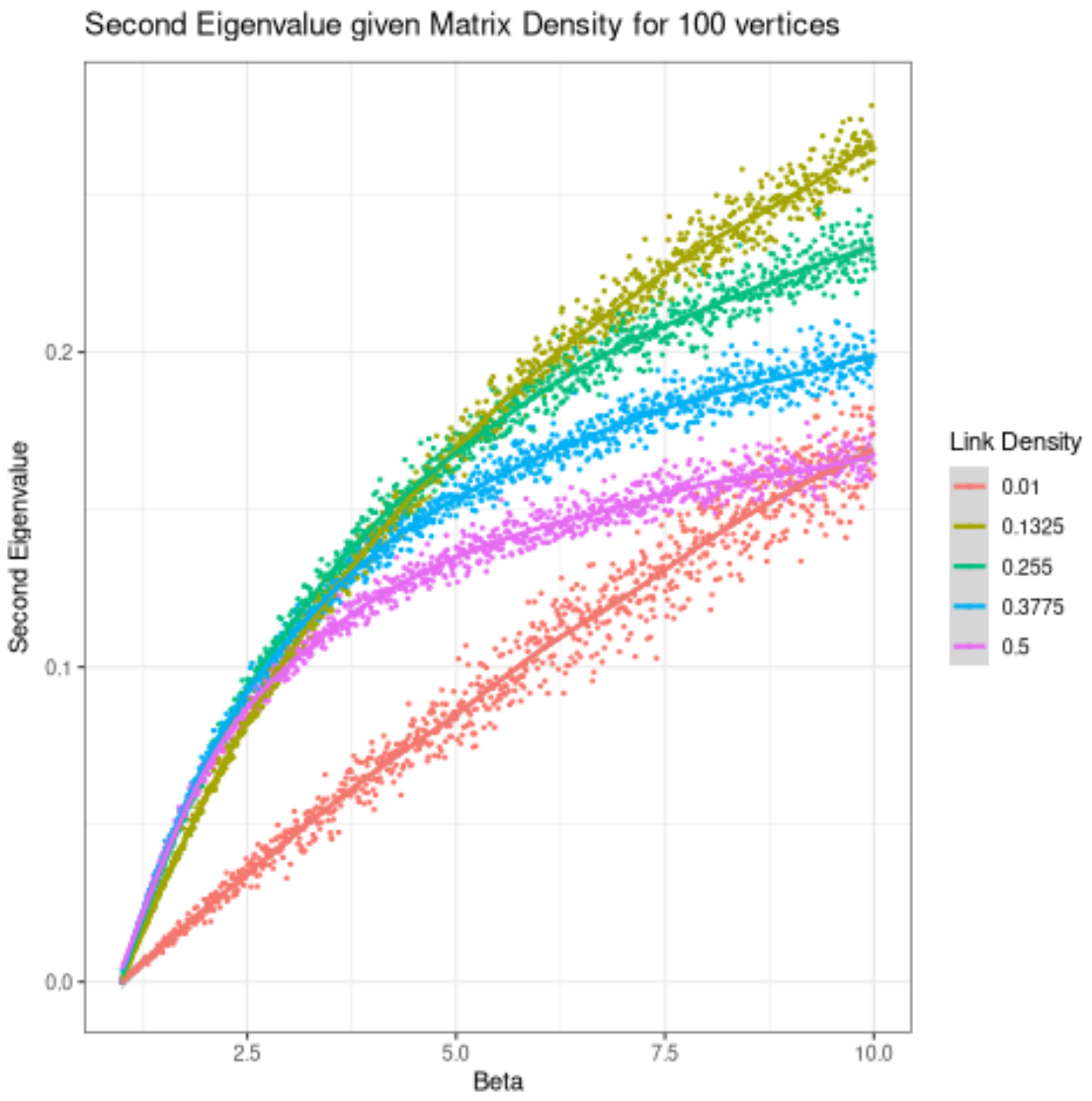

Figure 8:
Second Eigenvalue (ξ_2) for a variety of generated *Erdos Renyi* graphs

```
ggplot(data2[30:nrow(data2),], mapping = aes(col = factor(p), x = sqrt(beta), y = eigenvalue2)) +
geom_point(size = 0.5) +
stat_smooth() +
scale_size_continuous(range = c(0.1,1)) +
labs(x = "sqrt( )", y = TeX("Second Eigenvalue"), title = TeX("Second Eigenvalue given Matrix Density for 100 nodes") ) +
guides(col = guide_legend("Link Density"))  +
theme_bw()
```

Listing 15: Create a plot of the randomly generated *Erdos-Renyi* graphs

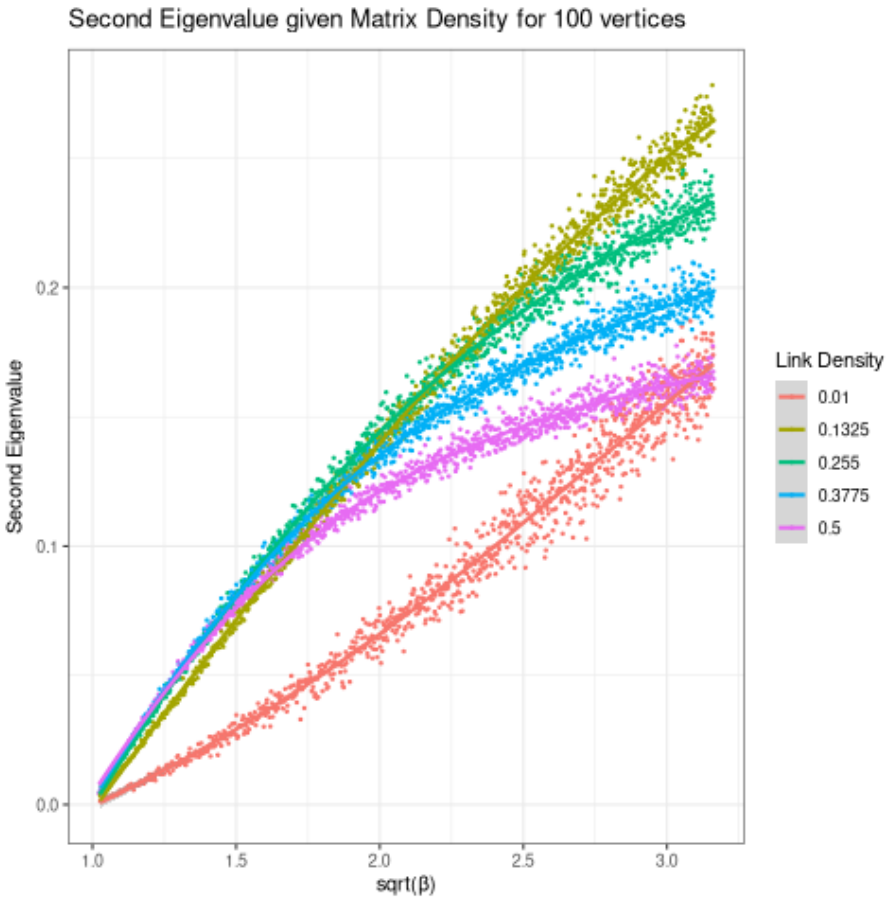

Figure 9:
Second Eigenvalue plotted against ($\sqrt{\xi_2}$) for a variety of generated *Erdos Renyi* graphs

```
filename <- "resources/erdosData_constant_dens.rds"

if (file.exists(filename)) {
    data2 <- readRDS(filename)
} else {
    p    <- 5/100
    beta <- seq(from = 1   , to = 10   , length.out = 1000)
    size <- seq(from = 100, to = 1000, length.out = 5    )

    data2 <- sim_graphs(filename, p, beta, size)
}
```

Listing 16: Produce a data set of a variety of sizes ranging from 100 to 1000 nodes.

```
        ggplot(data2, mapping = aes(col = factor(size), x = (beta), y =
        ↪  eigenvalue2)) +
        geom_point(size = 0.5) +
        stat_smooth() +
        scale_size_continuous(range = c(0.1,1)) +
        labs(x = " ", y = TeX("Second Eigenvalue"), title = TeX("Second
        ↪  Eigenvalue for uniform degree"), subtitle = "mean degree ÷
        ↪  size = 5%" ) +
        guides(col = guide_legend("Size"))  +
        theme_bw()
```

Listing 17: Create a plot of of randomly generated *Erdos-Renyi* graphs for a variety of sizes

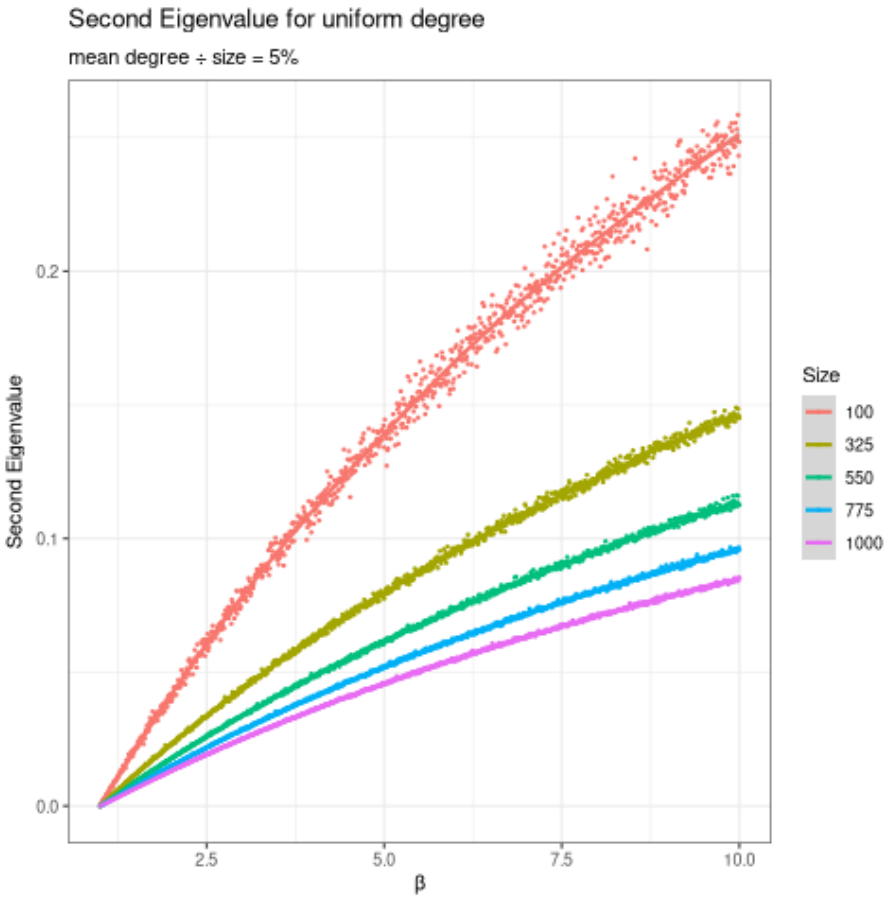

Figure 10: Second eigenvalue plotted against $\{beta$
for a variety of differently sized randomly generated *Erdos-Renyi* graphs

```
        ggplot(data2, mapping = aes(col = factor(size), x = sqrt(beta),
        ↪  y = eigenvalue2)) +
        geom_point(size = 0.5) +
        stat_smooth() +
        scale_size_continuous(range = c(0.1,1)) +
        labs(x = TeX("\\sqrt{ }"), y = TeX("Second Eigenvalue"), title
        ↪  = TeX("Second Eigenvalue for uniform degree"), subtitle =
        ↪  "mean degree ÷ size = 5%" ) +
        guides(col = guide_legend("Size"))  +
        theme_bw()
```

Listing 18: Create a plot of of randomly generated *Erdos-Renyi* graphs for a variety of sizes, plotted against $\sqrt{\beta}$

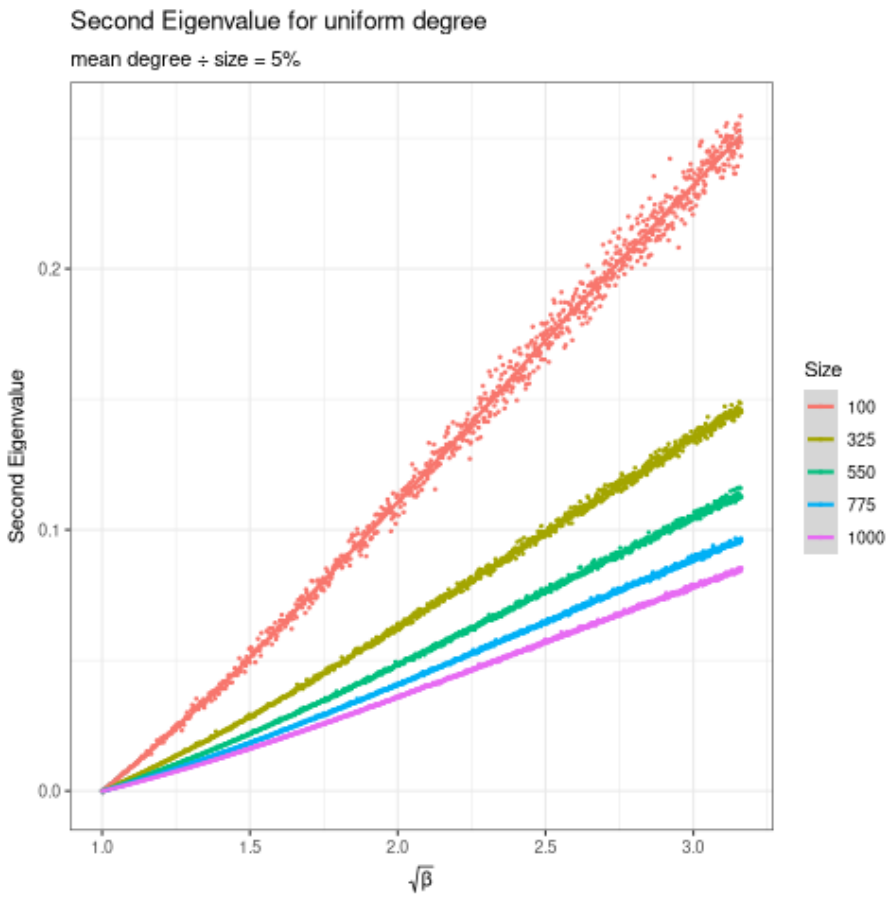

Figure 11: Second eigenvalue plotted against $\sqrt{beta}$ for a variety of differently sized randomly generated *Erdos-Renyi* graphs

The plot shown in 11 demonstrates a very strong linear relationship between ξ_2 and $\sqrt{\beta}$ for a variety of constant link densities, the variance however is still non-constant over $\sqrt{\beta}$.

8.3 Conclusion

The plots seem to indicate that for random graphs generally, there is a positive relationship between β and ξ_2.

9 Barabasi Albert Graphs

9.1 Theory

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 Barabassi-Albert graph [4].

The Erdos Renyi game assumes that the number of nodes is constant from beginning to end, clearly this is not true for networks such as the web. Consider a graph constructed node by node where each time a new node is introduced it is randomly connected to m other nodes with a constant probability. Despite the probability of connecting to any given node being constant as in the Erdos Renyi game, such a graph will be different and favor nodes introduced earlier with respect to the number edges. This shows that the presence of network growth is an import feature in modeling networks.

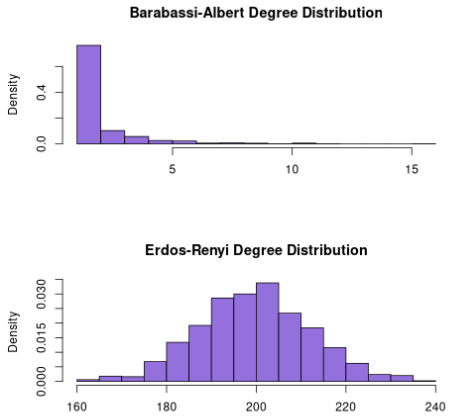

Figure 12: Histograms of degree distribution of *Erdos-Renyi* and *Barabassi-Albert* graphs produced in listing 19

Simply considering growth however is not sufficient to simulate graphs with a degree distribution consistent with the web [32, Ch. 7].

When introducing a new node, the probability of linking to any other node is not uniformly random rather it would be expected that links to more popular nodes would be made. A simple approach is to presume that the probability of linking from one node to another is proportional to the number of links, i.e. a node with twice as many links will be twice as likely to receive a link from a new node.

These two distinguishing features departing from the *Erdos Renyi* model, known as *Growth* and *Preferential Attachment*, are what set the Barabassi-Albert model apart from the Erdos-Renyi model and why it is better suited to modeling networks such as the web. [2, Ch. 7]

The Barabassi Albert model implements this scale free approach and predicts the following model of the degree of a node:

$$\mathrm{P}\left(k\right) \propto m^2 k^{-\gamma}$$

Where $\mathrm{P}\left(k\right)$ is the probability of a node having k edges, m is the nodes added per step and γ is a constant.

The Barabassi Albert model can be solved analytically to predict that values of $\gamma = 3$ would represent networks such as the internet, this value is independent of m. In practice values of $\gamma_{\text{in}} = 2.1$ and $\gamma_{\text{out}} = 2.45$ are observed [4], such a graph is said to be *scale free* [22, §10.7.2].

```
        layout(matrix(1:2, nrow = 2))
        col  <- "Mediumpurple"
        n <- 1000
        hist(
        igraph::degree(igraph::sample_pa(n, 0.2)),
        binwidth = 0.3,
        xlab = "",
        main = "Barabassi-Albert Degree Distribution",
        col = col, freq = FALSE
        )

        hist(igraph::degree(igraph::erdos.renyi.game(n, 0.2)),
            main= "Erdos-Renyi Degree Distribution",
            col = col,
            binwidth = 0.3,
            xlab = "",
            freq = FALSE )
```

Listing 19: Simulate Erdos-Renyi and Barabassi-Albert graphs in order to measure the degree distribution, shown in 12

9.2 Modeling

At each step of the Barabassi-Albert algorithm, one node is added and linked to m other nodes with a probability proportional to the in-degree of the destination vertex raised to the power of γ

This means that the average of the unweighted adjacency matrix would be given by $p = \text{mean}(\mathbf{A}) \in \left[\frac{m}{n}, \frac{m+1}{n}\right]$ and the relationship identified in § 8 can be determined from the parameters of a scale-free network.

In practice though measuring m (as the mean out degree) will be imprecise and the adjacency matrix will be weighted any way, so taking the mean of the weighted adjacency matrix would still be necessary to implement this type of model in practice.

A function to generate a random Barabassi-Albert graph and return the value of ξ_2 corresponding to the Power Walk method is provided in listing 20, another function to map this over the Cartesian product of input variables is provided in listing 21 and finally a dataset containing 9 different link densities was created in listing 22.

This data is plotted by listing 23 and shown in figure 13.

Due to the relationship between size and density in the barabassi albert model this approach seems to work even better for BA graphs, which is precisely the type of graph where we might anticipate the use of the power walk method.

```r
random_graph_pa <- function(m, beta, size) {
    g1 <- igraph::sample_pa(n = size, power = 3, m = m)
    A <- igraph::get.adjacency(g1) # Row to column
    A <- Matrix::t(A)

#     A_dens <- mean(A)
    T      <- PageRank::power_walk_prob_trans(A, beta = beta)
    tr     <- sum(diag(T))
    e2     <- eigen(T, only.values = TRUE)$values[2] # R orders
    ↪  by descending magnitude
    return(c(abs(e2), tr))
}
```

Listing 20: A function to build a random graph using the Barabassi-Albert Model and return the value of ξ_2 corresponding to the *Power Walk* method.

```r
sim_graphs_pa <- function(filename, m, beta, size) {

    input_var <- expand.grid("m" = m, "beta" = beta, "size" =
    ↪  size)

    nc <- length(random_graph_pa(1, 1, 1))
    Y  <- matrix(ncol = nc, nrow = nrow(input_var))
    for (i in 1:nrow(input_var)) {
        X     <- as.vector(input_var[i,])
        Y[i,] <-  random_graph_pa(X$m, X$beta, X$size)
        print(i/nrow(input_var))
    }
    if (sum(abs(Y) != abs(Re(Y))) == 0) {
        Y <- Re(Y)
    }
    Y <- as.data.frame(Y); colnames(Y) <- c("eigenvalue2",
    ↪  "trace")
    data2 <- cbind(input_var, Y)
    saveRDS(data2, filename)
    return(data2)
}
```

Listing 21: Return ξ_2 values by mapping the `random_graph_pa` function from listing 20 over the Cartesian product of input variables.

```
filename <- "resources/BAData.rds"

if (file.exists(filename)) {
    data2 <- readRDS(filename)
} else {
    m    <- seq(from = 1, to = 9, length.out = 3)
    beta <- seq(from = 1, to = 6, length.out = 30)
    sz   <- seq(from = 100, to = 500, length.out = 3) %>% rev()
    # Big numbers first

    input_var <- expand.grid("m" = m, "beta" = beta, "size" =
    sz)

    data2 <- sim_graphs_pa(filename, p, beta, size)
}
```

Listing 22: map the `sim_graphs_pa` function over the Cartesian product of various input variables

```
data2$p <- round((data2$m/data2$size), 2)

ggplot(data2, mapping = aes(col = factor(p), x = beta, y =
eigenvalue2)) +
geom_point(size = 0.5) +
stat_smooth(method = 'lm', size = 0.4) +
scale_size_continuous(range = c(0.1,1)) +
labs(x = "Beta", y = TeX("Second Eigenvalue"), title =
TeX("Second Eigenvalue given Matrix Density") ) +
guides(col = guide_legend("Link Density (by m/n)"))  +
theme_bw()
```

Listing 23: Plot $\xi_2 \sim \beta$ for discrete values of p, shown in figure 13

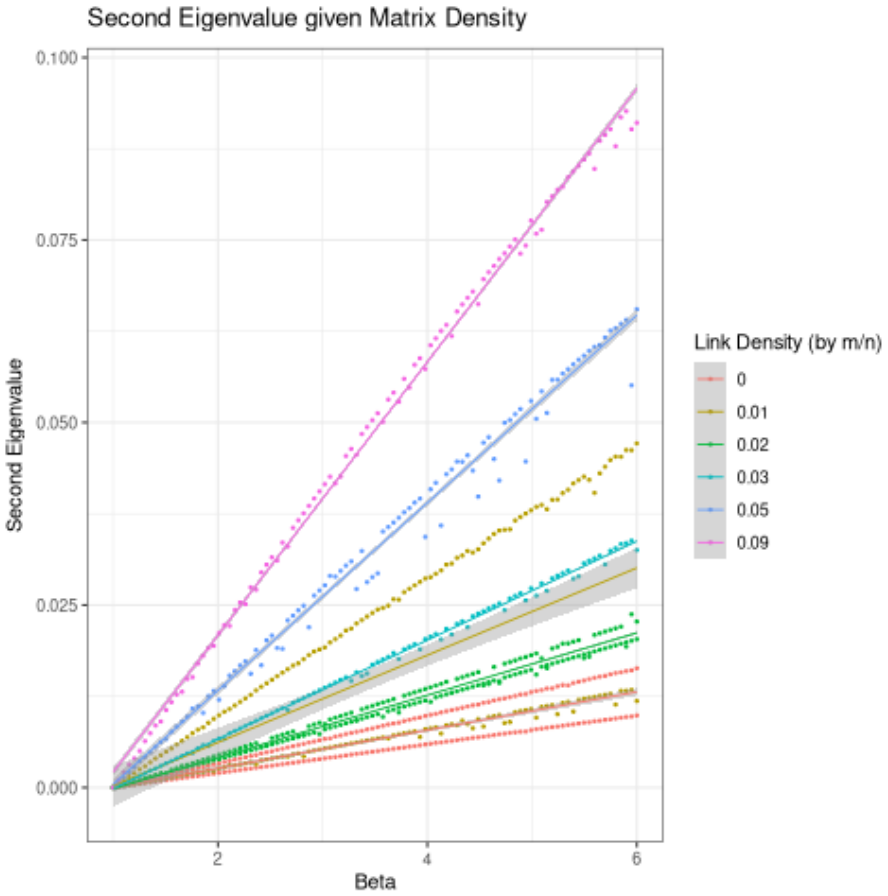

Figure 13: Plot of $\xi_2 \sim \beta$ for various values of $\text{mean}(\mathbf{A})$ corresponding to the *Power Walk* method. Produced by listing 23

The low link densities are such that there is no need for a log/root transform either, such a transform was found to be unsuitable in § 8.2 so this is encouraging.

To determine the relationship that the link density p may have, the value of β will be fixed and the behavior of $\xi_2 \sim p$ can be investigated. A corresponding data set is produced in listing 24, plotted in listing 25 and shown in figure 14.

```
filename <- "resources/BAData_constant_size.rds"

if (file.exists(filename)) {
    data2 <- readRDS(filename)
} else {
    m       <- seq(from = 1, to = 9, length.out = 15)
    beta    <- 5
    size    <- seq(from = 100, to = 500, length.out = 7)

    data2 <- sim_graphs_pa(filename, m, beta, size)
}
```

Listing 24: l:baData$_{\text{constantsize}}$

```
data2$p <- round((data2$m/data2$size), 2)

names(data2)
ggplot(data2, mapping = aes(col = factor(round(size)), x = p, y
↪  = eigenvalue2)) +
geom_point(size = 0.5) +
stat_smooth(method = 'lm', size = 0.4, se = FALSE) +
scale_size_continuous(range = c(0.1,1)) +
labs(x = "p", y = TeX("Second Eigenvalue"), title = TeX("Second
↪  Eigenvalue given Matrix Density") ) +
guides(col = guide_legend("Size"))  +
theme_bw()
```

Listing 25: l:ba$_{\text{dataconstantsizeplot}}$

This seems to suggest that the effect on the eigenvalue that p has depends on the size of the graph, so p and size interact with respect to ξ_2.

Hence an appropriate model will consider the value of p, $p*n$ and β, where n is the size of the graph.

9.2.1 Model

A simple model to consider the variables found to influence ξ_2 is multiple linear regression, although it is a very simple model it is easy to interpret and may offer insights into the behavior of ξ_2 in response to different graphs.

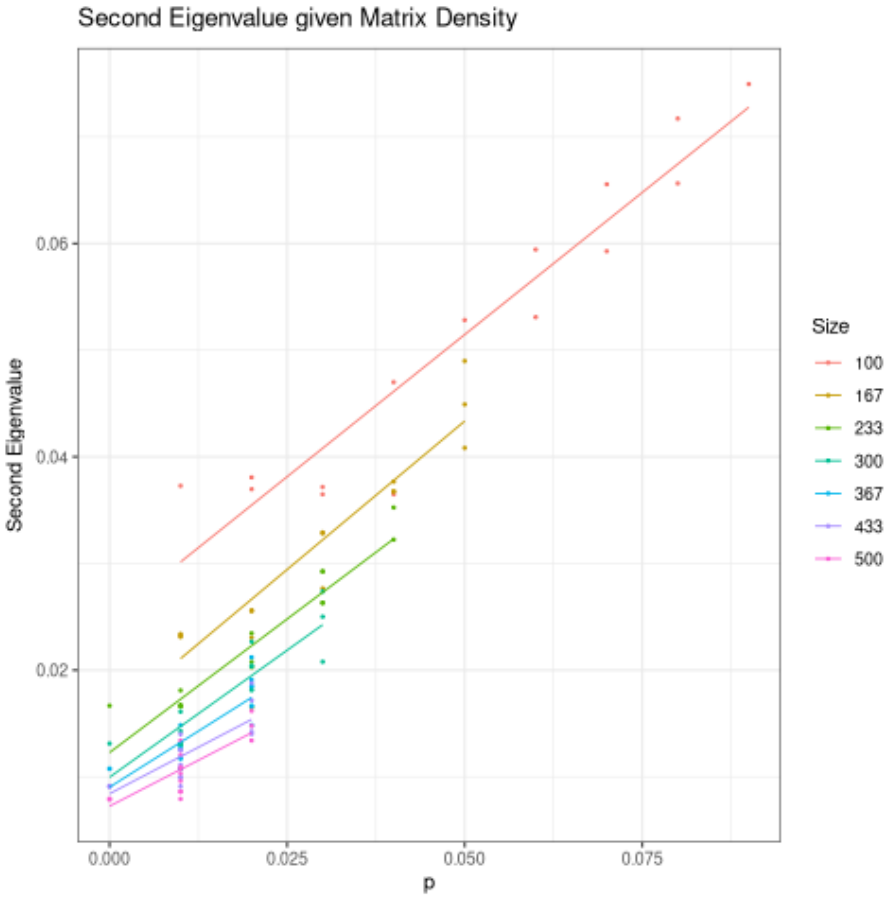

Figure 14: Second Eigenvalue plotted against matrix density for a variety of sizes from randomly generated *Barabassi-Albert* graphs

More data was generated and a model fitted to training data in listing 26. A corresponding residual histogram for the testing data is provided in 15 and a summary of the model provided in listing 27.

In fitting the model in listing 26 the identity $p = \frac{m}{n}$ was implemented because taking the mean value of a large matrix is resource intensive and would have reduced the number of data points in the analysis.

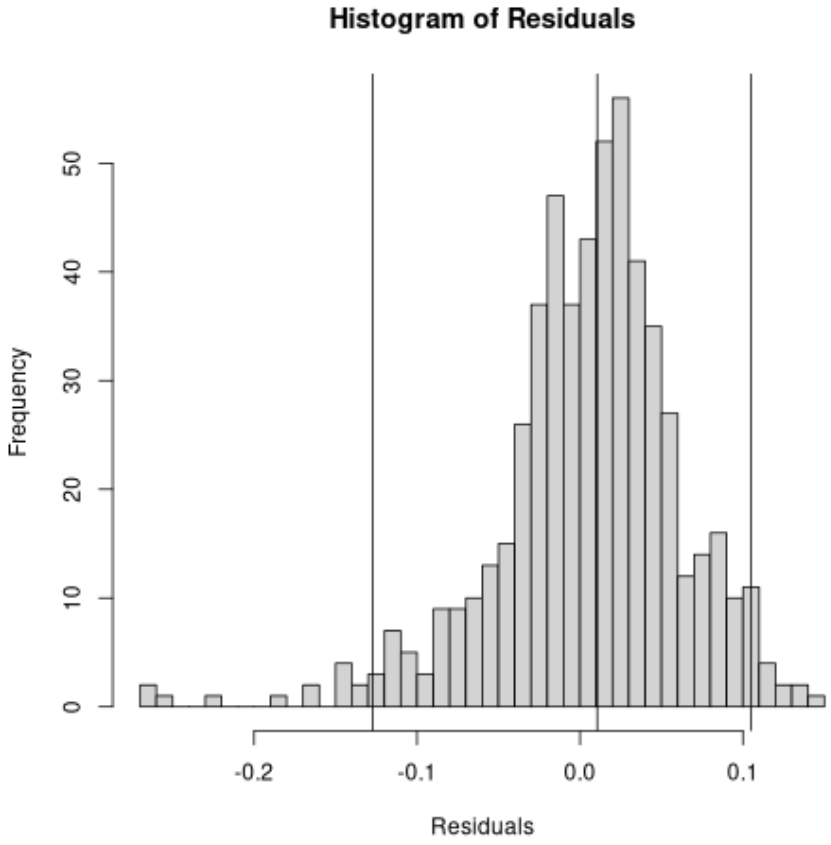

Figure 15: Histogram of the test set residuals for the multiple linear regression model fitted in listing 26, these residuals appear reasonably normal and are centered around zero, this suggests that the model may be an appropriate estimate on the domain that it was fitted.

The residuals appear reasonably normal, this suggests that the model may be an appropriate estimation over the domain on which it was fitted, the corresponding model is given by:

$$|\xi_2| = 1.34p - \frac{3.4n}{10^5} + \frac{3.9\beta}{10^3} - \frac{1.4pn}{10^3} \tag{52}$$

$$p = \text{mean}\left(\mathbf{A}\right) \tag{53}$$

For an unweighted adjacency matrix from the internet we may expect:

```r
filename <- "resources/BAData_lots.rds"

# Load the DAta
if (file.exists(filename)) {
    data_lots <- readRDS(filename)
} else {
    m       <- seq(from = 1, to = 9, length.out = 10)
    beta    <- seq(from = 1, to = 20, length.out = 40)
    size    <- seq(from = 100, to = 500, length.out = 5) %>%
    ↪  rev()
    size    <- c(size, 750, 1000)

    data_lots <- sim_graphs_pa(filename, m, beta, size)
}

# Create the link density variable
data2   <- data_lots
data2$p <- data2$m/data2$size

# Use 80% testing data
n   <- nrow(data2)
r   <- 0.8
train <- sample(x = 1:nrow(data2), size = r*n)

# Fit the Model
mod       <- lm(eigenvalue2 ~ 0 + p*size + p + beta, data =
↪  data2, subset = train)
test_pred <- predict(object = mod)[-train]
test_res  <- test_pred-data2$eigenvalue2[-train]

# Make the Plot
q <- quantile(test_res, c(0.025, 0.5, 0.975))
quantile(test_res)
cil <- q[1]
cih <- q[2]
cir <- q[3]

hist(test_res, breaks = 50, xlab = "Residuals",
    main = "Histogram of Residuals")
abline(v = c(cil, cih, cir))
```

Listing 26: l:mlm$_{\text{histba}}$

```
summary(mod)
```

```
Call:
lm(formula = eigenvalue2 ~ 0 + p * size + p + beta, data = data2)

Residuals:
    Min        1Q    Median        3Q       Max
-0.108811 -0.009673 -0.001948  0.007289  0.069467

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
p           1.348e+00  2.661e-02  50.651   <2e-16 ***
size       -3.404e-05  1.292e-06 -26.358   <2e-16 ***
beta        3.923e-03  5.205e-05  75.367   <2e-16 ***
p:size     -1.446e-03  1.683e-04  -8.592   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01805 on 2796 degrees of freedom
Multiple R-squared:  0.9089,Adjusted R-squared:  0.9088
F-statistic:  6975 on 4 and 2796 DF,  p-value: < 2.2e-16
```

Listing 27: Summarise the Coefficients of the model.

$$p \approx \frac{m}{n} \tag{54}$$

$$= \frac{\text{mean}\,(\text{out} - \text{degree})}{n} \tag{55}$$

For larger scale free graphs $p \to 0$ so this model seems to suggest that the eigenvalue will be reduced for larger graphs.

To test the model, we can measure the eigenvalue resulting from values within the domain of the model and from outside the domain:

```r
m         <- 1
beta      <- 3
size <- n <- 50


g <- igraph::sample_pa(power = 3, n = n, m = m)
A <- t(as.matrix(igraph::get.adjacency(g)))
W <- PageRank::power_walk_prob_trans(A, beta = beta)

p <- mean(A)
(e2 <- abs(eigen(W, only.values = TRUE)$values[2]) )

abs(e2_pred <-
p      *   1.344
+ size   * - 3.393e-05
+ beta   *   3.918e-03
+ p*size * - 1.437e-03
)
```

```
[1] 0.03689031
[1] 0.03499164
```

```r
m         <- 1
beta      <- 3
size <- n <- 2000


g <- igraph::sample_pa(power = 3, n = n, m = m)
A <- t(as.matrix(igraph::get.adjacency(g)))
W <- PageRank::power_walk_prob_trans(A, beta = beta)

p <- mean(A)
(e2 <- abs(eigen(W, only.values = TRUE)$values[2]) )

abs(e2_pred <-
p      *   1.344
+ size   * - 3.393e-05
+ beta   *   3.918e-03
+ p*size * - 1.437e-03
)
```

```
[1] 0.001329781
[1] 0.03968164
```

Unfortunately this suggests that for large graphs this model is not a good fit, which means that this model

can only provide insight into the behavior of ξ_2 for graphs with less than 1000 nodes.

10 Relating the Power Walk to the Random Surfer

It has been shown that $\xi_2 \leq \alpha$ for a transition probability matrix corresponding the random surfer model and that $\xi_2 = \alpha$ if there are at least two closed sub-graphs in the initial network [18]. This is demonstrated by figure 5 and figure 4 where $\alpha = \xi_2 = 0.8123456789$.

Finding a connection between the random surfer and the the power walk methods could provide insight into the relationship of ξ_2 and the parameters of the power walk method.

10.1 Introduction

Consider the equation:

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

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

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

10.2 Value of [1st Term]

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

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

Using this property we get the following

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

$$\left(\beta - 1\right)\mathbf{A}_{i,j} = \begin{cases} 0, & \mathbf{A}_{i,j} = 0 \\ \beta - 1, & \mathbf{A}_{i,j} = 1 \end{cases}$$

This means we have

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

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

10.3 Value of {2nd Term}

$$\begin{aligned}\mathbf{OD_B^{-1}} &= \begin{pmatrix} 1 & 1 & 1 & \\ 1 & 1 & 1 & \cdots \\ 1 & 1 & 1 & \\ & \vdots & & \ddots \end{pmatrix} \begin{pmatrix} \frac{1}{\delta_1} & 1 & 1 & \\ 1 & \frac{1}{\delta_2} & 1 \cdots & \\ 1 & 1 & \frac{1}{\delta_3} & \\ & \vdots & & \ddots \end{pmatrix} \\ &= n \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \\ \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \cdots \\ \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \\ & \vdots & & \ddots \end{pmatrix} \begin{pmatrix} \frac{1}{\delta_1} & 1 & 1 & \\ 1 & \frac{1}{\delta_2} & 1 & \cdots \\ 1 & 1 & \frac{1}{\delta_3} & \\ & \vdots & & \ddots \end{pmatrix} \\ &= n\mathbf{E}{\mathbf{D_B}}^{-1}\end{aligned}$$

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

- $\mathbf{E}_{i,j} = \frac{1}{n}$
- ${\mathbf{D_B}}^{-1}_{k,k} = \frac{1}{\delta_k}$
- 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 $\frac{1}{\texttt{colSums}(\mathbf{B})} = \vec{1}\mathbf{B}$, but if there were zeros in a column, it would be necessary to swap out the \$0\$s for \$1\$s 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:

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

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:

$$= \mathbf{D_A}\mathbf{D_A}^{-1}\mathbf{A}_{i,j}\left(\beta - 1\right) + n\mathbf{E}\mathbf{D_B}^{-1}$$

But the next step requires showing that:

$$\left(\beta - 1\right)\mathbf{D_A}\mathbf{D_B}^{-1} = \mathbf{I} - n\mathbf{D}_B^{-1}$$

10.4 Equate the Power Walk to the Random Surfer

Define the matrix $\mathbf{D_M}$:

$$\mathbf{D_M} = \mathrm{diag}\left(\texttt{colSums}\left(\mathbf{M}\right)\right) = \mathrm{diag}\left(\vec{1}\mathbf{M}\right) \tag{56}$$

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 transition probability matrix:

$$\mathbf{D_A}^{-1} : \left[\mathbf{D_A}^{-1}\right]_i = \begin{cases} 0, & \left[\mathbf{D_A}\right]_i = 0 \\ \left[\frac{1}{\mathbf{D_A}}\right], & \left[\mathbf{D_A}\right]_i \neq 0 \end{cases} \tag{57}$$

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

$$\mathbf{D}_B^{-1} = \frac{1}{\mathrm{diag}\left(\vec{1}\beta^{\mathbf{A}}\right)} \tag{58}$$

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

Power Walk

$$\mathbf{T} = \boxed{\mathbf{A}\mathbf{D_A}^{-1}}\mathbf{D_A}\left(\beta - 1\right)\mathbf{D_B}^{-1} + \boxed{\mathbf{E}}n\mathbf{D_B}^{-1} \tag{59}$$

Random Surfer

$$\mathbf{T} = \alpha\boxed{\mathbf{A}\mathbf{D_A}^{-1}} + \left(1 - \alpha\right)\boxed{\mathbf{E}} \tag{60}$$

So these are equivalent when:

$$\mathbf{D_A}\,(\beta-1)\,\mathbf{D_{B^{-1}}} = \mathbf{I}\alpha \tag{61}$$

$$\begin{aligned} \vec{1}\,(1-\alpha) &= -n\mathbf{D_B^{-1}} \\ \implies \vec{1}\alpha &= \vec{1} - n\mathbf{D_B^{-1}} \end{aligned} \tag{62}$$

Hence we have:

$$\mathbf{D_A}\,(\beta-1)\,\mathbf{D_B^{-1}} = \vec{1}\alpha = \mathbf{I} - n\mathbf{D_B^{-1}} \tag{63}$$

Solving for β:

$$\beta\mathbf{J} = (1-\Theta)\,\Theta^{-1}$$

where: [17]

- $\Theta = \mathbf{D_A}\mathbf{D_B^{-1}}$

If β is set accordingly then by (63):

$$\begin{aligned} \mathbf{A}\,(\beta-1)\,\mathbf{D_B^{-1}} &= \alpha = \mathbf{I} - n\mathbf{D_B^{-1}} \\ \implies \mathbf{A}\,(\beta-1)\,\mathbf{D_B^{-1}} &= \mathbf{I} - n\mathbf{D_B^{-1}} \end{aligned} \tag{64}$$

And setting $\Gamma = \mathbf{I} - n\mathbf{D_B^{-1}}$ from (62) and putting in (59) we have:

$$\begin{aligned} \mathbf{T} &= \boxed{\mathbf{A}\mathbf{D_A^{-1}}}\mathbf{D_A}\,(\beta-1)\,\mathbf{D_B^{-1}} + \boxed{\mathbf{E}}n\mathbf{D_B^{-1}} \\ \mathbf{T} &= \Gamma\boxed{\mathbf{A}\mathbf{D_A^{-1}}} + (1-\Gamma)\,\boxed{\mathbf{E}} \end{aligned}$$

$$\mathbf{T} = \Gamma\mathbf{A}\mathbf{D_A^{-1}} + (1-\Gamma)\,\mathbf{E} \tag{65}$$

Where $\mathbf{E}$ is square matrix of $\frac{1}{n}$ as in (15) (22)

10.5 Conclusion

More investigation will be required to determine further connections between these two methods.

[17] This is similar to a signmoid 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

11 Conclusion

In this report an approach to implement and investigate the random surfer and random walk *PageRank* methods was presented and a linear model was implemented to try and provide insight into the behavior of the second eigenvalue of the probability transition matrix corresponding to the power walk method.

The model appears not to perform well for graphs over 1000 nodes, this is unfortunate and limits the amount of insight the model can provide into the behavior of the power walk method.

A relationship between the random surfer and power walk method was also presented, this relationship appears to be restricted to unweighted adjacency matrices, it is not clear however if this could be expanded.

Further investigation is required to understand the relationship between the second eigenvalue and the method parameters of the power walk method, such further investigation should focus on investigating the relationship between the power walk and random surfer models and implementing non-linear models.

12 Appendix

1 Matrix Density

The R code displayed in listing 28 creates a plot of a sparse matrix shown in figure 16

1.1 Graph Diagrams

Graph Diagrams shown in 3.2.2 where produced using `DOT` (see [31, 12]).

1.2 *R* Packages

The packages used for analysis in this report are provided in listing 29

```r
        library(Matrix)
        library(igraph)
        n <- 200
        m <- 5
        power <- 1
        g <- igraph::sample_pa(n = n, power = power, m = m, directed =
        ↪  FALSE)
        plot(g)
        A <- t(get.adjacency(g))
        plot(A)
        image(A)


        # Create a Plotting Region
        par(pty = "s", mai = c(0.1, 0.1, 0.4, 0.1))


        # create the image

        title=paste0("Undirected Barabassi Albert Graph with
        ↪  parameters:\n Power = ", power, "; size = ", n, ";
        ↪  Edges/step = ", round(m))
        image(A, axes = FALSE, frame.plot = TRUE, main = title, xlab =
        ↪  "", ylab = "",  )
```

Listing 28: ***R*** code to produce an image illustrating the density of a simulated Barabassi-Albert graph, the *Barabassi-Albert* graph is a good analogue for the link structure of the internet [22, 3, 4] see the output in figure 16

```r
  if (require("pacman")) {
      library(pacman)
    }else{
      install.packages("pacman")
      library(pacman)
    }

    pacman::p_load(tidyverse, Matrix, igraph, plotly, plot3d, mise,
    ↪  docstring, mise, corrplot, latex2exp)
#   options(scipen=20) # Resist Scientific Notation
```

Listing 29: Implemented Packages used in this report

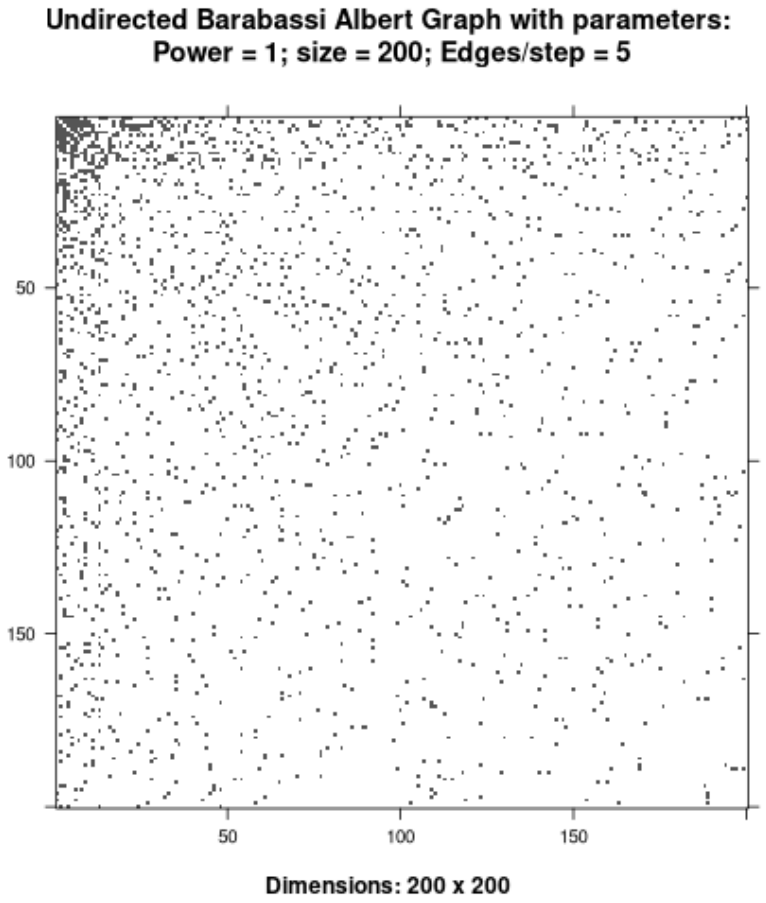

Figure 16: Plot of the adjacency matrix corresponding to a Barabassi-Albert (i.e. *Scale Free*) Graph produced by listing 28, observe the matrix is quite sparse.

Bibliography

[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. 8).

[2] Albert-László Barabási. *Linked: The New Science of Networks*. Cambridge, Mass: Perseus Pub, 2002. 280 pp. ISBN: 978-0-7382-0667-7 (cit. on p. 48).

[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. URL: https://iopscience.iop.org/article/10.1088/2058-7058/14/7/32 (visited on 10/11/2020) (cit. on pp. 48, 64).

[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. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378437100000182 (visited on 10/11/2020) (cit. on pp. 48, 49, 64).

[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), pp. 3–33. ISSN: 1573-7594. DOI: 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), pp. 92–128. ISSN: 15335399. DOI: 10.1145/1052934.1052938. URL: http://portal.acm.org/citation.cfm?doid=1052934.1052938 (visited on 08/18/2020) (cit. on pp. 6, 38).

[7] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. "PageRank as a Function of the Damping Factor". In: *Proceedings of the 14th International Conference on World Wide Web*. WWW '05. New York, NY, USA: Association for Computing Machinery, May 10, 2005, pp. 557–566. ISBN: 978-1-59593-046-0. DOI: 10.1145/1060745.1060827. URL: http://doi.org/10.1145/1060745.1060827 (visited on 08/19/2020) (cit. on p. 38).

[8] Michael Brinkmeier. "PageRank Revisited". In: *ACM Transactions on Internet Technology* 6.3 (Aug. 2006), pp. 282–301. ISSN: 15335399. DOI: 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).

[9] Kurt Bryan and Tanya Leise. "The $25,000,000,000 Eigenvector: The Linear Algebra behind Google". In: *SIAM Review* 48.3 (2006), pp. 569–581. ISSN: 0036-1445. JSTOR: 20453840 (cit. on p. 38).

[10] 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).

[11] Wai Ki Ching and Michael K. Ng. *Markov Chains: Models, Algorithms and Applications*. International Series in Operations Research and Management Science 83. New York, N.Y: Springer, 2006. 205 pp. ISBN: 978-0-387-29335-6 978-0-387-29337-0 (cit. on p. 8).

[12] *DOT (Graph Description Language)*. In: *Wikipedia*. June 11, 2020. URL: https://en.wikipedia.org/w/index.php?title=DOT_(graph_description_language)&oldid=961944797 (visited on 10/09/2020) (cit. on p. 63).

[13] Douglas Bates and Martin Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*. Version 1.2-18. Nov. 27, 2019. URL: https://CRAN.R-project.org/package=Matrix (visited on 10/17/2020) (cit. on p. 13).

[14] 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 pp. 4, 9).

[15] 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. URL: http://onlinelibrary.wiley.com/doi/abs/10.1002/asmb.656 (visited on 08/19/2020) (cit. on p. 6).

[16] 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. 8).

[17] 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. URL: http://doi.org/10.1145/2488388.2488433 (visited on 10/09/2020) (cit. on p. 5).

[18] Taher Haveliwala and Sepander Kamvar. "The Second Eigenvalue of the Google Matrix". In: *Stanford Technical Report* (2003). URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwi5x7iBhqnrAhVTfisKHQp8CrYQFjAAegQIBRAB&url=https%3A%2F%2Fnlp.stanford.edu%2Fpubs%2Fsecondeigenvalue.pdf&usg=AOvVaw3Em9lm2qOuWEN23PXhUS8J (cit. on pp. 38, 59).

[19] *Intel® Math Kernel Library Reference Manual; Sparse Matrix Storage Formats*. URL: https://scc.ustc.edu.cn/zlsc/sugon/intel/mkl/mkl_manual/GUID-9FCEB1C4-670D-4738-81D2-F378013412B0.htm (visited on 10/24/2020) (cit. on p. 13).

[20] 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. URL: http://www.sciencedirect.com/science/article/pii/S0024379504000023 (visited on 08/19/2020) (cit. on p. 6).

[21] 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. URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.23118 (visited on 08/21/2020) (cit. on p. 6).

[22] 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. 4–6, 8–12, 49, 64).

[23] 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. URL: http://www.sciencedirect.com/science/article/pii/S016975529800110X (visited on 08/19/2020) (cit. on pp. 3, 11).

[24] Ron Larson and Bruce H. Edwards. *Elementary Linear Algebra*. 2nd ed. Lexington, Mass: D.C. Heath, 1991. 592 pp. ISBN: 978-0-669-24592-9 (cit. on p. 14).

[25] 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. 8).

[26] Nathanael Ackerman, Cameron Freer,Alex Kruckman, and Rehana Patel. *Properly Erodic Structures*. Oct. 25, 2017. URL: https://math.mit.edu/~freer/papers/properly-ergodic-structures.pdf (visited on 10/10/2020) (cit. on p. 4).

[27] Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan. "Stable Algorithms for Link Analysis". In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '01. New York, NY, USA: Association for Computing Machinery, Sept. 1, 2001, pp. 258–266. ISBN: 978-1-58113-331-8. DOI: 10.1145/383952.384003. URL: http://doi.org/10.1145/383952.384003 (visited on 08/19/2020) (cit. on p. 38).

[28] 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. URL: http://doi.org/10.1145/2537734.2537749 (visited on 07/31/2020) (cit. on pp. 4, 11, 14).

[29] Alfred Renyi and Erdos Paul. "On Random Graphs I". In: *Publ. math. debrecen* 6.290-297 (1959), p. 18. URL: http://www.leonidzhukov.net/hse/2016/networks/papers/erdos-1959-11.pdf (cit. on p. 39).

[30] 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).

[31] *The DOT Language*. URL: https://graphviz.org/doc/info/lang.html (visited on 10/09/2020) (cit. on p. 63).

[32] Rui Zeng et al. "A Practical Simulation Method for Social Networks". In: 144 (2013), p. 8 (cit. on p. 48).