Delft University of Technology
Faculty Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Google PageRank and Reduced-Order Modelling

Report for the
Delft Institute of Applied Mathematics
as part of

the degree of

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

HUGO DE LOOIJ

Delft, Netherlands
June 2013

Copyright © 2013 by Hugo de Looij. All rights reserved.
BSc report APPLIED MATHEMATICS

“Google PageRank and Reduced-Order Modelling”

HUGO DE LOOIJ

Delft University of Technology

Thesis advisor
Dr. N.V. Budko

Other members of the graduation committee
Prof.dr.ir. C. Vuik
Dr. J.G. Spandaw
Dr. J.A.M. de Groot

June, 2013
Delft
## Contents

1 Introduction  
2 Preliminaries  
3 The PageRank Method   
   3.1 The basic model  
   3.2 Random surfer  
   3.3 Ranking  
   3.4 The hyperlink matrix $H$  
   3.5 The stochastic matrix $S$  
   3.6 The Google matrix $G$  
   3.7 PageRank  
4 Calculating the PageRank vector  
   4.1 The Power Method  
   4.2 Implementation of the Power Method  
   4.3 A direct method  
   4.4 The Jacobi method  
   4.5 Optimized Jacobi  
5 Numerical Experiments  
   5.1 Data  
   5.2 Convergence criteria  
   5.3 Numerical Results  
   5.4 Discussion  
6 Random Teleportation Parameter  
   6.1 Expected PageRank  
   6.2 Optimized Power Method  
   6.3 Reduced-order modelling  
   6.4 Numerical experiments  
   6.5 Discussion  
7 Conclusion & Discussion  
8 References  
A Appendix  
   A.1 Constructing the hyperlink/Google matrices  
      A.1.1 surfer.m  
      A.1.2 loadCaliforniaMatrix.m  
      A.1.3 loadStanfordMatrix.m  
      A.1.4 loadSBMatrix.m  
      A.1.5 DanglingNodeVector.m  
      A.1.6 GoogleMatrix.m  
   A.2 Numerical Algorithms  
      A.2.1 PowerMethod.m  
      A.2.2 JacobiMethodS.m  
      A.2.3 JacobiMethodH.m
A.2.4 OptimizedJacobiMethodH.m ............................... 50
A.2.5 OptimalBeta.m ........................................... 51
A.2.6 Compare.m ............................................... 52
A.3 Numerical Methods for the expected PageRank vector ........ 53
   A.3.1 OptimizedPowerMethod.m ............................. 53
   A.3.2 Arnoldi.m ............................................ 53
   A.3.3 ReducedOrderMethod.m ............................... 54
   A.3.4 Compare2.m ........................................... 54
   A.3.5 PlotResidual.m ....................................... 56
1 Introduction

The world wide web consists of almost 50 billion web pages. From this chaos of websites, users would like to visit a couple of sites about a certain subject. But how could a user possibly know which of these billion web pages he should visit?

This is where search engines come into play. Not only do search engines filter the world wide web for a specific search query, they also sort the internet by importance. In this way, search engines could suggest a user to visit certain web pages. However, how can a web page be rated by importance? Since the world wide web is so huge, this cannot be done manually. Hence algorithms need to be developed so this job can be done automatically.

Google is the first search engine that managed to do this effectively. The secret behind Google is the PageRank method. This method is developed in 1996 by the founders of Google, Larry Page and Sergey Brin. Every page on the world wide web is given a score of importance (also called the PageRank value). Whenever a user is making a search request, Google sorts all web pages which satisfy the search requests by their PageRank value. The user will receive these search results in this order.

The Google PageRank method is a completely objective way to rate a web page. It is based on a mathematical model which only uses the structure of the internet. In this paper, this method is discussed. We will see that the PageRank vector (the vector containing each PageRank value) is defined as an eigenvector of the so-called Google matrix.

The Google matrix, however, is extremely large; it is almost 50 billion by 50 billion in size. Therefore, calculating an eigenvector is far from an easy task. In chapter 3 we will develop and discuss numerical methods to approximate this eigenvector as fast as possible. Then these numerical methods are put to the test, using multiple small Google matrices, corresponding to multiple subsets of web pages in the world wide web. The tests and results can be found in chapter 4.

In the last chapter we will make an adjustment to the model. Previously, one of the parameters in the model is considered to be a deterministic value. Things get much more interesting if this parameter is considered to be stochastic. The PageRank vector depends on the value of this parameter, and our goal will be to calculate the expected PageRank vector. We will develop an algorithm which uses the shift-invariance of the Krylov space of the hyperlink matrix. The expected PageRank can then be approximated using this algorithm. This reduced-order algorithm will turn out to be far more efficient than the ‘standard’ approximations.

It is worth noting that the whole theory in this paper is based on the original paper of Google PageRank by Brin and Page in 1998, see [1]. It is almost certain that Google has further developed this model. The current model is kept secret, and thus the theory and results might not be completely up to date. It has been stated by Google, however, that the current algorithm used by Google is still based on the original PageRank model.
2 Preliminaries

Theorem 1 (Gershgorin’s circle theorem). Let $A$ be an $n \times n$ matrix. Each eigenvalue $\lambda$ of $A$ satisfies
\[
\lambda \in \bigcup_{i=1}^{n} \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j=1, j \neq i}^{n} |a_{ij}| \right\}
\]
Thus, every eigenvalue of $A$ is contained in the union of disks with center $a_{ii}$ and radius $\sum_{j=1, j \neq i}^{n} |a_{ij}|$.

Lemma 2. Let $A$ be an $n \times n$ matrix. $\lambda \in \mathbb{C}$ is an eigenvalue of $A$ if and only if $\lambda$ is an eigenvalue of $A^T$.

Definition 3. An $n \times n$ matrix $A$ is called (row)-stochastic if the following properties are true:
1. $a_{ij} \in \mathbb{R}$ and $a_{ij} \geq 0$ for all $1 \leq i, j \leq n$.
2. $\sum_{j=1}^{n} a_{ij} = 1$ for all $1 \leq j \leq n$.

Definition 4. An $n \times n$ matrix $A$ is called sub-stochastic if the following properties are true:
1. $a_{ij} \in \mathbb{R}$ and $a_{ij} \geq 0$ for all $1 \leq i, j \leq n$.
2. $\sum_{j=1}^{n} a_{ij} \leq 1$ for all $1 \leq j \leq n$.

Lemma 5. Let $A$ be an $n \times n$ stochastic matrix. Then 1 is an eigenvalue of $A$. Furthermore, this is the largest eigenvalue (i.e. if $\lambda$ is an eigenvalue of $A$, then $|\lambda| \leq 1$).

Proof: Let $e = (1, 1, \ldots, 1)^T$ be the $n \times 1$ vector where each component is 1. Then
\[
Ae = A \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} a_{1i} \cdot 1 \\ \sum_{i=1}^{n} a_{2i} \cdot 1 \\ \vdots \\ \sum_{i=1}^{n} a_{ni} \cdot 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = 1e
\]
Thus, 1 is indeed an eigenvalue of $A$. By applying Gershgorin’s Theorem on $A$, we find that for each eigenvalue $z$ of $A$ there exists some $1 \leq i \leq n$ such that $|z - a_{ii}| \leq \sum_{j=1, j \neq i}^{n} |a_{ij}|$.
This yields
\[
|z| = |z - a_{ii} + a_{ii}| \leq |z - a_{ii}| + |a_{ii}| \leq \sum_{j=1, j \neq i}^{n} |a_{ij}| + |a_{ii}| = \sum_{j=1}^{n} |a_{ij}| = \sum_{j=1}^{n} a_{ij} = 1
\]
In other words, 1 is the largest eigenvalue of $A$. 

8
Theorem 6 (Geometric series for matrices). Suppose $A$ is an $n \times n$ matrix such that its spectral radius $\rho(A) < 1$ (i.e. $|\lambda| < 1$ for all eigenvalues $\lambda$ of $A$). Then $\sum_{n=0}^{\infty} A^n$ converges and 

$$ (I - A)^{-1} = \sum_{n=0}^{\infty} A^n $$

For the PageRank model, we will also discuss Markov chains. The transition matrix of a Markov chain is a (sub)-stochastic matrix, and it is important to look at some properties of these matrices.

Definition 7. Let $A$ be an $n \times n$ sub-stochastic matrix. $A$ is said to be irreducible if for each $(i, j)$ there exists some $n \in \mathbb{N}$ such that $(A^n)_{ij} > 0$. If this property does not hold, $A$ is called reducible.

Definition 8. An $n \times n$ matrix is said to be aperiodic if for each $1 \leq i \leq n$ one has $\gcd\{n \in \mathbb{N}_{\geq 1} : (A^n)_{ii} > 0\} > 1$. In particular, if $A_{ii} > 0$ for each $i$, $A$ is periodic. $A$ is called periodic if and only if $A$ is not aperiodic.

Definition 9. A matrix $A$ is primitive if and only if it is irreducible and aperiodic.
3 The PageRank Method

The PageRank method is an algorithm designed by the founders of Google, Sergey Brin and Larry Page. This algorithm is the main reason why Google turned out to be the most popular search engine. Every page on the internet is given a score of importance. The larger this number, the earlier this page can be found in the search results. In the following paragraphs we will discuss how this score is being determined.

3.1 The basic model

PageRank is based on the hyperlink structure of the world wide web. We can view the internet as a directed graph. The nodes should be interpreted as the web pages of the world wide web and an arc from node $i$ to node $j$ should be seen as a hyperlink on webpage $i$ to webpage $j$. See figure 1 for a small example. We will use this example to illustrate a couple of definitions.

![Figure 1: A small network with 5 webpages.](image)

The goal of Brin and Page, the founders of Google, was to construct a method to rank each page on the internet. They invented the PageRank method, which uses only the structure of the internet. If many different webpages all link to the same page (such as node 2 in our example), it makes sense to view this page as important.

However, the amount of links to a certain page is very easily manipulated and does not give a clear idea how important a page should be. Brin and Page were the first to also look at the quality of an outgoing link. If an important page links to an external webpage, this webpage should be marked as important as well. This idea, that except for the quantity of incoming links the quality is relevant for the PageRank value of a webpage, is the main reason why Google PageRank is such a successful model.

3.2 Random surfer

The PageRank method is based on a random surfer. Suppose we are surfing on the internet, clicking links randomly. This can be interpreted as a Markov chain where the transition probability $p_{ij}$ is uniformly distributed between the set of pages that $i$ links to. We repeat this process an infinite amount of times. The fraction of the time the surfers visits a certain website can be seen as the PageRank of this site. Clearly, this fraction is greater for a certain webpage if more pages link to this webpage. In our example (see figure 1) node 2 will have a greater PageRank. However, every time the random surfer visits this node, it will also visit
node 1. Hence node 1 will have a good PageRank value as well, even though it has only 1 incoming link. This illustrates why the quality of links also matters.

These fractions - if they exist - form a stationary distribution corresponding to the Markov chain of the random surfer. Denote this distribution by \( p \). We know from Markov theory that this distribution satisfies the linear system \( p^T = p^T A \), with \( A \) the transition matrix of the Markov chain. This equality allows us to define the PageRank vector in a more rigorous way using only linear algebra. Because a stationary distribution does not need to exist nor does it need to be unique, some modifications to the model need to be made before the PageRank vector can be defined.

3.3 Ranking

**Definition 10.** Let \( p \) be an \( n \times 1 \) vector. We call \( p \) a **ranking** if it has the following properties:

- \( p_i = \sum_{j \in I(i)} \frac{1}{|U(j)|} p_j \) for all \( 1 \leq i \leq n \).
- \( p_i \geq 0 \) for all \( 1 \leq i \leq n \).
- \( \sum_{i=1}^{n} p_i = 1 \).

Here \( I(i) \) is the collection of pages which link to node \( i \) and \( |U(j)| \) is the amount of outer links on the same page. This is well defined, because if \( U(j) = 0 \), page \( j \) does not link to any other page, hence \( j \not\in I(i) \).

Therefore, a stationary distribution of the Markov chain is a ranking. To illustrate this definition, a vector \( p \) is a ranking of our example in figure 1 satisfies

\[
\begin{align*}
p_1 &= p_2 \\
p_2 &= \frac{1}{4} p_1 + p_3 + \frac{1}{2} p_4 + p_5 \\
p_3 &= 0 \\
p_4 &= \frac{1}{2} p_1 \\
p_5 &= \frac{1}{2} p_4
\end{align*}
\]

A solution of this system such that \( \sum_{i=1}^{5} p_i = 1 \) and \( p_j \geq 0 \) for all \( 1 \leq j \leq 5 \) is the vector

\[
p = \begin{pmatrix}
\frac{4}{11} \\
\frac{4}{11} \\
0 \\
\frac{2}{11} \\
\frac{1}{11}
\end{pmatrix}
\]

One can easily show that for any other solution \( q \) of this linear system, we have \( q = kp \) for some \( k \in \mathbb{R} \). The property \( \sum_{i=1}^{5} p_i = 1 \) guarantees the uniqueness of the ranking \( p \). The larger the number \( p_i \), the more important page \( i \). In our example we should see page 1 and 2 as the upper results, then following page 4, 5, and 3 respectively.
3.4 The hyperlink matrix $H$

We have seen that $p$ is a ranking if it satisfies a linear system. Therefore, we will construct a matrix such that we can make the definition more convenient. Define the (Boolean) variable $L_{ij}$ by $L_{ij} = 1$ if page $i$ links to page $j$, and $L_{ij} = 0$ otherwise. Let $n$ be the amount of web pages.

**Definition 11.** The out degree of a page $i$ is defined by $c_i = \sum_{j=1}^{n} L_{ij}$.

The out degree of a page is simply the amount of pages that it links to. In the world wide web, this number is usually 0 to 20 but can vary greatly. As we will see, the case $c_i = 0$ will require some caution.

**Definition 12.** A node $i$ is called a **dangling node** if its out degree $c_i = 0$.

Dangling nodes are those pages that do not link to any other pages. See figure 2. In this example, page 6 is a dangling node. The world wide web consists mainly of dangling nodes: images, scripts, pdf files and other common files found on the internet often do not link to any other page. We will see in the next paragraph that dangling nodes require us to make an adjustment to the model of the random surfer.

**Definition 13.** We define the $n \times n$ hyperlink matrix $H$ by

$$H_{ij} = \begin{cases} 
\frac{1}{c_i} & \text{if } c_i \neq 0 \text{ and } L_{ij} = 1 \\
0 & \text{otherwise}
\end{cases}$$

See figure 2 for an example.

![Figure 2: An example of a network with a dangling node.](image-url)
The following matrix is the hyperlink matrix corresponding to the web of this figure. Note that page 6 is a dangling node – in the matrix $H$, this corresponds to a zero row.

$$H = \begin{pmatrix}
0 & 1/2 & 1/2 & 0 & 0 & 0 \\
0 & 0 & 1/2 & 0 & 1/2 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1/3 & 1/3 & 0 & 0 & 1/3 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}$$

**Theorem 14.** Let $p$ be a ranking. let $H$ be the corresponding hyperlink matrix. Then $p^T H = p^T$.

*Proof:* For the $i^{th}$ component of $p^T$ we have

$$(p^T)_i = p_i = \sum_{j \in I(i)} \frac{1}{|U(j)|} p_j = \sum_{j \in I(i)} \frac{1}{c_j} p_j = \sum_{j \in I(i)} H_{ij} p_j = \sum_{j=1}^{n} H_{ij} p_j = (H^T p)_i = (p^T H)_i \quad \Box$$

Note that by taking the transpose of this equality, we get the following more convenient equality

$$H^T p = p$$

This means that a ranking $p$ is a right eigenvector of matrix $H^T$ corresponding to eigenvalue 1. We will look at some properties of this matrix $H$.

- The hyperlink matrix $H$ is **sub-stochastic**, i.e. all elements are non-negative and the sum of each row is at most 1. After all, let $1 \leq i \leq n$ be arbitrary. There are two possibilities:
  1. $c_i = 0$. Hence $i$ is a Dangling node. Since this page does not link to any other page, we have $\sum_{j=1}^{n} H_{ij} = 0 \leq 1$.
  2. $c_i \neq 0$. Then
     $$\sum_{j=1}^{n} H_{ij} = \sum_{j:L_{ij}=1}^{n} \frac{1}{c_i} = \sum_{j:L_{ij}=1}^{n} \frac{1}{\sum_{j:L_{ij}=1}} = 1$$

- Generally, $H$ is reducible. This is because the world wide web most likely consists of multiple disjoint subsets.

- Generally, $H$ is periodic. There only needs to be one node with period larger or equal to 2. This can happen if two web pages link to each other but to no other node.

- $H$ is a very sparse matrix. The world wide web consists of multiple billions of pages, but every page links to only a very few other pages. So most components $H_{ij}$ are equal to zero. An advantage of this is that it requires much less memory to store and calculate this matrix. Also, computations with this matrix are much faster. We will use these properties of $H$ to minimise computation time and memory storage issues for the methods to calculate the PageRank value.
3.5 The stochastic matrix \( S \)

We have seen that \( p \) is a ranking if it satisfies the equality \( H^T p = p \). However, there is no guarantee that such an \( p \) exists nor does it need to be unique. Brin and Page have modified this matrix in such a way that it became stochastic and primitive. The Perron-Frobenius Theorem then guarantees the existence and uniqueness of this solution. This will then allow us to define the PageRank vector.

Note that \( H \) is not a stochastic matrix. The random surfer will have no idea what to do when it ends up on a dangling node. We will therefore construct an artificial transition probability for dangling nodes. For this, we will need to define a personalisation vector.

**Definition 15.** an \( n \times 1 \) vector \( v \) is called a **personalisation vector** if it satisfies the following two properties:

- \( v_i > 0 \) for all \( 1 \leq i \leq n \), and
- \( \sum_{i=1}^{n} v_i = 1 \).

We can also denote the second property by \( e^T v = 1 \). Here \( e \) is the \( n \times 1 \) uniform vector: every element of \( e \) is equal to 1. This notation will turn out to be more convenient in the proofs.

Just like the name suggest, the personalisation vector can be different for different users. A component \( v_i \) for a specific person should be seen as the probability that (s)he goes to page \( i \). If a person very commonly searches for information about sport, then \( v_i \) will generally be larger for a page which is about sport. The personalisation vector can therefore be chosen to adapt the PageRank vector for the interests of a person.

**Definition 16.** The \( n \times 1 \) **dangling node vector** \( d \) is given by

\[
d_i = \begin{cases} 
1 & \text{if } c_i = 0 \\
0 & \text{if } c_i \neq 0 
\end{cases}
\]

**Definition 17.** The \( n \times n \) matrix \( S \) is defined as

\[ S = H + dv^T \]

The matrix \( S \) is essentially equal to \( H \), but every zero row is replaced by the personalisation vector. The interpretation for this is that whenever the random surfer visits a dangling node, it will visit another web page randomly according to the personalisation (probability) vector.

Usually, we suppose that \( v = e/n \), i.e. the transition probability \( p_{ij} \) is uniformly distributed if node \( i \) is a dangling node. This is the ‘democratic’ personalisation vector. In this case, our example in figure 2 yields the stochastic matrix

\[
S = \begin{pmatrix}
0 & 1/2 & 1/2 & 0 & 0 & 0 \\
0 & 0 & 1/2 & 0 & 1/2 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1/3 & 1/3 & 0 & 0 & 1/3 \\
1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6
\end{pmatrix}
\]

14
We will consider a couple of important properties of $S$.

- The matrix $S$ is **stochastic**: every element of $S$ is non-negative and the sum of every row of $S$ is exactly equal to 1. This is easy to see:

  1. If $c_i \neq 0$, then $d_i = 0$ and $\sum_{j=1}^n H_{ij} = 1$. It follows that $\sum_{j=1}^n S_{ij} = \sum_{j=1}^n H_{ij} + 0 = 1$.

  2. If $c_i = 0$, then $d_i = 1$ and $\sum_{j=1}^n H_{ij} = 0$. Since $v$ is a personalisation vector, we have $\sum_{j=1}^n v_j = 1$. It follows that $\sum_{j=1}^n S_{ij} = \sum_{j=1}^n H_{ij} + d_i \sum_{j=1}^n v_j = 0 + 1 \cdot 1 = 1$.

- Generally, $S$ is periodic and reducible.

- $S$ is a sparse matrix, but much less sparse than $H$. This is because a relatively large amount of pages on the world wide web are dangling nodes. The consequence of this is that all the corresponding rows of $S$ are completely non-zero, since the personalisation vector is by definition strictly larger than zero. However, this is not a very big problem, since $S$ is a rank-one update of the very sparse matrix $H$. We will see this in chapter 4.

### 3.6 The Google matrix $G$

The last step of Brin and Page was to force the matrix to become irreducible and aperiodic. This guarantees the uniqueness of the PageRank vector. We introduce the possibility of teleportation for the random surfer model.

**Definition 18.** The **teleportation parameter** $\alpha$ is a the probability that the random surfer will follow a link. We require that $0 \leq \alpha < 1$.

This parameter $\alpha$ represents the probability that a link is being clicked. The random surfer will with a probability of $1 - \alpha$ go to a random page on the internet according to the distribution of the personalisation vector. This kind of teleportation happens when, for example, a user enters a different link in the address bar.

For small values of $\alpha$, the random surfer will almost always teleport to another page. Therefore, the random surfer will rarely click links and thus the structure of the web will not affect the PageRank vector much. If, on the other hand, $\alpha$ is close to one, the numerical methods to approximate the PageRank vector will converge very slowly. We will see this in chapter 4. Therefore, Google[1] used $\alpha = 0.85$ for their model. In chapter 5 we will discuss what will happen if $\alpha$ is considered to be a stochastic variable instead of a deterministic value.

**Definition 19.** The **Google matrix** based on the personalisation vector $v$ and teleportation parameter $\alpha$ is defined as

$$G = \alpha S + (1 - \alpha)ev^T$$

Thus, this matrix corresponds to the transition matrix of the Markov chain with teleportation. This matrix has some very important properties.

- $G$ is **stochastic**. This follows from the fact that both $S$ and $ev^T$ are stochastic. $G$ is a convex combination of the two and therefore is stochastic as well.
• The matrix $G$ is nowhere sparse. Because $v > 0$ and $\alpha < 1$, it follows that for any element $G_{ij}$ we have $G_{ij} \geq (1 - \alpha)v_j > 0$. This is a very bad property, but as we will see, this will not cause any problems. One does not need to store $G$ and the matrix-vector products with $G$ can be computed very efficiently using definition 19.

• The matrix $G$ is irreducible. This follows from the fact that each element $G_{ij}$ is larger than zero. For each combination of nodes $i$ and $j$, the probability that the surfer will visit node $j$ immediately after leaving node $i$ is at least equal to $(1 - \alpha)v_j > 0$.

• $G$ is aperiodic. This is implied by the fact that $G_{ii} > 0$ for all nodes $i$.

• Therefore, $G$ is primitive.

3.7 PageRank

The matrix $G$ is stochastic and primitive. Hence, we will define the PageRank vector as the unique stationary distribution of the corresponding Markov chain.

Definition 20. (PageRank). The PageRank vector $\pi$ is the unique vector which satisfies:

• $G^T \pi = \pi$,
• $\sum_{i=1}^{n} \pi_i = 1$. We can also write this as $\pi^T e = 1$.
• $\pi_i > 0$ for all $1 \leq i \leq n$.

To prove that this vector is uniquely defined, we will show that 1 is an eigenvalue of $G^T$ with an algebraic multiplicity of one. For this, we will use the Perron-Frobenius theorem. See [5] for a proof of this theorem.

Theorem 21 (Perron-Frobenius). Let $A$ be an $n \times n$ primitive matrix. Then there exists an eigenvalue $\lambda$ which is strictly greater in absolute value than all other eigenvalues. Furthermore, all elements of an eigenvector corresponding to this eigenvalue have the same sign.

We will prove a few lemma’s. These lemma’s will be used to prove some properties of the spectrum of the matrix $G^T$. Furthermore, these lemma’s will be reused in chapter 3 to prove some very important theorems.

Lemma 22. For each $\lambda \in \mathbb{C}$ with $|\lambda| > \alpha$, the matrix $\lambda I - \alpha S^T$ is invertible.

Proof: Let $\lambda \in \mathbb{C}$ such that $|\lambda| > \alpha$. We will use Gershgorins Circle theorem to show that 0 is not an eigenvalue of $M := \lambda I - \alpha S$, which implies that $M$ is invertible. Let $1 \leq i \leq n$ be arbitrary. Note that the diagonal element $M_{ii}$ is equal to $M_{ii} = \lambda - \alpha S_{ii}$. Furthermore, we have

$$\sum_{j \neq i} |M_{ij}| = \sum_{j \neq i} \alpha S_{ij} = \alpha \left( \sum_{j=1}^{n} S_{ij} - S_{ii} \right) = \alpha (1 - S_{ii}) = \alpha - \alpha S_{ii}$$

since $S$ is a stochastic matrix. Thus, by the reverse triangle inequality, we get

$$|M_{ii} - 0| \geq |\lambda - \alpha S_{ii}| = |\lambda| - |\alpha S_{ii}| \geq |\lambda| - \alpha S_{ii}$$

where

$$\sum_{j \neq i} |M_{ij}|$$

and

$$\sum_{j \neq i} |M_{ij}|$$
In other words, 0 does not lie in the disk with center $M_{ii}$ and radius $\sum_{j \neq i} |M_{ij}|$. Since $i$ was arbitrary, the union of these disks does not contain 0. Hence, by Gershgorin’s Theorem, zero is not an eigenvalue of $M$. Finally, by Lemma 2, 0 is not an eigenvalue of $\lambda I - \alpha S^T$ either, which proves the Lemma.

**Lemma 23.** Let $\mathbf{p}$ be an eigenvector of $G^T$ corresponding to an eigenvalue $\lambda$. Suppose that $|\lambda| > \alpha$ holds. Then $\mathbf{p} = (1 - \alpha)(\mathbf{e}^T \mathbf{p})(\lambda I - \alpha S^T)^{-1}\mathbf{v}$.

**Proof:** By writing out the definition of the Google matrix and by using the fact that $G^T \mathbf{p} = \lambda \mathbf{p}$, we get

\[
\begin{align*}
\lambda \mathbf{p} &= G^T \mathbf{p} = (\alpha S + (1 - \alpha)\mathbf{e} \mathbf{v}^T)^T \mathbf{p} \\
&= (\alpha S^T + (1 - \alpha)\mathbf{v} \mathbf{e}^T) \mathbf{p} = \alpha S^T \mathbf{p} + (1 - \alpha)\mathbf{v} (\mathbf{e}^T \mathbf{p}) \\
\implies (\lambda I - \alpha S^T) \mathbf{p} &= (1 - \alpha)(\mathbf{e}^T \mathbf{p})\mathbf{v}
\end{align*}
\]

Note that the matrix $(\lambda I - \alpha S^T)$ is invertible by Lemma 22, since we have assumed that $|\lambda| > \alpha$. This yields

\[
\mathbf{p} = (1 - \alpha)(\mathbf{e}^T \mathbf{p})(\lambda I - \alpha S^T)^{-1}\mathbf{v}
\]

which completes the proof.

**Lemma 24.** Suppose $\lambda$ is an eigenvalue of $G^T$ and $|\lambda| > \alpha$. Then $(1 - \alpha)\mathbf{v}^T(\lambda I - \alpha S)^{-1}\mathbf{e} = 1$.

**Proof:** Let $\mathbf{p}$ be an eigenvector corresponding to an eigenvalue $\lambda$. Suppose furthermore that $|\lambda| > \alpha$. Note that if $\mathbf{e}^T \mathbf{p} = 0$ should hold, Lemma 23 yields that $\mathbf{p} = 0$. But this is not an eigenvector. Thus, we can assume that $\mathbf{e}^T \mathbf{p} \neq 0$. Define $\mathbf{q} = \frac{\mathbf{p}}{\mathbf{e}^T \mathbf{p}}$. This is still an eigenvector of $G^T$ corresponding to the same eigenvalue $\lambda$. Since $\mathbf{e}^T \mathbf{q} = 1$, it follows from Lemma 23 that

\[
\mathbf{q} = (1 - \alpha)(I - \alpha S^T)\mathbf{v}
\]

It follows that

\[
(1 - \alpha)\mathbf{v}^T(\lambda I - \alpha S)^{-1}\mathbf{e} = (1 - \alpha)\mathbf{e}^T((\lambda I - \alpha S)^{-1})^T \mathbf{v} = (1 - \alpha)\mathbf{e}^T((\lambda I - \alpha S)^{-1})^T \mathbf{e} = \mathbf{e}^T(1 - \alpha)(\lambda I - \alpha S^T)^{-1}\mathbf{v} = \mathbf{e}^T \mathbf{q} = 1
\]

This lemma can be used to give an upper bound for each eigenvalue.
Theorem 25 (Eigenvalues of the Google matrix). Let $G$ be the Google matrix and denote $\lambda_1, \lambda_2, \ldots, \lambda_n$ as the eigenvalues of $G^T$ in descending absolute value. Then:

1. $\lambda_1 = 1$, and
2. $|\lambda_2| \leq \alpha$.

Proof: The first property is fairly straightforward. Since $G$ is a stochastic matrix, we have $Ge = e$ by Lemma 5. Furthermore, by the same Lemma, any eigenvalue $\lambda$ of $G$ satisfies $|\lambda| \leq 1$. So $1$ is the largest eigenvalue of $G$. The spectrum of the transpose of a matrix is equal to the spectrum of the matrix itself, hence $\lambda_1 = 1$.

We will use Lemma 24 to prove the second statement. Suppose $\lambda$ is an eigenvalue of $G^T$ and $|\lambda| > \alpha$. It follows that

$$
(1 - \alpha)v^T(\lambda I - \alpha S)^{-1}e = \frac{1 - \alpha}{\lambda}v^T \left(I - \frac{\alpha}{\lambda}S\right)^{-1}e = \frac{1 - \alpha}{\lambda}v^T \sum_{n=0}^{\infty} \left(\frac{\alpha}{\lambda}S\right)^n e
$$

$$
= \frac{1 - \alpha}{\lambda} \sum_{n=0}^{\infty} \left[\left(\frac{\alpha}{\lambda}\right)^n v^T S^ne\right] = \frac{1 - \alpha}{\lambda} \sum_{n=0}^{\infty} \left[\left(\frac{\alpha}{\lambda}\right)^n v^T e\right]
$$

$$
= \frac{1 - \alpha}{\lambda} \sum_{n=0}^{\infty} \left(\frac{\alpha}{\lambda}\right)^n = \frac{1 - \alpha}{\lambda} \cdot \frac{1}{1 - \alpha/\lambda} = \frac{1 - \alpha}{\lambda - \alpha}
$$

The sum $\sum_{n=0}^{\infty} \left(\frac{\alpha}{\lambda}S\right)^n$ is convergent since $\rho(\frac{\alpha}{\lambda}S) = \frac{\alpha}{\lambda}\rho(S) < \rho(S) \leq 1$. The last inequality follows from Lemma 5. Since $S$ is stochastic, $Se = e$. By applying this multiple times, we see that $S^ne = e$ for any $n \in \mathbb{N}$. Theorem 23 yields that $(1 - \alpha)v^T(\lambda I - \alpha S)^{-1}e = 1$, hence $\frac{1 - \alpha}{\lambda - \alpha} = 1$ must hold. So $\lambda = 1$.

However, $\lambda = 1$ is the largest eigenvalue of $G^T$ and $G^T$ is primitive. Thus, this eigenvalue must have an algebraic multiplicity one by the Perron-Frobenius theorem, i.e. it is not possible to have $\lambda_2 = \lambda_1 = 1$. It follows that $|\lambda_i| \leq \alpha$ must hold for all $i = 2, 3, \ldots, n$. \qed
Figure 3: The spectrum of a Google matrix.

Figure 3 illustrates the last theorem. The figure contains all eigenvalues (in blue) of the $5000 \times 5000$ (mathworks.com) Google matrix (see also paragraph 5.1). All eigenvalues but one are contained in the red disk of radius $\alpha$. The other eigenvalue is exactly equal to 1.

Furthermore, if $G$ has at least two irreducible closed subsets, the second eigenvalue $\lambda_2$ is exactly equal to $\alpha$. For a proof, see [6].

By Perron’s Theorem, each component of the eigenvector corresponding to eigenvalue 1 has the same sign, so we can scale this vector (denote this vector by $\pi$) such that $\pi^T e = 1$. $\pi$ is (strictly) positive, so it is indeed a probability vector. Thus, the PageRank vector is well defined. The fact that the second eigenvalue $|\lambda_2| \leq \alpha$ has some important consequences. For example, the convergence speed of the Power Method depends on the second eigenvalue of $G^T$. We will discuss this in the next chapter.
4 Calculating the PageRank vector

The PageRank vector is the unique vector $\pi$ such that $G^T \pi = \pi$ and $\pi^T e = 1$. The goal of Google was to be able to rank all webpages using the PageRank model. This corresponds with calculating the eigenvector $\pi$. However, the world wide web consists of tens of billions of pages, so calculating an eigenvector is not an easy task. In this chapter we will look at some numerical methods to calculate this vector as fast as possible. First, we will discuss the Power Method. Secondly, we will see that the PageRank vector satisfies a simple linear system, which we will solve with the Jacobi Method.

4.1 The Power Method

The Power Method is a well known numerical algorithm to calculate an eigenvector of a matrix corresponding to the largest eigenvalue. It is known to be extremely slow; its convergence speed depends on the difference between the first and second largest eigenvalue of the corresponding matrix, which is usually very small. However, as we have seen in Theorem 25, the second eigenvalue of $G^T$ is smaller than or equal to $\alpha$. Thus, the Power Method is very effective for our problem.

Brin and Page have applied the Power Method to approximate the PageRank vector. We will discuss this algorithm here.

**Algorithm 26.** Let $x^{(0)}$ be a starting vector. The Power Method is the following iterative process:

\[
k \leftarrow 0
\]
\[\text{while convergence not reached do}
    \]
\[x^{(k+1)} \leftarrow G^T x^{(k)}
\]
\[k \leftarrow k + 1
\]
\[\text{end while}
\]
\[\pi = x^{(k)}
\]

**Theorem 27.** The Power Method will always converge to the PageRank vector $\pi$, assuming that the starting vector $x^{(0)}$ satisfies $e^T x^{(0)} = 1$.

**Proof:** Suppose that the eigenvectors $v_1, v_2, \ldots, v_n$ of $G^T$ form a basis of $\mathbb{R}^n$. Then we can write the starting vector as $x^{(0)} = \sum_{i=1}^{n} c_i v_i$ for some coefficients $c_1, \ldots, c_n \in \mathbb{R}$. It follows that

\[
x^{(0)} = \sum_{i=1}^{n} c_i v_i
\]
\[
x^{(1)} = G^T x^{(0)} = G^T \sum_{i=1}^{n} c_i v_i = \sum_{i=1}^{n} c_i G^T v_i = \sum_{i=1}^{n} c_i \lambda_i v_i
\]
\[
x^{(2)} = G^T x^{(1)} = G^T \sum_{i=1}^{n} c_i \lambda_i v_i = \sum_{i=1}^{n} c_i \lambda_i G^T v_i = \sum_{i=1}^{n} c_i \lambda_i^2 v_i
\]
\[\vdots
\]
\[
x^{(k)} = G^T x^{(k-1)} = G^T \sum_{i=1}^{n} c_i \lambda_i^{k-1} v_i = \sum_{i=1}^{n} c_i \lambda_i^{k-1} G^T v_i = \sum_{i=1}^{n} c_i \lambda_i^k v_i
\]

By Theorem 25, we know that $\lambda_1 = 1$ and $|\lambda_i| \leq \alpha$ for all $2 \leq i \leq n$. So $\lambda_i^k$ goes to zero as
$k$ tends to infinity. We get
\[
\lim_{k \to \infty} x^{(k)} = \lim_{k \to \infty} c_1 \lambda_1^k v_1 = \lim_{k \to \infty} c_1 1^k \pi = c_1 \pi
\]
We assumed that $e^T x^{(0)} = 1$ holds. Suppose $e^T x^{(i)} = 1$ for some $i \in \mathbb{N}$. Then we also have
\[
e^T x^{(i+1)} = e^T G^T x^{(i)} = (Ge)^T x^{(i)} = e^T x^{(i)} = 1,
\]
because $G$ is a stochastic matrix. Hence $e^T x^{(i)} = 1$ for all $i \in \mathbb{N}$. So this is also true for the limit:
\[
1 = \lim_{k \to \infty} e^T x^{(k)} = e^T \lim_{k \to \infty} x^{(k)} = c_1 e^T \pi = c_1
\]
So the Power Method does indeed converge to the PageRank vector.

Obviously, it might happen that we cannot write $x^{(0)}$ as a linear combination of eigenvectors (i.e. $G^T$ is not diagonalisable). In this case, we can write $G^T = PJP^{-1}$, where $J$ is the Jordan form of $G^T$ and $P$ is the corresponding matrix containing the generalized eigenvectors. Each block $J_m$ of $J$ can be written as $\lambda I_m + N$, where $N$ is the matrix of all zeros except on its upper diagonal. This matrix $N$ is nilpotent and if $|\lambda| < 1$, $J_m^k \to 0$ as $k \to \infty$. There is only one eigenvalue of $G^T$ which does not satisfy $|\lambda| < 1$, which is $\lambda_1 = 1$. So the Power Method will converge to the eigenvector corresponding to this eigenvalue, which is the PageRank vector.

\[\square\]

**Theorem 28.** Suppose that the starting vector satisfies $e^T x^{(0)} = 1$. The error of the Power Method satisfies $||x^{(k)} - \pi|| = O(\alpha^k)$.

**Proof:** Once again, assume that $x^{(0)} = \sum_{i=1}^{n} c_i v_i$ for some constants $c_i$. Let $M = \max\{||c_i v_i|| : 2 \leq i \leq n\}$. By Theorem 25, $\lambda_1 = 1$ and $|\lambda_i| \leq \alpha$ for all $2 \leq i \leq n$. We find that
\[
||x^{(k)} - \pi|| = ||\pi + \sum_{i=2}^{n} c_i \lambda_i^k v_i - \pi|| = \sum_{i=2}^{n} ||c_i \lambda_i^k v_i|| = \sum_{i=2}^{n} |\lambda_i|^k ||c_i v_i|| \\
\leq \sum_{i=2}^{n} \alpha^k M = (n-1) M \alpha^k = O(\alpha^k)
\]

\[\square\]

For the Power Method, a starting vector $x^{(0)}$ is needed. Clearly, if $x^{(0)}$ is close to the PageRank vector $\pi$, the error will be smaller on iteration one. Thus, the Power Method will converge faster. It makes sense to use $x^{(0)} = v$ as the starting vector, since the random surfer will prioritize some web pages more than others according to the personalisation vector. Another option is to use $x^{(0)} = e/n$.

### 4.2 Implementation of the Power Method

The Power Method is a very simple numerical method to calculate the PageRank vector. This method only requires the repetitive calculation of matrix-vector products. However, this needs to be done with the matrix $G^T$, which is nowhere sparse as we have stated before. To prevent storage issues and slow computation time, it is preferred not to use the matrix $G$ but the very sparse matrix $H$. $G$ is by definition equal to:

\[
G = \alpha S + (1 - \alpha) e v^T = \alpha (H + d v^T) + (1 - \alpha) e v^T \\
= \alpha H + (\alpha d + (1 - \alpha) e) v^T
\]
Now, it follows that
\[
G^T x = (\alpha H + (\alpha d + (1 - \alpha) e) v^T) x = \alpha H^T x + v (\alpha d^T + (1 - \alpha) e^T) x \\
= \alpha H^T x + \alpha v (d^T x) + (1 - \alpha) v (e^T x) \\
= \alpha H^T x + (1 - \alpha + \alpha (d^T x)) v
\]

Here we used the fact that \(e^T x = 1\). This follows from the fact that the starting vector, and therefore also the next vectors, satisfy this equality. This gives us an easy and fast method to approximate the PageRank vector by using just the very sparse hyperlink matrix \(H\) and the dangling node vector \(d\).

### 4.3 A direct method

In the proof of Lemma 23, we have seen that the PageRank vector can also be calculated in a different way. We will further discuss this method now.

**Theorem 29.** The PageRank vector \(\pi\) is equal to
\[
\pi = (1 - \alpha) (I - \alpha S^T)^{-1} v
\]

**Proof:** Since \(e^T \pi = 1\) holds by definition of the PageRank vector, this theorem is a direct consequence of Lemma 23.

The matrix \(I - \alpha S^T\) is an important matrix with a lot of useful properties. The following properties are worth noting.

- All eigenvalues of \(I - \alpha S^T\) lie in the disk with center 1 and radius \(\alpha\). This follows directly from Gershgorin’s Theorem.
- \(I - \alpha S^T\) is invertible, as stated in Lemma 22.
- \(I - \alpha S^T\) is strictly diagonally dominant, or in other words, \(I - \alpha S^T\) is an \(M\)-matrix. For each row \(i\) the diagonal element is larger than the sum of the absolute value of the non-diagonal elements. We have seen this in the proof of Lemma 22.
- The row sums of \(I - \alpha S^T\) are exactly \(1 - \alpha\).

Furthermore, we can simplify Theorem 29 even more:

**Theorem 30.** Define \(x = (I - \alpha H^T)^{-1} v\). The PageRank vector \(\pi\) is equal to
\[
\pi = \frac{x}{e^T x}
\]

**Proof:** First, note that \(I - \alpha H^T\) is indeed invertible (we can apply the same proof as in Lemma 22 for the substochastic matrix \(H\)). Theorem 29 yields the equality \((I - \alpha S^T) \pi = (1 - \alpha) v\). Note that the sum of row \(i\) of the matrix \(H\) is equal to zero if \(i\) is a dangling node, and 1 otherwise. Hence \(H e = e - d\). We find
\[
G^T x = \frac{1}{e^T x} G^T x = \frac{x}{e^T x}.
\]

So \(G^T x = \frac{1}{e^T x} G^T x = \frac{x}{e^T x}\). But of course we also have \(e^T \frac{x}{e^T x} = 1\). Since the PageRank vector is unique, it must be equal to \(\pi = \frac{x}{e^T x}\). \(\square\)
The matrix $I - \alpha H^T$ satisfies many of the same properties as $I - \alpha S^T$:

1. All eigenvalues of $I - \alpha H^T$ lie in the disk with center 1 and radius $\alpha$.
2. $I - \alpha H^T$ is invertible.
3. $I - \alpha H^T$ is an $M$-matrix.
4. The row sums of $I - \alpha H^T$ are either 1 or $1 - \alpha$.

### 4.4 The Jacobi method

The last two theorems give us new ways to calculate the PageRank vector. We will use both theorems and the Jacobi method to approximate the PageRank. This method makes use of the fact that

$$(I - \alpha S^T)^{-1} = \sum_{n=0}^{\infty} (\alpha S^T)^n$$

This sum converges because $\rho(\alpha S^T) = \alpha \rho(S^T) \leq \alpha \cdot 1 < 1$. We approximate this series by a partial sum. This requires the computation of multiple matrix-matrix products, which is not preferred. The Jacobi Method is an efficient way of computing this partial sum.

**Algorithm 31.** The Jacobi method (applied to the matrix $S$) is:

$k \leftarrow 0$

$x^{(0)} \leftarrow (1 - \alpha)v$

**while** convergence not reached **do**

$x^{(k+1)} \leftarrow \alpha S^T x^{(k)} + (1 - \alpha)v$

$k \leftarrow k + 1$

**end while**

$\pi \leftarrow x^{(k)}$

**Theorem 32.** The Jacobi method converges to the PageRank vector $\pi$. Furthermore, the error after $k$ iterations is of order $O(\alpha^k)$.

**Proof:** By induction, it is easy to see that $x^{(k)} = (1 - \alpha) \sum_{n=0}^{k} (\alpha S^T)^n v$. This is clearly true for $k = 0$: $x^{(0)} = (1 - \alpha)v = (1 - \alpha) \sum_{n=0}^{0} (\alpha S^T)^n v$. Assume that the equality holds for some $k \in \mathbb{N}$. Then we also have $x^{(k+1)} = \alpha S^T x^{(k)} + (1 - \alpha)v = \alpha S^T (1 - \alpha) \sum_{n=0}^{k} (\alpha S^T)^n v + (1 - \alpha)v = (1 - \alpha) \sum_{n=1}^{k+1} (\alpha S^T)^n v + (1 - \alpha)v = (1 - \alpha) \sum_{n=0}^{k+1} (\alpha S^T)^n v$. Thus, the equality is true for all $k \in \mathbb{N}$. As $k$ tends to infinity, $(1 - \alpha) \sum_{n=0}^{k} (\alpha S^T)^n v$ tends to $(1 - \alpha) \sum_{n=0}^{\infty} (\alpha S^T)^n v = (1 - \alpha)(I - \alpha S^T)^{-1} v$. By Theorem 29, this value is equal to $\pi$.

Note that after $k$ iterations, the error of $x$ is equal to

$$\|\pi - x^{(k)}\| = \|(1 - \alpha)(I - \alpha S^T)^{-1} v - (1 - \alpha) \sum_{n=0}^{k} (\alpha S^T)^n v\|$$

$$= (1 - \alpha)\| (I - \alpha S^T)^{-1} v - \sum_{n=0}^{k} (\alpha S^T)^n v\|$$

$$= (1 - \alpha)\| \sum_{n=k+1}^{\infty} (\alpha S^T)^n v\|$$

$$\leq (1 - \alpha)\| \sum_{n=k+1}^{\infty} (\alpha S^T)^n v\|$$

$$\leq (1 - \alpha)\| \sum_{n=0}^{\infty} (\alpha S^T)^n v\|$$

$$\leq (1 - \alpha)\| \sum_{n=0}^{\infty} \alpha^n \| S^T \| \| v\|$$

$$\leq (1 - \alpha)\| \sum_{n=0}^{\infty} \alpha^n \| v\| \| S^T \|$$

$$\leq (1 - \alpha)\| \sum_{n=0}^{\infty} \alpha^n \| = (1 - \alpha)\| v\| \alpha^k = \mathcal{O}(\alpha^k)$$

as $k$ tends to infinity. Additionally, the largest eigenvalue of $\alpha H^T$ is important for the speed of this method. 

\[ \square \]
Note that we can also apply the Jacobi method on the matrix $\alpha H^T$ to approximate $x = (I - \alpha H^T)^{-1}v$. This is sufficient, since by Theorem 30, the PageRank vector is equal to $\pi = x/(e^T x)$. Hence the following algorithm can be used as well:

**Algorithm 33.** The Jacobi method (applied to the matrix $H$) is:

```plaintext
k ← 0
x^(0) ← v

while convergence not reached do
    x^(k+1) ← $\alpha H^T x^{(k)} + v$
    k ← k + 1
end while

\[ \pi ← x^{(k)}/(e^T x^{(k)}) \]
```

By the same proof as in Theorem 30, this algorithm also converges to $\pi$ and the error after $k$ iteration is of order $O(\alpha^k)$. The advantage of this method is that $H$ is much sparser than $S$, so the computation of the matrix-vector product $H^T x^{(k)}$ will be faster.

### 4.5 Optimized Jacobi

It is worth checking if we can optimize these methods. We can try the following shift-and-scale method. We have seen that if we want to calculate $\pi$, it sufficient to solve the linear system $(I - \alpha H^T)x = v$. Multiply this equation by a certain constant $\beta$ and then shift it back and forth. We get

\[
(I - \alpha H^T)x = v
\]

\[\Rightarrow\]

\[\beta(I - \alpha H^T)x = \beta v
\]

\[\Rightarrow I - (I - \beta(I - \alpha H^T))x = \beta v
\]

Define $Q_\beta := (I - \beta(I - \alpha H^T))$. $x$ now satisfies the equality $(I - Q_\beta)x = \beta v$. Thus, we could also apply the Jacobi algorithm to this matrix $Q_\beta$ for a certain value of $\beta$. It might happen that the largest eigenvalue of $Q_\beta$ is smaller than the largest eigenvalue of $\alpha H^T$ (in absolute value), which will imply that the Jacobi method will converge faster. Therefore, our goal is to find the value of $\beta$ such that $\rho(Q_\beta)$ is minimal.

Suppose $\lambda_1, \lambda_n, \ldots, \lambda_n$ are the eigenvalues of $\alpha H^T$ with corresponding eigenvectors $p_1, \ldots, p_n$. Then for each $1 \leq i \leq n$, we have

\[
Q_\beta p_i = (I - \beta(I - \alpha H^T))p_i = p_i - \beta(I - \alpha H^T)p_i = p_i - \beta p_i + \beta(\alpha H^T)p_i
\]

\[= (1 - \beta + \beta \lambda_i)p_i
\]

Thus, the eigenvalues of $Q_\beta$ are $1 - \beta + \beta \lambda_1, 1 - \beta + \beta \lambda_2, \ldots, 1 - \beta + \beta \lambda_n$. So the spectrum of $\alpha H^T$ gets multiplied by a factor $\beta$ and then gets shifted by $1 - \beta$.
Figure 4: Eigenvalues of the matrix $\alpha H^T$ for $\alpha = 0.85$.

See figure 4 for an example how the spectrum of $\alpha H^T$ can look like (here we have used an induced sub graph of the mathworks.com web) for $\alpha = 0.85$. Since $H$ is sub-stochastic, all eigenvalues lie in the red disk of center 0 and radius 0.85 by Gershgorin’s Theorem. In this example, there is an eigenvalue $\lambda_1$ exactly equal to 0.85 but there are no eigenvalues with real value less than the eigenvalue $\lambda_2 \approx -0.3234$. Thus, we can make the largest eigenvalue smaller by shifting all eigenvalues to the left. It is easy to see that it is optimal to shift these eigenvalues to the left until the right-most and left-most eigenvalues $\lambda_1'$ and $\lambda_2'$ have the same distance $r$ from zero, i.e.

$$
\lambda_1' = 1 - \beta + \beta \cdot -0.3234 = -r, \quad \text{and}
\lambda_2' = 1 - \beta + \beta \cdot 0.85 = r.
$$

By solving this system, we find that the optimal $\beta$ is equal to $\beta \approx 1.357$. For this value, the spectral radius is $r \approx 0.79$. This coincides with figure 5, which shows that the eigenvalues of $Q\beta$ for $\beta = 1.357$ lie in the disk with center 0 and radius 0.79.

So it is better to apply the Jacobi Method on $Q\beta$ for $\beta = 1.357$. The error after $k$ iterations is of order $O(0.79^k)$ for this method, while the error of the regular Jacobi Method is of order $O(0.85^k)$. It might not seem like a huge difference, but after 50 iterations, the error of the Optimized Jacobi Method is roughly $(0.85)^{50} \approx 40$ times as small. Thus, much less iterations will be needed and so we will be able to approximate the PageRank vector much faster.
The Optimized Jacobi Method has two disadvantages. The first is that it is very hard to find the optimal value of $\beta$. In our example, we simply calculated each eigenvalue, which makes it very easy to find the value $\beta$ such that the Jacobi Method is optimized. However, for very large matrices, it is not practical to calculate the eigenvalues. We do know, however, an upper bound to these eigenvalues. By Gershgorin’s Theorem, all eigenvalues of $Q_\beta$ lie in the disk with center $1 - \beta$ and radius $\alpha \beta$. Hence any eigenvalue $\lambda$ of $Q_\beta$ satisfies $|\lambda| \leq \max\{(1 - \beta) + \alpha \beta, (1 - \beta) - \alpha \beta\}$. One can easily check that this upper bound is minimal if and only if $\beta = 1$. Therefore, $Q_\beta = \alpha H^T$, which means that we cannot generally optimize the Jacobi method.

The other problem is the fact that the eigenvalues of $\alpha H^T$ usually fill the whole Gershgorin disk of radius $\alpha$. Therefore, a shift-and-scale approach will not be able to decrease the spectral radius of the matrix, which means that it is often not even possible to optimize the Jacobi Method in this way.
5 Numerical Experiments

In the previous chapter we have discussed multiple algorithms to approximate the PageRank vector π, namely the Power Method and the Jacobi method (applied to two different matrices). The error of both methods is of order $O(\alpha^k)$, where $k$ is the number of iterations. Furthermore, in some cases the Jacobi Method can be further optimised. In this chapter we will apply these algorithms to see whether there are differences with respect to the computation time. We will compare both the computation time and the amount of iterations needed.

5.1 Data

Before we can test the algorithms, we need data to calculate the link matrix $H$. Clearly, we do not know the complete graph of the world wide web, as the web consists of multiple billion of pages and links. Thus, we will construct smaller link matrices to be able to test the algorithms. We have done this with a couple of different methods. Each option yields a completely different matrix (in size or structure). All algorithms will be applied on each matrix to see what method benefits most of a certain property of the web.

- The first method of obtaining a link matrix is by getting a small subset of the web. Just like Google, we use a surfer to obtain this graph. Starting in any webpage, the surfer analyses the source code of this page and determines where this page links to. We add these links to the link matrix. Then, the surfer opens one of the new pages. This process is repeated until $n$ pages have been found and all pages have been checked for links. This will construct a very realistic web. We do have an option to filter certain pages, such as images or JavaScript files. These are files which a user generally does not want to see as search results. It is, however, also interesting to allow these pages. This will yield a matrix with a large amount of dangling nodes, and we will discuss why this matters for the algorithms.

The disadvantage of using a surfer is that it is very slow. It generally takes more than a day to obtain a link matrix with more than 10 thousand nodes. See A.1.1 for the Matlab code of the surfer we have made and used.

- Luckily for us, many other people have made their own surfer to obtain even larger link matrices. This matrices are posted online, so that anyone could use them for testing purposes. One of these matrices forms a collection of websites that are about California. It consists of roughly 10 thousand pages, and about 50 percent of these are dangling nodes. We thank Jon Kleinberg for this dataset [13].

We have also used two link matrices which are far larger in size. The first one is a collection of all (almost 300 thousand) web pages in the Stanford University website. The second one contains all web pages in the whole Stanford-Berkerley web and has almost 700 thousand pages and more than 7 million links. We thank Sep Kamvar for these datasets [14].

It is worth noting that both of these larger matrices only have a relative small number of dangling nodes. This will have consequences for the iteration time of each numerical method.

- The last and most obvious method is by simply generating a link matrix randomly.
This allows us to construct a large variety of link matrices. For example, one could easily construct a link matrix with more than a million nodes. It is also possible to choose the average amount of dangling nodes on the web or the average amount of links.

However, a randomly generated web might not be very realistic; there does not need to be any structure and the link matrix can even be aperiodic. A consequence of this is that all eigenvalues of $H$ appeared to be much smaller (in absolute value) compared with those of a ‘realistic’ link matrix. Since $\rho(H)$ becomes smaller, the numerical methods will converge a lot faster. Thus, since such results are not realistic, we will not use these matrices.

In summary, we have used the following link matrices to test the algorithms. See also the Matlab files A.1.1–A.1.4. As one can see, there is a large variety of amount of nodes, dangling nodes, and links.

<table>
<thead>
<tr>
<th>Name</th>
<th>Pages</th>
<th>Links</th>
<th>Dangling nodes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5000</td>
<td>1430936</td>
<td>244</td>
<td>mathworks.com (from our own crawler)</td>
</tr>
<tr>
<td>B</td>
<td>9664</td>
<td>16150</td>
<td>5027</td>
<td>Sites about California</td>
</tr>
<tr>
<td>C</td>
<td>281903</td>
<td>2312497</td>
<td>172</td>
<td>Stanford University Web</td>
</tr>
<tr>
<td>D</td>
<td>683446</td>
<td>7583376</td>
<td>4735</td>
<td>Stanford-Berkeley Web</td>
</tr>
</tbody>
</table>

\[ 5.2 \text{ Convergence criteria} \]

The numerical methods we discussed are iterative. The iterations need to stop when a satisfactory approximation of the PageRank vector has been found. But when is this the case? Let $\|\cdot\|$ be a norm. We say that an approximation $\mathbf{x}$ of $\pi$ is good enough if the relative residual is small enough, i.e.

\[
\frac{\|G^T \mathbf{x} - \mathbf{x}\|}{\|\mathbf{x}\|} < \varepsilon
\]
for some small number $\varepsilon > 0$. We picked $\varepsilon = 10^{-5}$ and let $\|\cdot\|$ be the one-norm. The reason for this choice is that the order of the PageRank values is important and not the exact PageRank value. If the sup norm is used, for example, the ordered PageRank vector might be completely different.

- Let $x^{(i)}$ be the approximation of $\pi$ by using the Power Method for $i$ iterations. The Power Method is repeated until $\|x^{(i+1)} - x^{(i)}\|/\|x^{(i)}\| < \varepsilon$, i.e. $e^T(x^{(i+1)} - x^{(i)}) < 10^{-5}$, since $e^T(x^{(j)}) = 1$ for all $j$.

- The Jacobi method (applied to the matrix $S$) after $i$ iterations yields an approximation $x^{(i)}$ such that $(I - \alpha S^T)x^{(i)} \approx (1 - \alpha)v$. The iteration will be repeated until the residual $\|((I - \alpha S^T)x^{(i)} - (1 - \alpha)v\| < \varepsilon$, i.e. $e^T((I - \alpha S^T)x^{(i)} - (1 - \alpha)v) < 10^{-5}$.

To prevent unnecessary computations, we make use of the fact that $\alpha S^T x^{(i)} = \alpha(H^T x^{(i)} + v(d^T x^{(i)}))$. This vector can then be reused to calculate the residual. See Matlab code A.2.2 for the implementation.

- For the Jacobi method applied to $H$, one needs to normalise $x^{(i)}$ to get an approximation of $\pi$. Thus, the iteration is repeated until $\|\alpha H^T x^{(i)} - v\|/(e^T x^{(i)}) < \varepsilon$, i.e. $e^T(\alpha H^T x^{(i)} - v) < 10^{-5}e^T x^{(i)}$.

5.3 Numerical Results

The Power Method and both Jacobi methods (applied to different matrices) will now be applied to these four matrices. We have seen that the value of $\alpha$ is important for the speed of each method. Furthermore, we will use two personalisation vectors to see if there is any difference; the uniform vector $w_1 = e/n$ and a random probability vector $w_2$.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$v$</th>
<th>Power Method</th>
<th>Jacobi ($S$)</th>
<th>Jacobi ($H$)</th>
<th>Optimized Jacobi</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>$w_1$</td>
<td>0.0685s (13 iterations)</td>
<td>0.0739s (16 iterations)</td>
<td>0.0687s (15 iterations)</td>
<td>0.0592s (13 iterations)</td>
</tr>
<tr>
<td>0.50</td>
<td>$w_2$</td>
<td>0.0662s (13 iterations)</td>
<td>0.0724s (16 iterations)</td>
<td>0.0683s (15 iterations)</td>
<td>0.0583s (13 iterations)</td>
</tr>
<tr>
<td>0.85</td>
<td>$w_1$</td>
<td>0.1365s (45 iterations)</td>
<td>0.1656s (60 iterations)</td>
<td>0.1532s (55 iterations)</td>
<td>0.1298s (48 iterations)</td>
</tr>
<tr>
<td>0.85</td>
<td>$w_2$</td>
<td>0.1341s (44 iterations)</td>
<td>0.1678s (60 iterations)</td>
<td>0.1524s (55 iterations)</td>
<td>0.1303s (48 iterations)</td>
</tr>
<tr>
<td>0.95</td>
<td>$w_1$</td>
<td>0.2704s (110 iterations)</td>
<td>0.3856s (167 iterations)</td>
<td>0.3418s (152 iterations)</td>
<td>0.2977s (134 iterations)</td>
</tr>
<tr>
<td>0.95</td>
<td>$w_2$</td>
<td>0.2682s (110 iterations)</td>
<td>0.3862s (167 iterations)</td>
<td>0.3443s (152 iterations)</td>
<td>0.2984s (134 iterations)</td>
</tr>
</tbody>
</table>

Table 1: The numerical methods applied to the 5000 $\times$ 5000 (Mathworks) link matrix $A$. Note that the results are the average out of 100 simulations.

We picked $\beta = 1.17$ for the Optimized Jacobi Method, as it seemed that this value will require the least amount of iterations (see also Matlab code A.2.5). As expected, the Optimized Jacobi Method is always faster than the regular Jacobi Method applied to $H$, and it is better to apply the Jacobi Method on $H$ than to apply it on $S$. The difference in computation time between the three methods between is relatively small. Interestingly, the Power Method becomes more favourable if $\alpha$ is close to 1.
There does not seem to be any difference in computation time regarding the choice of the personalisation vector. Thus, for the next experiments, we will only look at the uniform personalisation vector $v = e/n$.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Power Method</th>
<th>Jacobi ($S$)</th>
<th>Jacobi ($H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.0034s</td>
<td>0.0047s</td>
<td>0.0021s</td>
</tr>
<tr>
<td></td>
<td>(12 iterations)</td>
<td>(16 iterations)</td>
<td>(11 iterations)</td>
</tr>
<tr>
<td>0.85</td>
<td>0.0111s</td>
<td>0.0154s</td>
<td>0.0067s</td>
</tr>
<tr>
<td></td>
<td>(47 iterations)</td>
<td>(60 iterations)</td>
<td>(42 iterations)</td>
</tr>
<tr>
<td>0.95</td>
<td>0.0330s</td>
<td>0.0432s</td>
<td>0.0201s</td>
</tr>
<tr>
<td></td>
<td>(142 iterations)</td>
<td>(167 iterations)</td>
<td>(128 iterations)</td>
</tr>
</tbody>
</table>

Table 2: The numerical methods applied to the 9664 $\times$ 9664 (California) link matrix $B$.

When applying the numerical methods on the second matrix $B$, the results are somewhat different. There was no value of $\beta \neq 1$ that decreases the amount of iterations needed before the Optimized Jacobi Method converges, hence the Jacobi Method could not be made faster. It is remarkable that for this matrix, the Jacobi Method applied on $H$ is better than the Power Method. One reason for this is that about 48% of the nodes of this web are dangling nodes. The Jacobi Method applied on $H$ does not depend on the amount dangling nodes, hence the difference in iteration time.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Power Method</th>
<th>Jacobi ($S$)</th>
<th>Jacobi ($H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.3802s</td>
<td>0.4907s</td>
<td>0.4240s</td>
</tr>
<tr>
<td></td>
<td>(14 iterations)</td>
<td>(16 iterations)</td>
<td>(16 iterations)</td>
</tr>
<tr>
<td>0.85</td>
<td>0.9921s</td>
<td>1.3807s</td>
<td>1.1580s</td>
</tr>
<tr>
<td></td>
<td>(50 iterations)</td>
<td>(60 iterations)</td>
<td>(60 iterations)</td>
</tr>
<tr>
<td>0.95</td>
<td>2.7790s</td>
<td>3.8045s</td>
<td>3.0155s</td>
</tr>
<tr>
<td></td>
<td>(150 iterations)</td>
<td>(167 iterations)</td>
<td>(166 iterations)</td>
</tr>
</tbody>
</table>

Table 3: The numerical methods applied to the 281903 $\times$ 281903 (Stanford University) link matrix $C$.

For the large Stanford University matrix, the Power Method is only slightly faster than the Jacobi Method applied on $H$. It is interesting to see that even though this web contains almost 300 thousand nodes, each method will return an approximation of the PageRank vector in less than 5 seconds.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Power Method</th>
<th>Jacobi ($S$)</th>
<th>Jacobi ($H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.5609s</td>
<td>0.7747s</td>
<td>0.6011s</td>
</tr>
<tr>
<td></td>
<td>(14 iterations)</td>
<td>(16 iterations)</td>
<td>(16 iterations)</td>
</tr>
<tr>
<td>0.85</td>
<td>1.5586s</td>
<td>2.3120s</td>
<td>1.6927s</td>
</tr>
<tr>
<td></td>
<td>(52 iterations)</td>
<td>(60 iterations)</td>
<td>(59 iterations)</td>
</tr>
<tr>
<td>0.95</td>
<td>4.2054s</td>
<td>6.0926s</td>
<td>4.4134s</td>
</tr>
<tr>
<td></td>
<td>(154 iterations)</td>
<td>(167 iterations)</td>
<td>(164 iterations)</td>
</tr>
</tbody>
</table>

Table 4: The numerical methods applied to the 683446 $\times$ 683446 (Stanford-Berkeley) link matrix $D$. 
For the largest link matrix, the results are the same as before. Note that less than 1% of the nodes are dangling nodes, and thus the speed of the Power Method is not much affected by these nodes.

Another way to compare the numerical methods is by first fixing an amount of iterations. All methods will be iterated this many times, and the residual can be calculated afterwards. This comparison might be better, since the residual should not be calculated after each iteration. Figure 7 shows how large the residual of the approximation of $\pi$ is, using the Stanford-Berkeley link matrix $D$. The figure shows that Power Method is only slightly faster than the Jacobi Method applied to $H$. However, after a while, the Jacobi Method becomes faster.

![Figure 7: The three numerical methods compared using the link matrix $D$.](image)

### 5.4 Discussion

As we have seen in the previous chapter, each numerical method has a convergence rate of $O(\alpha^k)$. Therefore, it is not too unexpected to see that the speed of the three methods is about the same. In most cases, the Power Method is the fastest algorithm. However, the difference between this method and the Jacobi Method applied on $H$ is very small. In some cases, for example if the amount of dangling nodes in a web is large, the latter will converge faster.

We were only able to decrease the spectral radius of the mathworks.com link matrix $A$. Thus, the Optimized Jacobi Method is of limited use.

The Jacobi method is based on the equality $(I - \alpha S^T)\pi = (1 - \alpha)v$. As we will see in the next chapter, using this equality will have an additional advantage.

31
6 Random Teleportation Parameter

The teleportation parameter $\alpha$ is the probability that the random surfer will click on a link. The random surfer will, with a probability of $1 - \alpha$, go to any page on the web. The choice of this page is random and distributed according to the personalisation vector. We have seen that we can calculate the PageRank vector much faster if $\alpha$ is small. However, for a small $\alpha$, the structure of the web will hardly matter.

Google has stated that they used $\alpha = 0.85$. But why did they pick this exact value? Except for the fact that $\alpha$ shouldn’t be close to 0 or 1, there is no mathematical reason behind the choice $\alpha = 0.85$. Researchers such as [8] have proposed to use $\alpha = 0.5$ instead. We will look at this problem in a different way. The probability that a user enters a different link in the address bar obviously depends on the user. It therefore makes sense to see $\alpha$ as a distribution of teleportation parameters. Gleich et al.[9] did research about this parameter. With a large amount of data, they concluded that this parameter fits a certain Beta distribution, as we can see in the following figure.

![Figure 8: The distribution of the teleportation parameter $\alpha$ (Source: [9]).](image)

6.1 Expected PageRank

Since the PageRank depends on the value of $\alpha$, we will from now on denote the PageRank vector corresponding to a certain value of $\alpha$ by $\pi(\alpha)$. Suppose that we want a single vector that corresponds to the PageRank vector of everyone. The most straightforward choice is $\pi(E[\alpha])$: the PageRank vector calculated for the mean of all choices of $\alpha$. However, this does not yield a satisfactory vector. Instead, we will look at the expected PageRank vector.

**Definition 34.** Let $f : [0, 1] \rightarrow \mathbb{R}^+$ be the probability density function of the stochastic parameter $\alpha$. We define the expected PageRank vector $\langle \pi \rangle$ by

$$
\langle \pi \rangle = \int_0^1 \pi(\alpha)f(\alpha)d\alpha
$$

We assume that this density function of $\alpha$ is very small if $\alpha$ is close to 1, i.e. $f(\alpha) \approx 0$ for $\alpha \approx 1$. This is because we originally assumed that $0 \leq \alpha < 1$ should hold; if $\alpha = 1$, there is no teleportation, hence there is no guarantee that a unique PageRank vector exists.
It is easy to see that the expected PageRank is a probability vector as well. Since for each \( \alpha \) we have \( e^T \pi(\alpha) = 1 \), it follows that

\[
e^T(\langle \pi \rangle) = e^T \int_0^1 \pi(\alpha) f(\alpha) d\alpha = \int_0^1 e^T \pi(\alpha) f(\alpha) d\alpha \int_0^1 1 f(\alpha) d\alpha = 1
\]

Note that by Theorem 30, the expected PageRank vector satisfies

\[
\langle \pi \rangle = \int_0^1 (1 - \alpha)(I - \alpha S^T)^{-1} v f(\alpha) d\alpha = \int_0^1 \sum_{n = 0}^{\infty} (1 - \alpha)(\alpha S^T)^n v f(\alpha) d\alpha
\]

Additionally, since \( S, v, \) and \( f \) are all non-negative, the integral and summation can be interchanged. The fact that this is allowed is a direct consequence of the Monotone Convergence Theorem, see [12, p.82]. Thus,

\[
\langle \pi \rangle = \sum_{n = 0}^{\infty} \int_0^1 (1 - \alpha)(\alpha S^T)^n v f(\alpha) d\alpha
\]

Suppose that \( \alpha \) is uniformly distributed between 0 and \( r \) for some value \( r < 0 \), i.e. \( f(\alpha) = \frac{1}{r} \) for \( 0 \leq \alpha \leq r \) and 0 otherwise. In this case, it is possible to calculate this integral:

\[
\langle \pi \rangle = \sum_{n = 0}^{\infty} \int_0^1 (1 - \alpha)(\alpha S^T)^n v f(\alpha) d\alpha = \sum_{n = 0}^{\infty} \int_0^r (1 - \alpha)\alpha^n \cdot \frac{1}{r} d\alpha (S^T)^n v
\]

\[
= \frac{1}{r} \sum_{n = 0}^{\infty} \left[ \frac{\alpha^{n+1}}{n+1} - \frac{\alpha^{n+2}}{n+2} \right]_0^r (S^T)^n v
\]

However, for general density functions \( f \) there is no practical way of calculating \( \langle \pi \rangle \). Instead, we will try to approximate it. An obvious way to do so is by calculating the Riemann sum. Suppose that we have picked some numbers \( 0 = x_0 < x_1 < \ldots < x_k = 1 \) and meshpoints \( \alpha_i \in [x_{i-1}, x_i] \) for all \( 1 \leq i \leq k \). The PageRank vector is approximately

\[
\langle \pi \rangle \approx \sum_{i=1}^{k} \pi(\alpha_i) f(\alpha_i) \cdot [x_i - x_{i-1}]
\]

This requires the computation of the PageRank vector for \( k \) different values of \( \alpha \). In general, \( k \) must be large if we want a good approximation of \( \langle \pi \rangle \). We could approximate \( \pi(\alpha) \) for each value of \( \alpha \) with one of the numerical methods we have discussed, but as the approximation of a single PageRank vector corresponding to a specific value of \( \alpha \) already takes very long, this is not preferred. However, the Power Method has the advantage that it can easily be optimised so that the computation is faster.

### 6.2 Optimized Power Method

As we have seen, the convergence speed of the Power Method is of order \( O(\alpha^k) \). An advantage of the Power Method over the Jacobi Method is that one can pick a starting vector for the Power Method. Clearly, if this starting vector is close to the real PageRank vector, less iterations will be needed until a satisfactory approximation will be computed.
In paragraph 4.1, we have proposed to use the personalisation vector $v$ as the starting vector, since this might be the best consequent guess we can make. However, if the PageRank vector needs to be computed for many different values of $\alpha$, we already have an idea how the PageRank vector should look like. Thus, we can use the PageRank vector corresponding to a different value of $\alpha$ as our initial guess. Suppose that $\pi(\alpha)$ should be computed for $\alpha_1, \ldots, \alpha_m$, where $0 \leq \alpha_1 < \ldots < \alpha_m < 1$. We can apply the following algorithm:

**Algorithm 35.**

\[
\begin{align*}
\pi(\alpha_1) & \leftarrow \text{PowerMethod}(v) \\
\text{for } k = 2 \text{ to } m \text{ do} & \\
\pi(\alpha_k) & = \text{PowerMethod}(\pi(\alpha_{k-1})) \\
\text{end for}
\end{align*}
\]

Here PowerMethod($w$) stands for that the Power Method should be applied with the starting vector $w$. Note that $\pi(\alpha_1)$ can also be computed using a different numerical method, but for $k = 2, \ldots, m$ the Power Method needs to be applied.

This algorithm is based on the idea that $\pi(\alpha)$ is approximately equal to $\pi(\alpha')$ if $\alpha$ is close to $\alpha'$. In general, this is true. However, note that

\[
\pi = (1 - \alpha)(I - \alpha S^T)^{-1} v
\]

The smallest eigenvalue of $(I - \alpha S^T)$ is exactly $1 - \alpha$. If $\alpha$ is close to 1, this eigenvalue approximates 0. To be more precise, the 1-norm condition number of $I - \alpha S^T$ is equal to $\frac{1}{\lambda_{\min}(I - \alpha S^T)}$, see [7]. Note that as $\alpha$ tends to 1 the condition number tends to infinity. The consequence of this is that a small relative change in $(I - \alpha S^T)$ (i.e. a small change in $\alpha$) can produce a large change in $(I - \alpha S^T)^{-1}$. Therefore, the PageRank vector can differ greatly for two values $\alpha \approx \alpha'$ close to 1. So the starting error will increase as $\alpha_k$ increases, and thus more iterations may be needed until the Optimized Power Method converges.

In the next paragraph we will discuss a much more efficient (reduced-order) method that is able to approximate the PageRank vector for all values of $\alpha$.

### 6.3 Reduced-order modelling

By Theorem 29, the PageRank vector corresponding to a certain value of $\alpha$ is the solution of the linear system $(I - \alpha S^T)\pi(\alpha) = (1 - \alpha)v$. Furthermore, we have shown that it is sufficient to solve the system $(I - \alpha H^T)x(\alpha) = v$. Our goal is to approximate this solution $x(\alpha)$ (and thus for $\pi(\alpha)$) for many different values of $\alpha$. To do so, we will make use of the shift-invariance of the so-called Krylov space. The following approach is based on [10].

**Definition 36.** The Krylov space $K_m$ is defined by:

\[
K_m(A, w) = \text{span}(w, Aw, A^2w, \ldots, A^{m-1}w)
\]

Since the Krylov space $K_m(A, w)$ is a vector space, we have $K_m(A, w) = K_m(\beta A, w)$ for any $\beta \neq 0$. Because $w \in K_m(A, w)$, we can also shift this space without changing it. So $K_m(A, w) = K_m(I - \beta A, w)$ for any $\beta \neq 0$. 34
We will look at the Krylov space of the matrix $H^T$ with respect to the personalisation vector $v$. Then we will try to find a vector $x$ in this space such that $(I - \alpha H^T)x$ approximates $v$. If $x$ is then normalised with respect to its 1-norm, this vector approximates $\pi(\alpha)$. The following theorem is important; this guarantees that the algorithm will converge.

**Theorem 37.** The PageRank vector $\pi$ satisfies $\pi \in K_n(H^T, v)$.

**Proof:** First, note that $K_m(H^T, v) = K_m(I - \alpha H^T, v)$. Let the characteristic polynomial of $I - \alpha H^T$ be equal to $p(\lambda) = c_0 + c_1 \lambda + c_2 \lambda^2 + \ldots + \lambda^n$. By the Cayley-Hamilton theorem, $I - \alpha H^T$ satisfies

$$0 = p(I - \alpha H^T) = c_0 I + c_1 (I - \alpha H^T) + c_2 (I - \alpha H^T)^2 + \ldots + (I - \alpha H^T)^n$$

Since $(I - \alpha H^T)$ is invertible, 0 is not an eigenvalue. Hence $c_0 \neq 0$. By dividing by $c_0$, subtracting $I$ and finally multiplying by $(I - \alpha H^T)^{-1}$, we get

$$(I - \alpha H^T)^{-1} = -\frac{1}{c_0} (c_1 + c_2 (I - \alpha H^T) + \ldots + (I - \alpha H^T)^{n-1})$$

Hence,

$$x = (I - \alpha H^T)^{-1}v = -\frac{1}{c_0} (c_1 v + c_2 (I - \alpha H^T)v + \ldots + (I - \alpha H^T)^{n-1}v) \in K_n(I - \alpha H^T, v)$$

Thus, $\pi = \frac{x}{\|x\|} \in K_n(I - \alpha H^T, v) = K_n(H^T, v)$.

The last theorem shows why it makes sense to look for a solution of the equation $(I - \alpha H^T)x = v$ in the space $K_m(H^T, v)$. To use this space, we will construct an orthonormal basis by using the *Arnoldi algorithm* applied to the vectors $v, H^T v, (H^T)^2 v, \ldots, (H^T)^{n-1} v$. This is essentially a modified version of the Gram Schmidt algorithm.

**Algorithm 38.** The Arnoldi Algorithm applied to $w_1 = v/\|v\|_2$ is:

```python
for k = 1 to m do
    w_{k+1} \leftarrow H^T w_k
    for j = 1 to k do
        u_{j,k} \leftarrow \langle w_j, w_{k+1} \rangle
        w_{k+1} \leftarrow w_{k+1} - u_{j,k} w_j
    end for
    u_{k+1,k} \leftarrow \|w_{k+1}\|_2
    if u_{k+1,k} = 0 then
        Stop;
    end if
    w_{k+1} \leftarrow w_{k+1} / u_{k+1,k}
end for
```

**Theorem 39.** Assuming the Arnoldi Algorithm does not stop before $m$ iterations, it creates an orthonormal basis of $K_m(H^T, v)$.

**Proof:** By induction, we show that $w_k = p_{k-1}(H^T)v$ for some polynomial $p_{k-1}$ of degree $k - 1$. The fact that this is true is trivial for $k = 1$, since $w_1 = v/\|v\|_2$. Let the induction hypothesis hold for some $k \in \mathbb{N}$. Note that
\[
\mathbf{w}_{k+1} = \frac{1}{u_{k+1,k}} (H^T \mathbf{w}_k - \sum_{j=1}^{k} u_{j,k} \mathbf{w}_j) = \frac{1}{u_{k+1,k}} (H^T p_{k-1}(H^T) \mathbf{v} - \sum_{j=1}^{k} u_{j,k} p_{j-1}(H^T) \mathbf{v})
\]

This shows that \( \mathbf{w}_{k+1} = p_k(H^T) \mathbf{v} \), where the polynomial \( p_k \) is defined as \( p_k(x) = (xp_{k-1}(x) - \sum_{j=1}^{k} u_{j,k} p_{j-1}(x))/u_{k+1,k} \). Since \( p_{k-1} \) is assumed to have degree \( k - 1 \), the degree of \( p_k \) is exactly \( k \). Furthermore, the Arnoldi vectors are orthonormal by construction, hence they form an orthonormal basis of \( K_m(H^T, \mathbf{v}) \).

But what happens if the Arnoldi Algorithm breaks down before creating \( m \) Arnoldi vectors? Theorem 40 shows that this should be seen as a good thing.

**Theorem 40.** Suppose that the Arnoldi Algorithm stops after \( k < m \) iterations. Then the dimension of the Krylov space is maximal, i.e. \( K_k(H^T, \mathbf{v}) = K_{k+1}(H^T, \mathbf{v}) \) for all \( l \geq k \).

**Proof:** If the Arnoldi Algorithm breaks down at iteration \( k \), we must have \( u_{k+1,k} = 0 \) and therefore \( \mathbf{w}_{k+1} = 0 \). Thus, we also have

\[
H^T \mathbf{w}_k = \sum_{j=1}^{k} u_{j,k} \mathbf{w}_j
\]

Let \( \mathbf{z} \in K_k(H^T, \mathbf{v}) \) be arbitrary. By Theorem 39, we can write \( \mathbf{z} \) as the linear combination of Arnoldi vectors: \( \mathbf{z} = \sum_{j=1}^{k} c_j \mathbf{w}_j \) for some coefficients \( c_j \). It follows that

\[
H^T \mathbf{z} = H^T \sum_{j=1}^{k} c_j \mathbf{w}_j = \sum_{j=1}^{k} c_j H^T \mathbf{w}_j = \sum_{j=1}^{k-1} c_j H^T \mathbf{w}_j + c_k H^T \mathbf{w}_k = \sum_{j=1}^{k-1} c_j H^T \mathbf{w}_j + c_k \sum_{j=1}^{k} u_{j,k} \mathbf{w}_j
\]

which is an element of \( K_k(H^T, \mathbf{v}) \). Thus, \( K_{k+1}(H^T, \mathbf{v}) \subseteq K_k(H^T, \mathbf{v}) \). ‘\( \supseteq \)’ is trivial, hence \( K_k(H^T, \mathbf{v}) = K_{k+1}(H^T, \mathbf{v}) \). By induction, the theorem now follows.

The consequence of this theorem is that if the algorithm stops at iteration \( k \), we have made a basis of \( K_k(H^T, \mathbf{v}) = K_m(H^T, \mathbf{v}) \). By Theorem 37, this space contains the PageRank vector so there is no reason to expand this space.

Let \( \mathbf{W} \) be the \( n \times m \) matrix containing each Arnoldi vector \( \mathbf{w}_i \) on \( i \)'th column. Let \( \mathbf{U} \) be the \( m \times m \) matrix with coefficients \( u_{i,j} \). Note that \( \mathbf{U} \) is an upper Hessenberg matrix (i.e. \( U_{ij} \neq 0 \) implies \( i \leq j + 1 \)). Then it is possible[11, p161] to write Arnoldi Algorithm in a single equation as

\[
H^T \mathbf{W} = \mathbf{W} \mathbf{U} + u_{m+1,m} \mathbf{w}_{m+1} \mathbf{e}_m^T
\]

Here \( \mathbf{e}_m \) is the standard \( m \)'th unit \( m \times 1 \) vector. To be able to approximate the PageRank vector we will look for an approximation \( \mathbf{u} \in K_m(H^T, \mathbf{v}) \) such that \( (I - \alpha H^T) \mathbf{u} \approx \mathbf{v} \). Since \( \mathbf{u} \in K_m(H^T, \mathbf{v}) \), we can write \( \mathbf{u} = a_1 \mathbf{w}_1 + a_2 \mathbf{w}_2 + \ldots + a_m \mathbf{w}_m \). In shorter notation \( \mathbf{u} = \mathbf{W} \mathbf{a} \), where \( \mathbf{a} = [a_1, a_2, \ldots, a_m]^T \). Our goal is to find good coefficients, i.e. coefficient \( \mathbf{a} \) such that the residual is very small. By definition, this residual \( \mathbf{r} \) is equal to

\[
\mathbf{r} = \mathbf{v} - (I - \alpha H^T) \mathbf{u} = \mathbf{v} - \mathbf{u} + \alpha H^T \mathbf{u}
\]

36
Since \( \mathbf{v} \in \mathcal{K}_m(H^T, \mathbf{v}) \) and \( \mathbf{u} \in \mathcal{K}_m(H^T, \mathbf{v}) \), it follows that \( \mathbf{r} \in \mathcal{K}_{m+1}(H^T, \mathbf{v}) \). Hence we can write \( \mathbf{r} = b_1 \mathbf{w}_1 + \ldots + b_{m+1} \mathbf{w}_{m+1} \) for some coefficients \( b_i \). Because the vectors \( \mathbf{w} \) are orthonormal, the 2-norm of this residual is simply equal to \( (\sum_{i=1}^{m+1} |b_i|^2)^{1/2} \). Thus, the best approximation \( \mathbf{u} \) can be achieved by minimising this sum. We will take a slightly different approach. Instead, we can pick the vector \( \mathbf{a} \) in such a way that \( b_i = 0 \) for all \( 1 \leq i \leq m \). This can be seen as a projection of \( \mathbf{x}(\alpha) \) onto \( \mathcal{K}_m(H^T, \mathbf{v}) \). To do so, note that

\[
\mathbf{r} = \mathbf{v} - (I - \alpha H^T)W \mathbf{a} = \mathbf{v} - W \mathbf{a} + \alpha H^T W \mathbf{a} = \mathbf{v} - W \mathbf{a} + \alpha(I(HU + u_{m+1,m} \mathbf{u}_{m+1}^T) + 1) \mathbf{a}
\]

Since \( \mathbf{w}_1 = \mathbf{v}/\|\mathbf{v}\|_2 \), we have \( \|\mathbf{v}\|_2 W \mathbf{e}_1 = \mathbf{v} \). Therefore

\[
\mathbf{r} = W[\|\mathbf{v}\|_2 \mathbf{e}_1 - (I - \alpha U) \mathbf{a}] + \alpha u_{m+1,m} (\mathbf{e}_m^T \mathbf{a}) \mathbf{w}_{m+1}
\]

Define \( \mathbf{a} = \|\mathbf{v}\|_2 (I - \alpha U)^{-1} \mathbf{e}_1 \). We find that

\[
\mathbf{r} = \alpha u_{m+1,m} \mathbf{e}_m^T (I - \alpha U)^{-1} \mathbf{e}_1 \mathbf{w}_{m+1}
\]

So in other words, \( b_1 = b_2 = \ldots = b_m = 0 \). The residual is only \( b_{m+1} \), which is equal to \( b_{m+1} = \alpha u_{m+1,m} \mathbf{e}_m^T (I - \alpha U)^{-1} \mathbf{e}_1 = \alpha u_{m+1,m} c(\alpha) \), where \( c(\alpha) \) is the bottom-left element of the matrix \( (I - \alpha U)^{-1} \). Thus we are able to approximate \( \mathbf{x}(\alpha) \) (and therefore \( \pi(\alpha) \) as well) by simply calculating \( \mathbf{a}_m \). This requires the computation of the inverse of \( I - \alpha U \). The main advantage of this method is that this matrix is only \( m \times m \) large, where \( m \) is usually much smaller than \( n \). Hence this is called a reduced-order method. Inverting an \( m \times m \) matrix is generally much faster than computing matrix-vector products of size \( n \).

In short, if we want to calculate \( \pi(\alpha) \) for multiple values of \( \alpha \), we can apply the following algorithm:

**Algorithm 41.**

1. Pick \( m \) large enough.
2. Calculate the orthonormal matrix \( W \) and upper Hessenberg matrix \( U \) by using the Arnoldi algorithm applied to \( H^T \) and \( \mathbf{v} \) for \( m \) iterations.
3. For each value of \( \alpha \), approximate \( \mathbf{x}(\alpha) \) by

\[
\mathbf{x}(\alpha) \approx W \mathbf{a}_m = W \|\mathbf{v}\|_2 (I - \alpha U)^{-1} \mathbf{e}_1
\]

4. Finally, normalise this vector to get an approximation of the PageRank vector:

\[
\pi(\alpha) = \frac{\mathbf{x}(\alpha)}{\mathbf{e}^T \mathbf{x}(\alpha)}
\]

It is important to note that the Arnoldi algorithm is independent on the value of \( \alpha \). Thus, the Arnoldi algorithm only needs to be applied once. After we have done this, we can approximate \( \pi(\alpha) \) very efficiently for many different values of \( \alpha \). But how large does \( m \) need to be? We expect that the residual decreases as \( m \) increases, as the dimension of the Krylov space increases and the first \( m \) coefficients \( b_i \) of the residual are made \( 0 \). The following numerical results confirm this hypothesis and illustrate how large \( m \) needs to be.
Figure 9: The residual of the approximation of $\pi$ for different values of $(\alpha, m)$. This has been calculated using the $5000 \times 5000$ matrix $A$, see section 5.1. Note that the residual is approximately equal to the machine error if $m$ is large or $\alpha$ is small. The red line corresponds with a residual of $10^{-5}$.

Figure 9 shows the relative residual of the approximation of the PageRank vector for many different values of $m$ and $\alpha$. As one can see, the residual gets smaller if either $\alpha$ gets smaller or $m$ gets larger. Suppose that we require the residual to be less than $10^{-5}$. Then $\pi(\alpha)$ can be approximated for any combination of $(\alpha, m)$ below the red line such that the residual is less than $10^{-5}$. Suppose that we want to calculate $\pi(\alpha)$ for $\alpha = 0.01, 0.02, \ldots, 0.95$. The figure shows that $m = 80$ should give sufficient results.

Clearly, it is easy to see how large $m$ needs to be after we have applied the algorithm. In general, one needs to pick $m$ before applying the algorithm. A good way to do so is by creating the space $K_m(H^T, v)$ for a certain initial guess $m$ by using the Arnoldi algorithm. Then the approximation of the PageRank vector corresponding to the largest value of $\alpha$ should be calculated by using algorithm 41. If the residual of this approximation is larger than requested, $m$ is too small. One can then expand $K_m(H^T, v)$ by simply applying the Arnoldi algorithm further. We repeat this until the residual is made small enough. In the next section we will test this algorithm numerically.

6.4 Numerical experiments

We will compare this Reduced-Order Algorithm with the Optimal Power Method. Furthermore, to make the differences in computation time clearer, we will also test the normal Power Method and the Jacobi Method applied on $H$. 
The goal of each algorithm is to approximate the expected PageRank vector $\langle \pi \rangle$. As we have stated before, we do this by calculating a Riemann sum. This requires mesh points for $\alpha$. For each value of $\alpha$, $\pi(\alpha)$ is calculated using one of the numerical methods. The speed of each method depends on the values of the mesh points for $\alpha$. We will try multiple sets of $\alpha$ (for example, many different values of $\alpha$ and/or allowing values close to 1). The numerical methods will be applied to matrix $A$, $B$, $C$ and $D$ as can be found in paragraph 5.2.

We have assumed that $f(\alpha)$ is small if $\alpha$ is close to one. Therefore, we can choose not to calculate $\pi(\alpha)$ for such values of $\alpha$ without the total error getting too large. Suppose for example that we have mesh points $\alpha = 0.00, 0.01, \ldots, 0.90$. Table 5 shows the computation time of the expected PageRank vector using these numerical methods.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Reduced Order</th>
<th>Optimized Power M.</th>
<th>Power Method</th>
<th>Jacobi ($H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 40$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A$</td>
<td>0.154s</td>
<td>6.646s</td>
<td>7.250s</td>
<td>7.735s</td>
</tr>
<tr>
<td></td>
<td>(1038 iterations)</td>
<td>(1511 iterations)</td>
<td>(1782 iterations)</td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td>0.044s</td>
<td>0.207s</td>
<td>0.431s</td>
<td>0.263s</td>
</tr>
<tr>
<td></td>
<td>($m = 25$)</td>
<td>(623 iterations)</td>
<td>(1520 iterations)</td>
<td>(1339 iterations)</td>
</tr>
<tr>
<td>$C$</td>
<td>3.338s</td>
<td>39.02s</td>
<td>45.88s</td>
<td>52.00s</td>
</tr>
<tr>
<td></td>
<td>($m = 40$)</td>
<td>(1299 iterations)</td>
<td>(1660 iterations)</td>
<td>(1906 iterations)</td>
</tr>
<tr>
<td>$D$</td>
<td>7.459s</td>
<td>53.47s</td>
<td>63.79s</td>
<td>68.78s</td>
</tr>
<tr>
<td></td>
<td>($m = 40$)</td>
<td>(1307 iterations)</td>
<td>(1690 iterations)</td>
<td>(1888 iterations)</td>
</tr>
</tbody>
</table>

Table 5: Computation times for the expected PageRank $\langle \pi \rangle$ using mesh points $\alpha = 0.00, 0.01, \ldots, 0.90$

It appeared that for $m = 40$ (or even $m = 25$ for matrix $B$) the Krylov space $K_m(H^T, v)$ is large enough to approximate $\pi(\alpha)$ for all mesh points (i.e. for each mesh point, the residual is less than $10^{-5}$). The Reduced-Order algorithm clearly is the fastest method, usually being around 10 times as fast as any other algorithm. However, the real power of the Reduced-Order algorithm becomes even more noticeable if a more precise approximation of $\langle \pi \rangle$ is requested. In this case, more mesh points are needed.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Reduced Order</th>
<th>Optimized Power M.</th>
<th>Power Method</th>
<th>Jacobi ($H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 40$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A$</td>
<td>0.264s</td>
<td>60.46s</td>
<td>71.62s</td>
<td>78.96s</td>
</tr>
<tr>
<td></td>
<td>($m = 40$)</td>
<td>(9768 iterations)</td>
<td>(14799 iterations)</td>
<td>(17473 iterations)</td>
</tr>
<tr>
<td>$B$</td>
<td>0.181s</td>
<td>1.014s</td>
<td>4.012s</td>
<td>2.579s</td>
</tr>
<tr>
<td></td>
<td>($m = 25$)</td>
<td>(2641 iterations)</td>
<td>(14874 iterations)</td>
<td>(13064 iterations)</td>
</tr>
<tr>
<td>$C$</td>
<td>12.31s</td>
<td>314.0s</td>
<td>461.8s</td>
<td>505.2s</td>
</tr>
<tr>
<td></td>
<td>($m = 40$)</td>
<td>(8613 iterations)</td>
<td>(16234 iterations)</td>
<td>(18635 iterations)</td>
</tr>
<tr>
<td>$D$</td>
<td>30.77s</td>
<td>426.2s</td>
<td>625.3s</td>
<td>685.2s</td>
</tr>
<tr>
<td></td>
<td>($m = 40$)</td>
<td>(9003 iterations)</td>
<td>(16532 iterations)</td>
<td>(18527 iterations)</td>
</tr>
</tbody>
</table>

Table 6: Computation times for the expected PageRank $\langle \pi \rangle$ using mesh points $\alpha = 0.00, 0.001, \ldots, 0.899, 0.90$
Table 6 shows the computation time of each algorithm when $\langle \pi \rangle$ is calculated using a more precise grid $\alpha = 0.0, 0.001, \ldots, 0.90$. Clearly, the difference between the numerical methods is huge; sometimes the Reduced-Order algorithm is more than 200 times faster than any other algorithm. This is because this algorithm only needs to solve an $m \times m$ linear equation for each value of $\alpha$. The other algorithms require the computation of multiple matrix-vector products with $n$ elements for each value of $\alpha$. Since $n$ is much larger than $m$, this computation takes much longer.

The Reduced-Order algorithm has one disadvantage: large values of $\alpha$. Even though $f(\alpha) \approx 0$ if $\alpha \approx 1$, the contribution of $\pi(0.99)$ to the expected PageRank $\langle \pi \rangle$ might be significant. Thus, a larger mesh grid is required. As we have seen in previous paragraph, one needs to expand the Krylov space to prevent the residual of this approximation to become too large.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Reduced Order</th>
<th>Optimized Power M.</th>
<th>Power Method</th>
<th>Jacobi ($H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0.408s</td>
<td>10.36s</td>
<td>11.25s</td>
<td>12.18s</td>
</tr>
<tr>
<td></td>
<td>($m = 100$)</td>
<td>(2477 iterations)</td>
<td>(2891 iterations)</td>
<td>(3390 iterations)</td>
</tr>
<tr>
<td>$B$</td>
<td>0.073s</td>
<td>0.234s</td>
<td>0.869s</td>
<td>0.519s</td>
</tr>
<tr>
<td></td>
<td>($m = 40$)</td>
<td>(641 iterations)</td>
<td>(3513 iterations)</td>
<td>(2788 iterations)</td>
</tr>
<tr>
<td>$C$</td>
<td>34.48s</td>
<td>60.19s</td>
<td>83.06s</td>
<td>81.73s</td>
</tr>
<tr>
<td></td>
<td>($m = 180$)</td>
<td>(2470 iterations)</td>
<td>(3736 iterations)</td>
<td>(3606 iterations)</td>
</tr>
<tr>
<td>$D$</td>
<td>103.0s</td>
<td>90.33s</td>
<td>125.5s</td>
<td>115.0s</td>
</tr>
<tr>
<td></td>
<td>($m = 190$)</td>
<td>(2538 iterations)</td>
<td>(3849 iterations)</td>
<td>(3572 iterations)</td>
</tr>
</tbody>
</table>

Table 7: Computation times for the expected PageRank $\langle \pi \rangle$ using mesh points $\alpha = 0.00, 0.01, \ldots, 0.99$

Suppose that we use $\alpha = 0, 0.01, \ldots, 0.99$ as our mesh grid. The Reduced Order algorithm will need a larger Krylov space to keep the residual below $10^{-5}$. For matrix $D$ for example, this space should be expanded from $m = 40$ to $m = 190$. By [11, p165], the Arnoldi algorithm is computationally very expensive; the total amount of flops is of the order $O(nm^2)$. Each new vector is made orthogonal to every other vector. This explains why the Reduced Order algorithm can become slightly slower than the Optimized Power Method.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Reduced Order</th>
<th>Optimized Power M.</th>
<th>Power Method</th>
<th>Jacobi ($H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0.687s</td>
<td>78.59s</td>
<td>107.7s</td>
<td>119.6s</td>
</tr>
<tr>
<td></td>
<td>($m = 100$)</td>
<td>(14888 iterations)</td>
<td>(26658 iterations)</td>
<td>(32608 iterations)</td>
</tr>
<tr>
<td>$B$</td>
<td>0.284s</td>
<td>1.140s</td>
<td>8.205s</td>
<td>4.997s</td>
</tr>
<tr>
<td></td>
<td>($m = 40$)</td>
<td>(2821 iterations)</td>
<td>(31461 iterations)</td>
<td>(26427 iterations)</td>
</tr>
<tr>
<td>$C$</td>
<td>74.99s</td>
<td>400.6s</td>
<td>768.4s</td>
<td>812.3s</td>
</tr>
<tr>
<td></td>
<td>($m = 180$)</td>
<td>(12173 iterations)</td>
<td>(33575 iterations)</td>
<td>(34727 iterations)</td>
</tr>
<tr>
<td>$D$</td>
<td>213.6s</td>
<td>582.9s</td>
<td>1181s</td>
<td>1185s</td>
</tr>
<tr>
<td></td>
<td>($m = 190$)</td>
<td>(12485 iterations)</td>
<td>(34522 iterations)</td>
<td>(34447 iterations)</td>
</tr>
</tbody>
</table>

Table 8: Computation times for the expected PageRank $\langle \pi \rangle$ using mesh points $\alpha = 0.00, 0.001, \ldots, 0.989, 0.99$

However, if we want a better approximation of $\langle \pi \rangle$, the amount of mesh points should be increased. As we have seen before, the Reduced Order method becomes by far the fastest way of calculation $\pi(\alpha)$ for these mesh points.
6.5 Discussion

To approximate the expected PageRank vector $\langle \pi \rangle$, one needs to calculate $\pi(\alpha)$ for many different values of $\alpha$. The Reduced Order algorithm we have discussed is a very efficient way of doing so. In some of our experiments, this algorithm is as much as 200 times faster than any other method we have discussed.

The power of the Reduced Order algorithm lies in the fact that the Arnoldi algorithm should only be applied once. After that, we only need to solve the simple $m \times m$ system $(I - \alpha U)x = e_1$. We have seen that $m$ can usually be very small compared to $n$ (in our examples $m = 200$ was sufficient), thus this system can be solved very quickly.

If $\pi(\alpha)$ for $\alpha$ close to 1 should be calculated using this Reduced Order algorithm, the Krylov space should be made larger. We have seen that is computationally very expensive, and in some cases (when using a small amount of mesh points but allowing values close to 1) this might not be worth it. Another option we have not discussed is keeping $m$ small but applying some iterations of the (Optimized) Power Method to the approximations of $\pi(\alpha)$ for values of $\alpha$ close to one.
7 Conclusion & Discussion

In this paper we have discussed the PageRank model. We have illustrated the idea behind PageRank with the help of a random surfer. To make sure this vector is uniquely determined, modifications to this model have been made; artificial transition probabilities for dangling nodes have been implemented, as well as a probability to teleport to any web page on the internet.

Originally defined as the unique stationary distribution of a Markov chain, we defined the PageRank vector in a more rigorous way with the help of some linear algebra; the PageRank vector is defined as an eigenvector of the Google matrix corresponding to eigenvalue 1. The problem was to find an efficient way to calculate this eigenvector. We have discussed several numerical methods for this.

The first algorithm we have discussed is the well-known Power Method. This algorithm is often applied for PageRank problem. The error of this algorithm is of the order $O(\alpha^k)$, where $\alpha$ is the teleportation parameter and $k$ stands for the number of iterations. Furthermore, we have shown that the PageRank vector $\pi$ is equal to

$$\pi = (1 - \alpha)(I - \alpha S^T)^{-1}v$$

By noting that $(I - \alpha S^T)^{-1} = \sum_{n=0}^{\infty} (\alpha S^T)^n$, we proposed to approximate this series (and thus the PageRank vector) by a partial sum. The Jacobi Method is an efficient way of doing so. Moreover, we have shown that it is sufficient to solve the system $(I - \alpha H^T)x = v$. The PageRank vector can be calculated by normalising this solution. Thus it is also possible to approximate the PageRank vector by applying the Jacobi Method on the matrix $H^T$.

Additionally, we have shown that in some cases the Jacobi Method can be optimized. This depends on the spectrum of the hyperlink matrix. By a shift-and-scale approach, one can apply the Jacobi Method on a different matrix with a smaller spectral radius. However, the algorithm turned out to be of limited use. In most cases the regular Jacobi Method was already optimal.

In chapter 4 we have compared these algorithms numerically. For this, we have used 4 different hyperlink matrices, varying greatly in size and structure. One of these has been computed by using a surfer. For larger-scale testing purposes we have also used hyperlink matrices corresponding to a subset of the web with almost 700 thousand web pages. The three numerical methods turned out to have roughly the same speed. The Jacobi Method applied on $S$ is always the slowest and the Power Method and the Jacobi Method applied on $H$ often have the same speed. In some cases it is better to apply the Jacobi Method, for example when the amount of dangling nodes is relatively large.

The PageRank model has two important parameters: the teleportation parameter $\alpha$ and the personalisation vector $v$. For each person, the personalisation vector can be changed such that the PageRank vector better suits the interests of a person. The parameter $\alpha$ corresponds to the probability that a user clicks a link; this value can also depend on the user. Hence, in chapter 5 we have made a modification to the PageRank model and assumed $\alpha$ to be a stochastic variable. Now the PageRank vector is a function of $\alpha$. The goal was to approximate the expected PageRank:

$$\langle \pi \rangle = \int_0^1 \pi(\alpha)f(\alpha)d\alpha$$
Here $f$ corresponds to the probability density function of $\alpha$. To approximate this vector, $\pi(\alpha)$ should be calculated for many different values of $\alpha$. We have introduced a reduced-order algorithm that can do that very efficiently. This algorithm makes use of the shift-invariance of the Krylov space $K_m(H^T, v)$. An orthonormal basis of this space can be found by applying the Arnoldi algorithm. We have shown that the residual of the linear system $(I - \alpha H^T)x = v$ lives in the Krylov space $K_{m+1}(H^T, v)$ for any vector $x \in K_m(H^T, v)$, and by smartly picking this vector $x$, one can make the residual of the linear system very small. The vector $x$ can then be used to approximate $\pi(\alpha)$.

The only disadvantage of the Reduced Order algorithm is that one needs to expand the Krylov space if $\pi(\alpha)$ is requested for values $\alpha$ close to 1. Expanding this space is computationally very expensive.

The power of the Reduced Order algorithm lies in the fact that the basis of $K_m(H^T, v)$ should only be made once using the Arnoldi algorithm. Then one can find an approximation of $\pi(\alpha)$ for each value of $\alpha$ by simply solving multiple systems with only $m$ variables, where $m$ is usually much smaller than $n$. The numerical experiments in paragraph 6.4 showed that this algorithm is extraordinary effective. Instead of using the (Optimized) Power Method or Jacobi Method iteratively, one can make the computation of the expected PageRank vector $\langle \pi \rangle$ up to 200 times faster by applying this Reduced Order algorithm.
8 References

References


A Appendix

A.1 Constructing the hyperlink/Google matrices

The following Matlab files have been created to be able to construct link matrices. Furthermore, the Google matrix can be computed with these scripts.

A.1.1 surfer.m

function 

\[
\begin{align*}
[H, \text{names}] &= \text{surfer}(\text{root}, n) \\
\text{names} &= \text{cell}(n, 1); \\
\text{names}{\{}1\}\} &= \text{root}; \\
m &= 1; \quad \%\text{current amount of url's} \\
H &= \text{logical}(\text{sparse}(n, n)); \\
\text{banned} &= \{'.jpg', '.jpeg', '.png', '.bmp', '.tif', '.gif', '.ico', '.css', \\
&\quad '.js', \ldots \\
&\quad '.cgi', '.pdf', '.doc', '.pps', '.ppt', '.odt', '.rar', '.tar', '.dat', \\
&\quad '.exe', \ldots \\
&\quad '.jar', '.xml', 'lmscadsi', 'cybernet', 'w3.org', 'google', 'yahoo', \\
&\quad 'scripts', \ldots \\
&\quad 'netscape', 'shockwave', 'webex', 'fansonly', 'doubleclick', '#', " \} ; \%\text{banned files} \\
\text{truncate} &= \{'?','#'\}; \\
\text{for} \ j = 1:n \\
\text{try} \ \text{page} &= \text{uurlread}(\text{names}{\{}j\}\}); \\
\text{end} \\
\text{for} \ f = \text{strfind}(\text{page}, 'href="http:') \\
\text{link} &= \text{page}(f+6:f+4+\min(\text{strfind}(\text{page}(f+6:end), '" '))) ; \\
\text{for} \ i = 1:\text{length}(\text{truncate}) \\
\text{pos} &= \min(\text{strfind}(\text{link}, \text{truncate}{\{}i\}\}); \\
\text{if}(\text{isempty}(\text{pos})) \\
\text{link} &= \text{link}(1:\text{pos}-1); \\
\text{end} \\
\text{end} \\
\text{for} \ i = 1:m \\
\text{if}(\text{strcmpi}(\text{names}{\{}i\}\}, \text{link}))
% The url is known!
known = true;
H(j, i) = 1;
end
end

% If the url is not known, adding this link
if (~known && m ~= n)
% Is the url allowed?
skip = false;
for str = banned
    if (~isempty(strfind(link, str{1})))
        skip = true;
        break;
    end
end
if (~skip)
    m = m + 1;
    names{m} = link;
    H(m, j) = 1;
end
end

catch
    % We couldn’t open the url. Continuing.
end
end
end

A.1.2 loadCaliforniaMatrix.m

function [H, names] = loadCaliforniaMatrix
% Computes the hyperlink matrix of the California web (see
% http://www.cs.cornell.edu/Courses/cs685/2002fa/data/gr0.
% California).

fid = fopen(’gr0.California’);
filme = fgets(fid);

% Adding the nodes
names = {};
while ischar(tline) && tline(1) == ’n’
    pos = strfind(tline,’’);
    i = tline(pos(1)+1:pos(2)-1);
    j = tline(pos(2)+1:end);
    names = [names; cellstr(j)];
    filme = fgets(fid);
end

% Adding the arcs
i = []; j = [];  
while ischar(tline) && tline(1)=='e'  
    pos = strfind(tline,' ');  
    i = [i; str2num(tline(pos(1)+1:pos(2)-1))];  
    j = [j; str2num(tline(pos(2)+1:end))];  
    tline = fgets(fid);  
end  

i = i+1;  
j = j+1;  
n = length(names);  
fclose(fid);  

%Creating the (sparse) hyperlink matrix  
H = sparse(i,j,1,n,n);  
H = spdiags(1./max(1,sum(H,2)),0,n,n)*H;  
end  

A.1.3 loadStanfordMatrix.m  

function H = loadStanfordMatrix  
% Computes the hyperlink matrix of the Stanford University web (see  
% http://www.kamvar.org/assets/data/stanford-web.tar.gz).  
load stanford-web.dat;  
H = spconvert(stanford_web);  
end  

A.1.4 loadSBMatrix.m  

function [H, rooturls] = loadSBMatrix2  
% Computes the hyperlink matrix of the Stanford-Berkeley Web (see  
% http://www.kamvar.org/assets/data/stanford-berkeley-web.tar.gz).  

n = 683446;  
load stanford-berkeley-bool-sorted.dat;  
H = spconvert(stanford_berkeley_bool_sorted);  
% make the matrix square  
H(n,n)=0;  
H = H(1:n,1:n);  
% normalize the rows to sum to 1  
H = spdiags(1./max(1,sum(H,2)),0,n,n)*H;  
load stanford-berkeley-sorted-roots.dat;  
indices = stanford_berkeley_sorted_roots;  
indices = indices(find(indices<n));  
rooturls = textread('rooturls.txt','%s');  
rooturls = rooturls(1:max(size(indices)));  
end
A.1.5 DanglingNodeVector.m

function d = DanglingNodeVector(H)
    % Computes the Dangling Node vector of the corresponding link matrix H.
    d = (sum(H,2) == 0);
end

A.1.6 GoogleMatrix.m

function G = GoogleMatrix(H, alpha, v)
    % Computes the Google Matrix corresponding to H, alpha and v
    % Note that G is not a sparse matrix; hence this method cannot be used for
    % large link matrices.
    n = length(H);
    if (nargin<3)
        v = ones(n,1)/n;
    end
    if (nargin<2)
        alpha = 0.85;
    end
    d = DanglingNodeVector(H);
    G = alpha*(H + d*v') + (1-alpha)*ones(n,1)*v';
end

A.2 Numerical Algorithms

The next Matlab files are the numerical methods we have discussed in chapter 4.

A.2.1 PowerMethod.m

function [pi, iter] = PowerMethod(H, alpha, v, startvector, error)
    % Approximates the PageRank vector such that the residual
    % is less than error. This uses the Power Method with
    % startvector as
    % initial guess
    n = length(H);
    e = ones(n,1);
    d = DanglingNodeVector(H);
    norm = @(x) sum(abs(x)); % l1-norm
    if(nargin < 5)
        error = 1e-5;
    end
    if(nargin < 3)
        v = e / n;
    end
if(nargin < 4)
    startvector = v;
end
if(nargin < 2)
    alpha = 0.85;
end
K = alpha*H';
pi = startvector; %initial guess
max = 300; %maximum amount of iterations
for iter = 2:max
    piprevious = pi;
    pi = K*pi + (1-alpha+alpha*sum(d.*pi))*v;
    res = norm(pi-piprevious);
    if(res < error)
        break;
    end
end

A.2.2 JacobiMethodS.m

function [pi, iter] = JacobiMethodS(H, alpha, v, error)
% Approximates the PageRank vector such that the residual
% is less than error. This uses the Jacobi method applied
% to S

n = length(H);
e = ones(n,1);
d = DanglingNodeVector(H);
norm = @(x) sum(abs(x)); %l–norm
if(nargin < 4)
    error = 1e-5;
end
if(nargin < 3)
    v = e / n;
end
if(nargin < 2)
    alpha = 0.85;
end
K = alpha*H';
pi = (1-alpha)*v;
max = 300; %maximum amount of iterations
alphaspi = K*pi + sum(d.*pi)*alpha*v;
for iter = 2:max
    pi = alphaspi + (1-alpha)*v;
    alphaspi = K*pi + sum(d.*pi)*alpha*v;
    res = norm(pi-alphaspi - (1-alpha)*v);
if (res < error)
    break;
end
end

A.2.3 JacobiMethodH.m

function [pi, iter] = JacobiMethodH(H, alpha, v, error)
    % Approximates the PageRank vector such that the residual
    % is less than error. This uses the Jacobi method applied
    % to H

    n = length(H);
    e = ones(n,1);
    norm = @(x) sum(abs(x));  \% 1-norm
    if (nargin < 4)
        error = 1e-5;
    end
    if (nargin < 3)
        v = e / n;
    end
    if (nargin < 2)
        alpha = 0.85;
    end

    K = alpha*H';
    pi = v;
    max = 300;  \% maximum amount of iterations
    alphahpi = K*pi;
    for iter = 2:max
        pi = alphahpi + v;
        alphahpi = K*pi;
        res = norm(pi-alphahpi - v);
        if (res < error*sum(pi))
            break;
        end
    end
    pi = pi / sum(pi);
end

A.2.4 OptimizedJacobiMethodH.m

function [pi, iter] = OptimizedJacobiMethodH(H, beta, alpha, v, error)
    % Approximates the PageRank vector such that the residual
    % is less than error. This uses the Optimized Jacobi method
    % applied
    % to H, with (required) shifting parameter beta

    n = length(H);
e = ones(n,1);

norm = @(x) sum(abs(x));  %l--norm
if(nargin < 5)
    error = 1e-5;
end
if(nargin < 4)
    v = e / n;
end
if(nargin < 3)
    alpha = 0.85;
end

K = H';
max = 300;  %maximum amount of iterations
pi = beta*(1-alpha)*v;
Qpi = (1-beta)*pi+(beta*alpha)*(K*pi);
for iter = 2:max
    pi = Qpi + v;
    Qpi = (1-beta)*pi+(beta*alpha)*(K*pi);
    res = norm(pi-Qpi - v);
    if(res < error*sum(pi))
        break;
    end
end
pi = pi/sum(pi);

A.2.5 OptimalBeta.m

function optbeta = BestBeta(H,betas,alpha,v,error)
% For each value of beta, the PageRank is computed using the
% Optimized
% Jacobi Method. Returns the value of beta such that the total
% iterations needed is minimal.

n = length(H);
if(nargin < 5)
    error = 1e-5;
end
if(nargin < 4)
    v = ones(n,1) / n;
end
if(nargin < 3)
    alpha = 0.85;
end

opt = Inf;
optbeta = NaN;
for beta = betas
\[
\text{iter} = \text{OptimizedJacobiMethodH}(H, beta, alpha, v, error);
\]

if (iter < opt)
    opt = iter;
    optbeta = beta;
end
end

A.2.6 Compare.m

function Compare(H, alpha, v, beta)

% Compares the numerical methods discussed in chapter 3. If no value of
% beta is given (or beta=1), the Optimized Jacobi Method will not be used.

n = length(H);
if (nargin < 4)
    beta = 1; % the Optimized Jacobi Method will not be used
end
if (nargin < 3)
    v = ones(n,1)/n;
end
if (nargin < 2)
    alpha = 0.85;
end

% Power Method:
tic; [~, iter] = PowerMethod(H, alpha, v);
disp(['Power Method: ', num2str(toc), 's (', num2str(iter), ' iterations)']);

% Jacobi S
tic; [~, iter] = JacobiMethodS(H, alpha, v);
disp(['Jacobi Method (S): ', num2str(toc), 's (', num2str(iter), ' iterations)']);

% Jacobi H
tic; [~, iter] = JacobiMethodH(H, alpha, v);
disp(['Jacobi Method (H): ', num2str(toc), 's (', num2str(iter), ' iterations)']);
if (beta == 1)
% Optimized Jacobi H
tic; [~, iter] = OptimizedJacobiMethodH(H, beta, alpha, v);
disp(['Optimized Jacobi Method (beta=', num2str(beta), '): ', num2str(toc), 's (', num2str(iter), ' iterations)']);
end
end
A.3 Numerical Methods for the expected PageRank vector

The next Matlab files have been made and used to approximate the expected PageRank vector.

A.3.1 OptimizedPowerMethod.m

function \([\pi, \text{iter}] = \text{OptimizedPowerMethod}(H, \text{alphas}, v)\)
\% Approximates \(\pi(\text{alphas})\) for the given values of alpha by using the 
\% Optimized Power Method. For the best results, we assume that 
\% alphas is 
\% strictly increasing.

\n\n\text{n = length}(H);
\text{if} (\text{nargin}<3)
\hspace{1em}v = \text{ones}(n, 1)/n;
\text{end}
\text{m = length}(\text{alphas});
\text{iter} = 0;

\pi = \text{zeros}(n, m);
\pi(:, 1) = \text{PowerMethod}(H, \text{alphas}(1), v);
\text{for} \text{i} = 2: \text{length}(\text{alphas})
\hspace{1em}[\pi(:, i), k] = \text{PowerMethod}(H, \text{alphas}(i), v, \pi(:, i-1));
\hspace{1em}\text{iter} = \text{iter} + k;
\text{end}
\text{end}

A.3.2 Arnoldi.m

function \([W, U] = \text{Arnoldi}(A, m, v)\)
\% Applies the Arnoldi Algorithm on the matrix A. \(W\) is the 
\% constructed 
\% orthogonal matrix, \(U\) the corresponding upper Hessenberg matrix.

\n\n\text{n = length}(A);
\text{if} (\text{nargin}<3)
\hspace{1em}v = \text{ones}(n, 1)/n;
\text{end}
\text{W} = \text{zeros}(n, m);
\text{norm} = @(x) \text{sqrt}(\text{sum}(\text{abs}(x.^2)));
\text{W}(:, 1) = v / \text{norm}(v);
\text{U} = \text{zeros}(m, m);
\text{for} \text{k} = 1:m
\hspace{1em}z = A\text{W}(:, k);
\hspace{1em}\text{for} \text{j} = 1:k
\hspace{2em}\text{U}(j,k) = \text{sum(W(:, j).*z)};
\hspace{1em}\text{end}
\hspace{1em}\text{end}
\[ z = z - U(j,k) \ast W(:,j); \]

\[ \text{if}(j==m) \]
\[ \text{break; } \]

\[ U(k+1,k) = \text{norm}(z); \]
\[ \text{if}(U(k+1,k)==0) \]
\[ \text{disp('The Arnoldi Method broke down!')}; \]
\[ m = k; \]
\[ \text{break; } \]

\[ W(:,k+1) = z / U(k+1,k); \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{if}(\text{argn}<4) \]
\[ v = \text{ones}(n,1)/n; \]
\[ \text{end} \]
\[ e1 = [1; \text{zeros}(m-1,1)]; \]
\[ \text{pi} = \text{zeros}(n, \text{length}(\text{alphas})); \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{A.3.3 ReducedOrderMethod.m} \]
\[ \text{function } \text{pi} = \text{ReducedOrderMethod}(H, \text{alphas}, m, v) \]
\[ \text{% Approximates } \pi(\alpha) \text{ for the given values of } \alpha \text{ by using the} \]
\[ \text{% Optimized Power Method. For the best results, we assume that} \]
\[ \text{% strictly increasing.} \]
\[ n = \text{length}(H); \]
\[ \text{if}(\text{nargin}<4) \]
\[ v = \text{ones}(n,1)/n; \]
\[ \text{end} \]
\[ e1 = [1; \text{zeros}(m-1,1)]; \]
\[ \text{pi} = \text{zeros}(n, \text{length}(\text{alphas})); \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{A.3.4 Compare2.m} \]
\[ \text{54} \]
function Compare2(H, alphas, m, v)
% Approximates pi(alpha) for the given values of alpha by using
% the Optimized Power Method, the Jacobi Method applied on H and
% S, and finally by using the Reduced Order Algorithm. m stands
% for the size of the Krylov space. Note that for storage issues,
% no PageRank vector will be stored.

n = length(H);
if (nargin < 4)
    v = ones(n, 1) / n;
end

e1 = [1; zeros(m-1, 1)];

disp('Reduced Order Method: ');
tic;
% Calculating W and U using H'
[W, U] = Arnoldi(H', m);
for i = 1:length(alphas)
    pi = W * (eye(m) - alphas(i) * U) \ e1;
    pi = pi / sum(pi);
end
toc;

iter = 0;
disp('Optimal Power Method: ');
tic;
pi = v;
for i = 1:length(alphas)
    [pi, k] = PowerMethod(H, alphas(i), pi);
    iter = iter + k;
end
toc; iter

iter = 0;
disp('Normal Power Method: ');
tic;
for i = 1:length(alphas)
    [pi, k] = PowerMethod(H, alphas(i));
    iter = iter + k;
end
toc; iter

iter = 0;
disp('Normal Jacobi H Method: ');
tic;
for i = 1:length(alphas)
    [pi, k] = JacobiMethodH(H, alphas(i));
    iter = iter + k;
end
toc; iter

55
A.3.5 PlotResidual.m

function PlotResidual(H,ms,alphas,v)
    % Using the Krylov space of dimension m, this function plots the
    % residual of the approximation of pi(alpha) for each value of
    % alpha.

    n = length(H);
    if (nargin<4)
        v = ones(n,1)/n;
    end
    norm = @(x) sqrt(sum(abs(x.^2)));
    [m, alpha] = meshgrid(ms,alphas);
    residual = zeros(size(m));
    d = DanglingNodeVector(H);

    % For each value of m, we approximate pi(alpha) for all alpha.
    for k = 1:length(ms)
        pi = ReducedOrderMethod(H,alphas,ms(k),v);
        for j = 1:length(alphas)
            residual(j,k) = norm(pi(:,j)-(alphas(j)*H'*pi(:,j) + sum(d.*pi(:,j)*alphas(j)*v) - (1-alphas(j))*v) / norm(v);
        end
    end

    % Plotting the residual
    figure;
    xlabel('m');
    ylabel('alpha');
    zlabel('residual');
    set(gca, 'zscale', 'log')